BIP32-Ed25519
Hierarchical Deterministic Keys over a Non-linear Keyspace

Dmitry Khovratovich and Jason Law
Evernym, Inc.

7 June 2016

1 Introduction

1.1 Ed25519

Ed25519 is an elliptic curve standard for digital signatures developed in [1] and fully described as an IETF draft in [3]. This standard is notable for high speed, constant-time implementations, and no requirement for randomness in signature generation.

The coordinates (x, y) are pairs of elements of the prime field $\mathbb{F}_p$, where $p = 2^{255} - 19$.

The base point B is

$$(X(P), Y(P)) = ($$
$$15112221349535400772501151409588531511454012693041857206046113283949847762202,$$
$$46316835694926478169428394003475163141307993866256225615783033603165251855960).$$

It has a base order n of $2^{252} + 27742317777372353535851937790883648493$.

The infinity point is $(0, 0)$ and the identity point is $(0, 1)$.

1.1.1 Keys and signatures in Ed25519

We will denote cryptographic hashing function SHA-512 by $H_{512}()$ and SHA-256 by $H_{256}()$.

The private key $\widetilde{k}$ is a 32-byte cryptographically-secure random value. The public key A is derived as follows:

1. Hash the 32-byte private key $\widetilde{k}$ using H_{512}, storing the digest in a 64-byte buffer, denoted k. We call k an *extended private key*. Split k into the lower 32 bytes k_L and upper 32 bytes k_R.
2. Modify k_L: the lowest 3 bits of the first byte of are cleared[1], the highest bit of the last byte is cleared, and the second highest bit of the last byte is set[2].
3. Interpret k_L as a little-endian integer and perform a fixed-base scalar multiplication $[k_L]B$.
4. The public key A is the encoding of the point $[k_L]B$ as follows. First encode the y coordinate (in the range $0 \leq y < p$) as a little-endian string of 32 octets. The most significant bit of the final octet is always zero. To form the encoding of the point $[k_L]B$, copy the least significant bit of the x coordinate to the most significant bit of the final octet. The result is the public key.

Signature for message M is produced as follows:

1. Compute $H_{512}(k_R||M)$ and interpret the result as a little-endian-encoded integer r. Compute $r \leftarrow r \pmod{n}$.
2. Compute point $[r]B$ and let R be its encoding.
3. Compute $x \leftarrow H_{512}(R||A||M)$, and interpret the 64-byte digest as a little-endian integer.
4. Compute $S = (r + x \cdot a) \pmod{n}$.
5. The string $R||S$ is the signature.

[1] This is done for key compatibility with Curve25519, where it serves as a countermeasure against low-order attack.
[2] This is done to avoid timing leakage in multiplication algorithms that search for highest active bit.

2 BIP32

The idea of BIP32 is a hierarchy of keys with private-public key homomorphism. Here, there is a single private-public key pair, and a number of enumerated child key pairs. A child public key can be derived from the parent public key using a secret (or selectively shared) chain code and a child index, and the private child key is derived from the private parent key using the same values.

Full specification of BIP32 can be found in [5]. Here is the short summary:

- A base point B of order n is selected on the elliptic curve secp256k1.
- The root private key k is generated. It determines the root public key $A = [k]B$.
- The root chain code c is generated, possibly together with k as a MAC of some master secret.
- Root has 2^{32} children, indexed from 0 to $2^{32} - 1$. For each $i < 2^{31}$ the child private key k_i is $k + f_1(c, A, i) \pmod n$ for some public f_1, so that the corresponding public key A_i is $A + [f_1(c, A, i)]B$, where addition is performed on curve points.
- Child may have children too, for which its chain code is $f_2(c, A, i)$ for some another public f_2. Thus we have a tree of descendants from the original keypair.
- Child with $i \geq 2^{31}$ is called hardened as its private key is determined by parent private key: $k_i = k + f_3(c, k, i)$. Such children do not have children.

The crucial property of this concept is that child public keys can be determined by anyone knowing the chain code (or l codes, if the child is l nodes away from the root).

3 BIP32+Ed25519: concept

The curve secp256k1 has an important property – its private keys form an affine (and even linear) space. This is not the case for the curve Ed25519; however, it is possible to modify the BIP32 proposal slightly so that all the produced extended private keys lie in some affine space.

The crucial security requirement to the extended private key in Ed25519 is that it has certain bits set and cleared (as documented in Section 1.1.1) to avoid a class of attacks based on points of small order.

