Network Working Group
Internet-Draft
Obsoletes: 8229 (if approved)
Intended status: Standards Track
Expires: November 7, 2020

V. Smyslov
ELVIS-PLUS

May 6, 2020

TCP Encapsulation of IKE and IPsec Packets
draft-smyslov-ipsecme-rfc8229bis-00

Abstract

This document describes a method to transport Internet Key Exchange Protocol (IKE) and IPsec packets over a TCP connection for traversing network middleboxes that may block IKE negotiation over UDP. This method, referred to as "TCP encapsulation", involves sending both IKE packets for Security Association establishment and Encapsulating Security Payload (ESP) packets over a TCP connection. This method is intended to be used as a fallback option when IKE cannot be negotiated over UDP.

TCP encapsulation for IKE and IPsec was defined in [RFC8229]. This document updates specification for TCP encapsulation by including additional calarifications obtained during implementation and deployment of this method. This documents makes RFC8229 obsolete.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
   1.1. Prior Work and Motivation 4
2. Terminology and Notation 4
3. Configuration 5
4. TCP-Encapsulated Header Formats 6
   4.1. TCP-Encapsulated IKE Header Format 6
   4.2. TCP-Encapsulated ESP Header Format 7
5. TCP-Encapsulated Stream Prefix 7
6. Applicability 8
   6.1. Recommended Fallback from UDP 8
7. Using TCP Encapsulation 9
   7.1. Connection Establishment and Teardown 9
   7.2. Retransmissions 11
   7.3. Cookies and Puzzles 11
   7.4. Error Handling in the IKE_SA_INIT 12
   7.5. NAT Detection Payloads 13
   7.6. Considerations for Keep-Alives and Dead Peer Detection . 13
   7.7. Implications of TCP Encapsulation on IPsec SA Processing 14
8. Interaction with IKEv2 Extensions 14
   8.1. MOBIKE Protocol 14
   8.2. IKE Redirect 15
   8.3. IKEv2 Session Resumption 16
   8.4. IKEv2 Protocol Support for High Availability 16
   8.5. IKEv2 Fragmentation 17
9. Middlebox Considerations 17
10. Performance Considerations 17
   10.1. TCP-in-TCP 17
   10.2. Added Reliability for Unreliable Protocols 18
   10.3. Quality-of-Service Markings 19
   10.4. Maximum Segment Size 19
   10.5. Tunneling ECN in TCP 19

11. Security Considerations 19
12. IANA Considerations 20
13. References 20
13.1. Normative References 20
13.2. Informative References 21

Appendix A. Using TCP Encapsulation with TLS 23
Appendix B. Example Exchanges of TCP Encapsulation with TLS 1.2 24
B.1. Establishing an IKE Session 24
B.2. Deleting an IKE Session 25
B.3. Re-establishing an IKE Session 26
B.4. Using MOBIKE between UDP and TCP Encapsulation 27
Acknowledgements 29
Author's Address 29

1. Introduction

The Internet Key Exchange Protocol version 2 (IKEv2) [RFC7296] is a protocol for establishing IPsec Security Associations (SAs), using IKE messages over UDP for control traffic, and using Encapsulating Security Payload (ESP) [RFC4303] messages for encrypted data traffic. Many network middleboxes that filter traffic on public hotspots block all UDP traffic, including IKE and IPsec, but allow TCP connections through because they appear to be web traffic. Devices on these networks that need to use IPsec (to access private enterprise networks, to route Voice over IP calls to carrier networks, or because of security policies) are unable to establish IPsec SAs. This document defines a method for encapsulating IKE control messages as well as IPsec data messages within a TCP connection.

Using TCP as a transport for IPsec packets adds a third option to the list of traditional IPsec transports:

1. Direct. Currently, IKE negotiations begin over UDP port 500. If no Network Address Translation (NAT) device is detected between the Initiator and the Responder, then subsequent IKE packets are sent over UDP port 500, and IPsec data packets are sent using ESP.

2. UDP Encapsulation [RFC3948]. If a NAT is detected between the Initiator and the Responder, then subsequent IKE packets are sent over UDP port 4500 with four bytes of zero at the start of the

UDP payload, and ESP packets are sent out over UDP port 4500. Some peers default to using UDP encapsulation even when no NAT is detected on the path, as some middleboxes do not support IP protocols other than TCP and UDP.

3. TCP Encapsulation. If the other two methods are not available or appropriate, IKE negotiation packets as well as ESP packets can be sent over a single TCP connection to the peer.

Direct use of ESP or UDP encapsulation should be preferred by IKE implementations due to performance concerns when using TCP encapsulation (Section 10). Most implementations should use TCP

encapsulation only on networks where negotiation over UDP has been attempted without receiving responses from the peer or if a network is known to not support UDP.

1.1. Prior Work and Motivation

Encapsulating IKE connections within TCP streams is a common approach to solve the problem of UDP packets being blocked by network middleboxes. The specific goals of this document are as follows:

- To promote interoperability by defining a standard method of framing IKE and ESP messages within TCP streams.

- To be compatible with the current IKEv2 standard without requiring modifications or extensions.

- To use IKE over UDP by default to avoid the overhead of other alternatives that always rely on TCP or Transport Layer Security (TLS) [RFC5246][RFC8446].

Some previous alternatives include:

Cellular Network Access
Interworking Wireless LAN (IWLAN) uses IKEv2 to create secure connections to cellular carrier networks for making voice calls and accessing other network services over Wi-Fi networks. 3GPP has recommended that IKEv2 and ESP packets be sent within a TLS connection to be able to establish connections on restrictive networks.

ISAKMP over TCP
Various non-standard extensions to the Internet Security Association and Key Management Protocol (ISAKMP) have been deployed that send IPsec traffic over TCP or TCP-like packets.

Secure Sockets Layer (SSL) VPNs
Many proprietary VPN solutions use a combination of TLS and IPsec in order to provide reliability. These often run on TCP port 443.

IKEv2 over TCP
IKEv2 over TCP as described in [I-D.ietf-ipsecme-ike-tcp] is used to avoid UDP fragmentation.

