Network Working Group
Internet-Draft
Updates: 7748 (if approved)
Intended status: Informational
Expires: May 7, 2020

R. Barnes
Cisco
J. Alwen
Wickr
S. Corretti
IOHK

November 04, 2019

Homomorphic Multiplication for X25519 and X448
draft-barnes-cfrg-mult-for-7748-00

Abstract

In some contexts it is useful for holders of the private and public parts of an elliptic curve key pair to be able to independently apply an updates to those values, such that the resulting updated public key corresponds to the updated private key. Such updates are straightforward for older elliptic curves, but for X25519 and X448, the "clamping" prescribed for scalars requires some additional processing. This document defines a multiplication procedure that can be used to update X25519 and X448 key pairs. This algorithm can fail to produce a result, but only with negligible probability. Failures can be detected by the holder of the private key.

Note to Readers

Source for this draft and an issue tracker can be found at https://github.com/bifurcation/draft-barnes-cfrg-mult-for-7748 [1].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 7, 2020.

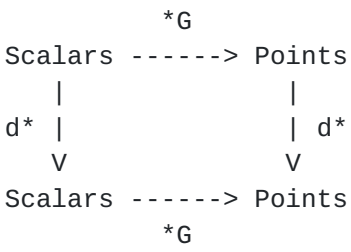
Table of Contents

1. Introduction

In some contexts it is useful for holders of the private and public parts of an elliptic curve key pair to be able to independently apply an updates to those values, such that the resulting updated public key corresponds to the updated private key. [[TODO: Cite examples (e.g. HKD like BIP32, Tor Hidden Service Identity Blinding, MLS), security properties]]

Such updates are straightforward with traditional elliptic curve groups, such as the NIST and Brainpool curve groups [NISTCurves][RFC5639], or with the proposed Ristretto groups [I-D.hdevalence-cfrg-ristretto]. In these groups, multiplication of

points by scalars is a homomorphism with regard to multiplication of scalars, so a key pair can be updated by multiplying the private key and the same "delta" scalar. In other words, the following diagram commutes for all "d", where "d*" represents scalar multiplication by "d" and "*G" represents multiplication of a scalar with the base point of the curve:



The X25519 and X448 functions defined in RFC 7748 [RFC7748], however, require scalars to be "clamped" before point multiplication is performed, which breaks this homomorphism. In particular, scalars are passed through the "decodeScalar25519" or "decodeScalar448" functions, respectively, which force a high-order bit to be set. Since this high-order bit is not guaranteed to be set in the product of two such numbers, the product of of two scalars may not represent a valid private key. In fact, there are points on Curve25519/Curve448 which are not X25519/X448 public keys, because their discrete logs do not have the correct high bit set.

Fortunately, X25519 and X448 use only one coordinate to represent curve points, which means they are insensitive to the sign of the point, so a scalar private key and its negative result in the same public key. And if a given scalar does not have the correct bit set, then its negative modulo the curve order almost certainly does. (We quantify these ideas below.) This allows us to construct an amended multiplication routine that succeeds with overwhelming probability, and where the failure cases are detectable by the holder of the private key.

The remainder of this document describes these algorithms, quantifies the failure cases and the resulting probabilities of failure, and discusses how failures can be detected.

2. Updating X25519 / X448 key pairs

The following procedures allow for updating X25519/X448 public keys and private keys in constant time. The values "sk" and "pk" represent the private and public keys of the key pair, and the value "d" represents a "delta" scalar being applied. All arithmetic operations are performed modulo the order "n" of the relevant curve:

- X25519:
  - "x = 0x14def9dea2f79cd65812631a5cf5d3ed"
  - "n = 8 * (2^253 + x)"
  - "b = 254"
- X448:
  - "x = 0x8335dc163bb124b65129c96fde933d8d723a70aadc873d6d54a7bb0d"
  - "n = 4 * (2^446 - x)"
  - "b = 447"

```
def updatePublic(d, pk):
  return CurveN(d, pkI)

def updatePrivate(d, sk):
  dc = decodeScalar(d)
  skc = decodeScalar(sk)
  skP = dc * skc
  skN = n - skP
  cP = (skP >> b) & 1
  cswap(1 - cP, skP, skN)
  return skP
```

The pulic operation is clearly just a normal DH operation in the relevant curve. The private operation computes the product of the delta with the private key as well as its negative modulo the curve order. If the product does not have the correct bit set, then the cswap operation ensures that that the negative is returned.