Our modifications are summarized as follows:

1. We work with the extended private key (64-byte) k in Ed25519, instead of the original 32-byte key $\widetilde{k}$. All extended keys have the bits set and cleared exactly as specified in [3]. Signing and verifying procedures remain the same as in Section 1.1.1. However, some Ed25519 libraries have a signing function that takes the original 32-byte secret key and expands it for every signing, while other libraries sign using the extended key. Due to the way child Ed25519 keys are derived, they will not have the 32-byte secret key. Although it's a trivial change to modify existing code to support this, you may want to consider this when selecting an encryption library. We use Bernstein's NaCl [2] which uses the extended key.
2. We admit only those $\widetilde{k}$ such that the third highest bit of the last byte of root private key k_L is zero.
3. The function f_1 in the BIP32 specification has longer output M to affect the 64-byte extended key.
4. The part M_L of M that affects the k_L part of the extended private key is trimmed to 28 bytes in order to guarantee that the second highest bit in the last byte of k_L is always 1.
5. The maximum number of levels in the tree is $2^{20} = 1048576$.
6. The functions f_1 and f_2, which originally take distinct parts of the single HMAC output, now require one HMAC call each.
7. Integers are serialized in the little-endian format (least significant bytes first), and byte strings are interpreted as little-endian integers.

4 BIP32+Ed25519: specification

This section describes the BIP32 proposal adapted for the use of the curve Ed25519 instead of the Bitcoin curve. The new proposal is called BIP32+Ed25519.

The master secrets of BIP32+Ed25519 and root extended private keys are backward compatible with Ed25519 keys. In other words, all BIP32+Ed25519 master secrets are valid Ed25519 private keys, and root extended private keys are valid Ed25519 extended private keys.

4.1 Root keys

Let $\widetilde{k}$ be 256-bit master secret. Then derive $k = H_{512}(\widetilde{k})$ and denote its left 32-byte by k_L and right one by k_R. If the third highest bit of the last byte of k_L is *not* zero, discard $\widetilde{k}$. Otherwise additionally set the bits in k_L as follows: the lowest 3 bits of the first byte of are cleared, the highest bit of the last byte is cleared, the second highest bit of the last byte is set. The resulting pair (k_L, k_R) is the extended root private key, and $A = [k_L]B$ is the root public key. Derive $c = H_{256}(0x01||\widetilde{k})$ and call it the root chain code.

4.2 Child keys

Root key can have 2^{32} child keys indexed from 0 to $2^{32} - 1$; the first 2^{31} of them can have their own children and so on. The procedure of child key derivation is the same for the root and its children.

Let c^P be the chain code. Let $k^P = (k_L^P, k_R^P)$ be the extended private key and A^P the public key.

Denote HMAC-SHA512 with key K by $HM_K()$.

4.3 Private child key

Extended private key $k_i = (k_L, k_R)$ for child i is produced as follows:

1. $$\begin{cases} Z \leftarrow HM_{c^P}(0x02||A^P||i), & i < 2^{31}; \\ Z \leftarrow HM_{c^P}(0x00||k^P||i), & i \geq 2^{31}. \end{cases}$$

   where A^P is serialized as little-endian 32-byte string, k^P is viewed as (k_L^P, k_R^P) and both values are serialized as little-endian, and i is serialized as little-endian 4-byte string.

2. $$k_L \leftarrow 8Z_L + k_L^P, \tag{1}$$
   $$k_R \leftarrow Z_R + k_R^P \pmod{2^{256}}. \tag{2}$$

   where Z_L is the left 28-byte part of Z interpreted as 224-bit integer using the little-endian representation, and Z_R is the right 32-byte part of Z interpreted as 256-bit integer. If k_L is divisible by n, discard the child.

3. The child chain code is defined as

   $$\begin{cases} c_i \leftarrow HM_{c^P}(0x03||A^P||i), & i < 2^{31}; \\ c_i \leftarrow HM_{c^P}(0x01||k^P||i), & i \geq 2^{31}. \end{cases}$$

   where the output of the HMAC function truncated to the right 32 bytes.

The child public key A_i is derived as $A_i = [k_L]B$.

4.4 Public child key

Public key A_i for child i is produced as follows:

1. $$\begin{cases} Z \leftarrow HM_{c^P}(0x02||A^P||i), & i < 2^{31}; \\ Z \leftarrow HM_{c^P}(0x00||k^P||i), & i \geq 2^{31}. \end{cases}$$

   where A^P is serialized as little-endian 32-byte string, and i is serialized as little-endian 4-byte string. *Note that the public key for $i \geq 2^{31}$ can be computed only by the private key owner and those knowing the parent private key.*

2. $$A_i = A^P + [8Z_L]B,$$

   where Z_L is the left 28-byte part of Z interpreted as 224-bit integer using the little-endian representation. If A_i is the identity point $(0, 1)$, discard the child.

3. The child chain code is defined as

   $$\begin{cases} c_i \leftarrow HM_{c^P}(0x03||A^P||i), & i < 2^{31}; \\ c_i \leftarrow HM_{c^P}(0x01||k^P||i), & i \geq 2^{31}. \end{cases}$$

   where the output of the HMAC function truncated to the right 32 bytes.

