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

Dmitry Khovratovich
Evernym, Inc.
University of Luxembourg
khovratovich@gmail.com

Jason Law
Evernym, Inc.
jason.law@evernym.com

***Abstract*—We show how to adapt the Bitcoin BIP32 proposal for deterministic key generation for the Ed25519 curve which has non-linear key space. We also demonstrate that the alternative proposal by Chain.com is insecure and deviates from the EdDSA standard.**

I. Introduction

Bitcoin [6] users tend to use every public/private key pair infrequently to minimize private key leakage and privacy loss. It is is a common practice to generate a new key pair for each transaction, and a set of such key pairs is called a wallet.

The Bitcoin Improvement Proposal 32 (shortly, BIP32) [9] specifies the deterministic generation of wallet keys from a single seed to minimize the amount of secret information needed to operate with the wallet. Bitcoin uses elliptic curve signatures on curve secp256k1, where private keys form a linear space. Thanks to this fact, a new public key can be generated by adding[1] $[x]B$ to the public key $[k]B$, where B is the base point, k is the private key, and $[x]$ is scalar, so that the new private key scalar will be $[k]+[x]$. BIP32 provides a deterministic procedure to compute x as a function of some input parameters, and this way of wallet generation is widely used in Bitcoin clients.

Ed25519 is an elliptic curve standard for digital signatures developed in [2] and notable for high speed, constant-time implementations, and no requirement for randomness in signature generation. It is natural to try to adapt the BIP32 for Ed25519 so that architectures using Ed25519 could use the deterministic key generation (this is helpful when users, not necessarily of cryptocurrencies, would like to generate a number of keys/identities for themselves out of the same seed).

However, Ed25519 is also notable for its public key generation involving hashing and bit manipulations, which seemingly prevents its use for BIP32. We show that it is in fact possible with only minor modifications to BIP32 and working with "after-hash" private keys of Ed25519. Thus we maintaining almost complete compatibility to both standards, and a potential implementation can reuse most of the code.

Shortly before the submission of this paper, we were informed of another approach to the same problem by Chain.com [4]. Apparently, a more aggressive adaptation strategy made their specification not only incompatible to the Ed25519 signature generation, but also re-enabled a side-channel attack vector via timing leakage.

II. BIP32

The BIP32 standard [9] specifies not just a derivation of several keys for a wallet, but a hierarchy of keys. We start with a single private-public key pair and generate 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 public *child index*. Simultaneously, the private child key is derived from the private parent key using the same values. This allows the user to generate a new public key on an untrusted machine and use it at some transaction as a receiver, and compute the private key only when he decides to spend the transaction.

Full specification of BIP32 can be found in [9]. Here is the short summary, crucial to our paper:

- A base point B of order n is the Bitcoin base point on the elliptic curve secp256k1.
- The 256-bit *root private key* k and the *root chain code* c are generated as $(k,c) \leftarrow F_w(S)$, where F is HMAC-SHA512, the key w is a public string, and S is a seed. We compute the *root public key* $A \leftarrow [k]B$.
- The root has 2^{32} children, indexed from 0 to $2^{32}-1$. For each $i < 2^{31}$ we compute the 512-bit $Z \leftarrow F_c(A,i)$ and split it into the 256-bit parts $Z_L||Z_R$.
- The child private key k_i is $\langle [k]+[Z_L] \bmod n\rangle$, so that the corresponding public key A_i is $A+[Z_L]B$.
- A child may have children too, for which its chain code is $c_i \leftarrow Z_R$. Thus we have a tree of descendants from the original key pair.
- A child with $i \geq 2^{31}$ is called *hardened* as its private key is determined by parent private key: $Z \leftarrow F_c(0\text{x}00,k,i)$ and $k_i \leftarrow \langle [k]+[Z_L] \bmod n\rangle$. Such children do not have children.

[1] Here and later we denote strings by s, their interpretation as integers by $[s]$, and the representation of integer q as a string by $\langle q\rangle$.

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).

III. Ed25519

A. Setup

Ed25519 is described in [2] and is a part of an IETF draft in [5].

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

The base point B has a base order n of $2^{252} +$ 27742317777372353535851937790883648493. The infinity point is $(0, 0)$ and the identity point is $(0, 1)$.

B. Keys and signature

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

1) Let $H_{512}()$ be SHA512. 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[2], the highest bit of the last byte is cleared, and the second highest bit of the last byte is set[3].
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 $r \leftarrow r \bmod n$.
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 $S = (r + x \cdot [k_L]) \bmod n$.
5) The string $R||S$ is the signature.

IV. BIP32-Ed25519

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 III-B) to avoid a class of timing attacks and for compatibility with Curve25519 keys, where small-order-group attacks are relevant.

