LWIP WG 2011-03-28

Minimal IKEv2

Tero Kivinen <kivinen@iki.fi>
AuthenTec

draft-kivinen-ipsecme-ikev2-minimal-00.txt

Example Use Case

- Garage door opener
  - Two buttons:
    - one to unlock and open door
    - another to close and lock the door
  - One led for feedback
  - Uses two-way radio communications
  - Obviously needs some kind of security
  - Battery powered

Example protocol

- Protocol can be very simple:
  - Send packet to server to start open/close door
  - Get packet back to acknowledge the command
  - Get status messages every second while door is moving
  - Get final message when operation is done

Protocol effects

- Device only wakes up when button is pressed
  - It always initiates the communication, it does not need to listen radio when it is sleeping, and it cannot reply to any messages while sleeping
- Device stays awake for some time after the button is pressed and if receives status packet blinks led and waits for more status packets.
- After certain timeout device goes back to sleep

What this means for IKEv2

- Device only needs work as IKEv2 Initiator
  - No need to work as IKEv2 Responder
- Only creates one IKEv2 SA and one IPsec SA
  - No need to support SA management operations like creating new IPsec SAs, rekeying, deleting SAs, etc.
- No need to do NAT-T, Configuration payloads, EAP authentication, Cookies, Multiple child SAs etc
- The server end would most likely be some kind of Home area network server (PC or similar).
- Pre-shared keys or RAW RSA keys authentication
  - No X.509 certificates

Authentication

- Pre-shared keys
  - Shared key printed on paper or in electronic form
  - Typed in to the home area gateway
- Raw RSA keys
  - Fingerprint of device is distributed as Pre-shared keys
  - Device imprints to first home area gateway it connects to
  - Some form of reset can be implemented to allow reimprinting

Implementation

- I created a prototype implementation of the minimal IKEv2 protocol usable for such scenarios and it took me less than a day to write the code and less than 1000 lines of perl source code.
  - I implemented sending ICMP Ping packet as didn't want to start writing server end to answer my requests...
- Implementing minimal IKEv2 is very simple compared to full implementation.
- There are some optimizations which can be done when only supporting minimal set of features.

Examples of Optimizations

- Message ID and Window code
  - In IKEv2 there is requirement to keep track of the Message IDs received and transmitted to protect replays
  - Minimal implementation
    - Sends only IKE_SA_INIT and IKE_AUTH
      - No need to keep track of transmitted Message IDs
    - Does not do anything useful based on received messages (only sends empty acknowledgement or error)
      - No need to keep track of received Message IDs

Running code 1/2

```perl
#!/usr/local/bin/perl
# -*- perl -*-
########################################################################
# mini-ike.pl -- Minimal IKIv2 initiator
# Copyright (c) 2010 Tero Kivinen
# All Rights Reserved.
########################################################################
#         Program: mini-ike.pl
#         $Source: /u/kivinen/RCS/ikeparser.pl,v $
#         Author : $Author: kivinen $
#
#         (C) Tero Kivinen 2010 <kivinen@iki.fi>
#
#         Creation          : 23:53 Nov  9 2010 kivinen
#         Last Modification : 15:40 Nov 23 2010 kivinen
#         Last check in     : $Date: 2002/01/30 22:23:03 $
#         Revision number   : $Revision: 1.9 $
#         State             : $State: Exp $
#         Version           : 1.785
#         Edit time         : 190 min
#
#         Description       : Minimal IKEv2 initiator
#
#         $Log: ikeparser.pl,v $
#         $EndLog$
#
#
########################################################################
# initialization

require 5.6.0;
package IkeV2;
use strict;
use Getopt::Long;
use Crypt::Rijndael;
use Digest::HMAC_SHA1 qw(hmac_sha1);
use Crypt::DH;
use Crypt::Random qw(makerandom_octet);
use Math::BigInt;
use Socket;

########################################################################
# Configure

my($ikev2, $ipsec);

$$ikev2{srchost} = "0.0.0.0";
$$ikev2{dsthost} = "172.30.4.59";
# $$ikev2{dsthost} = "127.0.0.1";
$$ikev2{srcport} = 5500;

# $$ikev2{dsthost} = "172.30.4.74";
# $$ikev2{dstport} = 5501;

# $$ikev2{dsthost} = "172.30.4.69";
# $$ikev2{id} = 'k@77.fi';

$$ikev2{srchost} = "172.30.4.74";
# $$ikev2{dsthost} = "172.30.4.189";
$$ikev2{dstport} = 500;
# $$ikev2{cipher} = 'null';
$$ikev2{cipher} = 'aes';
$$ikev2{id} = 'tk@iki.fi';
$$ikev2{sharedsecret} = "foo";

########################################################################
# Negotiate IKE

$ipsec = ikev2($ikev2);
print_hash($ipsec);

########################################################################
# Send ICMP echo request

my($packet, $pad, $i, $iv, $cipher, $addr);

# ICMP echo request
$packet = pack("CCnnna*", 8, 0, 0, 0x1234, 1, "test" . ("1234567890" x
6));
substr($packet, 2, 2) = pack("n", checksum($packet, 0, 20));

# Tunnel mode header
$packet = pack("CCnnnCCna4a4a*", 0x45, 0, 20 + length($packet),
               1, 0, 255, 1, 0,
               inet_aton($$ikev2{srchost}), inet_aton($$ikev2{dsthost}),
               $packet);
substr($packet, 10, 2) = pack("n", checksum(substr($packet, 0, 20)));

# Padding
if ((length($packet) + 2) % 16 != 0) {
    $pad = (16 - ((length($packet) + 2) % 16));
    for($i = 0; $i < $pad; $i++) {
        $packet .= chr($i);
    }
} else {
    $pad = 0;
}
$packet .= chr($pad - 1) . chr(4);

hexl_print("Packet before encryption", $packet);

if ($$ikev2{cipher} ne 'null') {
    $cipher = new Crypt::Rijndael($$ipsec{cipher_key_out},
                                  Crypt::Rijndael::MODE_CBC);
    $iv = generate_random($ikev2, 16);
    $cipher->set_iv($iv);
    $packet = $cipher->encrypt($packet);
} else {
    $iv = '';
}

$packet = pack("a4Na*a*", $$ipsec{spi_out}, 0, $iv, $packet);
$packet .= substr(hmac_sha1($packet, $$ipsec{auth_key_out}), 0, 12);
hexl_print("Sending esp packet", $packet);

$addr = sockaddr_in(0, inet_aton($$ikev2{dsthost}));
socket(RAW, PF_INET, SOCK_RAW, 50) || die "socket: $!";
send(RAW, $packet, 0, $addr)                  || die "Send: $!";
```