2. Terminology and Notation

This document distinguishes between the IKE peer that initiates TCP connections to be used for TCP encapsulation and the roles of Initiator and Responder for particular IKE messages. During the

course of IKE exchanges, the role of IKE Initiator and Responder may swap for a given SA (as with IKE SA rekeys), while the Initiator of the TCP connection is still responsible for tearing down the TCP connection and re-establishing it if necessary. For this reason, this document will use the term "TCP Originator" to indicate the IKE peer that initiates TCP connections. The peer that receives TCP connections will be referred to as the "TCP Responder". If an IKE SA is rekeyed one or more times, the TCP Originator MUST remain the peer that originally initiated the first IKE SA.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Configuration

One of the main reasons to use TCP encapsulation is that UDP traffic may be entirely blocked on a network. Because of this, support for TCP encapsulation is not specifically negotiated in the IKE exchange. Instead, support for TCP encapsulation must be pre-configured on both

the TCP Originator and the TCP Responder.

Implementations MUST support TCP encapsulation on TCP port 4500, which is reserved for IPsec NAT traversal.

Beyond a flag indicating support for TCP encapsulation, the configuration for each peer can include the following optional parameters:

- Alternate TCP ports on which the specific TCP Responder listens for incoming connections. Note that the TCP Originator may initiate TCP connections to the TCP Responder from any local port.

- An extra framing protocol to use on top of TCP to further encapsulate the stream of IKE and IPsec packets. See Appendix B for a detailed discussion.

Since TCP encapsulation of IKE and IPsec packets adds overhead and has potential performance trade-offs compared to direct or UDP-encapsulated SAs (as described in Section 10), implementations SHOULD prefer ESP direct or UDP-encapsulated SAs over TCP-encapsulated SAs when possible.

4. TCP-Encapsulated Header Formats

Like UDP encapsulation, TCP encapsulation uses the first four bytes of a message to differentiate IKE and ESP messages. TCP encapsulation also adds a 16-bit Length field that precedes every message to define the boundaries of messages within a stream. The message length plus the length of the field itself is sent this field. If the first 32 bits of the message are zeros (a non-ESP marker), then the contents comprise an IKE message. Otherwise, the contents comprise an ESP message. Authentication Header (AH) messages are not supported for TCP encapsulation.

Note, that since TCP header is longer than UDP header, and TCP encapsulation also requires prepending of 16-bit Length field, some very long ESP and IKE messages that could be sent over UDP cannot be

encapsulated in TCP, because their total length after encapsulation would exceed 65535 and thus could not be represented in Length field.

Although a TCP stream may be able to send very long messages, implementations SHOULD limit message lengths to typical UDP datagram ESP payload lengths. The maximum message length is used as the effective MTU for connections that are being encrypted using ESP, so the maximum message length will influence characteristics of inner connections, such as the TCP Maximum Segment Size (MSS).

Note that this method of encapsulation will also work for placing IKE and ESP messages within any protocol that presents a stream abstraction, beyond TCP.

4.1. TCP-Encapsulated IKE Header Format

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Non-ESP Marker                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                      IKE header [RFC7296]                     ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1

The IKE header is preceded by a 16-bit Length field in network byte order that specifies the length of the IKE message (including the non-ESP marker) within the TCP stream. As with IKE over UDP port

4500, a zeroed 32-bit non-ESP marker is inserted before the start of the IKE header in order to differentiate the traffic from ESP traffic between the same addresses and ports.

o Length (2 octets, unsigned integer) - Length of the IKE packet, including the Length field and non-ESP marker. Valuse 0 and 1 MUST NOT appear in this field. The receiver MUST treat these values in the Length field as fatal error and MUST close TCP

session in this case.

4.2. TCP-Encapsulated ESP Header Format

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                     ESP header [RFC4303]                      ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2

The ESP header is preceded by a 16-bit Length field in network byte order that specifies the length of the ESP packet within the TCP stream.

The Security Parameter Index (SPI) field [RFC7296] in the ESP header MUST NOT be a zero value.

- o Length (2 octets, unsigned integer) - Length of the ESP packet, including the Length field. Valuse 0 and 1 MUST NOT appear in this field. The receiver MUST treat these values in the Length field as fatal error and MUST close TCP session in this case.

5. TCP-Encapsulated Stream Prefix

Each stream of bytes used for IKE and IPsec encapsulation MUST begin with a fixed sequence of six bytes as a magic value, containing the characters "IKETCP" as ASCII values. This value is intended to identify and validate that the TCP connection is being used for TCP encapsulation as defined in this document, to avoid conflicts with the prevalence of previous non-standard protocols that used TCP port 4500. This value is only sent once, by the TCP Originator only, at the beginning of any stream of IKE and ESP messages.

If other framing protocols are used within TCP to further encapsulate

or encrypt the stream of IKE and ESP messages, the stream prefix must be at the start of the TCP Originator's IKE and ESP message stream within the added protocol layer (Appendix B). Although some framing protocols do support negotiating inner protocols, the stream prefix should always be used in order for implementations to be as generic as possible and not rely on other framing protocols on top of TCP.

```
 0      1      2      3      4      5
+------+------+------+------+------+------+
| 0x49 | 0x4b | 0x45 | 0x54 | 0x43 | 0x50 |
+------+------+------+------+------+------+
```

Figure 3

6. Applicability

TCP encapsulation is applicable only when it has been configured to be used with specific IKE peers. If a Responder is configured to use TCP encapsulation, it MUST listen on the configured port(s) in case any peers will initiate new IKE sessions. Initiators MAY use TCP encapsulation for any IKE session to a peer that is configured to support TCP encapsulation, although it is recommended that Initiators should only use TCP encapsulation when traffic over UDP is blocked.

Since the support of TCP encapsulation is a configured property, not a negotiated one, it is recommended that if there are multiple IKE endpoints representing a single peer (such as multiple machines with different IP addresses when connecting by Fully Qualified Domain Name, or endpoints used with IKE redirection), all of the endpoints equally support TCP encapsulation.

If TCP encapsulation is being used for a specific IKE SA, all messages for that IKE SA and its Child SAs MUST be sent over a TCP connection until the SA is deleted or IKEv2 Mobility and Multihoming (MOBIKE) is used to change the SA endpoints and/or the encapsulation protocol. See Section 8.1 for more details on using MOBIKE to transition between encapsulation modes.

