Network Working Group
Request for Comments: 4718
Category: Informational

P. Eronen
Nokia
P. Hoffman
VPN Consortium

October 2006

IKEv2 Clarifications and Implementation Guidelines

Status of This Memo

Copyright Notice

Abstract

This document clarifies many areas of the IKEv2 specification. It does not to introduce any changes to the protocol, but rather provides descriptions that are less prone to ambiguous interpretations. The purpose of this document is to encourage the development of interoperable implementations.

Table of Contents

1. Introduction

This document clarifies many areas of the IKEv2 specification that may be difficult to understand to developers not intimately familiar with the specification and its history. The clarifications in this document come from the discussion on the IPsec WG mailing list, from experience in interoperability testing, and from implementation issues that have been brought to the editors' attention.

IKEv2/IPsec can be used for several different purposes, including IPsec-based remote access (sometimes called the "road warrior" case), site-to-site virtual private networks (VPNs), and host-to-host protection of application traffic. While this document attempts to consider all of these uses, the remote access scenario has perhaps received more attention here than the other uses.

This document does not place any requirements on anyone and does not use [RFC2119] keywords such as "MUST" and "SHOULD", except in quotations from the original IKEv2 documents. The requirements are given in the IKEv2 specification [IKEv2] and IKEv2 cryptographic algorithms document [IKEv2ALG].

In this document, references to a numbered section (such as "Section 2.15") mean that section in [IKEv2]. References to mailing list messages or threads refer to the IPsec WG mailing list at ipsec@ietf.org. Archives of the mailing list can be found at <http://www.ietf.org/mail-archive/web/ipsec/index.html>.

2. Creating the IKE_SA

2.1. SPI Values in IKE_SA_INIT Exchange

Normal IKE messages include the initiator's and responder's Security Parameter Indexes (SPIs), both of which are non-zero, in the IKE header. However, there are some corner cases where the IKEv2 specification is not fully consistent about what values should be used.

First, Section 3.1 says that the Responder's SPI "...MUST NOT be zero in any other message" (than the first message of the IKE_SA_INIT exchange). However, the figure in Section 2.6 shows the second IKE_SA_INIT message as "HDR(A,0), N(COOKIE)", contradicting the text in 3.1.

Since the responder's SPI identifies security-related state held by the responder, and in this case no state is created, sending a zero value seems reasonable.

Second, in addition to cookies, there are several other cases when the IKE_SA_INIT exchange does not result in the creation of an IKE_SA (for instance, INVALID_KE_PAYLOAD or NO_PROPOSAL_CHOSEN). What responder SPI value should be used in the IKE_SA_INIT response in this case?

Since the IKE_SA_INIT request always has a zero responder SPI, the value will not be actually used by the initiator. Thus, we think sending a zero value is correct also in this case.

If the responder sends a non-zero responder SPI, the initiator should not reject the response only for that reason. However, when retrying the IKE_SA_INIT request, the initiator will use a zero responder SPI, as described in Section 3.1: "Responder's SPI [...] This value MUST be zero in the first message of an IKE Initial Exchange (including repeats of that message including a cookie) [...]". We believe the intent was to cover repeats of that message due to other reasons, such as INVALID_KE_PAYLOAD, as well.

(References: "INVALID_KE_PAYLOAD and clarifications document" thread, Sep-Oct 2005.)

2.2. Message IDs for IKE_SA_INIT Messages

The Message ID for IKE_SA_INIT messages is always zero. This includes retries of the message due to responses such as COOKIE and INVALID_KE_PAYLOAD.

This is because Message IDs are part of the IKE_SA state, and when the responder replies to IKE_SA_INIT request with N(COOKIE) or N(INVALID_KE_PAYLOAD), the responder does not allocate any state.

(References: "Question about N(COOKIE) and N(INVALID_KE_PAYLOAD) combination" thread, Oct 2004. Tero Kivinen's mail "Comments of draft-eronen-ipsec-ikev2-clarifications-02.txt", 2005-04-05.)

2.3. Retransmissions of IKE_SA_INIT Requests

When a responder receives an IKE_SA_INIT request, it has to determine whether the packet is a retransmission belonging to an existing "half-open" IKE_SA (in which case the responder retransmits the same response), or a new request (in which case the responder creates a new IKE_SA and sends a fresh response).

The specification does not describe in detail how this determination is done. In particular, it is not sufficient to use the initiator's SPI and/or IP address for this purpose: two different peers behind a single NAT could choose the same initiator SPI (and the probability of this happening is not necessarily small, since IKEv2 does not require SPIs to be chosen randomly). Instead, the responder should do the IKE_SA lookup using the whole packet or its hash (or at the minimum, the Ni payload which is always chosen randomly).

For all other packets than IKE_SA_INIT requests, looking up right IKE_SA is of course done based on the recipient's SPI (either the initiator or responder SPI depending on the value of the Initiator bit in the IKE header).

2.4. Interaction of COOKIE and INVALID_KE_PAYLOAD

There are two common reasons why the initiator may have to retry the IKE_SA_INIT exchange: the responder requests a cookie or wants a different Diffie-Hellman group than was included in the KEi payload. Both of these cases are quite simple alone, but it is not totally obvious what happens when they occur at the same time, that is, the IKE_SA_INIT exchange is retried several times.

The main question seems to be the following: if the initiator receives a cookie from the responder, should it include the cookie in only the next retry of the IKE_SA_INIT request, or in all subsequent retries as well? Section 3.10.1 says that:

> "This notification MUST be included in an IKE_SA_INIT request retry if a COOKIE notification was included in the initial response."

This could be interpreted as saying that when a cookie is received in the initial response, it is included in all retries. On the other hand, Section 2.6 says that:

> "Initiators who receive such responses MUST retry the IKE_SA_INIT with a Notify payload of type COOKIE containing the responder supplied cookie data as the first payload and all other payloads unchanged."

Including the same cookie in later retries makes sense only if the "all other payloads unchanged" restriction applies only to the first retry, but not to subsequent retries.

It seems that both interpretations can peacefully coexist. If the initiator includes the cookie only in the next retry, one additional roundtrip may be needed in some cases:

```
   Initiator                    Responder
  -----------                  -----------
   HDR(A,0), SAi1, KEi, Ni -->
                            <-- HDR(A,0), N(COOKIE)
   HDR(A,0), N(COOKIE), SAi1, KEi, Ni  -->
                            <-- HDR(A,0), N(INVALID_KE_PAYLOAD)
   HDR(A,0), SAi1, KEi', Ni -->
                            <-- HDR(A,0), N(COOKIE')
   HDR(A,0), N(COOKIE'), SAi1, KEi',Ni -->
                            <-- HDR(A,B), SAr1, KEr, Nr
```

An additional roundtrip is needed also if the initiator includes the cookie in all retries, but the responder does not support this functionality. For instance, if the responder includes the SAi1 and KEi payloads in cookie calculation, it will reject the request by sending a new cookie (see also Section 2.5 of this document for more text about invalid cookies):

```
   Initiator                    Responder
  -----------                  -----------
   HDR(A,0), SAi1, KEi, Ni -->
                            <-- HDR(A,0), N(COOKIE)
   HDR(A,0), N(COOKIE), SAi1, KEi, Ni  -->
                            <-- HDR(A,0), N(INVALID_KE_PAYLOAD)
   HDR(A,0), N(COOKIE), SAi1, KEi', Ni -->
                            <-- HDR(A,0), N(COOKIE')
   HDR(A,0), N(COOKIE'), SAi1, KEi',Ni -->
                            <-- HDR(A,B), SAr1, KEr, Nr
```

If both peers support including the cookie in all retries, a slightly shorter exchange can happen:

```
   Initiator                    Responder
  -----------                  -----------
   HDR(A,0), SAi1, KEi, Ni -->
                            <-- HDR(A,0), N(COOKIE)
   HDR(A,0), N(COOKIE), SAi1, KEi, Ni  -->
                            <-- HDR(A,0), N(INVALID_KE_PAYLOAD)
   HDR(A,0), N(COOKIE), SAi1, KEi', Ni -->
                            <-- HDR(A,B), SAr1, KEr, Nr
```

This document recommends that implementations should support this shorter exchange, but it must not be assumed the other peer also supports the shorter exchange.

In theory, even this exchange has one unnecessary roundtrip, as both the cookie and Diffie-Hellman group could be checked at the same time:

```
   Initiator                     Responder
  -----------                   -----------
   HDR(A,0), SAi1, KEi, Ni -->
                             <-- HDR(A,0), N(COOKIE),
                                           N(INVALID_KE_PAYLOAD)
   HDR(A,0), N(COOKIE), SAi1, KEi',Ni -->
                             <-- HDR(A,B), SAr1, KEr, Nr
```

However, it is clear that this case is not allowed by the text in Section 2.6, since "all other payloads" clearly includes the KEi payload as well.

(References: "INVALID_KE_PAYLOAD and clarifications document" thread, Sep-Oct 2005.)

2.5. Invalid Cookies

There has been some confusion what should be done when an IKE_SA_INIT request containing an invalid cookie is received ("invalid" in the sense that its contents do not match the value expected by the responder).

The correct action is to ignore the cookie and process the message as if no cookie had been included (usually this means sending a response containing a new cookie). This is shown in Section 2.6 when it says "The responder in that case MAY reject the message by sending another response with a new cookie [...]".

Other possible actions, such as ignoring the whole request (or even all requests from this IP address for some time), create strange failure modes even in the absence of any malicious attackers and do not provide any additional protection against DoS attacks.

(References: "Invalid Cookie" thread, Sep-Oct 2005.)

3. Authentication

3.1. Data Included in AUTH Payload Calculation

Section 2.15 describes how the AUTH payloads are calculated; this calculation involves values prf(SK_pi,IDi') and prf(SK_pr,IDr'). The text describes the method in words, but does not give clear definitions of what is signed or MACed (i.e., protected with a message authentication code).

The initiator's signed octets can be described as:

```
InitiatorSignedOctets = RealMessage1 | NonceRData | MACedIDForI
GenIKEHDR = [ four octets 0 if using port 4500 ] | RealIKEHDR
RealIKEHDR =  SPIi | SPIr |  . . . | Length
RealMessage1 = RealIKEHDR | RestOfMessage1
NonceRPayload = PayloadHeader | NonceRData
InitiatorIDPayload = PayloadHeader | RestOfIDPayload
RestOfInitIDPayload = IDType | RESERVED | InitIDData
MACedIDForI = prf(SK_pi, RestOfInitIDPayload)
```

The responder's signed octets can be described as:

```
ResponderSignedOctets = RealMessage2 | NonceIData | MACedIDForR
GenIKEHDR = [ four octets 0 if using port 4500 ] | RealIKEHDR
RealIKEHDR =  SPIi | SPIr |  . . . | Length
RealMessage2 = RealIKEHDR | RestOfMessage2
NonceIPayload = PayloadHeader | NonceIData
ResponderIDPayload = PayloadHeader | RestOfIDPayload
RestOfRespIDPayload = IDType | RESERVED | InitIDData
MACedIDForR = prf(SK_pr, RestOfRespIDPayload)
```

3.2. Hash Function for RSA Signatures

Section 3.8 says that RSA digital signature is "Computed as specified in section 2.15 using an RSA private key over a PKCS#1 padded hash."

Unlike IKEv1, IKEv2 does not negotiate a hash function for the IKE_SA. The algorithm for signatures is selected by the signing party who, in general, may not know beforehand what algorithms the verifying party supports. Furthermore, [IKEv2ALG] does not say what algorithms implementations are required or recommended to support. This clearly has a potential for causing interoperability problems, since authentication will fail if the signing party selects an algorithm that is not supported by the verifying party, or not acceptable according to the verifying party's policy.

This document recommends that all implementations support SHA-1 and use SHA-1 as the default hash function when generating the signatures, unless there are good reasons (such as explicit manual configuration) to believe that the peer supports something else.

Note that hash function collision attacks are not important for the AUTH payloads, since they are not intended for third-party verification, and the data includes fresh nonces. See [HashUse] for more discussion about hash function attacks and IPsec.