Init/Configuration

ICMP

```perl
# Receive response
my($rin, $rout, $nfound);

$rin = '';
vec($rin, fileno(RAW), 1) = 1;
$nfound = select($rout = $rin, undef, undef, 10);
if ($nfound != 0) {
    my($spi, $seq, $iv, $mac, $cipher);
    my($proto, $pad, $type, $code, $checksum, $id, $data);

    $addr = recv(RAW, $packet, 1280, 0);
    die "Recv failed: $!"if (!defined($addr));
    hexl_print("Received esp packet", $packet);

    # Remove IP header
    $packet = substr($packet, 20);

    # Check mac
    $mac = substr(hmac_sha1(substr($packet, 0, -12),
                            $$ipsec{auth_key_in}), 0, 12);
    die "MAC check failed" if ($mac ne substr($packet, -12));

    ($spi, $seq, $iv, $packet) = unpack("NNa16a*", $packet);
    die "Invalid SPI" if ($spi ne $$ipsec{spi_in});


    $packet = substr($packet, 0, -12);
    $cipher = new Crypt::Rijndael($$ipsec{cipher_key_in},
                                  Crypt::Rijndael::MODE_CBC);
    $cipher->set_iv($iv);
    $packet = $cipher->decrypt($packet);

    hexl_print("Decrypted esp packet", $packet);

    $proto = ord(substr($packet, -1, 1));
    $pad = ord(substr($packet, -2, 1));
    $packet = substr($packet, 0, -($pad + 2));
    die "No IP in IP" if ($proto != 4);

    # Remove IP header
    $proto = ord(substr($packet, 9, 1));
    $packet = substr($packet, 20);
    die "Not ICMP" if ($proto != 1);
    die "Invalid icmmp  checksum" if (checksum($packet) != 0);
    ($type, $code, $checksum, $id, $seq, $data) = unpack("CCnnna*",
$packet);
    die "Not ICMP echo reply" if ($type != 0);
    hexl_print("Received ICMP echo packet, type = $type, code =
$code, " .
               "chk = $checksum, id = $id, seq = $seq", $data);
} else {
    die "Receive timeout";
}
exit(0);

########################################################################
#
# %$ipsec_params = ikev2(%$ikev2)

sub ikev2 {
    my($ikev2) = @_;

    $$ikev2{ike_sa_init_i} = generate_ike_sa_init($ikev2);
    $$ikev2{ike_sa_init_r} = do_exchange($ikev2, $
$ikev2{ike_sa_init_i});
    parse_ike_sa_init($ikev2, $$ikev2{ike_sa_init_r});

    calculate_keys($ikev2);

    $$ikev2{ike_auth_i} = generate_ike_auth($ikev2);
    $$ikev2{ike_auth_r} = do_exchange($ikev2, $$ikev2{ike_auth_i});
    parse_ike_auth($ikev2, $$ikev2{ike_auth_r});

    return ipsec_params($ikev2);
}

########################################################################
#
# Generate generic header
# $packet = generate_hdr($ikev2, $next_payload, $exchange_type,
#                        $flags, $message_id, $rest_of_packet);

sub generate_hdr {
    my($ikev2, $next_payload, $exchange_type, $flags,
       $message_id, $rest_of_packet) = @_;

    return pack("a8a8CCCCNNa*",
                $$ikev2{spi_i}, $$ikev2{spi_r},
                $next_payload, 0x20, $exchange_type, $flags,
$message_id,
                28 + length($rest_of_packet), $rest_of_packet);
}

########################################################################
#
# Generate generic header
# $packet = generate_gen_hdr($ikev2, $next_payload, $payload_data);

sub generate_gen_hdr {
    my($ikev2, $next_payload, $payload_data) = @_;

    return pack("Cxna*", $next_payload, 4 + length($payload_data),
                $payload_data);
}

########################################################################
#
# Generate sa payload (without generic header)
# $payload = generate_ike_sa_payload($ikev2);

sub generate_ike_sa_payload {
    my($ikev2) = @_;
    my($transforms);

    # ENCR: AES-CBC 128 bit
    $transforms = generate_gen_hdr($ikev2, 3,
                                   pack("Cxnnn",
                                        1, 12, 0x8000 | 14, 128));
    # PRF: PRF_HMAC_SHA1
    $transforms .= generate_gen_hdr($ikev2, 3, pack("Cxn", 2, 2));
    # Auth: AUTH_HMAC_SHA1_96
    $transforms .= generate_gen_hdr($ikev2, 3, pack("Cxn", 3, 2));
    # Diffie-Hellman: 1024-bit MODP
    $transforms .= generate_gen_hdr($ikev2, 0, pack("Cxn", 4, 2));
    # 1st proposal, protocol IKE = 1, no SPI, 4 transforms
    return generate_gen_hdr($ikev2, 0,
                            pack("CCCCa*",
                                 1, 1, 0, 4, $transforms));
}
```