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 [5]. Signing and verifying procedures remain the same as in Section III-B. However, some Ed25519 libraries have a signing function that takes the 32-byte $\widetilde{k}$ and expands it for every signing, while other libraries sign using the extended 64-bit k. Due to the way child Ed25519 keys are derived, they will not have the 32-byte master secret key. We note that the NaCl library [3] provides the necessary interface to the extended private key.
2) We admit only those $\widetilde{k}$ such that the third highest bit of the last byte of k_L is zero.
3) We make another HMAC call to take a longer private key into account.
4) The string Z_L 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 child k_L is always 1.
5) The maximum number of levels in the tree is $2^{20} =$ 1048576.
6) Integers are serialized in the little-endian format (least significant bytes first), and byte strings are interpreted as little-endian integers.

V. 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 scheme is outlined at Figure 1.

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.

A. 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 k_L 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 $A \leftarrow [k_L]B$ is the root public key after encoding. Derive $c \leftarrow H_{256}(0\text{x}01||\widetilde{k})$, where H_{256} is SHA-256, and call it *the root chain code*.

B. Child keys

The 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

[2] This is done for key compatibility with Curve25519, where it serves as a countermeasure against low-order attack.

[3] This is done to avoid timing leakage in multiplication algorithms that search for highest active bit.

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. Recall that F_K stands for HMAC-SHA512 with key K.

C. Private child key

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

1)
$$\begin{cases} Z \leftarrow F_{c^P}(\text{0x02}||A^P||i), & i < 2^{31}; \\ Z \leftarrow F_{c^P}(\text{0x00}||k^P||i), & i \geq 2^{31}. \end{cases}$$
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)
$$k_L \leftarrow \langle 8[Z_L] + [k_L^P] \rangle, \tag{1}$$
$$k_R \leftarrow \langle [Z_R] + [k_R^P] \bmod 2^{256} \rangle. \tag{2}$$
where Z_L is the left 28-byte part of Z, and Z_R is the right 32-byte part of Z. If k_L is divisible by the base order n, discard the child.

3) The child chain code is defined as
$$\begin{cases} c_i \leftarrow F_{c^P}(\text{0x03}||A^P||i), & i < 2^{31}; \\ c_i \leftarrow F_{c^P}(\text{0x01}||k^P||i), & i \geq 2^{31}. \end{cases}$$
where the output of F is truncated to the right 32 bytes.

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

D. Public child key

Public key A_i for child i is produced as follows:

1)
$$\begin{cases} Z \leftarrow F_{c^P}(\text{0x02}||A^P||i), & i < 2^{31}; \\ Z \leftarrow F_{c^P}(\text{0x00}||k^P||i), & i \geq 2^{31}. \end{cases}$$
where A^P is serialized as little-endian 32-byte string, and i is serialized as little-endian 4-byte string. Note that like in the original BIP32 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)
$$A_i \leftarrow A^P + [8Z_L]B,$$
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
$$\begin{cases} c_i \leftarrow F_{c^P}(\text{0x03}||A^P||i), & i < 2^{31}; \\ c_i \leftarrow F_{c^P}(\text{0x01}||k^P||i), & i \geq 2^{31}. \end{cases}$$
where the output of the HMAC function truncated to the right 32 bytes.

E. 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 $m \rightarrow i_1 \rightarrow \ldots \rightarrow i_l$ from the original parent m to a child at level l thus uniquely identifies the node.

F. Security

a) 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.

b) 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.

VI. 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 [7] can be used to produce extended keys that are longer than 64 bytes, such as in the Ed448 curve [5] 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.

VII. Insecurity of the BIP32-Ed25519 proposal by Chain

Earlier in 2016 Chain.com introduced its own proposal on generating hierarchical deterministic keys on the Ed25519 curve [4]. Here we review it and identify weaknesses that lead to side-channel attacks.

A. Key generation

The root private key $k_L||k_R$ is generated exactly as described in Section III-B.

The root extended public key is $[k_L]B||k_R$, thus making k_R public (for some reason it is called *salt* in the proposal).