Another reasonable choice would be to use the hash function that was used by the CA when signing the peer certificate. However, this does not guarantee that the IKEv2 peer would be able to validate the AUTH payload, because the same code might not be used to validate certificate signatures and IKEv2 message signatures, and these two routines may support a different set of hash algorithms. The peer could be configured with a fingerprint of the certificate, or certificate validation could be performed by an external entity using [SCVP]. Furthermore, not all CERT payloads types include a signature, and the certificate could be signed with some algorithm other than RSA.

Note that unlike IKEv1, IKEv2 uses the PKCS#1 v1.5 [PKCS1v20] signature encoding method (see next section for details), which includes the algorithm identifier for the hash algorithm. Thus, when the verifying party receives the AUTH payload it can at least determine which hash function was used.

(References: Magnus Alstrom's mail "RE:", 2005-01-03. Pasi Eronen's reply, 2005-01-04. Tero Kivinen's reply, 2005-01-04. "First draft of IKEv2.1" thread, Dec 2005/Jan 2006.)

3.3. Encoding Method for RSA Signatures

Section 3.8 says that the RSA digital signature is "Computed as specified in section 2.15 using an RSA private key over a PKCS#1 padded hash."

The PKCS#1 specification [PKCS1v21] defines two different encoding methods (ways of "padding the hash") for signatures. However, the Internet-Draft approved by the IESG had a reference to the older PKCS#1 v2.0 [PKCS1v20]. That version has only one encoding method for signatures (EMSA-PKCS1-v1_5), and thus there is no ambiguity.

Note that this encoding method is different from the encoding method used in IKEv1. If future revisions of IKEv2 provide support for other encoding methods (such as EMSA-PSS), they will be given new Auth Method numbers.

(References: Pasi Eronen's mail "RE:", 2005-01-04.)

3.4. Identification Type for EAP

Section 3.5 defines several different types for identification payloads, including, e.g., ID_FQDN, ID_RFC822_ADDR, and ID_KEY_ID. EAP [EAP] does not mandate the use of any particular type of identifier, but often EAP is used with Network Access Identifiers (NAIs) defined in [NAI]. Although NAIs look a bit like email addresses (e.g., "joe@example.com"), the syntax is not exactly the same as the syntax of email address in [RFC822]. This raises the question of which identification type should be used.

This document recommends that ID_RFC822_ADDR identification type is used for those NAIs that include the realm component. Therefore, responder implementations should not attempt to verify that the contents actually conform to the exact syntax given in [RFC822] or [RFC2822], but instead should accept any reasonable looking NAI.

For NAIs that do not include the realm component, this document recommends using the ID_KEY_ID identification type.

(References: "need your help on this IKEv2/i18n/EAP issue" and "IKEv2 identifier issue with EAP" threads, Aug 2004.)

3.5. Identity for Policy Lookups When Using EAP

When the initiator authentication uses EAP, it is possible that the contents of the IDi payload is used only for AAA routing purposes and selecting which EAP method to use. This value may be different from the identity authenticated by the EAP method (see [EAP], Sections 5.1 and 7.3).

It is important that policy lookups and access control decisions use the actual authenticated identity. Often the EAP server is implemented in a separate AAA server that communicates with the IKEv2 responder using, e.g., RADIUS [RADEAP]. In this case, the authenticated identity has to be sent from the AAA server to the IKEv2 responder.

(References: Pasi Eronen's mail "RE: Reauthentication in IKEv2", 2004-10-28. "Policy lookups" thread, Oct/Nov 2004. RFC 3748, Section 7.3.)

3.6. Certificate Encoding Types

Section 3.6 defines a total of twelve different certificate encoding types, and continues that "Specific syntax is for some of the certificate type codes above is not defined in this document." However, the text does not provide references to other documents that would contain information about the exact contents and use of those values.

Without this information, it is not possible to develop interoperable implementations. Therefore, this document recommends that the following certificate encoding values should not be used before new specifications that specify their use are available.

```
     PKCS #7 wrapped X.509 certificate    1
     PGP Certificate                      2
     DNS Signed Key                       3
     Kerberos Token                       6
     SPKI Certificate                     9
```

This document recommends that most implementations should use only those values that are "MUST"/"SHOULD" requirements in [IKEv2]; i.e., "X.509 Certificate - Signature" (4), "Raw RSA Key" (11), "Hash and URL of X.509 certificate" (12), and "Hash and URL of X.509 bundle" (13).

Furthermore, Section 3.7 says that the "Certificate Encoding" field for the Certificate Request payload uses the same values as for Certificate payload. However, the contents of the "Certification Authority" field are defined only for X.509 certificates (presumably covering at least types 4, 10, 12, and 13). This document recommends that other values should not be used before new specifications that specify their use are available.

The "Raw RSA Key" type needs one additional clarification. Section 3.6 says it contains "a PKCS #1 encoded RSA key". What this means is a DER-encoded RSAPublicKey structure from PKCS#1 [PKCS1v21].

3.7. Shared Key Authentication and Fixed PRF Key Size

Section 2.15 says that "If the negotiated prf takes a fixed-size key, the shared secret MUST be of that fixed size". This statement is correct: the shared secret must be of the correct size. If it is not, it cannot be used; there is no padding, truncation, or other processing involved to force it to that correct size.

This requirement means that it is difficult to use these pseudo-random functions (PRFs) with shared key authentication. The authors think this part of the specification was very poorly thought out, and using PRFs with a fixed key size is likely to result in interoperability problems. Thus, we recommend that such PRFs should not be used with shared key authentication. PRF_AES128_XCBC [RFC3664] originally used fixed key sizes; that RFC has been updated to handle variable key sizes in [RFC4434].

Note that Section 2.13 also contains text that is related to PRFs with fixed key size: "When the key for the prf function has fixed length, the data provided as a key is truncated or padded with zeros as necessary unless exceptional processing is explained following the formula". However, this text applies only to the prf+ construction, so it does not contradict the text in Section 2.15.

(References: Paul Hoffman's mail "Re: ikev2-07: last nits", 2003-05-02. Hugo Krawczyk's reply, 2003-05-12. Thread "Question about PRFs with fixed size key", Jan 2005.)

3.8. EAP Authentication and Fixed PRF Key Size

As described in the previous section, PRFs with a fixed key size require a shared secret of exactly that size. This restriction applies also to EAP authentication. For instance, a PRF that requires a 128-bit key cannot be used with EAP since [EAP] specifies that the MSK is at least 512 bits long.

(References: Thread "Question about PRFs with fixed size key", Jan 2005.)

3.9. Matching ID Payloads to Certificate Contents

In IKEv1, there was some confusion about whether or not the identities in certificates used to authenticate IKE were required to match the contents of the ID payloads. The PKI4IPsec Working Group produced the document [PKI4IPsec] which covers this topic in much more detail. However, Section 3.5 of [IKEv2] explicitly says that the ID payload "does not necessarily have to match anything in the CERT payload".

3.10. Message IDs for IKE_AUTH Messages

According to Section 2.2, "The IKE_SA initial setup messages will always be numbered 0 and 1." That is true when the IKE_AUTH exchange does not use EAP. When EAP is used, each pair of messages has their message numbers incremented. The first pair of AUTH messages will have an ID of 1, the second will be 2, and so on.

(References: "Question about MsgID in AUTH exchange" thread, April 2005.)

4. Creating CHILD_SAs

4.1. Creating SAs with the CREATE_CHILD_SA Exchange

Section 1.3’s organization does not lead to clear understanding of what is needed in which environment. The section can be reorganized with subsections for each use of the CREATE_CHILD_SA exchange (creating child SAs, rekeying IKE SAs, and rekeying child SAs.)

The new Section 1.3 with subsections and the above changes might look like the following.

NEW-1.3 The CREATE_CHILD_SA Exchange

> The CREATE_CHILD_SA Exchange is used to create new CHILD_SAs and to rekey both IKE_SAs and CHILD_SAs. This exchange consists of a single request/response pair, and some of its function was referred to as a phase 2 exchange in IKEv1. It MAY be initiated by either end of the IKE_SA after the initial exchanges are completed.
>
> All messages following the initial exchange are cryptographically protected using the cryptographic algorithms and keys negotiated in the first two messages of the IKE exchange. These subsequent messages use the syntax of the Encrypted Payload described in section 3.14. All subsequent messages include an Encrypted Payload, even if they are referred to in the text as "empty".
>
> The CREATE_CHILD_SA is used for rekeying IKE_SAs and CHILD_SAs. This section describes the first part of rekeying, the creation of new SAs; Section 2.8 covers the mechanics of rekeying, including moving traffic from old to new SAs and the deletion of the old SAs. The two sections must be read together to understand the entire process of rekeying.

Either endpoint may initiate a CREATE_CHILD_SA exchange, so in this section the term initiator refers to the endpoint initiating this exchange. An implementation MAY refuse all CREATE_CHILD_SA requests within an IKE_SA.

The CREATE_CHILD_SA request MAY optionally contain a KE payload for an additional Diffie-Hellman exchange to enable stronger guarantees of forward secrecy for the CHILD_SA or IKE_SA. The keying material for the SA is a function of SK_d established during the establishment of the IKE_SA, the nonces exchanged during the CREATE_CHILD_SA exchange, and the Diffie-Hellman value (if KE payloads are included in the CREATE_CHILD_SA exchange). The details are described in sections 2.17 and 2.18.

If a CREATE_CHILD_SA exchange includes a KEi payload, at least one of the SA offers MUST include the Diffie-Hellman group of the KEi. The Diffie-Hellman group of the KEi MUST be an element of the group the initiator expects the responder to accept (additional Diffie-Hellman groups can be proposed). If the responder rejects the Diffie-Hellman group of the KEi payload, the responder MUST reject the request and indicate its preferred Diffie-Hellman group in the INVALID_KE_PAYLOAD Notification payload. In the case of such a rejection, the CREATE_CHILD_SA exchange fails, and the initiator SHOULD retry the exchange with a Diffie-Hellman proposal and KEi in the group that the responder gave in the INVALID_KE_PAYLOAD.

NEW-1.3.1 Creating New CHILD_SAs with the CREATE_CHILD_SA Exchange

A CHILD_SA may be created by sending a CREATE_CHILD_SA request. The CREATE_CHILD_SA request for creating a new CHILD_SA is:

```
    Initiator                                 Responder
   -----------                               -----------
    HDR, SK {[N+], SA, Ni, [KEi],
               TSi, TSr}        -->
```

The initiator sends SA offer(s) in the SA payload, a nonce in the Ni payload, optionally a Diffie-Hellman value in the KEi payload, and the proposed traffic selectors for the proposed CHILD_SA in the TSi and TSr payloads. The request can also contain Notify payloads that specify additional details for the CHILD_SA: these include IPCOMP_SUPPORTED, USE_TRANSPORT_MODE, ESP_TFC_PADDING_NOT_SUPPORTED, and NON_FIRST_FRAGMENTS_ALSO.

The CREATE_CHILD_SA response for creating a new CHILD_SA is:

```
                              <--    HDR, SK {[N+], SA, Nr,
                                            [KEr], TSi, TSr}
```

The responder replies with the accepted offer in an SA payload, and a Diffie-Hellman value in the KEr payload if KEi was included in the request and the selected cryptographic suite includes that group. As with the request, optional Notification payloads can specify additional details for the CHILD_SA.

The traffic selectors for traffic to be sent on that SA are specified in the TS payloads in the response, which may be a subset of what the initiator of the CHILD_SA proposed.

The text about rekeying SAs can be found in Section 5.1 of this document.

4.2. Creating an IKE_SA without a CHILD_SA

CHILD_SAs can be created either by being piggybacked on the IKE_AUTH exchange, or using a separate CREATE_CHILD_SA exchange. The specification is not clear about what happens if creating the CHILD_SA during the IKE_AUTH exchange fails for some reason.

Our recommendation in this situation is that the IKE_SA is created as usual. This is also in line with how the CREATE_CHILD_SA exchange works: a failure to create a CHILD_SA does not close the IKE_SA.