ICMP send/response

IKE_INIT_SA/ IKE_AUTH

Headers

Payload generation

```perl
########################################################################
# Generate ke payload (without generic header)
# $payload = generate_ke_payload($ikev2);

sub generate_ke_payload {
    my($ikev2) = @_;
    my($dh, $pub);

    $dh = Crypt::DH->new(p => Math::BigInt-
>new("0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA
63B139B22514A08798E3404DDEF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245E485B5
76625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7EDEE386BFB5A899FA5AE9F24117C4B1FE649286
651ECE65381FFFFFFFFFFFFFFFF"),
                         g => Math::BigInt->new("2"));
    $dh->generate_keys;
    $pub = $dh->pub_key()->as_hex;
    $pub =~ s/^0x//g;
    $$ikev2{dh} = $dh;
    if (length($pub) != 256) {
        $pub = "0" x (256 - length($pub)) . $pub;
    }
    return pack("nxxH*", 2, $pub);
}

########################################################################
# Generate nonce payload (without generic header)
# $payload = generate_nonce_payload($ikev2);

sub generate_nonce_payload {
    my($ikev2) = @_;
    $$ikev2{nonce_i} = generate_random($ikev2, 16);
    return $$ikev2{nonce_i};
}

########################################################################
# Generate id payload (without generic header)
# $payload = generate_id_payload($ikev2);

sub generate_id_payload {
    my($ikev2) = @_;
    return pack("Cxxxa*", 3, $$ikev2{id});
}

########################################################################
# Generate auth payload (without generic header)
# $payload = generate_auth_payload($ikev2);

sub generate_auth_payload {
    my($ikev2) = @_;
    my($auth_data, $signed);

    $signed = $$ikev2{ike_sa_init_i} . $$ikev2{p_ike_sa}{40}{nonce} .
        hmac_sha1(generate_id_payload($ikev2), $$ikev2{sk_pi});
    printf(STDERR "Signed data:\n%s", bin_to_hexl($signed));
    $auth_data = hmac_sha1($signed, hmac_sha1("Key Pad for IKEv2",
                                              $$ikev2{sharedsecret}));
    printf(STDERR "Auth data:\n%s", bin_to_hexl($auth_data));
    return pack("Cxxxa*", 2, $auth_data);
}

########################################################################
# Generate Child sa payload (without generic header)
# $payload = generate_ipsec_sa_payload($ikev2);

sub generate_ipsec_sa_payload {
    my($ikev2) = @_;
    my($transforms);

    if ($$ikev2{cipher} eq 'null') {
        # ENCR: null
        $transforms = generate_gen_hdr($ikev2, 3, pack("Cxn", 1, 11));
    } else {
        # ENCR: AES-CBC 128 bit
        $transforms = generate_gen_hdr($ikev2, 3,
                                       pack("Cxnnn",
                                            1, 12, 0x8000 | 14, 128));
    }
    # Auth: AUTH_HMAC_SHA1_96
    $transforms .= generate_gen_hdr($ikev2, 3, pack("Cxn", 3, 2));
    # ESN - no ESN
    $transforms .= generate_gen_hdr($ikev2, 0, pack("Cxn", 5, 0));
    # 1st proposal, protocol ESP = 3, 4 byte SPI, 3 transforms
    return generate_gen_hdr($ikev2, 0,
                            pack("CCCCNa*",
                                 1, 3, 4, 3, 0x12345678, $transforms));
}

########################################################################
# Generate tsi payload (without generic header)
# $payload = generate_tsi_payload($ikev2);

sub generate_tsi_payload {
    my($ikev2) = @_;
    return pack("CxxxCCnnnNN", 1, 7, 0, 16, 0, 65535, 0, 0xffffffff);
}

########################################################################
# Generate tsr payload (without generic header)
# $payload = generate_tsi_payload($ikev2);

sub generate_tsr_payload {
    my($ikev2) = @_;
    return pack("CxxxCCnnnNN", 1, 7, 0, 16, 0, 65535, 0, 0xffffffff);
}

########################################################################
# Generate notify payload (without generic header)
# $payload = generate_notify_payload($ikev2);

sub generate_notify_payload {
    my($ikev2) = @_;
    return pack("CCn", 0, 0, 16384);
}
```