6.1. Recommended Fallback from UDP

Since UDP is the preferred method of transport for IKE messages, implementations that use TCP encapsulation should have an algorithm for deciding when to use TCP after determining that UDP is unusable. If an Initiator implementation has no prior knowledge about the network it is on and the status of UDP on that network, it SHOULD always attempt to negotiate IKE over UDP first. IKEv2 defines how to use retransmission timers with IKE messages and, specifically,

IKE_SA_INIT messages [RFC7296]. Generally, this means that the implementation will define a frequency of retransmission and the maximum number of retransmissions allowed before marking the IKE SA as failed. An implementation can attempt negotiation over TCP once it has hit the maximum retransmissions over UDP, or slightly before to reduce connection setup delays. It is recommended that the initial message over UDP be retransmitted at least once before falling back to TCP, unless the Initiator knows beforehand that the network is likely to block UDP.

When switching from UDP to TCP a new IKE_SA_INIT exchange MUST be initiated with new Initiator's SPI and with recalculated content of NAT_DETECTION_SOURCE_IP notification.

7. Using TCP Encapsulation

7.1. Connection Establishment and Teardown

When the IKE Initiator uses TCP encapsulation, it will initiate a TCP connection to the Responder using the configured TCP port. The first bytes sent on the stream MUST be the stream prefix value (Section 5). After this prefix, encapsulated IKE messages will negotiate the IKE SA and initial Child SA [RFC7296]. After this point, both encapsulated IKE (Figure 1) and ESP (Figure 2) messages will be sent over the TCP connection. The TCP Responder MUST wait for the entire stream prefix to be received on the stream before trying to parse out any IKE or ESP messages. The stream prefix is sent only once, and only by the TCP Originator.

In order to close an IKE session, either the Initiator or Responder SHOULD gracefully tear down IKE SAs with DELETE payloads. Once the SA has been deleted, the TCP Originator SHOULD close the TCP connection if it does not intend to use the connection for another IKE session to the TCP Responder. If the connection is left idle and the TCP Responder needs to clean up resources, the TCP Responder MAY close the TCP connection.

An unexpected FIN or a TCP Reset on the TCP connection may indicate a loss of connectivity, an attack, or some other error. If a DELETE payload has not been sent, both sides SHOULD maintain the state for their SAs for the standard lifetime or timeout period. The TCP Originator is responsible for re-establishing the TCP connection if it is torn down for any unexpected reason. Since new TCP connections may use different ports due to NAT mappings or local port allocations changing, the TCP Responder MUST allow packets for existing SAs to be received from new source ports.

A peer MUST discard a partially received message due to a broken connection.

Whenever the TCP Originator opens a new TCP connection to be used for an existing IKE SA, it MUST send the stream prefix first, before any IKE or ESP messages. This follows the same behavior as the initial TCP connection.

If a TCP connection is being used to resume a previous IKE session, the TCP Responder can recognize the session using either the IKE SPI from an encapsulated IKE message or the ESP SPI from an encapsulated ESP message. If the session had been fully established previously, it is suggested that the TCP Originator send an UPDATE_SA_ADDRESSES message if MOBIKE is supported, or an informational message (a keep-alive) otherwise.

The TCP Responder MUST NOT accept any messages for the existing IKE session on a new incoming connection, unless that connection begins with the stream prefix. If either the TCP Originator or TCP Responder detects corruption on a connection that was started with a valid stream prefix, it SHOULD close the TCP connection. The connection can be determined to be corrupted if there are too many subsequent messages that cannot be parsed as valid IKE messages or ESP messages with known SPIs, or if the authentication check for an ESP message with a known SPI fails. Implementations SHOULD NOT tear down a connection if only a single ESP message has an unknown SPI, since the SPI databases may be momentarily out of sync. If there is instead a syntax issue within an IKE message, an implementation MUST send the INVALID_SYNTAX notify payload and tear down the IKE SA as usual, rather than tearing down the TCP connection directly.

A TCP Originator SHOULD only open one TCP connection per IKE SA, over which it sends all of the corresponding IKE and ESP messages. This helps ensure that any firewall or NAT mappings allocated for the TCP connection apply to all of the traffic associated with the IKE SA equally.

Similarly, a TCP Responder SHOULD at any given time send packets for an IKE SA and its Child SAs over only one TCP connection. It SHOULD

choose the TCP connection on which it last received a valid and decryptable IKE or ESP message. In order to be considered valid for choosing a TCP connection, an IKE message must be successfully decrypted and authenticated, not be a retransmission of a previously received message, and be within the expected window for IKE message IDs. Similarly, an ESP message must pass authentication checks and be decrypted, and must not be a replay of a previous message.

Since a connection may be broken and a new connection re-established by the TCP Originator without the TCP Responder being aware, a TCP Responder SHOULD accept receiving IKE and ESP messages on both old and new connections until the old connection is closed by the TCP Originator. A TCP Responder MAY close a TCP connection that it perceives as idle and extraneous (one previously used for IKE and ESP messages that has been replaced by a new connection).

Multiple IKE SAs MUST NOT share a single TCP connection, unless one is a rekey of an existing IKE SA, in which case there will temporarily be two IKE SAs on the same TCP connection.

7.2. Retransmissions

Section 2.1 of [RFC7296] describes how IKEv2 deals with unreliability of UDP protocol. In brief, the exchange Initiator is responsible for retransmissions and must retransmit requests message until response message is received. If no reply is received after several retransmissions, the SA is deleted. The Responder never retransmits but must resend the response message in case it receives retransmitted request.

When IKEv2 uses reliable transport protocol the retransmission rules change as follows:

- o the exchange Initiator SHOULD NOT retransmit request message; if no response is received within some reasonable period of time, the IKE SA is deleted

- o if TCP connection is broken and then restored while the exchange Initiator is waiting for the response, it MUST retransmit the request and continue to wait for the response

o the exchange Responder acts as described in Section 2.1 of [RFC7296], i.e. using TCP encapsulation doesn't change the its behavior

7.3. Cookies and Puzzles

IKEv2 provides a DoS attack protection mechanism called Cookie, which is described in Section 2.6 of [RFC7296]. [RFC8019] extends this mechanism for protection against DDoS attacks by means of Client Puzzles. Both mechanisms allow the Responder to keep no state until the Initiator proves its IP address is real (and solves puzzle in the latter case).