The list of responses in the IKE_AUTH exchange that do not prevent an IKE_SA from being set up include at least the following: NO_PROPOSAL_CHOSEN, TS_UNACCEPTABLE, SINGLE_PAIR_REQUIRED, INTERNAL_ADDRESS_FAILURE, and FAILED_CP_REQUIRED.

(References: "Questions about internal address" thread, April 2005.)

4.3. Diffie-Hellman for First CHILD_SA

Section 1.2 shows that IKE_AUTH messages do not contain KEi/KEr or Ni/Nr payloads. This implies that the SA payload in IKE_AUTH exchange cannot contain Transform Type 4 (Diffie-Hellman Group) with any other value than NONE. Implementations should probably leave the transform out entirely in this case.

4.4. Extended Sequence Numbers (ESN) Transform

The description of the ESN transform in Section 3.3 has be proved difficult to understand. The ESN transform has the following meaning:

- o A proposal containing one ESN transform with value 0 means "do not use extended sequence numbers".

- o A proposal containing one ESN transform with value 1 means "use extended sequence numbers".

- o A proposal containing two ESN transforms with values 0 and 1 means "I support both normal and extended sequence numbers, you choose". (Obviously this case is only allowed in requests; the response will contain only one ESN transform.)

In most cases, the exchange initiator will include either the first or third alternative in its SA payload. The second alternative is rarely useful for the initiator: it means that using normal sequence numbers is not acceptable (so if the responder does not support ESNs, the exchange will fail with NO_PROPOSAL_CHOSEN).

Note that including the ESN transform is mandatory when creating ESP/AH SAs (it was optional in earlier drafts of the IKEv2 specification).

(References: "Technical change needed to IKEv2 before publication", "STRAW POLL: Dealing with the ESN negotiation interop issue in IKEv2" and "Results of straw poll regarding: IKEv2 interoperability issue" threads, March-April 2005.)

4.5. Negotiation of ESP_TFC_PADDING_NOT_SUPPORTED

The description of ESP_TFC_PADDING_NOT_SUPPORTED notification in Section 3.10.1 says that "This notification asserts that the sending endpoint will NOT accept packets that contain Flow Confidentiality (TFC) padding".

However, the text does not say in which messages this notification should be included, or whether the scope of this notification is a single CHILD_SA or all CHILD_SAs of the peer.

Our interpretation is that the scope is a single CHILD_SA, and thus this notification is included in messages containing an SA payload negotiating a CHILD_SA. If neither endpoint accepts TFC padding, this notification will be included in both the request proposing an SA and the response accepting it. If this notification is included in only one of the messages, TFC padding can still be sent in one direction.

4.6. Negotiation of NON_FIRST_FRAGMENTS_ALSO

NON_FIRST_FRAGMENTS_ALSO notification is described in Section 3.10.1 simply as "Used for fragmentation control. See [RFC4301] for explanation."

[RFC4301] says "Implementations that will transmit non-initial fragments on a tunnel mode SA that makes use of non-trivial port (or ICMP type/code or MH type) selectors MUST notify a peer via the IKE NOTIFY NON_FIRST_FRAGMENTS_ALSO payload. The peer MUST reject this proposal if it will not accept non-initial fragments in this context. If an implementation does not successfully negotiate transmission of non-initial fragments for such an SA, it MUST NOT send such fragments over the SA."

However, it is not clear exactly how the negotiation works. Our interpretation is that the negotiation works the same way as for IPCOMP_SUPPORTED and USE_TRANSPORT_MODE: sending non-first fragments is enabled only if NON_FIRST_FRAGMENTS_ALSO notification is included in both the request proposing an SA and the response accepting it. In other words, if the peer "rejects this proposal", it only omits NON_FIRST_FRAGMENTS_ALSO notification from the response, but does not reject the whole CHILD_SA creation.

4.7. Semantics of Complex Traffic Selector Payloads

As described in Section 3.13, the TSi/TSr payloads can include one or more individual traffic selectors.

There is no requirement that TSi and TSr contain the same number of individual traffic selectors. Thus, they are interpreted as follows: a packet matches a given TSi/TSr if it matches at least one of the individual selectors in TSi, and at least one of the individual selectors in TSr.

For instance, the following traffic selectors:

```
TSi = ((17, 100, 192.0.1.66-192.0.1.66),
       (17, 200, 192.0.1.66-192.0.1.66))
TSr = ((17, 300, 0.0.0.0-255.255.255.255),
       (17, 400, 0.0.0.0-255.255.255.255))
```

would match UDP packets from 192.0.1.66 to anywhere, with any of the four combinations of source/destination ports (100,300), (100,400), (200,300), and (200, 400).

This implies that some types of policies may require several CHILD_SA pairs. For instance, a policy matching only source/destination ports (100,300) and (200,400), but not the other two combinations, cannot be negotiated as a single CHILD_SA pair using IKEv2.

(References: "IKEv2 Traffic Selectors?" thread, Feb 2005.)

4.8. ICMP Type/Code in Traffic Selector Payloads

The traffic selector types 7 and 8 can also refer to ICMP type and code fields. As described in Section 3.13.1, "For the ICMP protocol, the two one-octet fields Type and Code are treated as a single 16-bit integer (with Type in the most significant eight bits and Code in the least significant eight bits) port number for the purposes of filtering based on this field."

Since ICMP packets do not have separate source and destination port fields, there is some room for confusion what exactly the four TS payloads (two in the request, two in the response, each containing both start and end port fields) should contain.

The answer to this question can be found from [RFC4301] Section 4.4.1.3.

To give a concrete example, if a host at 192.0.1.234 wants to create a transport mode SA for sending "Destination Unreachable" packets (ICMPv4 type 3) to 192.0.2.155, but is not willing to receive them over this SA pair, the CREATE_CHILD_SA exchange would look like this:

```
Initiator                    Responder
-----------                 -----------
HDR, SK { N(USE_TRANSPORT_MODE), SA, Ni,
          TSi(1, 0x0300-0x03FF, 192.0.1.234-192.0.1.234),
          TSr(1, 65535-0, 192.0.2.155-192.0.2.155) } -->

   <-- HDR, SK { N(USE_TRANSPORT_MODE), SA, Nr,
                 TSi(1, 0x0300-0x03FF, 192.0.1.234-192.0.1.234),
                 TSr(1, 65535-0, 192.0.2.155-192.0.2.155) }
```

Since IKEv2 always creates IPsec SAs in pairs, two SAs are also created in this case, even though the second SA is never used for data traffic.

An exchange creating an SA pair that can be used both for sending and receiving "Destination Unreachable" places the same value in all the port:

```
   Initiator                  Responder
  -----------                -----------
   HDR, SK { N(USE_TRANSPORT_MODE), SA, Ni,
             TSi(1, 0x0300-0x03FF, 192.0.1.234-192.0.1.234),
             TSr(1, 0x0300-0x03FF, 192.0.2.155-192.0.2.155) } -->

      <-- HDR, SK { N(USE_TRANSPORT_MODE), SA, Nr,
                    TSi(1, 0x0300-0x03FF, 192.0.1.234-192.0.1.234),
                    TSr(1, 0x0300-0x03FF, 192.0.2.155-192.0.2.155) }
```

(References: "ICMP and MH TSs for IKEv2" thread, Sep 2005.)

4.9. Mobility Header in Traffic Selector Payloads

Traffic selectors can use IP Protocol ID 135 to match the IPv6 mobility header [MIPv6]. However, the IKEv2 specification does not define how to represent the "MH Type" field in traffic selectors.

At some point, it was expected that this will be defined in a separate document later. However, [RFC4301] says that "For IKE, the IPv6 mobility header message type (MH type) is placed in the most significant eight bits of the 16 bit local "port" selector". The direction semantics of TSi/TSr port fields are the same as for ICMP and are described in the previous section.

(References: Tero Kivinen’s mail "Issue #86: Add IPv6 mobility header message type as selector", 2003-10-14. "ICMP and MH TSs for IKEv2" thread, Sep 2005.)

4.10. Narrowing the Traffic Selectors

Section 2.9 describes how traffic selectors are negotiated when creating a CHILD_SA. A more concise summary of the narrowing process is presented below.

- o If the responder’s policy does not allow any part of the traffic covered by TSi/TSr, it responds with TS_UNACCEPTABLE.

- o If the responder’s policy allows the entire set of traffic covered by TSi/TSr, no narrowing is necessary, and the responder can return the same TSi/TSr values.

- o Otherwise, narrowing is needed. If the responder’s policy allows all traffic covered by TSi[1]/TSr[1] (the first traffic selectors in TSi/TSr) but not entire TSi/TSr, the responder narrows to an acceptable subset of TSi/TSr that includes TSi[1]/TSr[1].

- o If the responder's policy does not allow all traffic covered by TSi[1]/TSr[1], but does allow some parts of TSi/TSr, it narrows to an acceptable subset of TSi/TSr.

In the last two cases, there may be several subsets that are acceptable (but their union is not); in this case, the responder arbitrarily chooses one of them and includes ADDITIONAL_TS_POSSIBLE notification in the response.

4.11. SINGLE_PAIR_REQUIRED

The description of the SINGLE_PAIR_REQUIRED notify payload in Sections 2.9 and 3.10.1 is not fully consistent.

We do not attempt to describe this payload in this document either, since it is expected that most implementations will not have policies that require separate SAs for each address pair.

Thus, if only some part (or parts) of the TSi/TSr proposed by the initiator is (are) acceptable to the responder, most responders should simply narrow TSi/TSr to an acceptable subset (as described in the last two paragraphs of Section 2.9), rather than use SINGLE_PAIR_REQUIRED.

4.12. Traffic Selectors Violating Own Policy

Section 2.9 describes traffic selector negotiation in great detail. One aspect of this negotiation that may need some clarification is that when creating a new SA, the initiator should not propose traffic selectors that violate its own policy. If this rule is not followed, valid traffic may be dropped.

This is best illustrated by an example. Suppose that host A has a policy whose effect is that traffic to 192.0.1.66 is sent via host B encrypted using Advanced Encryption Standard (AES), and traffic to all other hosts in 192.0.1.0/24 is also sent via B, but encrypted using Triple Data Encryption Standard (3DES). Suppose also that host B accepts any combination of AES and 3DES.

If host A now proposes an SA that uses 3DES, and includes TSr containing (192.0.1.0-192.0.1.0.255), this will be accepted by host B. Now, host B can also use this SA to send traffic from 192.0.1.66, but those packets will be dropped by A since it requires the use of AES for those traffic. Even if host A creates a new SA only for 192.0.1.66 that uses AES, host B may freely continue to use the first SA for the traffic. In this situation, when proposing the SA, host A should have followed its own policy, and included a TSr containing ((192.0.1.0-192.0.1.65),(192.0.1.67-192.0.1.255)) instead.

In general, if (1) the initiator makes a proposal "for traffic X (TSi/TSr), do SA", and (2) for some subset X' of X, the initiator does not actually accept traffic X' with SA, and (3) the initiator would be willing to accept traffic X' with some SA' (!=SA), valid traffic can be unnecessarily dropped since the responder can apply either SA or SA' to traffic X'.

(References: "Question about "narrowing" ..." thread, Feb 2005. "IKEv2 needs a "policy usage mode"..." thread, Feb 2005. "IKEv2 Traffic Selectors?" thread, Feb 2005. "IKEv2 traffic selector negotiation examples", 2004-08-08.)

4.13. Traffic Selector Authorization

IKEv2 relies on information in the Peer Authorization Database (PAD) when determining what kind of IPsec SAs a peer is allowed to create. This process is described in [RFC4301] Section 4.4.3. When a peer requests the creation of an IPsec SA with some traffic selectors, the PAD must contain "Child SA Authorization Data" linking the identity authenticated by IKEv2 and the addresses permitted for traffic selectors.

For example, the PAD might be configured so that authenticated identity "sgw23.example.com" is allowed to create IPsec SAs for 192.0.2.0/24, meaning this security gateway is a valid "representative" for these addresses. Host-to-host IPsec requires similar entries, linking, for example, "fooserver4.example.com" with 192.0.1.66/32, meaning this identity a valid "owner" or "representative" of the address in question.