Payloads

```perl
########################################################################
# Generate encrypted payload (without generic header)
# $payload = generate_encr_payload($ikev2);
# The ICV is added here, but it is replaced with correct value in
calculate_icv

sub generate_encr_payload {
    my($ikev2, $data) = @_;
    my($iv, $pad, $cipher);

    $iv = generate_random($ikev2, 16);

    if (length($data) % 16 != 15) {
        $pad = (15 - (length($data) % 16));
        $data .= "\0" x $pad;
    } else {
        $pad = 0;
    }
    $data .= chr($pad);

    $cipher = new Crypt::Rijndael($$ikev2{sk_ei},
Crypt::Rijndael::MODE_CBC);
    $cipher->set_iv($iv);
    $data = $cipher->encrypt($data);
    return $iv . $data . ("\0" x 12);
}

########################################################################
# Calculate ICV
# $packet = calculate_icv($ikev2, $packet);

sub calculate_icv {
    my($ikev2, $packet) = @_;

    $packet = substr($packet, 0, -12);
    $packet .= substr(hmac_sha1($packet, $$ikev2{sk_ai}), 0, 12);
    return $packet;
}

########################################################################
# Generate IKE_SA_INIT packet
# $request_packet = generate_ike_sa_init($ikev2);

sub generate_ike_sa_init {
    my($ikev2) = @_;
    my($packet);

    $$ikev2{spi_i} = generate_random($ikev2, 8);
    $$ikev2{spi_r} = "\0" x 8;

    $packet = generate_gen_hdr($ikev2, 34,
                               generate_ike_sa_payload($ikev2));
    $packet .= generate_gen_hdr($ikev2, 40,
                               generate_ke_payload($ikev2));
    $packet .= generate_gen_hdr($ikev2, 0,
                               generate_nonce_payload($ikev2));
    return generate_hdr($ikev2, 33, 34, 8, 0, $packet);
}

########################################################################
# Parse SA payload
# parse_sa($ikev2, $payload_hash, $payload_str);

sub parse_sa {
    my($ikev2, $hash, $payload) = @_;
    my($len, $next);

    ($next, $len, $$hash{prop_num},
     $$hash{proto_id}, $$hash{spi_size}, $$hash{num_trans}) =
         unpack("CxnCCCC", $payload);
    die "Proposal len" if ($len < 8 + $$hash{spi_size});
    die "Proposal num" if ($$hash{prop_num} != 1);
    die "Proposal next" if ($next != 0);
    $$hash{spi} = substr($payload, 8, $$hash{spi_size})
        if ($$hash{spi_size} != 0);
    $payload = substr($payload, 8 + $$hash{spi_size});

    while (length($payload) > 4) {
        my(%transform, $attr, $type);
        ($next, $len, $type,
         $transform{id}) = unpack("CxnCxn", $payload);
        die "Transform len" if ($len > length($payload));
        die "Transform type dup" if (defined($$hash{$type}));
        $attr = substr($payload, 8, $len - 8);
        if (length($attr) == 4) {
            ($transform{a_type}, $transform{a_value}) =
                unpack("nn", $attr);
            $transform{a_type} = $transform{a_type} & 0x7fff;
        } elsif (length($attr) != 0) {
            die "Invalid transform attributes";
        }
        $$hash{$type} = \%transform;
        $payload = substr($payload, $len);
        return if (length($payload) == 0 && $next == 0);
        die "Transform next" if ($next != 3);
    }
    die "Invalid transform struct";
}

########################################################################
# Parse KE payload
# parse_ke($ikev2, $payload_hash, $payload_str);

sub parse_ke {
    my($ikev2, $hash, $payload) = @_;
    ($$hash{dh_group}, $$hash{dh_value}) =
        unpack("nxxa*", $payload);
}

########################################################################
# Parse id payload
# parse_id($ikev2, $payload_hash, $payload_str);

sub parse_id {
    my($ikev2, $hash, $payload) = @_;
    ($$hash{type}, $$hash{data}) = unpack("Cxxxa*", $payload);
    printf(STDERR "data for mac:\n%s", bin_to_hexl($payload));
    $$ikev2{maced_id_for_r} = hmac_sha1($payload, $$ikev2{sk_pr});
}

########################################################################
# Parse auth payload
# parse_auth($ikev2, $payload_hash, $payload_str);

sub parse_auth {
    my($ikev2, $hash, $payload) = @_;
    ($$hash{type}, $$hash{data}) = unpack("Cxxxa*", $payload);
    die "Auth type not preshared keys" if ($$hash{type} != 2);
}
```