If "updatePrivate" and "updatePublic" are called on both halves of a key pair, and "updatePrivate" produces a valid private key (with the relevant high bit set), then the output private and public keys will correspond to each other. (That is, the public key will equal the private key times the base point.) If the "updatePrivate" function does not return a valid private key, then the update has failed, and the delta "d" cannot be used with this key pair.

3. Failure Cases

An update of a private key "sk" by a delta "d" fails if and only if neither "d*sk" or "n - d*sk" has the relevant bit set. In this section, we will assume that for uniform "d", this product "c = d*sk"

is uniformly distributed among scalars modulo "n". From this assumption, we can describe the set of values "c" for which updates fail, and thus estimate the probability that an update will fail.

In general, an update fails if neither "c" nor its negative has the relevant high bit set, i.e., if they are not in the range "[M, N]", where "M = 2^b" and "N = 2^{b+1} - 1". So our failure criterion is:

```
    (c < M || c > N) && ((n-c) < M || (n-c) > N)
<=> (c < M || c > N) && (c < n-N || c > n-M)
```

So the probability of failures is proportional to the size of the set where this conditions applies. In the following subsections, we will calculate these values for X25519 and X448.

3.1. X25519

In the case of X25519, the following values apply:

```
    b = 254
    n = 8 * (2^253 + x)
      = 2^255 + 8x
    M = 2^254
    N = 2^255 - 1
n - M = (2^255 + 8x) - 2^254
      = 2^254 + 8x
n - N = 8x + 1
```

Thus we have "n - N < M < n-M < N", so the failure set "F" and the failure probability "|F|/n" are as follows:

```
    F = [0, n-N) U (N, n]
      = [0, 8x + 1) U (2^255 - 1, 2^255 + 8x]
|F|/n = (2 * 8x) / (2^255 + 8x)
      < 2^130 / 2^255 (since x < 2^125)
      = 2^-125
```

3.2. X448

In the case of Curve448, the following values apply:

```
    b = 447
    n = 4 * (2^446 - x)
      = 2^448 - 4x
    M = 2^447
    N = 2^448 - 1
n - M = (2^448 - 4x) - 2^447
      = 2^447 - 4x
n - N = 1 - 4x
```

Thus we have "n - N < 0 < n - M < M < N", so the failure set "F" and the failure probability "|F|/n" are as follows:

```
    F = (n-M, M)
      = (2^447 - 4x, 2^447)
|F|/n = (4x - 1) / (2^448 - 4x)
      < 2^226 / 2^448           (since x < 2^224)
      = 2^-222
```

4. Protocol Considerations

Protocols making use of the update mechanism defined in this document should account for the possibility that updates can fail. As described above, entities updating private keys can tell when the update fails. However, entities that hold only the public key of a key pair will not be able to detect such a failure. So when this mechanism is used in a given protocol context, it should be possible for the private-key updater to inform other actors in the protocol that a failure has occurred.

5. Security Considerations

[[TODO: Comment on whether this algorithm achieves the security objective stated in the introduction]]

The major security concerns with this algorithm are implementation-related. The "updatePrivate" function requires access to the private key in ways that are typically not exposed by HSMs or other limited-access crypto libraries. Implementing key updates with such limited interfaces will require either exporting the private key or implementing the update algorithm internally to the HSM/library. (The latter obviously being preferable.)

As an algorithm involving secret key material, the "updatePrivate" function needs to be implemented in in a way that does not leak secrets through side channels. While the algorithm specified above is logically constant-time, it reques that multiplication, subtraction, and conditional swap be implemented in constant time.

6. IANA Considerations

This document makes no request of IANA.

7. References

7.1. Normative References

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

7.2. Informative References

[I-D.hdevalence-cfrg-ristretto]
Valence, H., Grigg, J., Tankersley, G., Valsorda, F., and I. Lovecruft, "The ristretto255 Group", draft-hdevalence-cfrg-ristretto-01 (work in progress), May 2019.

[NISTCurves]
"Digital Signature Standard (DSS)", National Institute of Standards and Technology report, DOI 10.6028/nist.fips.186-4, July 2013.

[RFC5639] Lochter, M. and J. Merkle, "Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation", RFC 5639, DOI 10.17487/RFC5639, March 2010, <https://www.rfc-editor.org/info/rfc5639>.

