Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 10, 2017

Y. Nir
Check Point
S. Josefsson
SJD

August 9, 2016

Curve25519 and Curve448 for IKEv2 Key Agreement
draft-ietf-ipsecme-safecurves-03

Abstract

This document describes the use of Curve25519 and Curve448 for ephemeral key exchange in the Internet Key Exchange (IKEv2) protocol.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The "Elliptic Curves for Security" document [RFC7748] describes two elliptic curves: Curve25519 and Curve448, as well as the X25519 and X448 functions for performing key agreement (Diffie-Hellman) operations with these curves. The curves and functions are designed for both performance and security.

Almost ten years ago the "ECP Groups for IKE and IKEv2" document [RFC4753] specified the first elliptic curve Diffie-Hellman groups for the Internet Key Exchange protocol (IKEv2 - [RFC7296]). These were the so-called NIST curves. The state of the art has advanced since then. More modern curves allow faster implementations while making it much easier to write constant-time implementations free from time-based side-channel attacks. This document defines two such curves for use in IKE. See [Curve25519] for details about the speed and security of the Curve25519 function.

1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Curve25519 & Curve448

All cryptographic computations are done using the X25519 and X448 functions defined in [RFC7748]. All related parameters (for example, the base point) and the encoding (in particular, pruning the least/ most significant bits and use of little-endian encoding) are inherited from [RFC7748].

An ephemeral Diffie-Hellman key exchange using Curve25519 or Curve448 goes as follows: Each party picks a secret key d uniformly at random and computes the corresponding public key. "X" is used below to denote either X25519 or X448, and "G" is used to denote the corresponding base point:

```
pub_mine = X(d, G)
```

Parties exchange their public keys (see Section 3.1) and compute a shared secret:

```
SHARED_SECRET = X(d, pub_peer).
```

This shared secret is used directly as the value denoted g^ir in section 2.14 of RFC 7296. It is 32 octets when Curve25519 is used, and 56 octets when Curve448 is used.

3. Use and Negotiation in IKEv2

The use of Curve25519 and Curve448 in IKEv2 is negotiated using a Transform Type 4 (Diffie-Hellman group) in the SA payload of either an IKE_SA_INIT or a CREATE_CHILD_SA exchange. The value TBA1 is used for the group defined by Curve25519 and the value TBA2 is used for the group defined by Curve448.

3.1. Key Exchange Payload

The diagram for the Key Exchange Payload from section 3.4 of RFC 7296 is copied below for convenience:

```
                     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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Diffie-Hellman Group Num    |           RESERVED            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                       Key Exchange Data                       ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Payload Length - For Curve25519 the public key is 32 octets, so the Payload Length field will be 40, and for Curve448 the public key is 56 octets, so the Payload Length field will be 64.
- The Diffie-Hellman Group Num is TBA1 for Curve25519, or TBA2 for Curve448.

- The Key Exchange Data is the 32 or 56 octets as described in section 6 of [RFC7748]

3.2. Recipient Tests

This document matches the discussion in [RFC7748] related to receiving and accepting incompatible point formats. In particular, receiving entities MUST mask the most-significant bit in the final byte for X25519 (but not X448), and implementations MUST accept non-canonical values. See section 5 of [RFC7748] for further discussion.

4. Security Considerations

Curve25519 and Curve448 are designed to facilitate the production of high-performance constant-time implementations. Implementors are encouraged to use a constant-time implementation of the functions. This point is of crucial importance if the implementation chooses to reuse its supposedly ephemeral key pair for many key exchanges, which some implementations do in order to improve performance.

Curve25519 is intended for the ~128-bit security level, comparable to the 256-bit random ECP group (group 19) defined in RFC 4753, also known as NIST P-256 or secp256r1. Curve448 is intended for the ~224-bit security level.

While the NIST curves are advertised as being chosen verifiably at random, there is no explanation for the seeds used to generate them. In contrast, the process used to pick these curves is fully documented and rigid enough so that independent verification has been done. This is widely seen as a security advantage, since it prevents the generating party from maliciously manipulating the parameters.

Another family of curves available in IKE, generated in a fully verifiable way, is the Brainpool curves [RFC6954]. For example, brainpoolP256 (group 28) is expected to provide a level of security comparable to Curve25519 and NIST P-256. However, due to the use of pseudo-random prime, it is significantly slower than NIST P-256, which is itself slower than Curve25519.

5. IANA Considerations

IANA is requested to assign two values from the IKEv2 "Transform Type 4 - Diffie-Hellman Group Transform IDs" registry, with names "Curve25519" and "Curve448" and this document as reference. The Recipient Tests field should also point to this document:

Number	Name	Recipient Tests	Reference
TBA1	Curve25519	RFCxxxx Section 3.2	RFCxxxx
TBA2	Curve448	RFCxxxx Section 3.2	RFCxxxx

Table 1: New Transform Type 4 Values

6. Acknowledgements

Curve25519 was designed by D. J. Bernstein and the parameters for Curve448 ("Goldilocks") is by Mike Hamburg. The specification of algorithms, wire format and other considerations are in RFC 7748 by Adam Langley, Mike Hamburg, and Sean Turner.

The example in Appendix A was calculated using the master version of OpenSSL, retrieved on August 4th, 2016.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC7296] Kivinen, T., Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 7296, October 2014.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, January 2016.

7.2. Informative References

[Curve25519]
Bernstein, J., "Curve25519: New Diffie-Hellman Speed Records", LNCS 3958, February 2006, <http://dx.doi.org/10.1007/11745853_14>.

[RFC4753] Fu, D. and J. Solinas, "ECP Groups For IKE and IKEv2", RFC 4753, January 2007.

[RFC6954] Merkle, J. and M. Lochter, "Using the Elliptic Curve Cryptography (ECC) Brainpool Curves for the Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 6954, July 2013.

Appendix A. Numerical Example for Curve25519

Suppose we have both the initiator and the responder generating private keys by generating 32 random octets. As usual in IKEv2 and its extension, we will denote Initiator values with the suffix _i and responder values with the suffix _r:

```
random_i = 75 1f b4 30 86 55 b4 76 b6 78 9b 73 25 f9 ea 8c
           dd d1 6a 58 53 3f f6 d9 e6 00 09 46 4a 5f 9d 94

random_r = 0a 54 64 52 53 29 0d 60 dd ad d0 e0 30 ba cd 9e
           55 01 ef dc 22 07 55 a1 e9 78 f1 b8 39 a0 56 88
```

These numbers need to be fixed by unsetting some bits as described in section 5 of RFC 7748. This affects only the first and last octets of each value:

```
fixed_i =  70 1f b4 30 86 55 b4 76 b6 78 9b 73 25 f9 ea 8c
           dd d1 6a 58 53 3f f6 d9 e6 00 09 46 4a 5f 9d 54

fixed_r =  08 54 64 52 53 29 0d 60 dd ad d0 e0 30 ba cd 9e
           55 01 ef dc 22 07 55 a1 e9 78 f1 b8 39 a0 56 48
```

The actual private keys are considered to be encoded in little-endian format:

```
d_i = 549D5F4A460900E6D9F63F53586AD1DD8CEAF925739B78B676B4558630B41F70

d_r = 4856A039B8F178E9A1550722DCEF01559ECDBA30E0D0ADDD600D295352645408
```

The public keys are generated from this using the formula in Section 2:

```
pub_i = X25519(d_i, G) =
             48 d5 dd d4 06 12 57 ba 16 6f a3 f9 bb db 74 f1
             a4 e8 1c 08 93 84 fa 77 f7 90 70 9f 0d fb c7 66

pub_r = X25519(d_r, G) =
             0b e7 c1 f5 aa d8 7d 7e 44 86 62 67 32 98 a4 43
             47 8b 85 97 45 17 9e af 56 4c 79 c0 ef 6e ee 25
```

And this is the value of the Key Exchange Data field in the key exchange payload described in Section 3.1. The shared value is calculated as in Section 2:

```
SHARED_SECRET = X25519(d_i, pub_r) = X25519(d_r, pub_i) =
             c7 49 50 60 7a 12 32 7f-32 04 d9 4b 68 25 bf b0
             68 b7 f8 31 9a 9e 37 08-ed 3d 43 ce 81 30 c9 50
```

Authors' Addresses

Yoav Nir
Check Point Software Technologies Ltd.
5 Hasolelim st.
Tel Aviv 6789735
Israel

Email: ynir.ietf@gmail.com

Simon Josefsson
SJD AB

Email: simon@josefsson.org