IKE_SA_INIT packet

Payload parsing

Running code 2/2

Payload parsing

```
########################################################################
# Check auth payload
# $ok = check_auth($ikev2);

sub check_auth {
    my($ikev2) = @_;
    my($signed, $auth_data);

    printf(STDERR "nonce_i:\n%s", bin_to_hexl($$ikev2{nonce_i}));
    printf(STDERR "maced_id_for_r:\n%s", bin_to_hexl($$ikev2{maced_id_for_r}));
    $signed = $$ikev2{ike_sa_init_r} . $$ikev2{nonce_i} . $$ikev2{maced_id_for_r};
    printf(STDERR "Signed data:\n%s", bin_to_hexl($signed));
    $auth_data = hmac_sha1($signed, hmac_sha1("Key Pad for IKEv2",
                                              $$ikev2{sharedsecret}));
    printf(STDERR "Auth data:\n%s", bin_to_hexl($auth_data));
    die "Auth data does not match" if ($auth_data ne $$ikev2{p_ike_auth}{39}
{data});
    return 1;
}

########################################################################
# Parse nonce payload
# parse_nonce($ikev2, $payload_hash, $payload_str);

sub parse_nonce {
    my($ikev2, $hash, $payload) = @_;
    $$hash{nonce} = $payload;
}

########################################################################
# Parse notify payload
# parse_notify($ikev2, $payload_hash, $payload_str);

sub parse_notify {
    my($ikev2, $hash, $payload) = @_;
    ($$hash{proto_id}, $$hash{spi_size}, $$hash{type}) =
        unpack("CCn", $payload);
    $$hash{spi} = substr($payload, 4, $$hash{spi_size})
        if ($$hash{spi_size} != 0);
    $$hash{data} = substr($payload, 4 + $$hash{spi_size})
        if (length($payload) > 4 + $$hash{spi_size});
}

########################################################################
# Parse ts payload
# parse_ts($ikev2, $payload_hash, $payload_str);

sub parse_ts {
    my($ikev2, $hash, $payload) = @_;
    my($rest, $i, $len);

    ($$hash{num}, $rest) = unpack("Cxxxa*", $payload);
    for($i = 0; $i < $$hash{num}; $i++) {
        my(%ts);
        ($ts{type}, $ts{protocol}, $len, $ts{start_port}, $ts{end_port}, $rest) =
            unpack("CCnnna*", $rest);
        die "Invalid len" if (length($rest) < $len - 8);
        if ($ts{type} == 7 && $len == 16) {
            ($ts{ip_start}, $ts{ip_end}, $rest) = unpack("a4a4a*", $rest);
        } elsif ($ts{type} == 8 && $len == 40) {
            ($ts{ip_start}, $ts{ip_end}, $rest) = unpack("a16a16a*", $rest);
        } else {
            die "Invalid traffic selector type, or len";
        }
        $$hash{$i} = \%ts;
    }
    die "Junk after traffic selectors" if ($rest ne "");
    return
}
```

Parse packet

```
########################################################################
# Parse packet
# $packet_hash = parse_packet($ikev2, $packet);

sub parse_packet {
    my($ikev2, $packet) = @_;
    my(%parsed_packet, $type, $rest, $next, $len, $crit);

    ($parsed_packet{hdr_spi_i}, $parsed_packet{hdr_spi_r},
     $type, $parsed_packet{hdr_version}, $parsed_packet{hdr_exchange_type},
     $parsed_packet{hdr_flags}, $parsed_packet{hdr_message_id},
     $len, $rest) =
         unpack("a8a8CCCCNNa*", $packet);

    die "Message length" if ($len > length($packet));

    if ($type == 46) {
        my($iv, $mac, $cipher, $pad);

        $mac = substr(hmac_sha1(substr($packet, 0, -12),
                                $$ikev2{sk_ar}), 0, 12);
        ($type, $crit, $len, $iv, $rest) = unpack("CCna16a*", $rest);

        die "Payload length" if ($len - 4 - 16 > length($rest));
        die "MAC check failed" if ($mac ne substr($rest, -12));

        $rest = substr($rest, 0, -12);
        $cipher = new Crypt::Rijndael($$ikev2{sk_er}, Crypt::Rijndael::MODE_CBC);
        $cipher->set_iv($iv);
        $rest = $cipher->decrypt($rest);
        $pad = ord(substr($rest, -1, 1));
        $rest = substr($rest, 0, -($pad + 1));
        die "Encrypted payload is critical" if ($crit & 0x80);
    }

    while ($type != 0) {
        my(%payload, $data);
        ($next, $crit, $len) = unpack("CCn", $rest);
        die "Payload length" if ($len > length($rest));
        $data = substr($rest, 4, $len - 4);

        if ($type == 33) { parse_sa($ikev2, \%payload, $data);
        } elsif ($type == 34) { parse_ke($ikev2, \%payload, $data);
        } elsif ($type == 35) { parse_id($ikev2, \%payload, $data);
        } elsif ($type == 36) { parse_id($ikev2, \%payload, $data);
        } elsif ($type == 39) { parse_auth($ikev2, \%payload, $data);
        } elsif ($type == 40) { parse_nonce($ikev2, \%payload, $data);
#       } elsif ($type == 41) { parse_notify($ikev2, \%payload, $data);
        } elsif ($type == 44) { parse_ts($ikev2, \%payload, $data);
        } elsif ($type == 45) { parse_ts($ikev2, \%payload, $data);
        } elsif ($crit & 0x80) {
            die "Unknown critical payload: $type";
        }

        if (%payload) {
            if (!defined($parsed_packet{$type})) {
                $parsed_packet{$type} = \%payload;
            } else {
                die "Duplicate payload type $type";
            }
        }

        $type = $next;
        $rest = substr($rest, $len);
    }
    print_hash(\%parsed_packet);
    return \%parsed_packet;
}
```