As noted in [RFC4301], "It is necessary to impose these constraints on creation of child SAs to prevent an authenticated peer from spoofing IDs associated with other, legitimate peers." In the example given above, a correct configuration of the PAD prevents sgw23 from creating IPsec SAs with address 192.0.1.66 and prevents fooserver4 from creating IPsec SAs with addresses from 192.0.2.0/24.

It is important to note that simply sending IKEv2 packets using some particular address does not imply a permission to create IPsec SAs with that address in the traffic selectors. For example, even if sgw23 would be able to spoof its IP address as 192.0.1.66, it could not create IPsec SAs matching fooserver4's traffic.

The IKEv2 specification does not specify how exactly IP address assignment using configuration payloads interacts with the PAD. Our interpretation is that when a security gateway assigns an address using configuration payloads, it also creates a temporary PAD entry linking the authenticated peer identity and the newly allocated inner address.

It has been recognized that configuring the PAD correctly may be difficult in some environments. For instance, if IPsec is used between a pair of hosts whose addresses are allocated dynamically using Dynamic Host Configuration Protocol (DHCP), it is extremely difficult to ensure that the PAD specifies the correct "owner" for each IP address. This would require a mechanism to securely convey address assignments from the DHCP server and link them to identities authenticated using IKEv2.

Due to this limitation, some vendors have been known to configure their PADs to allow an authenticated peer to create IPsec SAs with traffic selectors containing the same address that was used for the IKEv2 packets. In environments where IP spoofing is possible (i.e., almost everywhere) this essentially allows any peer to create IPsec SAs with any traffic selectors. This is not an appropriate or secure configuration in most circumstances. See [Aura05] for an extensive discussion about this issue, and the limitations of host-to-host IPsec in general.

5. Rekeying and Deleting SAs

5.1. Rekeying SAs with the CREATE_CHILD_SA Exchange

Continued from Section 4.1 of this document.

NEW-1.3.2 Rekeying IKE_SAs with the CREATE_CHILD_SA Exchange

The CREATE_CHILD_SA request for rekeying an IKE_SA is:

```
   Initiator                                 Responder
  -----------                               -----------
   HDR, SK {SA, Ni, [KEi]} -->
```

The initiator sends SA offer(s) in the SA payload, a nonce in the Ni payload, and optionally a Diffie-Hellman value in the KEi payload.

The CREATE_CHILD_SA response for rekeying an IKE_SA is:

```
                              <--    HDR, SK {SA, Nr, [KEr]}
```

The responder replies (using the same Message ID to respond) with the accepted offer in an SA payload, a nonce in the Nr payload, and, optionally, a Diffie-Hellman value in the KEr payload.

The new IKE_SA has its message counters set to 0, regardless of what they were in the earlier IKE_SA. The window size starts at 1 for any new IKE_SA. The new initiator and responder SPIs are supplied in the SPI fields of the SA payloads.

NEW-1.3.3 Rekeying CHILD_SAs with the CREATE_CHILD_SA Exchange

The CREATE_CHILD_SA request for rekeying a CHILD_SA is:

```
Initiator                                  Responder
-----------                              -----------
HDR, SK {N(REKEY_SA), [N+], SA,
    Ni, [KEi], TSi, TSr}  -->
```

The leading Notify payload of type REKEY_SA identifies the CHILD_SA being rekeyed, and it contains the SPI that the initiator expects in the headers of inbound packets. In addition, the initiator sends SA offer(s) in the SA payload, a nonce in the Ni payload, optionally a Diffie-Hellman value in the KEi payload, and the proposed traffic selectors in the TSi and TSr payloads. The request can also contain Notify payloads that specify additional details for the CHILD_SA.

The CREATE_CHILD_SA response for rekeying a CHILD_SA is:

```
                          <--    HDR, SK {[N+], SA, Nr,
                                       [KEr], TSi, TSr}
```

The responder replies with the accepted offer in an SA payload, and a Diffie-Hellman value in the KEr payload if KEi was included in the request and the selected cryptographic suite includes that group.

The traffic selectors for traffic to be sent on that SA are specified in the TS payloads in the response, which may be a subset of what the initiator of the CHILD_SA proposed.

5.2. Rekeying the IKE_SA vs. Reauthentication

Rekeying the IKE_SA and reauthentication are different concepts in IKEv2. Rekeying the IKE_SA establishes new keys for the IKE_SA and resets the Message ID counters, but it does not authenticate the parties again (no AUTH or EAP payloads are involved).

While rekeying the IKE_SA may be important in some environments, reauthentication (the verification that the parties still have access to the long-term credentials) is often more important.

IKEv2 does not have any special support for reauthentication. Reauthentication is done by creating a new IKE_SA from scratch (using IKE_SA_INIT/IKE_AUTH exchanges, without any REKEY_SA notify payloads), creating new CHILD_SAs within the new IKE_SA (without REKEY_SA notify payloads), and finally deleting the old IKE_SA (which deletes the old CHILD_SAs as well).

This means that reauthentication also establishes new keys for the IKE_SA and CHILD_SAs. Therefore, while rekeying can be performed more often than reauthentication, the situation where "authentication lifetime" is shorter than "key lifetime" does not make sense.

While creation of a new IKE_SA can be initiated by either party (initiator or responder in the original IKE_SA), the use of EAP authentication and/or configuration payloads means in practice that reauthentication has to be initiated by the same party as the original IKE_SA. IKEv2 base specification does not allow the responder to request reauthentication in this case; however, this functionality is added in [ReAuth].

(References: "Reauthentication in IKEv2" thread, Oct/Nov 2004.)

5.3. SPIs When Rekeying the IKE_SA

Section 2.18 says that "New initiator and responder SPIs are supplied in the SPI fields". This refers to the SPI fields in the Proposal structures inside the Security Association (SA) payloads, not the SPI fields in the IKE header.

(References: Tom Stiemerling's mail "Rekey IKE SA", 2005-01-24. Geoffrey Huang's reply, 2005-01-24.)

5.4. SPI When Rekeying a CHILD_SA

Section 3.10.1 says that in REKEY_SA notifications, "The SPI field identifies the SA being rekeyed."

Since CHILD_SAs always exist in pairs, there are two different SPIs. The SPI placed in the REKEY_SA notification is the SPI the exchange initiator would expect in inbound ESP or AH packets (just as in Delete payloads).

5.5. Changing PRFs When Rekeying the IKE_SA

When rekeying the IKE_SA, Section 2.18 says that "SKEYSEED for the new IKE_SA is computed using SK_d from the existing IKE_SA as follows:

```
SKEYSEED = prf(SK_d (old), [g^ir (new)] | Ni | Nr)"
```

If the old and new IKE_SA selected a different PRF, it is not totally clear which PRF should be used.

Since the rekeying exchange belongs to the old IKE_SA, it is the old IKE_SA's PRF that is used. This also follows the principle that the same key (the old SK_d) should not be used with multiple cryptographic algorithms.

Note that this may work poorly if the new IKE_SA's PRF has a fixed key size, since the output of the PRF may not be of the correct size. This supports our opinion earlier in the document that the use of PRFs with a fixed key size is a bad idea.

(References: "Changing PRFs when rekeying the IKE_SA" thread, June 2005.)

5.6. Deleting vs. Closing SAs

The IKEv2 specification talks about "closing" and "deleting" SAs, but it is not always clear what exactly is meant. However, other parts of the specification make it clear that when local state related to a CHILD_SA is removed, the SA must also be actively deleted with a Delete payload.

In particular, Section 2.4 says that "If an IKE endpoint chooses to delete CHILD_SAs, it MUST send Delete payloads to the other end notifying it of the deletion". Section 1.4 also explains that "ESP and AH SAs always exist in pairs, with one SA in each direction. When an SA is closed, both members of the pair MUST be closed."

5.7. Deleting a CHILD_SA Pair

Section 1.4 describes how to delete SA pairs using the Informational exchange: "To delete an SA, an INFORMATIONAL exchange with one or more delete payloads is sent listing the SPIs (as they would be expected in the headers of inbound packets) of the SAs to be deleted. The recipient MUST close the designated SAs."

The "one or more delete payloads" phrase has caused some confusion. You never send delete payloads for the two sides of an SA in a single message. If you have many SAs to delete at the same time (such as the nested example given in that paragraph), you include delete payloads for the inbound half of each SA in your Informational exchange.

5.8. Deleting an IKE_SA

Since IKE_SAs do not exist in pairs, it is not totally clear what the response message should contain when the request deleted the IKE_SA.

Since there is no information that needs to be sent to the other side (except that the request was received), an empty Informational response seems like the most logical choice.

(References: "Question about delete IKE SA" thread, May 2005.)

5.9. Who is the original initiator of IKE_SA

In the IKEv2 document, "initiator" refers to the party who initiated the exchange being described, and "original initiator" refers to the party who initiated the whole IKE_SA. However, there is some potential for confusion because the IKE_SA can be rekeyed by either party.

To clear up this confusion, we propose that "original initiator" always refers to the party who initiated the exchange that resulted in the current IKE_SA. In other words, if the "original responder" starts rekeying the IKE_SA, that party becomes the "original initiator" of the new IKE_SA.

(References: Paul Hoffman's mail "Original initiator in IKEv2", 2005-04-21.)

5.10. Comparing Nonces

Section 2.8 about rekeying says that "If redundant SAs are created though such a collision, the SA created with the lowest of the four nonces used in the two exchanges SHOULD be closed by the endpoint that created it."

Here "lowest" uses an octet-by-octet (lexicographical) comparison (instead of, for instance, comparing the nonces as large integers). In other words, start by comparing the first octet; if they're equal, move to the next octet, and so on. If you reach the end of one nonce, that nonce is the lower one.

(References: "IKEv2 rekeying question" thread, July 2005.)

5.11. Exchange Collisions

Since IKEv2 exchanges can be initiated by both peers, it is possible that two exchanges affecting the same SA partly overlap. This can lead to a situation where the SA state information is temporarily not synchronized, and a peer can receive a request it cannot process in a normal fashion. Some of these corner cases are discussed in the specification, some are not.

Obviously, using a window size greater than one leads to infinitely more complex situations, especially if requests are processed out of order. In this section, we concentrate on problems that can arise even with window size 1.

(References: "IKEv2: invalid SPI in DELETE payload" thread, Dec 2005/ Jan 2006. "Problem with exchanges collisions" thread, Dec 2005.)

5.11.1. Simultaneous CHILD_SA Close

Probably the simplest case happens if both peers decide to close the same CHILD_SA pair at the same time:

```
   Host A                        Host B
  --------                      --------
   send req1: D(SPIa) -->
                              <-- send req2: D(SPIb)
                              --> recv req1
                              <-- send resp1: ()
   recv resp1
   recv req2
   send resp2: () -->
                              --> recv resp2
```

This case is described in Section 1.4 and is handled by omitting the Delete payloads from the response messages.

5.11.2. Simultaneous IKE_SA Close

Both peers can also decide to close the IKE_SA at the same time. The desired end result is obvious; however, in certain cases the final exchanges may not be fully completed.

```
   Host A                        Host B
  --------                      --------
   send req1: D() -->
                             <-- send req2: D()
                             --> recv req1
```

At this point, host B should reply as usual (with empty Informational response), close the IKE_SA, and stop retransmitting req2. This is because once host A receives resp1, it may not be able to reply any longer. The situation is symmetric, so host A should behave the same way.

```
   Host A                        Host B
  --------                      --------
                             <-- send resp1: ()
   send resp2: ()
```

Even if neither resp1 nor resp2 ever arrives, the end result is still correct: the IKE_SA is gone. The same happens if host A never receives req2.