7.3. URIs

[1] https://github.com/bifurcation/draft-barnes-cfrg-mult-for-7748

Appendix A. Test Vectors

All values are presented as little-endian integers. An implementation should verify that the following relations hold:

- sk1 = updatePriv(d, sk0)
- pk1 = updatePub(d, pk0)
- pk1 = CurveN(sk1)

A.1. X25519

Successful update (no swap):

```
sk0 ff08989a80d6042f21cafd4af63f4334cc9a08e9a18a55f28739dd9c96762b36
pk0 4d8ce30b0322cbf5caa30d654f14e986a774fd2a50135165818ef3ed02566462
d   7ceff33b5fa2e095c37f773bdcc747e0071bea02d6b58f7a6c4283b1fea5df39
dC  78eff33b5fa2e095c37f773bdcc747e0071bea02d6b58f7a6c4283b1fea5df79
skC f808989a80d6042f21cafd4af63f4334cc9a08e9a18a55f28739dd9c96762b76
skP c848d5753e4d06e9d475d972537b4b8f32c7c81a97c538ced90aa0f64414e661
skN a056d97194cb8cd7dd70e3a4a153ac17ce3837e5683ac73126f55f09bbeb191e
cP  1
sk1 c848d5753e4d06e9d475d972537b4b8f32c7c81a97c538ced90aa0f64414e661
pk1 ee043d01816ba2da7cc37421ecbfd8c947afd57b18344dcfe77071929c02e061
```

Successful update (swap):

```
sk0 61b64d3177801e6a8bb742b235edccf32736c76ee0bfd62c9b4a204e7f2dd544
pk0 1d415ce35bff1279fb7392ea5ec6e856f670c289d2e4bd2161c876bb7d662a7b
d   7ceff33b5fa2e095c37f773bdcc747e0071bea02d6b58f7a6c4283b1fea5df39
dC  78eff33b5fa2e095c37f773bdcc747e0071bea02d6b58f7a6c4283b1fea5df79
skC 60b64d3177801e6a8bb742b235edccf32736c76ee0bfd62c9b4a204e7f2dd544
skP 786eb7c972f788e1d97f14eba675801dbdece5e93183502b65ec6abe2b525c5e
skN f030f71d60210adfd866a82c4e59778943131a16ce7cafd49a139541d4ada321
cP  0
sk1 786eb7c972f788e1d97f14eba675801dbdece5e93183502b65ec6abe2b525c5e
pk1 a854ba19d7de3d73531501566b4e4e51ab263d743316525a86d6ecc2bff5036a
```

Failed update:

```
sk0 e09ec60ffb39ef974324161d749df7881124492d369906147ea3a64086c1e857
pk0 984398c1dc572f13a20dd046901daf6716615e37d8c701ed75fb3871af14d374
d   7ceff33b5fa2e095c37f773bdcc747e0071bea02d6b58f7a6c4283b1fea5df39
dC  78eff33b5fa2e095c37f773bdcc747e0071bea02d6b58f7a6c4283b1fea5df79
skC e09ec60ffb39ef974324161d749df7881124492d369906147ea3a64086c1e857
skP 289faee7d21893c0b2e6bc17f5cef7a600000000000000000000000000000000
skN 4000000000000000000000000000000000000000000000000000000000000080
cP  0
sk1 289faee7d21893c0b2e6bc17f5cef7a600000000000000000000000000000000
pk1 af6c6be037cc0622e88a735a98f77ac06f372bad8542bc0f65c0c580b095ae4e
```