The connection-oriented nature of TCP and transport brings additional considerations for using these mechanisms. In general, Cookie

provides less value in case of TCP encapsulation, because when the Responder receives the IKE_SA_INIT request the TCP session has already been established, so the Initiator's IP address has been verified. Moreover, TCP Responder creates state as far as the SYN packer is received (unless SYN Cookies described in [RFC4987] are employed), that violates the stateless nature of IKEv2 Cookies. So, it makes little sense to send Cookie request in this situation, unless the Responder in concerned with the possibility of TCP Sequence Number attacks (see [RFC6528] for details). On the other hand, Puzzles still remain useful and their use requires using Cookies.

The following considerations are applicable for using Cookie and Puzzle mechanisms in case of TCP encapsulation:

o the exchange Responder SHOULD NOT request Cookie unless it has good reason to do it (like a concern of the possibility of TCP Sequence Number attacks or Puzzle request is sent in the same message)

o if the Responder chooses to send Cookie request (possibly along with Puzzle request), then the TCP connection that the IKE_SA_INIT request message was received over SHOULD be closed, so that the Responder remains stateless at least until the Cookie (or Puzzle Solution) is returned

* note, that if this TCP connection is closed, then the Responder MUST NOT include the Initiator's TCP port into the Cookie calculation (*), since the Cookie will be returned over a new TCP connection with a different port

o the exchange Initiator acts as described in Section 2.6 of [RFC7296] and Section 7 of [RFC8019], i.e. using TCP encapsulation doesn't change the Initiator's behavior

(*) Examples of Cookie calculation methods are given in Section 2.6 of [RFC7296] and in Section 7.1.1.3 of [RFC8019] and they don't include transport protocol ports. However these examples are given for illustrative purposes, since Cookie generation algorithm is a local matter and some implementations might include port numbers, that won't work with TCP encapsulation.

7.4. Error Handling in the IKE_SA_INIT

Section 2.21.1 of [RFC7296] describes how error notifications should be handled in the IKE_SA_INIT exchange. In particular, it is advised that the Initiator should not act immediately after receiving error notification and should instead wait some time for valid response,

since the IKE_SA_INIT messages are completely unauthenticated. This advice has little sense in case of TCP encapsulation. If the Initiator receives the response message over TCP, then either this message is genuine and was sent by the peer, or the TCP session was hijacked and the message is forged, but in this case no genuine messages from the Responder will be received.

So, in case of TCP encapsulation the Initiator SHOULD NOT wait for additional messages in case it receives error notification from the Responder in the IKE_SA_INIT exchange.

7.5. NAT Detection Payloads

When negotiating over UDP port 500, IKE_SA_INIT packets include NAT_DETECTION_SOURCE_IP and NAT_DETECTION_DESTINATION_IP payloads to determine if UDP encapsulation of IPsec packets should be used. These payloads contain SHA-1 digests of the SPIs, IP addresses, and ports as defined in [RFC7296]. IKE_SA_INIT packets sent on a TCP

connection SHOULD include these payloads with the same content as when sending over UDP and SHOULD use the applicable TCP ports when creating and checking the SHA-1 digests.

If a NAT is detected due to the SHA-1 digests not matching the expected values, no change should be made for encapsulation of subsequent IKE or ESP packets, since TCP encapsulation inherently supports NAT traversal. Implementations MAY use the information that a NAT is present to influence keep-alive timer values.

If a NAT is detected, implementations need to handle transport mode TCP and UDP packet checksum fixup as defined for UDP encapsulation in [RFC3948].

7.6. Considerations for Keep-Alives and Dead Peer Detection

Encapsulating IKE and IPsec inside of a TCP connection can impact the strategy that implementations use to detect peer liveness and to maintain middlebox port mappings. Peer liveness should be checked using IKE informational packets [RFC7296].

In general, TCP port mappings are maintained by NATs longer than UDP port mappings, so IPsec ESP NAT keep-alives [RFC3948] SHOULD NOT be sent when using TCP encapsulation. Any implementation using TCP encapsulation MUST silently drop incoming NAT keep-alive packets and not treat them as errors. NAT keep-alive packets over a TCP-encapsulated IPsec connection will be sent as an ESP message with a one-octet-long payload with the value 0xFF.

Note that, depending on the configuration of TCP and TLS on the connection, TCP keep-alives [RFC1122] and TLS keep-alives [RFC6520] may be used. These MUST NOT be used as indications of IKE peer liveness.

7.7. Implications of TCP Encapsulation on IPsec SA Processing

Using TCP encapsulation makes affects some aspects of IPsec SA processing.

1. Section 8.1 of [RFC4301] requires all tunnel mode IPsec SAs to be

able to copy the Don't Fragment (DF) bit from inner IP header to the outer (tunnel) one. With TCP encapsulation it's generally impossible, because TCP/IP stack manages DF bit in the outer IP header, and usually the stack ensures that the DF bit is set for TCP packets to avoid IP fragmentation.

2. The other feature that is degraded with TCP encapsulation is an ability to split traffic of different QoS classes into different IPsec SAs, created by a single IKE SA. In this case the Differentiated Services Code Point (DSCP) field is usually copied from the inner IP header to the outer (tunnel) one, ensuring that IPsec traffic of each SA receives the corresponding level of service. With TCP encapsulation all IPsec SAs created by a single IKE SA will share a single TCP connection and thus will receive the same level of service (see Section 10.3). If this functionality is needed, implementations should create several IKE SAs over TCP and assign a corresponding DSCP value to each of them.

8. Interaction with IKEv2 Extensions

8.1. MOBIKE Protocol

MOBIKE protocol, that allows IKEv2 SA to migrate between IP addresses, is defined in [RFC4555], and [RFC4621] further clarifies the details of the protocol. When an IKE session that has negotiated MOBIKE is transitioning between networks, the Initiator of the transition may switch between using TCP encapsulation, UDP encapsulation, or no encapsulation. Implementations that implement both MOBIKE and TCP encapsulation MUST support dynamically enabling and disabling TCP encapsulation as interfaces change.