5.11.3. Simultaneous CHILD_SA Rekeying

Another case that is described in the specification is simultaneous rekeying. Section 2.8 says

> "If the two ends have the same lifetime policies, it is possible that both will initiate a rekeying at the same time (which will result in redundant SAs). To reduce the probability of this happening, the timing of rekeying requests SHOULD be jittered (delayed by a random amount of time after the need for rekeying is noticed).
>
> This form of rekeying may temporarily result in multiple similar SAs between the same pairs of nodes. When there are two SAs eligible to receive packets, a node MUST accept incoming packets through either SA. If redundant SAs are created though such a collision, the SA created with the lowest of the four nonces used in the two exchanges SHOULD be closed by the endpoint that created it."

However, a better explanation on what impact this has on implementations is needed. Assume that hosts A and B have an existing IPsec SA pair with SPIs (SPIa1,SPIb1), and both start rekeying it at the same time:

```
   Host A                            Host B
  --------                          --------
   send req1: N(REKEY_SA,SPIa1),
      SA(..,SPIa2,..),Ni1,..  -->
                                <-- send req2: N(REKEY_SA,SPIb1),
                                       SA(..,SPIb2,..),Ni2,..
   recv req2 <--
```

At this point, A knows there is a simultaneous rekeying going on. However, it cannot yet know which of the exchanges will have the lowest nonce, so it will just note the situation and respond as usual.

```
   send resp2: SA(..,SPIa3,..),Nr1,.. -->
                                --> recv req1
```

Now B also knows that simultaneous rekeying is going on. Similarly as host A, it has to respond as usual.

```
                                <-- send resp1: SA(..,SPIb3,..),Nr2,..
    recv resp1 <--
                                --> recv resp2
```

At this point, there are three CHILD_SA pairs between A and B (the old one and two new ones). A and B can now compare the nonces. Suppose that the lowest nonce was Nr1 in message resp2; in this case, B (the sender of req2) deletes the redundant new SA, and A (the node that initiated the surviving rekeyed SA) deletes the old one.

```
   send req3: D(SPIa1) -->
                                <-- send req4: D(SPIb2)
                                --> recv req3
                                <-- send resp4: D(SPIb1)
   recv req4 <--
   send resp4: D(SPIa3) -->
```

The rekeying is now finished.

However, there is a second possible sequence of events that can happen if some packets are lost in the network, resulting in retransmissions. The rekeying begins as usual, but A’s first packet (req1) is lost.

```
      Host A                      Host B
     --------                    --------
      send req1: N(REKEY_SA,SPIa1),
         SA(..,SPIa2,..),Ni1,..  -->  (lost)
                              <-- send req2: N(REKEY_SA,SPIb1),
                                     SA(..,SPIb2,..),Ni2,..
      recv req2 <--
      send resp2: SA(..,SPIa3,..),Nr1,.. -->
                              --> recv resp2
                              <-- send req3: D(SPIb1)
      recv req3 <--
      send resp3: D(SPIa1) -->
                              --> recv resp3
```

From B’s point of view, the rekeying is now completed, and since it has not yet received A’s req1, it does not even know that these was simultaneous rekeying. However, A will continue retransmitting the message, and eventually it will reach B.

```
      resend req1 -->
                               --> recv req1
```

What should B do in this point? To B, it looks like A is trying to rekey an SA that no longer exists; thus failing the request with something non-fatal such as NO_PROPOSAL_CHOSEN seems like a reasonable approach.

```
                               <-- send resp1: N(NO_PROPOSAL_CHOSEN)
      recv resp1 <--
```

When A receives this error, it already knows there was simultaneous rekeying, so it can ignore the error message.

5.11.4. Simultaneous IKE_SA Rekeying

Probably the most complex case occurs when both peers try to rekey the IKE_SA at the same time. Basically, the text in Section 2.8 applies to this case as well; however, it is important to ensure that the CHILD_SAs are inherited by the right IKE_SA.

The case where both endpoints notice the simultaneous rekeying works the same way as with CHILD_SAs. After the CREATE_CHILD_SA exchanges, three IKE_SAs exist between A and B; the one containing the lowest nonce inherits the CHILD_SAs.

However, there is a twist to the other case where one rekeying finishes first:

```
   Host A                          Host B
  --------                        --------
   send req1:
      SA(..,SPIa1,..),Ni1,.. -->
                              <-- send req2: SA(..,SPIb1,..),Ni2,..
                              --> recv req1
                              <-- send resp1: SA(..,SPIb2,..),Nr2,..
   recv resp1 <--
   send req3: D() -->
                              --> recv req3
```

At this point, host B sees a request to close the IKE_SA. There's not much more to do than to reply as usual. However, at this point host B should stop retransmitting req2, since once host A receives resp3, it will delete all the state associated with the old IKE_SA and will not be able to reply to it.

```
                              <-- send resp3: ()
```

5.11.5. Closing and Rekeying a CHILD_SA

A case similar to simultaneous rekeying can occur if one peer decides to close an SA and the other peer tries to rekey it:

```
   Host A                          Host B
  --------                        --------
   send req1: D(SPIa) -->
                              <-- send req2: N(REKEY_SA,SPIb),SA,..
                              --> recv req1
```

At this point, host B notices that host A is trying to close an SA that host B is currently rekeying. Replying as usual is probably the best choice:

```
                              <-- send resp1: D(SPIb)
```

Depending on in which order req2 and resp1 arrive, host A sees either a request to rekey an SA that it is currently closing, or a request to rekey an SA that does not exist. In both cases, NO_PROPOSAL_CHOSEN is probably fine.

```
   recv req2
   recv resp1
   send resp2: N(NO_PROPOSAL_CHOSEN) -->
                              --> recv resp2
```

5.11.6. Closing a New CHILD_SA

Yet another case occurs when host A creates a CHILD_SA pair, but soon thereafter host B decides to delete it (possible because its policy changed):

```
 Host A                        Host B
--------                      --------
 send req1: [N(REKEY_SA,SPIa1)],
    SA(..,SPIa2,..),.. -->
                           --> recv req1
                    (lost) <-- send resp1: SA(..,SPIb2,..),..

                           <-- send req2: D(SPIb2)
 recv req2
```

At this point, host A has not yet received message resp1 (and is retransmitting message req1), so it does not recognize SPIb in message req2. What should host A do?

One option would be to reply with an empty Informational response. However, this same reply would also be sent if host A has received resp1, but has already sent a new request to delete the SA that was just created. This would lead to a situation where the peers are no longer in sync about which SAs exist between them. However, host B would eventually notice that the other half of the CHILD_SA pair has not been deleted. Section 1.4 describes this case and notes that "a node SHOULD regard half-closed connections as anomalous and audit their existence should they persist", and continues that "if connection state becomes sufficiently messed up, a node MAY close the IKE_SA".

Another solution that has been proposed is to reply with an INVALID_SPI notification that contains SPIb. This would explicitly tell host B that the SA was not deleted, so host B could try deleting it again later. However, this usage is not part of the IKEv2 specification and would not be in line with normal use of the INVALID_SPI notification where the data field contains the SPI the recipient of the notification would put in outbound packets.

Yet another solution would be to ignore req2 at this time and wait until we have received resp1. However, this alternative has not been fully analyzed at this time; in general, ignoring valid requests is always a bit dangerous, because both endpoints could do it, leading to a deadlock.

This document recommends the first alternative.

5.11.7. Rekeying a New CHILD_SA

Yet another case occurs when a CHILD_SA is rekeyed soon after it has been created:

```
   Host A                       Host B
  --------                     --------
   send req1: [N(REKEY_SA,SPIa1)],
      SA(..,SPIa2,..),..  -->
                    (lost) <-- send resp1: SA(..,SPIb2,..),..

                           <-- send req2: N(REKEY_SA,SPIb2),
                                  SA(..,SPIb3,..),..
   recv req2 <--
```

To host A, this looks like a request to rekey an SA that does not exist. Like in the simultaneous rekeying case, replying with NO_PROPOSAL_CHOSEN is probably reasonable:

```
   send resp2: N(NO_PROPOSAL_CHOSEN) -->
   recv resp1
```

5.11.8. Collisions with IKE_SA Rekeying

Another set of cases occurs when one peer starts rekeying the IKE_SA at the same time the other peer starts creating, rekeying, or closing a CHILD_SA. Suppose that host B starts creating a CHILD_SA, and soon after, host A starts rekeying the IKE_SA:

```
   Host A                       Host B
  --------                     --------
                           <-- send req1: SA,Ni1,TSi,TSr
   send req2: SA,Ni2,.. -->
                           --> recv req2
```

What should host B do at this point? Replying as usual would seem like a reasonable choice:

```
                           <-- send resp2: SA,Ni2,..
   recv resp2 <--
   send req3: D() -->
                           --> recv req3
```

Now, a problem arises: If host B now replies normally with an empty Informational response, this will cause host A to delete state associated with the IKE_SA. This means host B should stop retransmitting req1. However, host B cannot know whether or not host A has received req1. If host A did receive it, it will move the

CHILD_SA to the new IKE_SA as usual, and the state information will then be out of sync.

It seems this situation is tricky to handle correctly. Our proposal is as follows: if a host receives a request to rekey the IKE_SA when it has CHILD_SAs in "half-open" state (currently being created or rekeyed), it should reply with NO_PROPOSAL_CHOSEN. If a host receives a request to create or rekey a CHILD_SA after it has started rekeying the IKE_SA, it should reply with NO_ADDITIONAL_SAS.

The case where CHILD_SAs are being closed is even worse. Our recommendation is that if a host receives a request to rekey the IKE_SA when it has CHILD_SAs in "half-closed" state (currently being closed), it should reply with NO_PROPOSAL_CHOSEN. And if a host receives a request to close a CHILD_SA after it has started rekeying the IKE_SA, it should reply with an empty Informational response. This ensures that at least the other peer will eventually notice that the CHILD_SA is still in "half-closed" state and will start a new IKE_SA from scratch.

5.11.9. Closing and Rekeying the IKE_SA

The final case considered in this section occurs if one peer decides to close the IKE_SA while the other peer tries to rekey it.

```
   Host A                      Host B
  --------                    --------
   send req1: SA(..,SPIa1,..),Ni1 -->
                           <-- send req2: D()
                           --> recv req1
   recv req2 <--
```

At this point, host B should probably reply with NO_PROPOSAL_CHOSEN, and host A should reply as usual, close the IKE_SA, and stop retransmitting req1.

```
                           <-- send resp1: N(NO_PROPOSAL_CHOSEN)
   send resp2: ()
```

If host A wants to continue communication with B, it can now start a new IKE_SA.

5.11.10. Summary

If a host receives a request to rekey:

- o a CHILD_SA pair that the host is currently trying to close: reply with NO_PROPOSAL_CHOSEN.

- o a CHILD_SA pair that the host is currently rekeying: reply as usual, but prepare to close redundant SAs later based on the nonces.

- o a CHILD_SA pair that does not exist: reply with NO_PROPOSAL_CHOSEN.

- o the IKE_SA, and the host is currently rekeying the IKE_SA: reply as usual, but prepare to close redundant SAs and move inherited CHILD_SAs later based on the nonces.

- o the IKE_SA, and the host is currently creating, rekeying, or closing a CHILD_SA: reply with NO_PROPOSAL_CHOSEN.

- o the IKE_SA, and the host is currently trying to close the IKE_SA: reply with NO_PROPOSAL_CHOSEN.

If a host receives a request to close:

- o a CHILD_SA pair that the host is currently trying to close: reply without Delete payloads.

- o a CHILD_SA pair that the host is currently rekeying: reply as usual, with Delete payload.

- o a CHILD_SA pair that does not exist: reply without Delete payloads.

- o the IKE_SA, and the host is currently rekeying the IKE_SA: reply as usual, and forget about our own rekeying request.

- o the IKE_SA, and the host is currently trying to close the IKE_SA: reply as usual, and forget about our own close request.

If a host receives a request to create or rekey a CHILD_SA when it is currently rekeying the IKE_SA: reply with NO_ADDITIONAL_SAS.

If a host receives a request to delete a CHILD_SA when it is currently rekeying the IKE_SA: reply without Delete payloads.

5.12. Diffie-Hellman and Rekeying the IKE_SA

There has been some confusion whether doing a new Diffie-Hellman exchange is mandatory when the IKE_SA is rekeyed.