A.2. X448

Successful update (no swap):

```
   sk0 745310aa0942b1cf2d7d4a8eef25c572da5f647ae376e7f1f5dcacdd
       cc0d09419a0cb773d73d331e68e6f6485427de9ecddb8f73440e0012
   pk0 e1ff42e736266138310db4beab444fe68440b2f1459c729e537d9833
       6fa009ce25b3a3c57743577d995cf7f6f18e40f2fb228e5177c06219
   d   f50678b0c8505f04554b7f8e04b1ab1682b681f279df4d84129b07e0
       4bcfe59f328e9ee2bbb64e9ae94c776593e8bac549fe40d1a4e81371
   dC  f40678b0c8505f04554b7f8e04b1ab1682b681f279df4d84129b07e0
       4bcfe59f328e9ee2bbb64e9ae94c776593e8bac549fe40d1a4e813f1
   skC 745310aa0942b1cf2d7d4a8eef25c572da5f647ae376e7f1f5dcacdd
       cc0d09419a0cb773d73d331e68e6f6485427de9ecddb8f73440e0092
   skP 908dafad675a0b99fd6991d19e13c3b26f121a4881316601fc192e0a
       0737b99f44ead69b52684dd00ccb5ea34c1a8ea5d768510f34e873d6
   skN 3c86b1ffe2afd7f456d384652bf6efd2d0c73e73a53bd50fab75fae8
       f6c84660bb152964ad97b22ff334a15cb3e5715a2897aef0cb178c29
   cP  1
   sk1 908dafad675a0b99fd6991d19e13c3b26f121a4881316601fc192e0a
       0737b99f44ead69b52684dd00ccb5ea34c1a8ea5d768510f34e873d6
   pk1 40e213cb2c06c6ec327e80623ecb2625b7a474a9eb4eb7f8c601d148
       cd7734a59afbb5886efe1cca48b36353d750d076a7fec3f4686ffa27

   Successful update (swap):

   sk0 42e35e26d257aed97ec5d66528504acbbd4141dae7eefb232c30f6da
       8664ef38b083020f0931adaeb511143d80de6942c1096f33fe96c80e
   pk0 a79df64f2b9c3510ccf27825f7524791ede627ce76f4a174cf050521
       86e2994aa078cb2a605179cfcd33ec4e6747f222036025f6233c0268
   d   f50678b0c8505f04554b7f8e04b1ab1682b681f279df4d84129b07e0
       4bcfe59f328e9ee2bbb64e9ae94c776593e8bac549fe40d1a4e81371
   dC  f40678b0c8505f04554b7f8e04b1ab1682b681f279df4d84129b07e0
       4bcfe59f328e9ee2bbb64e9ae94c776593e8bac549fe40d1a4e813f1
   skC 40e35e26d257aed97ec5d66528504acbbd4141dae7eefb232c30f6da
       8664ef38b083020f0931adaeb511143d80de6942c1096f33fe96c88e
   skP b4ba86f8b4d83a0b4d192df1e0ab71645719e7ddf304fa94f155c3e6
       ff6c746fdc45aa45e94ab75a146d9f8bf50cc8c4f520bc7d72d4ba8b
   skN 1859dab49531a8820724e945e95d4121e9c071dd3268417cb539650c
       fe928b9023ba55ba16b548a5eb9260740af3373b0adf43828d2b4574
   cP  0
   sk1 b4ba86f8b4d83a0b4d192df1e0ab71645719e7ddf304fa94f155c3e6
       ff6c746fdc45aa45e94ab75a146d9f8bf50cc8c4f520bc7d72d4ba8b
   pk1 eeb52f6eeb3d1785077dca6c763c0489c85f22fe5e96a82c54153ac3
       33138c250b66445beb28986d3b7dbda1974049949906ab13f69151bc

   Failed update:
```

```
   sk0 48441950f8dd7ed277ed9727798b283906774ba0b3917fb21371c8f6
       2e164c3a069e224338d696a3dfe0c99b7277593949b8e555eb0766ed
   pk0 aaf96ee42f7752c54544225f129cd8bccb8ad834f65f6186d11cbe9b
       105087ebf04408e0159c726eacaa8975d0c39a9a6304dca2b5d6b2eb
   d   f50678b0c8505f04554b7f8e04b1ab1682b681f279df4d84129b07e0
       4bcfe59f328e9ee2bbb64e9ae94c776593e8bac549fe40d1a4e81371
   dC  f40678b0c8505f04554b7f8e04b1ab1682b681f279df4d84129b07e0
       4bcfe59f328e9ee2bbb64e9ae94c776593e8bac549fe40d1a4e813f1
   skC 48441950f8dd7ed277ed9727798b283906774ba0b3917fb21371c8f6
       2e164c3a069e224338d696a3dfe0c99b7277593949b8e555eb0766ed
   skP ecffffffffffffffffffffffffffffffffffffffffffffffffffffff
       ffffffffffffffffffffffffffffffffffffffffffffffffffffffff7f
   skN e01361ad4a0ae38d543d1637ca09b38540da58bb266d3b11a78f28f3
       fdffffffffffffffffffffffffffffffffffffffffffffffffffffff7f
   cP  0
   sk1 ecffffffffffffffffffffffffffffffffffffffffffffffffffffffff
       ffffffffffffffffffffffffffffffffffffffffffffffffffffffff7f
   pk1 a643f22b04d50faf004a08f6ff38acbf0cbca8591b1f07a70e269ce1
       4e240e5389a583eab63ab6d9d49d4fe051305f6676201d41df60b83d
```

Appendix B. Acknowledgements

Thanks to Mike Hamburg for reviewing an early version of the ideas that led to this document.

Authors' Addresses

Richard L. Barnes
Cisco

Email: rlb@ipv.sx

Joel Alwen
Wickr

Email: jalwen@wickr.com

Sandro Corretti
IOHK

Email: corettis@gmail.com