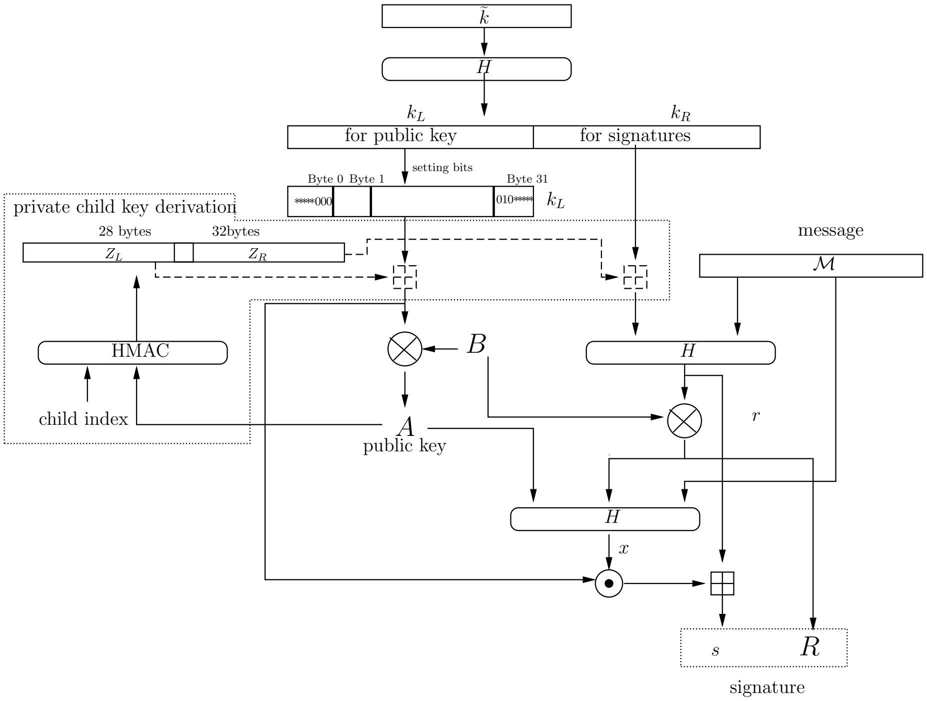

Fig. 1. BIP32-Ed25519.

B. Private child key

Extended private key $k = (k_L, k_R)$ is produced as follows:

$$\begin{cases} Z \leftarrow H_{512}(0\text{x}01||A^P||k_R^P||l(c)||c) & \text{for non-hardened keys.} \\ Z \leftarrow H_{512}(0\text{x}00||k_L^P||k_R^P||l(c)||c) & \text{for hardened keys.} \end{cases}$$

where A^P, k_R^P, k_L^P are serialized as little-endian 32-byte string, and c is a secret.

Then Z is divided into 256-bit parts Z_L and Z_R, and Z_L is modified as in Step 2, Section III-B. Then we set

$$[k_L] \leftarrow ([Z_L] + [k_L^P]) \bmod n; \quad k_R \leftarrow Z_R.$$

The public key are generated from this private key as $A = [k_L]B||k_R$.

C. Signatures

The signature generation is not compatible with Ed25519. Here is the modified version with the *difference in italic*:

1) *Compute* $H_{512}(0\text{x}02||k_L||k_R)$*, truncate it to the first 32 bytes and denote by* z*.*
2) Compute $H_{512}(z||M)$ and interpret the result as a little-endian-encoded integer r. Compute $r \leftarrow r \bmod n$.
3) Compute point $[r]B$ and let R be its encoding.
4) Compute $x \leftarrow H_{512}(R||A||M)$, and interpret the 64-byte digest as a little-endian integer.
5) Compute $S = (r + x \cdot [k_L]) \bmod n$.
6) The string $R||S$ is the signature.

Thus r is now dependent on k_L too, and the signature algorithm is different from that of Ed25519 (though similar). The signature verification procedure remains the same.

D. Weaknesses

We identify the following problems and weaknesses of the scheme.

1) The child private key is no longer a valid EdDSA extended private key. Since both Z_L and k_L^P have their highest bit cleared and the second highest bit set, the value $([Z_L] + [k_L^P])$ exceeds 2^{255} and thus will be reduced modulo n to a number smaller than n. The resulting string k_L has no guarantee on its highest bits set or cleared, thus contradicting the definition of the EdDSA keys.
2) The resulting extended private key is vulnerable to timing attacks on a bit-searching multiplication implementation [1]. Even if the value $([Z_L] + [k_L^P])$ is not reduced modulo n, there will be a guaranteed overflow in the second highest bit and thus the highest bit will be set to 1. This again contradicts the private key standard of Ed25519.
3) The signature procedure is no longer compatible with EdDSA. As a result, the signature generation code from existing libraries can not be fully reused.
4) The k_R string is exposed to public for unknown reason. We have not identified any attack vector, but the rationale for this design is unclear.
5) The authors write ”We believe the scheme is equivalent to RFC 6979 that derives the nonce by hashing the secret scalar. As an extra safety measure, the secret scalar is concatenated with the salt (which is not considered secret in this scheme) in order to make derivation

function not dependent solely on the key" [4]. We note that even though RFC 6979 [8] follows the EdDSA approach to deterministic signature generation, the salt is not involved. Moreover, RFC 6979 does not expose any part of the private key.

VIII. Acknowledgements

We thank the anonymous reviewers for fruitful comments that helped to improve the paper.

IX. Conclusion

We demonstrate how to adapt the hierarchical keys proposal from the Bitcoin curve to the Ed25519 curve by keeping all the code for the Ed25519 public key generation and signatures valid and compatible. Our approach can be simplified using a variable-length MAC. In turn, the approach by Chain.com is much harsher as it breaks compatibility with the signature procedure and introduces potential side-channel vulnerabilities.

References

[1] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[2] 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.

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

[4] Chain.com. Chain key derivation, 2016. https://chain.com/docs/protocol/specifications/chainkd.

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

[6] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009. http://www.bitcoin.org/bitcoin.pdf.

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

[8] Thomas Pornin. Rfc 6979: Deterministic usage of the digital signature algorithm (dsa) and elliptic curve digital signature algorithm (ecdsa), 2013. https://tools.ietf.org/html/rfc6979.

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