When a MOBIKE-enabled Initiator changes networks, the INFORMATIONAL exchange with the UPDATE_SA_ADDRESSES notification SHOULD be initiated first over UDP before attempting over TCP. If there is a response to the request sent over UDP, then the ESP packets should be sent directly over IP or over UDP port 4500 (depending on if a NAT

was detected), regardless of if a connection on a previous network was using TCP encapsulation. If no response is received within some period of time after several retransmissions, then the Initiator changes transport for this very exchange from UDP to TCP and

continues retransmitting. New INFORMATIONAL exchange MUST NOT be started in this situation. If the Responder only responds to the request sent over TCP, then the ESP packets should be sent over the TCP connection, regardless of if a connection on a previous network did not use TCP encapsulation.

Since switching from UDP to TCP happens in the same INFORMATIONAL exchange the content of the NAT_DETECTION_SOURCE_IP notification will in most cases be incorrect (since UDP and TCP source ports will most probably be different), and the peer will falsely think that there is a NAT in between. This should not cause problems because in this case all traffic will be encapsulated in TCP anyway, and TCP encapsulation is the same with regardless of NAT presence.

MOBIKE protocol defined the NO_NATS_ALLOWED notification that can be used to detect the presence of NAT between peer and to refuse to communicate in this situation. In case of TCP the NO_NATS_ALLOWED notification SHOULD be ignored because TCP generally has no problems with NAT boxes.

Section 3.7 of [RFC4555] describes an additional optional step in the process of changing IP addresses called Return Routability Check. It is performed by the responder in order to be sure that the new initiator's address is in fact routable. In case of TCP encapsulation this check has little value, since TCP handshake proves routability of the TCP Originator's address. So, in case of TCP encapsulation the Return Routability Check SHOULD NOT be performed.

8.2. IKE Redirect

Redirect mechanism for IKEv2 is defined in [RFC5685]. This mechanism allows security gateways to redirect clients to another gateway either during IKE SA establishment or after it is set up. If a client is connecting to a security gateway using TCP transport and then is being redirected to another security gateway, then the client must disregard the current transport. In other words, the client MUST again try first UDP and then fall back to TCP while establishing a new IKE SA, regardless of the transport of the SA the redirect notification was received over (unless the client's configuration instructs it to instantly use TCP for the gateway it is redirected to).

8.3. IKEv2 Session Resumption

Session resumption for IKEv2 is defined in [RFC5723]. Once IKE SA is established the server creates a resumption ticket where information about this SA is stored, and transfers this ticket to the client. The ticket may be later used to resume the IKE SA if it is deleted. In the event of resumption the client presents the ticket in a new exchange, called IKE_SESSION_RESUME. For the new SA some parameters are taken from the ticket and some are re-negotiated (more details are given in Section 5 of [RFC5723]). If TCP encapsulation was used in an old SA, then the client SHOULD resume this SA using TCP, without first trying to connect over UDP.

8.4. IKEv2 Protocol Support for High Availability

[RFC6311] defines a support for High Availability in IKEv2. The core idea is that in case of cluster failover a new active node immediately initiates the special INFORMATION exchange containing the IKEV2_MESSAGE_ID_SYNC notification, which instructs the client to skip some number of Message IDs that might not be synchronized yet between nodes at the time of failover.

The problem is that TCP states are much harder to synchronize than IKE states - it requires access to TCP/IP stack internals, which is not always available for IKE/IPsec implementations. If a cluster implementation doesn't synchronize TCP states between nodes, then after failover event the new active node will not have any TCP connection with the client, so the node cannot initiate the INFORMATIONAL exchange as required by [RFC6311]. Since the cluster usually acts as TCP Responder, the new active node cannot re-establish TCP connection, since only the TCP Originator can do it. And for the client the situation of cluster failover may remain unknown for long time if it has no IKE or ESP traffic to send. Once the client sends any ESP or IKEv2 packet, the cluster node will reply with TCP RST and the client (as TCP Originator) will restore the TCP connection so that the node will be able to initiate the INFORMATIONAL exchange informing the client about the cluster failover.

This memo makes the following recommendation: if support for High Availability in IKEv2 is negotiated and TCP transport is used and a client is TCP Originator, then the client SHOULD periodically send IKEv2 messages (e.g. by initiating liveness check exchange) whenever there is no any IKEv2 or ESP traffic. This differs from the recommendations given in Section 2.4 of [RFC7296] in the following: the liveness check should be periodically performed even if the client has nothing to send over ESP. The frequency of sending such

messages should be high enough to allow quick detection and restoring of broken TCP connection.

8.5. IKEv2 Fragmentation

IKE message fragmentation [RFC7383] is not required when using TCP encapsulation, since a TCP stream already handles the fragmentation of its contents across packets. Since fragmentation is redundant in this case, implementations might choose to not negotiate IKE fragmentation. Even if fragmentation is negotiated, an implementation SHOULD NOT send fragments when going over a TCP connection, although it MUST support receiving fragments.

If an implementation supports both MOBIKE and IKE fragmentation, it SHOULD negotiate IKE fragmentation over a TCP-encapsulated session in case the session switches to UDP encapsulation on another network.

9. Middlebox Considerations

Many security networking devices, such as firewalls or intrusion prevention systems, network optimization/acceleration devices, and NAT devices, keep the state of sessions that traverse through them.

These devices commonly track the transport-layer and/or application-layer data to drop traffic that is anomalous or malicious in nature. While many of these devices will be more likely to pass TCP-encapsulated traffic as opposed to UDP-encapsulated traffic, some may still block or interfere with TCP-encapsulated IKE and IPsec traffic.

A network device that monitors the transport layer will track the state of TCP sessions, such as TCP sequence numbers. TCP encapsulation of IKE should therefore use standard TCP behaviors to avoid being dropped by middleboxes.

10. Performance Considerations

Several aspects of TCP encapsulation for IKE and IPsec packets may negatively impact the performance of connections within a tunnel-mode IPsec SA. Implementations should be aware of these performance impacts and take these into consideration when determining when to

use TCP encapsulation. Implementations SHOULD favor using direct ESP or UDP encapsulation over TCP encapsulation whenever possible.

10.1. TCP-in-TCP

If the outer connection between IKE peers is over TCP, inner TCP connections may suffer negative effects from using TCP within TCP. Running TCP within TCP is discouraged, since the TCP algorithms

generally assume that they are running over an unreliable datagram layer.