Parse IKE_SA_INIT

```
########################################################################
# Parse IKE_SA_INIT packet
# parse_ike_sa_init($ikev2, $response_packet);

sub parse_ike_sa_init {
    my($ikev2, $packet) = @_;
    my($p);

    $p = parse_packet($ikev2, $packet);
    die "Wrong spi" if ($$p{hdr_spi_i} != $$ikev2{spi_i});
    die "Invalid major version" if (int($$p{hdr_version} / 16) != 2);
    die "Invalid exchange" if ($$p{hdr_exchange_type} != 34);
    die "Invalid flags" if ($$p{hdr_flags} != 0x20);
    die "Invalid message id" if ($$p{hdr_message_id} != 0);
    die "Missing sa" if (!defined($$p{33}));
    die "Missing ke" if (!defined($$p{34}));
    die "Missing nonce" if (!defined($$p{40}));
    die "Wrong num trans" if ($$p{33}{num_trans} != 4);
    die "Invalid proto id" if ($$p{33}{proto_id} != 1);
    die "Invalid sa payload" if ($$p{33}{spi_size} != 0);
    die "Invalid proposal 1" if ($$p{33}{1}{a_type} != 14 ||
                                 $$p{33}{1}{a_value} != 128 ||
                                 $$p{33}{1}{id} != 12);
    die "Invalid proposal 2" if (defined($$p{33}{2}{a_type}) ||
                                 $$p{33}{2}{id} != 2);
    die "Invalid proposal 3" if (defined($$p{33}{3}{a_type}) ||
                                 $$p{33}{3}{id} != 2);
    die "Invalid proposal 4" if (defined($$p{33}{4}{a_type}) ||
                                 $$p{33}{4}{id} != 2);
    die "Invalid ke" if ($$p{34}{dh_group} != 2);
    $$ikev2{spi_r} = $$p{hdr_spi_r};
    $$ikev2{p_ike_sa} = $p;
}
```

Generate IKE_AUTH

```
########################################################################
# Generate IKE_AUTH packet
# $request_packet = generate_ike_auth($ikev2);

sub generate_ike_auth {
    my($ikev2) = @_;
    my($packet, $encr_packet, $final_packet);

    $packet = generate_gen_hdr($ikev2, 39,
                               generate_id_payload($ikev2));
    $packet .= generate_gen_hdr($ikev2, 33,
                               generate_auth_payload($ikev2));
    $packet .= generate_gen_hdr($ikev2, 44,
                               generate_ipsec_sa_payload($ikev2));
    $packet .= generate_gen_hdr($ikev2, 45,
                               generate_tsi_payload($ikev2));
    $packet .= generate_gen_hdr($ikev2, 41,
                               generate_tsr_payload($ikev2));
    $packet .= generate_gen_hdr($ikev2, 0,
                               generate_notify_payload($ikev2));
    $encr_packet = generate_gen_hdr($ikev2, 35,
                                   generate_encr_payload($ikev2,
$packet));
    $final_packet = generate_hdr($ikev2, 46, 35, 8, 1, $encr_packet);
    return calculate_icv($ikev2, $final_packet);
}
```

Parse IKE_AUTH

```
########################################################################
# Parse IKE_AUTH packet
# parse_ike_sa_init($ikev2, $response_packet);

sub parse_ike_auth {
    my($ikev2, $packet) = @_;
    my($p);

    $p = parse_packet($ikev2, $packet);
    die "Wrong spi" if ($$p{hdr_spi_i} != $$ikev2{spi_i});
    die "Invalid major version" if (int($$p{hdr_version} / 16) != 2);
    die "Invalid exchange" if ($$p{hdr_exchange_type} != 35);
    die "Invalid flags" if ($$p{hdr_flags} != 0x20);
    die "Invalid message id" if ($$p{hdr_message_id} != 1);
    die "Missing auth" if (!defined($$p{39}));
    die "Missing id" if (!defined($$p{36}));
    die "Missing sa" if (!defined($$p{33}));
    die "Missing tsi" if (!defined($$p{44}));
    die "Missing tsr" if (!defined($$p{45}));
    die "Wrong num trans" if ($$p{33}{num_trans} != 3);
    die "Invalid proto id" if ($$p{33}{proto_id} != 3);
    die "Invalid sa payload" if ($$p{33}{spi_size} != 4);
    if ($$ikev2{cipher} eq 'null') {
        die "Invalid proposal 1 null" if ($$p{33}{1}{id} != 11);
    } else {
        die "Invalid proposal 1" if ($$p{33}{1}{a_type} != 14 ||
                                     $$p{33}{1}{a_value} != 128 ||
                                     $$p{33}{1}{id} != 12);
    }
    die "Invalid proposal 3" if (defined($$p{33}{3}{a_type}) ||
                                 $$p{33}{3}{id} != 2);
    die "Invalid proposal 5" if (defined($$p{33}{5}{a_type}) ||
                                 $$p{33}{5}{id} != 0);
    $$ikev2{p_ike_auth} = $p;
    die "Authentication failed" if (!check_auth($ikev2));
}
```

