A Note on Hierarchical Deterministic Keys for EC-KCDSA on Curve25519

Haim Bender   Samuel Dobson   Lior Yaffe

August 7, 2019

1 Introduction

Cryptocurrencies using a transactional model similar to that in Bitcoin [6] make use of asymmetric key-pairs to prove ownership of specific funds. Sending funds from one user to another is done by creating a transaction to an "address" generated by the receiver. The address is, for example, the hash of a public key along with a checksum, encoded in some human-readable format such as Base58 or Bech32. Spending those funds requires a signature on the spending transaction from that public key, proving both ownership and integrity of the spend transaction.

For better privacy in such cryptocurrencies, it is usually encouraged that a new address be generated for every transaction received, as addresses generated by the same person are not linkable. Generating many standalone keys, though, creates logistical and security issues when creating backups and so forth. Originally, clients would generate some number of keys (100 by default in the Bitcoin reference client) in advance, and cache these for use when a new address was required. Unfortunately this still required a backup of the wallet each time new keys were generated, and if the wallet was encrypted, it would have to be unlocked to generate these new keys.

To solve these issues, Hierarchical Deterministic (HD) address creation was proposed in Bitcoin Improvement Proposal 32 (BIP-32) by Pieter Wuille [8]. From a single "master" extended key-pair, any number of new key-pairs can be generated deterministically, and all these keys can be restored on any device from a backup of the master key alone (assuming the keys are generated according to the standard proposed in the BIP). In addition, it is possible to generate the public keys of these key-pairs even without access to the private keys, allowing generation of new addresses even when the wallet is encrypted or the private keys are stored on a separate device.

Because BIP-32 was designed for use in Bitcoin, it uses keys generated on the elliptic curve **secp256k1** (defined in [3]), with signature generation according to ECDSA on that curve. secp256k1 has a linear private-key space, meaning that the sum of two private keys corresponds to the public key calculated as the sum of the two respective public keys (as points on the curve). This property is used in the design of BIP-32. Unfortunately, this is not the case in Ed25519 [2], which uses a twisted Edwards curve bi-rationally equivalent to the Montgomery curve Curve25519 [1]. Ed25519 has a non-linear private-key space, because the generation of public key from private key involves hashing the private key with SHA-512. To overcome this, a HD wallet derivation scheme for Ed25519 was proposed by Khovratovich and Law [4].

In this note, we present a similar HD scheme for the signature algorithm EC-KCDSA, over Curve25519. We make use of the Ed25519 derivation system as above, because it has already been implemented on a number of platforms. We then show how to convert the keys generated by this scheme into keys compatible with EC-KCDSA.

2 Ed25519

Ed25519 is described in [2]. It uses the twisted Edwards curve E

$$-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2 \tag{1}$$

over the finite field $\mathbb{F}_q$ where $q = 2^{255} - 19$ is a prime.
This curve is bi-rationally equivalent to the Montgomery curve Curve25519 over the same field, which was introduced by Bernstein in 2006 [1], with the equivalence given by

$$\begin{aligned} x &= \frac{u}{v}\sqrt{486664} \\ y &= \frac{u-1}{u+1} \end{aligned} \tag{2}$$

The generator point $G \in E(\mathbb{F}_q)$ on this curve is defined to be the unique point on the curve where $y = 4/5$ and x is positive. G has order $\ell = 2^{252} + 27742317777372353535851937790883648493$, and cofactor $c = 3$ such that $\#E(\mathbb{F}_q) = 2^c\ell$. Let H denote the hash function SHA-512.

Key-pair generation A private key in Ed25519 is a 256-bit string k chosen uniformly at random. Let $s = \text{LeastSignificant32Bytes}(H(k))$, where LeastSignificant32Bytes interprets the bytes as a little-endian encoded integer. Modify s by clearing the lowest 3 bits, clearing the highest bit of the last byte, and setting the second highest bit of the last byte. We then define the Ed25519 public key as $A = sG$.

Signing Using private key k, we sign a message M in the following way. Let Let $r = H(\text{MostSignificant32Bytes}(H(k)), M)$, where MostSignificant32Bytes interprets the bytes as a little-endian encoded integer. Let $R = rG$ and $S \equiv r + H(R, A, M)s \pmod{\ell}$. Then (R, S) is the signature.