It seems that this case is allowed by the IKEv2 specification. Section 2.18 shows the Diffie-Hellman term (g^ir) in brackets. Section 3.3.3 does not contradict this when it says that including the D-H transform is mandatory: although including the transform is mandatory, it can contain the value "NONE".

However, having the option to skip the Diffie-Hellman exchange when rekeying the IKE_SA does not add useful functionality to the protocol. The main purpose of rekeying the IKE_SA is to ensure that the compromise of old keying material does not provide information about the current keys, or vice versa. This requires performing the Diffie-Hellman exchange when rekeying. Furthermore, it is likely that this option would have been removed from the protocol as unnecessary complexity had it been discussed earlier.

Given this, we recommend that implementations should have a hard-coded policy that requires performing a new Diffie-Hellman exchange when rekeying the IKE_SA. In other words, the initiator should not propose the value "NONE" for the D-H transform, and the responder should not accept such a proposal. This policy also implies that a successful exchange rekeying the IKE_SA always includes the KEi/KEr payloads.

(References: "Rekeying IKE_SAs with the CREATE_CHILD_SA exhange" thread, Oct 2005. "Comments of draft-eronen-ipsec-ikev2-clarifications-02.txt" thread, Apr 2005.)

6. Configuration Payloads

6.1. Assigning IP Addresses

Section 2.9 talks about traffic selector negotiation and mentions that "In support of the scenario described in section 1.1.3, an initiator may request that the responder assign an IP address and tell the initiator what it is."

This sentence is correct, but its placement is slightly confusing. IKEv2 does allow the initiator to request assignment of an IP address from the responder, but this is done using configuration payloads, not traffic selector payloads. An address in a TSi payload in a response does not mean that the responder has assigned that address to the initiator; it only means that if packets matching these traffic selectors are sent by the initiator, IPsec processing can be performed as agreed for this SA. The TSi payload itself does not give the initiator permission to configure the initiator’s TCP/IP stack with the address and use it as its source address.

In other words, IKEv2 does not have two different mechanisms for assigning addresses, but only one: configuration payloads. In the scenario described in Section 1.1.3, both configuration and traffic selector payloads are usually included in the same message, and they often contain the same information in the response message (see Section 6.3 of this document for some examples). However, their semantics are still different.

6.2. Requesting any INTERNAL_IP4/IP6_ADDRESS

When describing the INTERNAL_IP4/IP6_ADDRESS attributes, Section 3.15.1 says that "In a request message, the address specified is a requested address (or zero if no specific address is requested)". The question here is whether "zero" means an address "0.0.0.0" or a zero-length string.

Earlier, the same section also says that "If an attribute in the CFG_REQUEST Configuration Payload is not zero-length, it is taken as a suggestion for that attribute". Also, the table of configuration attributes shows that the length of INTERNAL_IP4_ADDRESS is either "0 or 4 octets", and likewise, INTERNAL_IP6_ADDRESS is either "0 or 17 octets".

Thus, if the client does not request a specific address, it includes a zero-length INTERNAL_IP4/IP6_ADDRESS attribute, not an attribute containing an all-zeroes address. The example in 2.19 is thus incorrect, since it shows the attribute as "INTERNAL_ADDRESS(0.0.0.0)".

However, since the value is only a suggestion, implementations are recommended to ignore suggestions they do not accept; or in other words, to treat the same way a zero-length INTERNAL_IP4_ADDRESS, "0.0.0.0", and any other addresses the implementation does not recognize as a reasonable suggestion.

6.3. INTERNAL_IP4_SUBNET/INTERNAL_IP6_SUBNET

Section 3.15.1 describes the INTERNAL_IP4_SUBNET as "The protected sub-networks that this edge-device protects. This attribute is made up of two fields: the first is an IP address and the second is a netmask. Multiple sub-networks MAY be requested. The responder MAY respond with zero or more sub-network attributes." INTERNAL_IP6_SUBNET is defined in a similar manner.

This raises two questions: first, since this information is usually included in the TSr payload, what functionality does this attribute add? And second, what does this attribute mean in CFG_REQUESTs?

For the first question, there seem to be two sensible interpretations. Clearly TSr (in IKE_AUTH or CREATE_CHILD_SA response) indicates which subnets are accessible through the SA that was just created.

The first interpretation of the INTERNAL_IP4/6_SUBNET attributes is that they indicate additional subnets that can be reached through this gateway, but need a separate SA. According to this interpretation, the INTERNAL_IP4/6_SUBNET attributes are useful mainly when they contain addresses not included in TSr.

The second interpretation is that the INTERNAL_IP4/6_SUBNET attributes express the gateway's policy about what traffic should be sent through the gateway. The client can choose whether other traffic (covered by TSr, but not in INTERNAL_IP4/6_SUBNET) is sent through the gateway or directly to the destination. According to this interpretation, the attributes are useful mainly when TSr contains addresses not included in the INTERNAL_IP4/6_SUBNET attributes.

It turns out that these two interpretations are not incompatible, but rather two sides of the same principle: traffic to the addresses listed in the INTERNAL_IP4/6_SUBNET attributes should be sent via this gateway. If there are no existing IPsec SAs whose traffic selectors cover the address in question, new SAs have to be created.

A couple of examples are given below. For instance, if there are two subnets, 192.0.1.0/26 and 192.0.2.0/24, and the client's request contains the following:

```
CP(CFG_REQUEST) =
  INTERNAL_IP4_ADDRESS()
TSi = (0, 0-65535, 0.0.0.0-255.255.255.255)
TSr = (0, 0-65535, 0.0.0.0-255.255.255.255)
```

Then a valid response could be the following (in which TSr and INTERNAL_IP4_SUBNET contain the same information):

```
CP(CFG_REPLY) =
  INTERNAL_IP4_ADDRESS(192.0.1.234)
  INTERNAL_IP4_SUBNET(192.0.1.0/255.255.255.192)
  INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
TSi = (0, 0-65535, 192.0.1.234-192.0.1.234)
TSr = ((0, 0-65535, 192.0.1.0-192.0.1.63),
       (0, 0-65535, 192.0.2.0-192.0.2.255))
```

In these cases, the INTERNAL_IP4_SUBNET does not really carry any useful information. Another possible reply would have been this:

```
CP(CFG_REPLY) =
  INTERNAL_IP4_ADDRESS(192.0.1.234)
  INTERNAL_IP4_SUBNET(192.0.1.0/255.255.255.192)
  INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
```

```
   TSi = (0, 0-65535, 192.0.1.234-192.0.1.234)
   TSr = (0, 0-65535, 0.0.0.0-255.255.255.255)
```

This would mean that the client can send all its traffic through the gateway, but the gateway does not mind if the client sends traffic not included by INTERNAL_IP4_SUBNET directly to the destination (without going through the gateway).

A different situation arises if the gateway has a policy that requires the traffic for the two subnets to be carried in separate SAs. Then a response like this would indicate to the client that if it wants access to the second subnet, it needs to create a separate SA:

```
   CP(CFG_REPLY) =
     INTERNAL_IP4_ADDRESS(192.0.1.234)
     INTERNAL_IP4_SUBNET(192.0.1.0/255.255.255.192)
     INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
   TSi = (0, 0-65535, 192.0.1.234-192.0.1.234)
   TSr = (0, 0-65535, 192.0.1.0-192.0.1.63)
```

INTERNAL_IP4_SUBNET can also be useful if the client's TSr included only part of the address space. For instance, if the client requests the following:

```
   CP(CFG_REQUEST) =
     INTERNAL_IP4_ADDRESS()
   TSi = (0, 0-65535, 0.0.0.0-255.255.255.255)
   TSr = (0, 0-65535, 192.0.2.155-192.0.2.155)
```

Then the gateway's reply could be this:

```
   CP(CFG_REPLY) =
     INTERNAL_IP4_ADDRESS(192.0.1.234)
     INTERNAL_IP4_SUBNET(192.0.1.0/255.255.255.192)
     INTERNAL_IP4_SUBNET(192.0.2.0/255.255.255.0)
   TSi = (0, 0-65535, 192.0.1.234-192.0.1.234)
   TSr = (0, 0-65535, 192.0.2.155-192.0.2.155)
```

It is less clear what the attributes mean in CFG_REQUESTs, and whether other lengths than zero make sense in this situation (but for INTERNAL_IP6_SUBNET, zero length is not allowed at all!). This document recommends that implementations should not include INTERNAL_IP4_SUBNET or INTERNAL_IP6_SUBNET attributes in CFG_REQUESTs.

For the IPv4 case, this document recommends using only netmasks consisting of some amount of "1" bits followed by "0" bits; for instance, "255.0.255.0" would not be a valid netmask for INTERNAL_IP4_SUBNET.

It is also worthwhile to note that the contents of the INTERNAL_IP4/6_SUBNET attributes do not imply link boundaries. For instance, a gateway providing access to a large company intranet using addresses from the 10.0.0.0/8 block can send a single INTERNAL_IP4_SUBNET attribute (10.0.0.0/255.0.0.0) even if the intranet has hundreds of routers and separate links.

(References: Tero Kivinen’s mail "Intent of couple of attributes in Configuration Payload in IKEv2?", 2004-11-19. Srinivasa Rao Addepalli’s mail "INTERNAL_IP4_SUBNET and INTERNAL_IP6_SUBNET in IKEv2", 2004-09-10. Yoav Nir’s mail "Re: New I-D: IKEv2 Clarifications and Implementation Guidelines", 2005-02-07. "Clarifications open issue: INTERNAL_IP4_SUBNET/NETMASK" thread, April 2005.)

6.4. INTERNAL_IP4_NETMASK

Section 3.15.1 defines the INTERNAL_IP4_NETMASK attribute and says that "The internal network’s netmask. Only one netmask is allowed in the request and reply messages (e.g., 255.255.255.0) and it MUST be used only with an INTERNAL_IP4_ADDRESS attribute".

However, it is not clear what exactly this attribute means, as the concept of "netmask" is not very well defined for point-to-point links (unlike multi-access links, where it means "you can reach hosts inside this netmask directly using layer 2, instead of sending packets via a router"). Even if the operating system’s TCP/IP stack requires a netmask to be configured, for point-to-point links it could be just set to 255.255.255.255. So, why is this information sent in IKEv2?

One possible interpretation would be that the host is given a whole block of IP addresses instead of a single address. This is also what Framed-IP-Netmask does in [RADIUS], the IPCP "subnet mask" extension does in PPP [IPCPSubnet], and the prefix length in the IPv6 Framed-IPv6-Prefix attribute does in [RADIUS6]. However, nothing in the specification supports this interpretation, and discussions on the IPsec WG mailing list have confirmed it was not intended. Section 3.15.1 also says that multiple addresses are assigned using multiple INTERNAL_IP4/6_ADDRESS attributes.

Currently, this document’s interpretation is the following: INTERNAL_IP4_NETMASK in a CFG_REPLY means roughly the same thing as INTERNAL_IP4_SUBNET containing the same information ("send traffic to these addresses through me"), but also implies a link boundary. For instance, the client could use its own address and the netmask to calculate the broadcast address of the link. (Whether the gateway will actually deliver broadcast packets to other VPN clients and/or other nodes connected to this link is another matter.)

An empty INTERNAL_IP4_NETMASK attribute can be included in a CFG_REQUEST to request this information (although the gateway can send the information even when not requested). However, it seems that non-empty values for this attribute do not make sense in CFG_REQUESTs.

Fortunately, Section 4 clearly says that a minimal implementation does not need to include or understand the INTERNAL_IP4_NETMASK attribute, and thus this document recommends that implementations should not use the INTERNAL_IP4_NETMASK attribute or assume that the other peer supports it.

(References: Charlie Kaufman's mail "RE: Proposed Last Call based revisions to IKEv2", 2004-05-27. Email discussion with Tero Kivinen, Jan 2005. Yoav Nir's mail "Re: New I-D: IKEv2 Clarifications and Implementation Guidelines", 2005-02-07. "Clarifications open issue: INTERNAL_IP4_SUBNET/NETMASK" thread, April 2005.)