PRF+

```
########################################################################
# Calculate PRF+
# $data = prf_plus($ret_len, $key, $data);

sub prf_plus {
    my($ret_len, $key, $data) = @_;
    my($buf, $cnt, $prev);

    $buf = '';
    $prev = '';
    $cnt = 1;
    while (length($buf) < $ret_len) {
        $prev = hmac_sha1($prev . $data . chr($cnt++), $key);
        $buf .= $prev
    }
    return $buf;
}
```

Key Calculation

```
########################################################################
# Calculate IKE SA Keys
# calculate_keys($ikev2);

sub calculate_keys {
    my($ikev2) = @_;
    my($g_ir, $skeyseed, $keymat, $pub);

    $pub = "0x" . unpack("H*", $$ikev2{p_ike_sa}{34}{dh_value});
    $g_ir = $$ikev2{dh}->compute_secret(Math::BigInt->new($pub))-
>as_hex;
    $g_ir =~ s/^0x//g;
    $g_ir = pack("H*", $g_ir);

    printf(STDERR "g_ir:\n%s", bin_to_hexl($g_ir));
    printf(STDERR "nonce_i:\n%s", bin_to_hexl($$ikev2{nonce_i}));
    printf(STDERR "nonce_r:\n%s", bin_to_hexl($$ikev2{p_ike_sa}{40}
{nonce}));

    $skeyseed = hmac_sha1($g_ir, $$ikev2{nonce_i} .
                          $$ikev2{p_ike_sa}{40}{nonce});

    printf(STDERR "skeyseed:\n%s", bin_to_hexl($skeyseed));

    printf(STDERR "spi_i:\n%s", bin_to_hexl($$ikev2{spi_i}));
    printf(STDERR "spi_r:\n%s", bin_to_hexl($$ikev2{spi_r}));

    $keymat = prf_plus(20 + 20 * 2 + 16 * 2 + 20 * 2, $skeyseed,
                       $$ikev2{nonce_i} . $$ikev2{p_ike_sa}{40}{nonce}
.
                       $$ikev2{spi_i} . $$ikev2{spi_r});

    printf(STDERR "keymat:\n%s", bin_to_hexl($keymat));

    $$ikev2{sk_d} = substr($keymat, 0, 20);
    $$ikev2{sk_ai} = substr($keymat, 20, 20);
    $$ikev2{sk_ar} = substr($keymat, 40, 20);
    $$ikev2{sk_ei} = substr($keymat, 60, 16);
    $$ikev2{sk_er} = substr($keymat, 76, 16);
    $$ikev2{sk_pi} = substr($keymat, 92, 20);
    $$ikev2{sk_pr} = substr($keymat, 112, 20);

    printf(STDERR "sk_d:\n%s", bin_to_hexl($$ikev2{sk_d}));
    printf(STDERR "sk_ai:\n%s", bin_to_hexl($$ikev2{sk_ai}));
    printf(STDERR "sk_ar:\n%s", bin_to_hexl($$ikev2{sk_ar}));
    printf(STDERR "sk_ei:\n%s", bin_to_hexl($$ikev2{sk_ei}));
    printf(STDERR "sk_er:\n%s", bin_to_hexl($$ikev2{sk_er}));
    printf(STDERR "sk_pi:\n%s", bin_to_hexl($$ikev2{sk_pi}));
    printf(STDERR "sk_pr:\n%s", bin_to_hexl($$ikev2{sk_pr}));
}
```

IPsec keys

```
########################################################################
# Calculate IPsec SA keys and parameters
# ipsec_params($ikev2);

sub ipsec_params {
    my($ikev2) = @_;
    my($keymat, %hash);

    if ($$ikev2{cipher} eq 'null') {
        $keymat = prf_plus(20 * 2, $$ikev2{sk_d},
                           $$ikev2{nonce_i} . $$ikev2{p_ike_sa}{40}
{nonce});
        $hash{cipher_key_out} = '';
        $hash{auth_key_out} = substr($keymat, 0, 20);
        $hash{cipher_key_in} = '';
        $hash{auth_key_in} = substr($keymat, 20, 20);
        $hash{spi_out} = $$ikev2{p_ike_auth}{33}{spi};
        $hash{spi_in} = 0x12345678;
    } else {
        $keymat = prf_plus(20 * 2 + 16 * 2, $$ikev2{sk_d},
                           $$ikev2{nonce_i} . $$ikev2{p_ike_sa}{40}
{nonce});
        $hash{cipher_key_out} = substr($keymat, 0, 16);
        $hash{auth_key_out} = substr($keymat, 16, 20);
        $hash{cipher_key_in} = substr($keymat, 36, 16);
        $hash{auth_key_in} = substr($keymat, 52, 20);
        $hash{spi_out} = $$ikev2{p_ike_auth}{33}{spi};
        $hash{spi_in} = 0x12345678;
    }
    printf(STDERR "cipher_out:\n%s",
bin_to_hexl($hash{cipher_key_out}));
    printf(STDERR "auth_out:\n%s", bin_to_hexl($hash{auth_key_out}));
    printf(STDERR "cipher_in:\n%s",
bin_to_hexl($hash{cipher_key_in}));
    printf(STDERR "auth_in:\n%s", bin_to_hexl($hash{auth_key_in}));
    printf(STDERR "spi_in:\n%s", bin_to_hexl($hash{spi_in}));
    printf(STDERR "spi_out:\n%s", bin_to_hexl($hash{spi_out}));
    return \%hash;
}
```