If the outer (tunnel) TCP connection experiences packet loss, this loss will be hidden from any inner TCP connections, since the outer connection will retransmit to account for the losses. Since the outer TCP connection will deliver the inner messages in order, any messages after a lost packet may have to wait until the loss is recovered. This means that loss on the outer connection will be interpreted only as delay by inner connections. The burstiness of inner traffic can increase, since a large number of inner packets may be delivered across the tunnel at once. The inner TCP connection may interpret a long period of delay as a transmission problem, triggering a retransmission timeout, which will cause spurious retransmissions. The sending rate of the inner connection may be unnecessarily reduced if the retransmissions are not detected as spurious in time.

The inner TCP connection's round-trip-time estimation will be affected by the burstiness of the outer TCP connection if there are long delays when packets are retransmitted by the outer TCP connection. This will make the congestion control loop of the inner TCP traffic less reactive, potentially permanently leading to a lower sending rate than the outer TCP would allow for.

TCP-in-TCP can also lead to increased buffering, or bufferbloat. This can occur when the window size of the outer TCP connection is reduced and becomes smaller than the window sizes of the inner TCP connections. This can lead to packets backing up in the outer TCP connection's send buffers. In order to limit this effect, the outer TCP connection should have limits on its send buffer size and on the rate at which it reduces its window size.

Note that any negative effects will be shared between all flows going through the outer TCP connection. This is of particular concern for any latency-sensitive or real-time applications using the tunnel. If such traffic is using a TCP-encapsulated IPsec connection, it is recommended that the number of inner connections sharing the tunnel be limited as much as possible.

10.2. Added Reliability for Unreliable Protocols

Since ESP is an unreliable protocol, transmitting ESP packets over a TCP connection will change the fundamental behavior of the packets. Some application-level protocols that prefer packet loss to delay (such as Voice over IP or other real-time protocols) may be negatively impacted if their packets are retransmitted by the TCP connection due to packet loss.

10.3. Quality-of-Service Markings

Quality-of-Service (QoS) markings, such as the Differentiated Services Code Point (DSCP) and Traffic Class, should be used with care on TCP connections used for encapsulation. Individual packets SHOULD NOT use different markings than the rest of the connection, since packets with different priorities may be routed differently and cause unnecessary delays in the connection.

10.4. Maximum Segment Size

A TCP connection used for IKE encapsulation SHOULD negotiate its MSS in order to avoid unnecessary fragmentation of packets.

10.5. Tunneling ECN in TCP

Since there is not a one-to-one relationship between outer IP packets and inner ESP/IP messages when using TCP encapsulation, the markings for Explicit Congestion Notification (ECN) [RFC3168] cannot be simply mapped. However, any ECN Congestion Experienced (CE) marking on inner headers should be preserved through the tunnel.

Implementations SHOULD follow the ECN compatibility mode for tunnel ingress as described in [RFC6040]. In compatibility mode, the outer tunnel TCP connection marks its packet headers as not ECN-capable. If upon egress, the arriving outer header is marked with CE, the

implementation will drop the inner packet, since there is not a distinct inner packet header onto which to translate the ECN markings.

11. Security Considerations

IKE Responders that support TCP encapsulation may become vulnerable to new Denial-of-Service (DoS) attacks that are specific to TCP, such as SYN-flooding attacks. TCP Responders should be aware of this additional attack surface.

TCP Responders should be careful to ensure that (1) the stream prefix "IKETCP" uniquely identifies incoming streams as streams that use the TCP encapsulation protocol and (2) they are not running any other protocols on the same listening port (to avoid potential conflicts).

Attackers may be able to disrupt the TCP connection by sending spurious TCP Reset packets. Therefore, implementations SHOULD make sure that IKE session state persists even if the underlying TCP connection is torn down.

If MOBIKE is being used, all of the security considerations outlined for MOBIKE apply [RFC4555].

Similarly to MOBIKE, TCP encapsulation requires a TCP Responder to handle changes to source address and port due to network or connection disruption. The successful delivery of valid IKE or ESP messages over a new TCP connection is used by the TCP Responder to determine where to send subsequent responses. If an attacker is able to send packets on a new TCP connection that pass the validation checks of the TCP Responder, it can influence which path future packets will take. For this reason, the validation of messages on the TCP Responder must include decryption, authentication, and replay checks.

Since TCP provides reliable, in-order delivery of ESP messages, the ESP anti-replay window size SHOULD be set to 1. See [RFC4303] for a complete description of the ESP anti-replay window. This increases the protection of implementations against replay attacks.

12. IANA Considerations

TCP port 4500 is already allocated to IPsec for NAT traversal. This port SHOULD be used for TCP-encapsulated IKE and ESP as described in this document.

This document updates the reference for TCP port 4500:

Keyword	Decimal	Description	Reference
ipsec-nat-t	4500/tcp	IPsec NAT-Traversal	[RFCXXXX]

Figure 4

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, DOI 10.17487/RFC3948, January 2005, <https://www.rfc-editor.org/info/rfc3948>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <https://www.rfc-editor.org/info/rfc4303>.

[RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <https://www.rfc-editor.org/info/rfc6040>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[RFC8019] Nir, Y. and V. Smyslov, "Protecting Internet Key Exchange Protocol Version 2 (IKEv2) Implementations from Distributed Denial-of-Service Attacks", RFC 8019, DOI 10.17487/RFC8019, November 2016, <https://www.rfc-editor.org/info/rfc8019>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

13.2. Informative References

[I-D.ietf-ipsecme-ike-tcp] Nir, Y., "A TCP transport for the Internet Key Exchange", draft-ietf-ipsecme-ike-tcp-01 (work in progress), December 2012.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC2817] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, DOI 10.17487/RFC2817, May 2000, <https://www.rfc-editor.org/info/rfc2817>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC4555] Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, DOI 10.17487/RFC4555, June 2006, <https://www.rfc-editor.org/info/rfc4555>.

[RFC4621] Kivinen, T. and H. Tschofenig, "Design of the IKEv2

Mobility and Multihoming (MOBIKE) Protocol", RFC 4621, DOI 10.17487/RFC4621, August 2006, <https://www.rfc-editor.org/info/rfc4621>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5685] Devarapalli, V. and K. Weniger, "Redirect Mechanism for the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5685, DOI 10.17487/RFC5685, November 2009, <https://www.rfc-editor.org/info/rfc5685>.