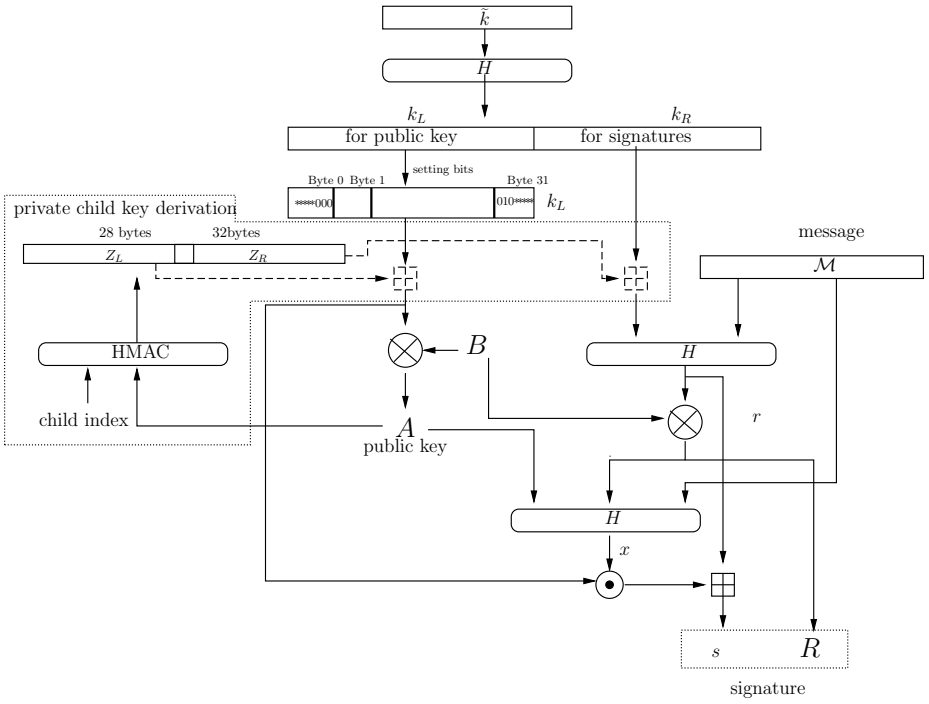

Figure 1: BIP32+Ed25519.

4.5 Key tree

Child with $i < 2^{31}$ can be a parent for his children with his own chain code c_i. Proceeding with this, we can create a tree of keys where each non-leaf node is a parent for its children and is a child for its parent. A path $m \to i_1 \to \ldots \to i_l$ from the original parent m to a child at level l thus uniquely identifies the node.

4.6 Security

Master key security We set 6 bits in the 512-bit extended key. Therefore, the extended key can be guessed after 2^{506} attempts by brute-force, or by trying 2^{256} different master secrets. Since the Ed25519 scheme claims the security level of 128 bits, our manipulations with the extended private key yield no security loss.

Child key collisions There are 2^{224} distinct M_L, so we expect that collisions in k_L and thus the public key collisions are possible for 2^{112} keys and more. However, such collisions do not help to forge signatures as the k_R part of the extended key is independent of k_L. Therefore, we do not see any degradation in the overall security because of our modifications.

It is easy to prove that all produced values a do not violate the requirements outlined in the Ed25519 specification:

- Highest bit of the last byte is cleared, second highest bit is set;
- Three lowest bits of the first byte are cleared.

Indeed, the root a has the form $a = 2^{254} + 8b$, where $b < 2^{250}$ since we clear the third highest bit. The value a^j at level j is defined as $a^j = a + \sum_{j'<j} 8M_L^{j'}$. Since $M_L < 2^{224}$, we get that $a^j \leq a + j2^{227}$. Since $j \leq 2^{20}$, we get $a^j \leq 2^{254} + 2^{253} + 2^{247}$. As it is divisible by 8, the requirements are satisfied.

5 Extension to other curves and primitives

Our method can be extended to handle other curves with non-linear keyspace and other hashing primitives. For instance, the SHA-512/256 hash function can be used to produce the 256-bit chain code. The SHAKE hash functions [4] can be used to produce extended keys that are longer than 64 bytes, such as in the Ed448 curve [3] with 224-bit security level. Finally, the HMAC calls can be replaced by the very recent variable-length KMAC (currently standardized by NIST) so that the (Z, c) pair is produced in a single call.

References

[1] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[2] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. NaCl: Networking and cryptography library, 2013. `https://nacl.cr.yp.to/`.

[3] S. Josefsson and I. Liusvaara. Edwards-curve digital signature algorithm (EdDSA), 2016. `https://tools.ietf.org/html/draft-irtf-cfrg-eddsa-05`.

[4] NIST. SHA-3 standard: Permutation-based hash and extendable-output functions, 2015. `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf`.

[5] Pieter Wuille. BIP 32: Hierarchical deterministic wallets, 2012. `https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki`.