Utility/debug

```
########################################################################
# Print hash

sub print_hash {
    my($hash, $indent) = @_;
    my($i, $j, $value, $print);

    $indent = "" if (!defined($indent));
    foreach $i (sort { return $a cmp $b; } (keys %{$hash})) {
        $value = $$hash{$i};
        if (ref($value) eq 'HASH') {
            printf("%s%-20s\n", $indent, $i);
            print_hash($value, $indent . "    ");
        } elsif (ref($value) eq 'ARRAY') {
            printf("%s%-20s\n", $indent, $i);
            for($j = 0; $j <= $#$value; $j++) {
                print_hash($$value[$j], $indent . "[" . $j . "]");
            }
        } elsif ($value =~ /^[:print:]*$/) {
            printf("%s%-20s: %s\n", $indent, $i, $value);
        } elsif ($value + 0 eq $value) {
            printf("%s%-20s: %d\n", $indent, $i, $value);
        } else {
            printf("%s%-20s\n%s", $indent, $i, bin_to_hexl($value,
$indent));
        }
    }
}
```

Do Exchange

```
########################################################################
# Do IKEv2 request / response exchange
# $response_packet = do_exchange($ikev2, $request_packet);

sub do_exchange {
    my($ikev2, $request) = @_;
    my($proto, $port, $paddr, $hispaddr);
    my($count, $timeout, $rin, $rout, $buf, $nfound);

    $proto = getprotobyname('udp');
    $paddr = sockaddr_in($$ikev2{srcport}, inet_aton($
$ikev2{srchost}));
    $hispaddr = sockaddr_in($$ikev2{dstport}, inet_aton($
$ikev2{dsthost}));

    socket(SOCKET, PF_INET, SOCK_DGRAM, $proto)   || die "socket: $!";
    bind(SOCKET, $paddr)                          || die "bind: $!";

    $count = 10;
    $timeout = 1;
    $rin = '';
    vec($rin, fileno(SOCKET), 1) = 1;

    while ($count-- > 0) {
        hexl_print("Sending packet", $request);
        if (!defined(send(SOCKET, $request, 0, $hispaddr))) {
            die "send: $!";
        }

        $nfound = select($rout = $rin, undef, undef, $timeout);
        if ($nfound != 0) {
            $hispaddr = recv(SOCKET, $buf, 1280, 0);
            if (!defined($hispaddr)) {
                die "Recv: $!";
            }
            hexl_print("Received packet", $buf);
            last;
        }
        $timeout *= 2;
        $timeout = 10 if ($timeout > 10);
    }
    die "Timeout" if ($count <= 0);
    close(SOCKET);
    return $buf;
}
```

Utility/debug

```
########################################################################
# Get random string with given length
# $str = generate_random($ikev2, $length)

sub generate_random {
    my($ikev2, $length) = @_;

    return makerandom_octet(Length => $length, Strength => 0);
}

########################################################################
# IP checksum
# $sum = checksum($packet);

sub checksum {
    my($packet) = @_;
    my($len, $num, $i, $chk, $short);

    $chk = 0;
    $len = length($packet);
    $num = int($len / 2);
    foreach $short (unpack("n$num", $packet)) {
        $chk += $short;
    }
    $chk += (ord(substr($packet, -1, 1)) << 8) if ($len % 2);
    $chk = ($chk >> 16) + ($chk & 0xffff);
    $chk = ($chk >> 16) + ($chk & 0xffff);
    return (~$chk) & 0xffff;
}

########################################################################
# hexl_print

sub hexl_print {
    my($txt, $value) = @_;
    print($txt, "\n", bin_to_hexl($value), "\n");
}

########################################################################
# bin_to_hexl

sub bin_to_hexl {
    my($data, $indent) = @_;
    my($len, $i, $j, $ret, $c);
    $indent = '' if (!defined($indent));
    $len = length($data);
    $ret = '';
    for($i = 0; $i < $len; $i += 16) {
        $ret .= sprintf("%s%08x: ", $indent, $i);
        for($j = 0; $j < 16; $j++) {
            if ($i + $j < $len) {
                $ret .= sprintf("%02x", ord(substr($data, $i + $j,
1)));
            } else {
                $ret .= "  ";
            }
            if (($j & 1) == 1) {
                $ret .= " ";
            }
        }
        $ret .= " ";
        for($j = 0; $j < 16; $j++) {
            if ($i + $j < $len) {
                $c = substr($data, $i + $j, 1);
                if ($c =~ /[[:print:]]/) {
                    $ret .= $c;
                } else {
                    $ret .= ".";
                }
            } else {
                $ret .= " ";
            }
        }
        $ret .= "\n";
    }
    return $ret;
}
```

Conclusions

- IKEv2 is very small protocol when only minimal features are implemented
- Certificate support would multiply the code size
- Pre-shared keys or RAW RSA keys are feasible options for authentication in this kind if use scenarios
- My draft describes more of those optimizations possible:
  - draft-kivinen-ipsecme-ikev2-minimal-00.txt