6.5. Configuration Payloads for IPv6

IKEv2 also defines configuration payloads for IPv6. However, they are based on the corresponding IPv4 payloads and do not fully follow the "normal IPv6 way of doing things".

A client can be assigned an IPv6 address using the INTERNAL_IP6_ADDRESS configuration payload. A minimal exchange could look like this:

```
   CP(CFG_REQUEST) =
     INTERNAL_IP6_ADDRESS()
     INTERNAL_IP6_DNS()
   TSi = (0, 0-65535, :: - FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF)
   TSr = (0, 0-65535, :: - FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF)

   CP(CFG_REPLY) =
     INTERNAL_IP6_ADDRESS(2001:DB8:0:1:2:3:4:5/64)
     INTERNAL_IP6_DNS(2001:DB8:99:88:77:66:55:44)
   TSi = (0, 0-65535, 2001:DB8:0:1:2:3:4:5 - 2001:DB8:0:1:2:3:4:5)
   TSr = (0, 0-65535, :: - FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF)
```

In particular, IPv6 stateless autoconfiguration or router advertisement messages are not used; neither is neighbor discovery.

The client can also send a non-empty INTERNAL_IP6_ADDRESS attribute in the CFG_REQUEST to request a specific address or interface identifier. The gateway first checks if the specified address is acceptable, and if it is, returns that one. If the address was not acceptable, the gateway will attempt to use the interface identifier with some other prefix; if even that fails, the gateway will select another interface identifier.

The INTERNAL_IP6_ADDRESS attribute also contains a prefix length field. When used in a CFG_REPLY, this corresponds to the INTERNAL_IP4_NETMASK attribute in the IPv4 case (and indeed, was called INTERNAL_IP6_NETMASK in earlier versions of the IKEv2 draft). See the previous section for more details.

While this approach to configuring IPv6 addresses is reasonably simple, it has some limitations: IPsec tunnels configured using IKEv2 are not fully-featured "interfaces" in the IPv6 addressing architecture [IPv6Addr] sense. In particular, they do not necessarily have link-local addresses, and this may complicate the use of protocols that assume them, such as [MLDv2]. (Whether they are called "interfaces" in some particular operating system is a different issue.)

(References: "VPN remote host configuration IPv6 ?" thread, May 2004. "Clarifications open issue: INTERNAL_IP4_SUBNET/NETMASK" thread, April 2005.)

6.6. INTERNAL_IP6_NBNS

Section 3.15.1 defines the INTERNAL_IP6_NBNS attribute for sending the IPv6 address of NetBIOS name servers.

However, NetBIOS is not defined for IPv6 and probably never will be. Thus, this attribute most likely does not make much sense.

(Pointed out by Bernard Aboba in the IP Configuration Security (ICOS) BoF at IETF62.)

6.7. INTERNAL_ADDRESS_EXPIRY

Section 3.15.1 defines the INTERNAL_ADDRESS_EXPIRY attribute as "Specifies the number of seconds that the host can use the internal IP address. The host MUST renew the IP address before this expiry time. Only one of these attributes MAY be present in the reply."

Expiry times and explicit renewals are primarily useful in environments like DHCP, where the server cannot reliably know when the client has gone away. However, in IKEv2 this is known, and the gateway can simply free the address when the IKE_SA is deleted.

Also, Section 4 says that supporting renewals is not mandatory. Given that this functionality is usually not needed, we recommend that gateways should not send the INTERNAL_ADDRESS_EXPIRY attribute. (And since this attribute does not seem to make much sense for CFG_REQUESTs, clients should not send it either.)

Note that according to Section 4, clients are required to understand INTERNAL_ADDRESS_EXPIRY if they receive it. A minimum implementation would use the value to limit the lifetime of the IKE_SA.

(References: Tero Kivinen’s mail "Comments of draft-eronen-ipsec-ikev2-clarifications-02.txt", 2005-04-05. "Questions about internal address" thread, April 2005.)

6.8. Address Assignment Failures

If the responder encounters an error while attempting to assign an IP address to the initiator, it responds with an INTERNAL_ADDRESS_FAILURE notification as described in Section 3.10.1. However, there are some more complex error cases.

First, if the responder does not support configuration payloads at all, it can simply ignore all configuration payloads. This type of implementation never sends INTERNAL_ADDRESS_FAILURE notifications. If the initiator requires the assignment of an IP address, it will treat a response without CFG_REPLY as an error.

A second case is where the responder does support configuration payloads, but only for particular type of addresses (IPv4 or IPv6). Section 4 says that "A minimal IPv4 responder implementation will ignore the contents of the CP payload except to determine that it includes an INTERNAL_IP4_ADDRESS attribute". If, for instance, the initiator includes both INTERNAL_IP4_ADDRESS and INTERNAL_IP6_ADDRESS in the CFG_REQUEST, an IPv4-only responder can thus simply ignore the IPv6 part and process the IPv4 request as usual.

A third case is where the initiator requests multiple addresses of a type that the responder supports: what should happen if some (but not all) of the requests fail? It seems that an optimistic approach would be the best one here: if the responder is able to assign at least one address, it replies with those; it sends INTERNAL_ADDRESS_FAILURE only if no addresses can be assigned.

(References: "ikev2 and internal_ivpn_address" thread, June 2005.)

7. Miscellaneous Issues

7.1. Matching ID_IPV4_ADDR and ID_IPV6_ADDR

When using the ID_IPV4_ADDR/ID_IPV6_ADDR identity types in IDi/IDr payloads, IKEv2 does not require this address to match anything in the TSi/TSr payloads. For example, in a site-to-site VPN between two security gateways, the gateways could authenticate each other as ID_IPV4_ADDR(192.0.1.1) and ID_IPV4_ADDR(192.0.2.1), and then create a CHILD_SA for protecting traffic between 192.0.1.55/32 (a host behind the first security gateway) and 192.0.2.240/28 (a network behind the second security gateway). The authenticated identities (IDi/IDr) are linked to the authorized traffic selectors (TSi/TSr) using "Child SA Authorization Data" in the Peer Authorization Database (PAD).

Furthermore, IKEv2 does not require that the addresses in ID_IPV4_ADDR/ID_IPV6_ADDR match the address in the IP header of the IKE packets. However, other specifications may place additional requirements regarding this. For example, [PKI4IPsec] requires that implementation must be capable of comparing the addresses in the ID_IPV4_ADDR/ID_IPV6_ADDR with the addresses in the IP header of the IKE packets, and this comparison must be enabled by default.

(References: "Identities types IP address,FQDN/user FQDN and DN and its usage in preshared key authentication" thread, Jan 2005. "Matching ID_IPV4_ADDR and ID_IPV6_ADDR" thread, May 2006.)

7.2. Relationship of IKEv2 to RFC 4301

The IKEv2 specification refers to [RFC4301], but it never clearly defines the exact relationship.

However, there are some requirements in the specification that make it clear that IKEv2 requires [RFC4301]. In other words, an implementation that does IPsec processing strictly according to [RFC2401] cannot be compliant with the IKEv2 specification.

One such example can be found in Section 2.24: "Specifically, tunnel encapsulators and decapsulators for all tunnel-mode SAs created by IKEv2 [...] MUST implement the tunnel encapsulation and decapsulation processing specified in [RFC4301] to prevent discarding of ECN congestion indications."

Nevertheless, the changes required to existing [RFC2401] implementations are not very large, especially since supporting many of the new features (such as Extended Sequence Numbers) is optional.

7.3. Reducing the Window Size

In IKEv2, the window size is assumed to be a (possibly configurable) property of a particular implementation and is not related to congestion control (unlike the window size in TCP, for instance).

In particular, it is not defined what the responder should do when it receives a SET_WINDOW_SIZE notification containing a smaller value than is currently in effect. Thus, there is currently no way to reduce the window size of an existing IKE_SA. However, when rekeying an IKE_SA, the new IKE_SA starts with window size 1 until it is explicitly increased by sending a new SET_WINDOW_SIZE notification.

(References: Tero Kivinen’s mail "Comments of draft-eronen-ipsec-ikev2-clarifications-02.txt", 2005-04-05.)

7.4. Minimum Size of Nonces

Section 2.10 says that "Nonces used in IKEv2 MUST be randomly chosen, MUST be at least 128 bits in size, and MUST be at least half the key size of the negotiated prf."

However, the initiator chooses the nonce before the outcome of the negotiation is known. In this case, the nonce has to be long enough for all the PRFs being proposed.

7.5. Initial Zero Octets on Port 4500

It is not clear whether a peer sending an IKE_SA_INIT request on port 4500 should include the initial four zero octets. Section 2.23 talks about how to upgrade to tunneling over port 4500 after message 2, but it does not say what to do if message 1 is sent on port 4500.

> IKE MUST listen on port 4500 as well as port 500.
>
> [...]
>
> The IKE initiator MUST check these payloads if present and if they do not match the addresses in the outer packet MUST tunnel all future IKE and ESP packets associated with this IKE_SA over UDP port 4500.
>
> To tunnel IKE packets over UDP port 4500, the IKE header has four octets of zero prepended and the result immediately follows the UDP header. [...]

The very beginning of Section 2 says "... though IKE messages may also be received on UDP port 4500 with a slightly different format (see section 2.23)."

That "slightly different format" is only described in discussing what to do after changing to port 4500. However, [RFC3948] shows clearly the format has the initial zeros even for initiators on port 4500. Furthermore, without the initial zeros, the processing engine cannot determine whether the packet is an IKE packet or an ESP packet.

Thus, all packets sent on port 4500 need the four-zero prefix; otherwise, the receiver won't know how to handle them.

7.6. Destination Port for NAT Traversal

Section 2.23 says that "an IPsec endpoint that discovers a NAT between it and its correspondent MUST send all subsequent traffic to and from port 4500".

This sentence is misleading. The peer "outside" the NAT uses source port 4500 for the traffic it sends, but the destination port is, of course, taken from packets sent by the peer behind the NAT. This port number is usually dynamically allocated by the NAT.

7.7. SPI Values for Messages outside an IKE_SA

The IKEv2 specification is not quite clear what SPI values should be used in the IKE header for the small number of notifications that are allowed to be sent outside an IKE_SA. Note that such notifications are explicitly not Informational exchanges; Section 1.5 makes it clear that these are one-way messages that must not be responded to.

There are two cases when such a one-way notification can be sent: INVALID_IKE_SPI and INVALID_SPI.

In case of INVALID_IKE_SPI, the message sent is a response message, and Section 2.21 says that "If a response is sent, the response MUST be sent to the IP address and port from whence it came with the same IKE SPIs and the Message ID copied."

In case of INVALID_SPI, however, there are no IKE SPI values that would be meaningful to the recipient of such a notification. Also, the message sent is now an INFORMATIONAL request. A strict interpretation of the specification would require the sender to invent garbage values for the SPI fields. However, we think this was not the intention, and using zero values is acceptable.

(References: "INVALID_IKE_SPI" thread, June 2005.)

7.8. Protocol ID/SPI Fields in Notify Payloads

Section 3.10 says that the Protocol ID field in Notify payloads "For notifications that do not relate to an existing SA, this field MUST be sent as zero and MUST be ignored on receipt". However, the specification does not clearly say which notifications are related to existing SAs and which are not.

Since the main purpose of the Protocol ID field is to specify the type of the SPI, our interpretation is that the Protocol ID field should be non-zero only when the SPI field is non-empty.

There are currently only two notifications where this is the case: INVALID_SELECTORS and REKEY_SA.

7.9. Which message should contain INITIAL_CONTACT

The description of the INITIAL_CONTACT notification in Section 3.10.1 says that "This notification asserts that this IKE_SA is the only IKE_SA currently active between the authenticated identities". However, neither Section 2.4 nor 3.10.1 says in which message this payload should be placed.

The general agreement is that INITIAL_CONTACT is best communicated in the first IKE_AUTH request, not as a separate exchange afterwards.

(References: "Clarifying the use of INITIAL_CONTACT in IKEv2" thread, April 2005. "Initial Contact messages" thread, December 2004. "IKEv2 and Initial Contact" thread, September 2004 and April 2005.)