Verifying To verify a signature (R, S) (where $0 < S < \ell$) on a message M with public key A, we check that $2^c SG = 2^c R + 2^c H(R, A, M)A$. This holds because

$$\begin{aligned} 2^c SG &= 2^c(r + H(R, A, M)s)G \\ &= 2^c rG + 2^c H(R, A, M)sG \\ &= 2^c R + 2^c H(R, A, M)A \end{aligned}$$

3 EC-KCDSA

EC-KCDSA is the Elliptic Curve variant of the Korean Certificate-based Digital Signature Algorithm [7, 5]. In this signature scheme, the signer has some certification data denoted Cert_Data, which contains at least a distinguished identifier of the signer, the signer's public key, and the domain parameters of the algorithm (for example, the curve). Here we consider EC-KCDSA using the elliptic curve Curve25519. Let $q = 2^{255} - 19$ as above, and let $E(\mathbb{F}_q)$ be Curve25519, with equation

$$y^2 = x^3 + 486662x^2 + x \tag{3}$$

over $\mathbb{F}_q$. Points on this curve are specified with only x coordinates (that is, compressed elliptic curve points). On this curve, the generator point G is such that $x = 9$, and has order ℓ as on the twisted Edwards curve above (the generator point on the twisted Edwards curve is equivalent to the generator point on Curve25519 by the bi-rational equivalence given above). Let H here denote the hash function SHA-256. The hash of the signer's certification data is denoted $z = H(\text{Cert\_Data})$.

Key-pair generation An EC-KCDSA private key in Curve25519 is a uniformly-randomly chosen integer $x \in \mathbb{Z}_\ell^*$. Let $\overline{x} \equiv x^{-1} \pmod{\ell}$. Then the public key is $A = \overline{x}G$.

Signing Using private key x to sign a message M, we first choose an ephemeral $k \in \mathbb{Z}_\ell^*$ uniformly at random. Then we compute $r = H(kG)$. Finally, let $s = x(k - r \oplus H(z, M)) \bmod \ell$. The signature is (r, s).

Verifying First, the verifier will check the validity of the signers certification data Cert_Data, and compute z. They should then verify that R and s are of the correct size and form. The verifier computes $e = r \oplus H(z, M) \pmod{\ell}$ and finally checks that $r = H(sY - eG)$. This holds because

$$\begin{aligned} sA - eG &= x(k - r \oplus H(z, M))A - eG \\ &= xkA - xeA - eG \\ &= kG \end{aligned}$$

because $xA = xx^{-1}G \equiv G \pmod{\ell}$.

4 Yaffe-Bender HD key derivation for EC-KCDSA

We now present the main result, a method for generating EC-KCDSA key-pairs on Curve25519 using the BIP32-Ed25519 HD derivation scheme directly, developed by Yaffe and Bender. Suppose we generate a key-pair $\{\text{PrivateEdDSA}, \text{PublicEdDSA}\}$ using this scheme. Then let

$$\begin{aligned} k_L &= \text{LeastSignificant32Bytes(SHA-512(PrivateEdDSA))} \\ k'_L &= \text{Clear lowest 3 bits of } k_L,\ \text{clear highest bit of the last byte,} \\ &\quad \text{set second highest bit of the last byte} \\ x &= (k'_L)^{-1} \pmod{\ell} \\ A &= \text{Convert\_ED25519\_to\_Curve25519\_public\_key(PublicEdDSA)} \end{aligned}$$

Return x as the private key and A as the public key.

This, in effect, combines the hashing and bit-modification stages of the public key generation in EdDSA directly into the private key. We must then invert it to ensure the correct relationship holds betwen x and A, because in EC-KCDSA, the private key is inverted to generate the public key.

Because the curve used by Ed25519 is birationally equivalent to Curve25519, we can directly convert the public key (point on the twisted Edwards curve) to a point on Curve25519. Points on Curve25519 are encoded with their x coordinate only, as discussed above, because the Montgomery ladder uses only x coordinates in its computation.

We now remark on the fact that the derivation of the EdDSA public key from the randomly selected private key involves setting and clearing certain bits. The clearing of the lowest 3 bits is done to ensure the key is a multiple of 8, preventing small-subgroup attacks, while the highest and second highest bits are fixed in order to prevent timing attacks on Curve25519. Other than these modifications, the distribution of the private keys x should be indistinguishable from uniformly random and suitable for use with EC-KCDSA.

References

[1] D. J. Bernstein. Curve25519: New diffie-hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography - PKC 2006*, pages 207–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[2] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, Sep 2012.

[3] Certicom Research. Sec 2: Recommended elliptic curve domain parameters, Jan. 2010. `https://www.secg.org/sec2-v2.pdf`.

[4] D. Khovratovich and J. Law. BIP32-Ed25519: Hierarchical deterministic keys over a non-linear keyspace. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pages 27–31, April 2017.

[5] C. H. Lim and P. J. Lee. A study on the proposed korean digital signature algorithm. In K. Ohta and D. Pei, editors, *Advances in Cryptology — ASIACRYPT'98*, pages 175–186, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[6] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `http://bitcoin.org/bitcoin.pdf`, 2008.

[7] K. T. F. Team. The korean certificate-based digital signature algorithm, 1998.

[8] P. Wuille. BIP 32: Hierarchical deterministic wallets. `https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki`. Accessed: 2019-05-12.