[RFC5723] Sheffer, Y. and H. Tschofenig, "Internet Key Exchange Protocol Version 2 (IKEv2) Session Resumption", RFC 5723, DOI 10.17487/RFC5723, January 2010, <https://www.rfc-editor.org/info/rfc5723>.

[RFC6311] Singh, R., Ed., Kalyani, G., Nir, Y., Sheffer, Y., and D. Zhang, "Protocol Support for High Availability of IKEv2/IPsec", RFC 6311, DOI 10.17487/RFC6311, July 2011, <https://www.rfc-editor.org/info/rfc6311>.

[RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <https://www.rfc-editor.org/info/rfc6520>.

[RFC6528] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, DOI 10.17487/RFC6528, February 2012, <https://www.rfc-editor.org/info/rfc6528>.

[RFC7383] Smyslov, V., "Internet Key Exchange Protocol Version 2 (IKEv2) Message Fragmentation", RFC 7383, DOI 10.17487/RFC7383, November 2014, <https://www.rfc-editor.org/info/rfc7383>.

[RFC8229] Pauly, T., Touati, S., and R. Mantha, "TCP Encapsulation of IKE and IPsec Packets", RFC 8229, DOI 10.17487/RFC8229, August 2017, <https://www.rfc-editor.org/info/rfc8229>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

Appendix A. Using TCP Encapsulation with TLS

This section provides recommendations on how to use TLS in addition to TCP encapsulation.

When using TCP encapsulation, implementations may choose to use TLS 1.2 [RFC5246] or TLS 1.3 [RFC8446] on the TCP connection to be able to traverse middleboxes, which may otherwise block the traffic.

If a web proxy is applied to the ports used for the TCP connection and TLS is being used, the TCP Originator can send an HTTP CONNECT message to establish an SA through the proxy [RFC2817].

The use of TLS should be configurable on the peers, and may be used as the default when using TCP encapsulation or may be used as a fallback when basic TCP encapsulation fails. The TCP Responder may expect to read encapsulated IKE and ESP packets directly from the TCP connection, or it may expect to read them from a stream of TLS data packets. The TCP Originator should be pre-configured to use TLS or not when communicating with a given port on the TCP Responder.

When new TCP connections are re-established due to a broken connection, TLS must be renegotiated. TLS session resumption is recommended to improve efficiency in this case.

The security of the IKE session is entirely derived from the IKE negotiation and key establishment and not from the TLS session (which in this context is only used for encapsulation purposes); therefore, when TLS is used on the TCP connection, both the TCP Originator and the TCP Responder SHOULD allow the NULL cipher to be selected for performance reasons. Note, that TLS 1.3 only supports AEAD algorithms and at the time of writing this document there was no recommended cipher suite for TLS 1.3 with the NULL cipher.

Implementations should be aware that the use of TLS introduces another layer of overhead requiring more bytes to transmit a given IKE and IPsec packet. For this reason, direct ESP, UDP encapsulation, or TCP encapsulation without TLS should be preferred in situations in which TLS is not required in order to traverse middleboxes.

Appendix B. Example Exchanges of TCP Encapsulation with TLS 1.2

B.1. Establishing an IKE Session

```
                Client                                    Server
              ----------                                ----------
     1)  -------------------- TCP Connection --------------------
         (IP_I:Port_I  -> IP_R:Port_R)
         TcpSyn                    ---------->
                                   <----------          TcpSyn,Ack
         TcpAck                    ---------->

     2)  --------------------- TLS Session ---------------------
         ClientHello               ---------->
                                                       ServerHello
                                                      Certificate*
                                                ServerKeyExchange*
                                   <----------     ServerHelloDone
         ClientKeyExchange
         CertificateVerify*
         [ChangeCipherSpec]
         Finished                  ---------->
                                                [ChangeCipherSpec]
                                   <----------            Finished

     3)  ---------------------- Stream Prefix --------------------
         "IKETCP"                  ---------->
     4)  ----------------------- IKE Session ---------------------
         Length + Non-ESP Marker   ---------->
         IKE_SA_INIT
         HDR, SAi1, KEi, Ni,
         [N(NAT_DETECTION_*_IP)]
                                   <------ Length + Non-ESP Marker
                                                       IKE_SA_INIT
                                               HDR, SAr1, KEr, Nr,
                                           [N(NAT_DETECTION_*_IP)]
         Length + Non-ESP Marker   ---------->
         first IKE_AUTH
         HDR, SK {IDi, [CERTREQ]
         CP(CFG_REQUEST), IDr,
         SAi2, TSi, TSr, ...}
                                   <------ Length + Non-ESP Marker
```

```
                                                first IKE_AUTH
                                   HDR, SK {IDr, [CERT], AUTH,
                                          EAP, SAr2, TSi, TSr}
     Length + Non-ESP Marker   ---------->
     IKE_AUTH + EAP
```



```
     repeat 1..N times
                               <------ Length + Non-ESP Marker
                                                IKE_AUTH + EAP
     Length + Non-ESP Marker   ---------->
     final IKE_AUTH
     HDR, SK {AUTH}
                               <------ Length + Non-ESP Marker
                                                final IKE_AUTH
                                 HDR, SK {AUTH, CP(CFG_REPLY),
                                            SA, TSi, TSr, ...}
     -------------- IKE and IPsec SAs Established ------------
     Length + ESP Frame        ---------->
```

Figure 5

1. The client establishes a TCP connection with the server on port 4500 or on an alternate pre-configured port that the server is listening on.

2. If configured to use TLS, the client initiates a TLS handshake. During the TLS handshake, the server SHOULD NOT request the client's certificate, since authentication is handled as part of IKE negotiation.

3. The client sends the stream prefix for TCP-encapsulated IKE (Section 5) traffic to signal the beginning of IKE negotiation.

4. The client and server establish an IKE connection. This example shows EAP-based authentication, although any authentication type may be used.