7.10. Alignment of Payloads

Many IKEv2 payloads contain fields marked as "RESERVED", mostly because IKEv1 had them, and partly because they make the pictures easier to draw. In particular, payloads in IKEv2 are not, in general, aligned to 4-octet boundaries. (Note that payloads were not aligned to 4-octet boundaries in IKEv1 either.)

(References: "IKEv2: potential 4-byte alignment problem" thread, June 2004.)

7.11. Key Length Transform Attribute

Section 3.3.5 says that "The only algorithms defined in this document that accept attributes are the AES based encryption, integrity, and pseudo-random functions, which require a single attribute specifying key width."

This is incorrect. The AES-based integrity and pseudo-random functions defined in [IKEv2] always use a 128-bit key. In fact, there are currently no integrity or PRF algorithms that use the key length attribute (and we recommend that they should not be defined in the future either).

For encryption algorithms, the situation is slightly more complex since there are three different types of algorithms:

- The key length attribute is never used with algorithms that use a fixed length key, such as DES and IDEA.

- The key length attribute is always included for the currently defined AES-based algorithms (Cipher Block Chaining (CBC), Counter (CTR) Mode, Counter with CBC-MAC (CCM), and Galois/Counter Mode (GCM)). Omitting the key length attribute is not allowed; if the proposal does not contain it, the proposal has to be rejected.

- For other algorithms, the key length attribute can be included but is not mandatory. These algorithms include, e.g., RC5, CAST, and BLOWFISH. If the key length attribute is not included, the default value specified in [RFC2451] is used.

7.12. IPsec IANA Considerations

There are currently three different IANA registry files that contain important numbers for IPsec: ikev2-registry, isakmp-registry, and ipsec-registry. Implementers should note that IKEv2 may use numbers different from those of IKEv1 for a particular algorithm.

For instance, an encryption algorithm can have up to three different numbers: the IKEv2 "Transform Type 1" identifier in ikev2-registry, the IKEv1 phase 1 "Encryption Algorithm" identifier in ipsec-registry, and the IKEv1 phase 2 "IPSEC ESP Transform Identifier" isakmp-registry. Although some algorithms have the same number in all three registries, the registries are not identical.

Similarly, an integrity algorithm can have at least the IKEv2 "Transform Type 3" identifier in ikev2-registry, the IKEv1 phase 2 "IPSEC AH Transform Identifier" in isakmp-registry, and the IKEv1 phase 2 ESP "Authentication Algorithm Security Association Attribute" identifier in isakmp-registry. And there is also the IKEv1 phase 1 "Hash Algorithm" list in ipsec-registry.

This issue needs special care also when writing a specification for how a new algorithm is used with IPsec.

7.13. Combining ESP and AH

The IKEv2 specification contains some misleading text about how ESP and AH can be combined.

IKEv2 is based on [RFC4301], which does not include "SA bundles" that were part of [RFC2401]. While a single packet can go through IPsec processing multiple times, each of these passes uses a separate SA, and the passes are coordinated by the forwarding tables. In IKEv2, each of these SAs has to be created using a separate CREATE_CHILD_SA exchange. Thus, the text in Section 2.7 about a single proposal containing both ESP and AH is incorrect.

Moreover, the combination of ESP and AH (between the same endpoints) had already become largely obsolete in 1998 when RFC 2406 was published. Our recommendation is that IKEv2 implementations should not support this combination, and implementers should not assume the combination can be made to work in an interoperable manner.

(References: "Rekeying SA bundles" thread, Oct 2005.)

8. Implementation Mistakes

Some implementers at the early IKEv2 bakeoffs didn't do everything correctly. This may seem like an obvious statement, but it is probably useful to list a few things that were clear in the document, but that some implementers didn't do. All of these things caused interoperability problems.

- o Some implementations continued to send traffic on a CHILD_SA after it was rekeyed, even after receiving an DELETE payload.

- o After rekeying an IKE_SA, some implementations did not reset their message counters to zero. One set the counter to 2, another did not reset the counter at all.

- o Some implementations could only handle a single pair of traffic selectors or would only process the first pair in the proposal.

- o Some implementations responded to a delete request by sending an empty INFORMATIONAL response and then initiated their own INFORMATIONAL exchange with the pair of SAs to delete.

- o Although this did not happen at the bakeoff, from the discussion there, it is clear that some people had not implemented message window sizes correctly. Some implementations might have sent messages that did not fit into the responder’s message windows, and some implementations may not have torn down an SA if they did not ever receive a message that they know they should have.

9. Security Considerations

This document does not introduce any new security considerations to IKEv2. If anything, clarifying complex areas of the specification can reduce the likelihood of implementation problems that may have security implications.

10. Acknowledgments

This document is mainly based on conversations on the IPsec WG mailing list. The authors would especially like to thank Bernard Aboba, Jari Arkko, Vijay Devarapalli, William Dixon, Francis Dupont, Alfred Hoenes, Mika Joutsenvirta, Charlie Kaufman, Stephen Kent, Tero Kivinen, Yoav Nir, Michael Richardson, and Joel Snyder for their contributions.

In addition, the authors would like to thank all the participants of the first public IKEv2 bakeoff, held in Santa Clara in February 2005, for their questions and proposed clarifications.

11. References

11.1. Normative References

[IKEv2] Kaufman, C., Ed., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[IKEv2ALG] Schiller, J., "Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2)", RFC 4307, December 2005.

[PKCS1v20] Kaliski, B. and J. Staddon, "PKCS #1: RSA Cryptography Specifications Version 2.0", RFC 2437, October 1998.

[PKCS1v21] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

11.2. Informative References

[Aura05] Aura, T., Roe, M., and A. Mohammed, "Experiences with Host-to-Host IPsec", 13th International Workshop on Security Protocols, Cambridge, UK, April 2005.

[EAP] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[HashUse] Hoffman, P., "Use of Hash Algorithms in IKE and IPsec", Work in Progress, July 2006.

[IPCPSubnet] Cisco Systems, Inc., "IPCP Subnet Mask Support Enhancements", http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121newft/121limit/121dc/121dc3/ipcp_msk.htm, January 2003.

[IPv6Addr] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[MIPv6] Johnson, D., Perkins, C., and J. Arkko, "Mobility Support in IPv6", RFC 3775, June 2004.

[MLDv2] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.

[NAI] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, December 2005.

[PKI4IPsec] Korver, B., "Internet PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX", Work in Progress, April 2006.

[RADEAP] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.

[RADIUS] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RADIUS6] Aboba, B., Zorn, G., and D. Mitton, "RADIUS and IPv6", RFC 3162, August 2001.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC2451] Pereira, R. and R. Adams, "The ESP CBC-Mode Cipher Algorithms", RFC 2451, November 1998.

[RFC2822] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[RFC3664] Hoffman, P., "The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE)", RFC 3664, January 2004.

[RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, January 2005.

[RFC4434] Hoffman, P., "The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE)", RFC 4434, February 2006.

[RFC822] Crocker, D., "Standard for the format of ARPA Internet text messages", RFC 822, August 1982.

[ReAuth] Nir, Y., "Repeated Authentication in Internet Key Exchange (IKEv2) Protocol", RFC 4478, April 2006.

[SCVP] Freeman, T., Housley, R., Malpani, A., Cooper, D., and T. Polk, "Simple Certificate Validation Protocol (SCVP)", Work in Progress, June 2006.

Appendix A. Exchanges and Payloads

This appendix contains a short summary of the IKEv2 exchanges, and what payloads can appear in which message. This appendix is purely informative; if it disagrees with the body of this document or the IKEv2 specification, the other text is considered correct.

Vendor-ID (V) payloads may be included in any place in any message. This sequence shows what are, in our opinion, the most logical places for them.

The specification does not say which messages can contain N(SET_WINDOW_SIZE). It can possibly be included in any message, but it is not yet shown below.

A.1. IKE_SA_INIT Exchange

```
request             --> [N(COOKIE)],
                        SA, KE, Ni,
                        [N(NAT_DETECTION_SOURCE_IP)+,
                         N(NAT_DETECTION_DESTINATION_IP)],
                        [V+]

normal response     <-- SA, KE, Nr,
(no cookie)             [N(NAT_DETECTION_SOURCE_IP),
                         N(NAT_DETECTION_DESTINATION_IP)],
                        [[N(HTTP_CERT_LOOKUP_SUPPORTED)], CERTREQ+],
                        [V+]
```

A.2. IKE_AUTH Exchange without EAP

```
request             --> IDi, [CERT+],
                        [N(INITIAL_CONTACT)],
                        [[N(HTTP_CERT_LOOKUP_SUPPORTED)], CERTREQ+],
                        [IDr],
                        AUTH,
                        [CP(CFG_REQUEST)],
                        [N(IPCOMP_SUPPORTED)+],
                        [N(USE_TRANSPORT_MODE)],
                        [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                        [N(NON_FIRST_FRAGMENTS_ALSO)],
                        SA, TSi, TSr,
                        [V+]
```

```
   response            <-- IDr, [CERT+],
                           AUTH,
                           [CP(CFG_REPLY)],
                           [N(IPCOMP_SUPPORTED)],
                           [N(USE_TRANSPORT_MODE)],
                           [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                           [N(NON_FIRST_FRAGMENTS_ALSO)],
                           SA, TSi, TSr,
                           [N(ADDITIONAL_TS_POSSIBLE)],
                           [V+]
```

A.3. IKE_AUTH Exchange with EAP

```
   first request       --> IDi,
                           [N(INITIAL_CONTACT)],
                           [[N(HTTP_CERT_LOOKUP_SUPPORTED)], CERTREQ+],
                           [IDr],
                           [CP(CFG_REQUEST)],
                           [N(IPCOMP_SUPPORTED)+],
                           [N(USE_TRANSPORT_MODE)],
                           [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                           [N(NON_FIRST_FRAGMENTS_ALSO)],
                           SA, TSi, TSr,
                           [V+]

   first response      <-- IDr, [CERT+], AUTH,
                           EAP,
                           [V+]

                     / --> EAP
   repeat 1..N times |
                     \ <-- EAP

   last request        --> AUTH

   last response       <-- AUTH,
                           [CP(CFG_REPLY)],
                           [N(IPCOMP_SUPPORTED)],
                           [N(USE_TRANSPORT_MODE)],
                           [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                           [N(NON_FIRST_FRAGMENTS_ALSO)],
                           SA, TSi, TSr,
                           [N(ADDITIONAL_TS_POSSIBLE)],
                           [V+]
```

A.4. CREATE_CHILD_SA Exchange for Creating/Rekeying CHILD_SAs

```
request             --> [N(REKEY_SA)],
                        [N(IPCOMP_SUPPORTED)+],
                        [N(USE_TRANSPORT_MODE)],
                        [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                        [N(NON_FIRST_FRAGMENTS_ALSO)],
                        SA, Ni, [KEi], TSi, TSr

response            <-- [N(IPCOMP_SUPPORTED)],
                        [N(USE_TRANSPORT_MODE)],
                        [N(ESP_TFC_PADDING_NOT_SUPPORTED)],
                        [N(NON_FIRST_FRAGMENTS_ALSO)],
                        SA, Nr, [KEr], TSi, TSr,
                        [N(ADDITIONAL_TS_POSSIBLE)]
```

A.5. CREATE_CHILD_SA Exchange for Rekeying the IKE_SA

```
request             --> SA, Ni, [KEi]

response            <-- SA, Nr, [KEr]
```

A.6. INFORMATIONAL Exchange

```
request             --> [N+],
                        [D+],
                        [CP(CFG_REQUEST)]

response            <-- [N+],
                        [D+],
                        [CP(CFG_REPLY)]
```

Authors' Addresses

Pasi Eronen
Nokia Research Center
P.O. Box 407
FIN-00045 Nokia Group
Finland

EMail: pasi.eronen@nokia.com

Paul Hoffman
VPN Consortium
127 Segre Place
Santa Cruz, CA 95060
USA

EMail: paul.hoffman@vpnc.org

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).