B.2. Deleting an IKE Session

```
                Client                                    Server
              ----------                                ----------
      1)  ----------------------- IKE Session ---------------------
          Length + Non-ESP Marker   ---------->
          INFORMATIONAL
          HDR, SK {[N,] [D,]
                 [CP,] ...}
                                    <------ Length + Non-ESP Marker
                                                     INFORMATIONAL
                                                HDR, SK {[N,] [D,]
                                                         [CP], ...}

      2)  ---------------------  TLS Session  ---------------------
          close_notify              ---------->
                                    <----------        close_notify
      3)  --------------------  TCP Connection  -------------------
          TcpFin                    ---------->
                                    <----------                 Ack
                                    <----------              TcpFin
          Ack                       ---------->
          --------------------  IKE SA Deleted  -------------------
```

Figure 6

1. The client and server exchange informational messages to notify IKE SA deletion.

2. The client and server negotiate TLS session deletion using TLS

CLOSE_NOTIFY.

3. The TCP connection is torn down.

The deletion of the IKE SA should lead to the disposal of the underlying TLS and TCP state.

B.3. Re-establishing an IKE Session


```
             Client                                   Server
           ----------                               ----------
  1)  --------------------  TCP Connection  -------------------
      (IP_I:Port_I  -> IP_R:Port_R)
      TcpSyn                     ---------->
                                 <----------          TcpSyn,Ack
      TcpAck                     ---------->
  2)  ---------------------  TLS Session  ---------------------
      ClientHello                ---------->
                                 <----------         ServerHello
                                              [ChangeCipherSpec]
                                                        Finished
      [ChangeCipherSpec]         ---------->
      Finished
  3)  ---------------------- Stream Prefix --------------------
      "IKETCP"                   ---------->
  4)  <---------------------> IKE/ESP Flow <------------------>
      Length + ESP Frame         ---------->
```

Figure 7

1. If a previous TCP connection was broken (for example, due to a TCP Reset), the client is responsible for re-initiating the TCP connection. The TCP Originator's address and port (IP_I and Port_I) may be different from the previous connection's address and port.

2. In the ClientHello TLS message, the client SHOULD send the session ID it received in the previous TLS handshake if available. It is up to the server to perform either an abbreviated handshake or a full handshake based on the session ID match.

3. After TCP and TLS are complete, the client sends the stream prefix for TCP-encapsulated IKE traffic (Section 5).

4. The IKE and ESP packet flow can resume. If MOBIKE is being used, the Initiator SHOULD send an UPDATE_SA_ADDRESSES message.

B.4. Using MOBIKE between UDP and TCP Encapsulation

```
                 Client                               Server
               ----------                           ----------
      (IP_I1:UDP500 -> IP_R:UDP500)
  1)  ----------------- IKE_SA_INIT Exchange -----------------
      (IP_I1:UDP4500 -> IP_R:UDP4500)
      Non-ESP Marker           ----------->
      Initial IKE_AUTH
```



```
      HDR, SK { IDi, CERT, AUTH,
      CP(CFG_REQUEST),
      SAi2, TSi, TSr,
      N(MOBIKE_SUPPORTED) }
                               <-----------      Non-ESP Marker
                                               Initial IKE_AUTH
                                          HDR, SK { IDr, CERT, AUTH,
                                                EAP, SAr2, TSi, TSr,
                                               N(MOBIKE_SUPPORTED) }
      <------------------ IKE SA Establishment --------------->

  2)  ------------ MOBIKE Attempt on New Network --------------
      (IP_I2:UDP4500 -> IP_R:UDP4500)
      Non-ESP Marker           ----------->
```

```
    INFORMATIONAL
    HDR, SK { N(UPDATE_SA_ADDRESSES),
    N(NAT_DETECTION_SOURCE_IP),
    N(NAT_DETECTION_DESTINATION_IP) }

3)  --------------------  TCP Connection  -------------------
    (IP_I2:Port_I -> IP_R:Port_R)
    TcpSyn                    ----------->
                              <-----------         TcpSyn,Ack
    TcpAck                    ----------->

4)  ---------------------  TLS Session  ---------------------
    ClientHello               ----------->
                                                  ServerHello
                                                 Certificate*
                                           ServerKeyExchange*
                              <-----------     ServerHelloDone
    ClientKeyExchange
    CertificateVerify*
    [ChangeCipherSpec]
    Finished                  ----------->
                                           [ChangeCipherSpec]
                              <-----------           Finished

5)  ---------------------- Stream Prefix --------------------
    "IKETCP"                   ---------->

6)  ----------------------- IKE Session ---------------------
    Length + Non-ESP Marker  ----------->
    INFORMATIONAL (Same as step 2)
    HDR, SK { N(UPDATE_SA_ADDRESSES),
    N(NAT_DETECTION_SOURCE_IP),
    N(NAT_DETECTION_DESTINATION_IP) }
```



```
                              <------- Length + Non-ESP Marker
                         HDR, SK { N(NAT_DETECTION_SOURCE_IP),
                                 N(NAT_DETECTION_DESTINATION_IP) }
7)  <----------------- IKE/ESP Data Flow ------------------->
```

Figure 8

1. During the IKE_SA_INIT exchange, the client and server exchange MOBIKE_SUPPORTED notify payloads to indicate support for MOBIKE.

2. The client changes its point of attachment to the network and receives a new IP address. The client attempts to re-establish the IKE session using the UPDATE_SA_ADDRESSES notify payload, but the server does not respond because the network blocks UDP traffic.

3. The client brings up a TCP connection to the server in order to use TCP encapsulation.

4. The client initiates a TLS handshake with the server.

5. The client sends the stream prefix for TCP-encapsulated IKE traffic (Section 5).

6. The client sends the UPDATE_SA_ADDRESSES notify payload on the TCP-encapsulated connection. Note that this IKE message is the same as the one sent over UDP in step 2; it should have the same message ID and contents.

7. The IKE and ESP packet flow can resume.

Acknowledgments

The following people provided valuable feedback and advices while preparing RFC8229: Stuart Cheshire, Delziel Fernandes, Yoav Nir, Christoph Paasch, Yaron Sheffer, David Schinazi, Graham Bartlett, Byju Pularikkal, March Wu, Kingwel Xie, Valery Smyslov, Jun Hu, and Tero Kivinen. Special thanks to Eric Kinnear for his implementation work.

Author would like to thank Tommy Pauly and Tero Kivinen for their valuable comments while preparing this document.

Author's Address

Valery Smyslov
ELVIS-PLUS
PO Box 81
Moscow (Zelenograd) 124460
RU

Phone: +7 495 276 0211
Email: svan@elvis.ru