XEdDSA and VXEdDSA

Content

1. Introduction.

This document describes Signal Protocol's XEdDSA and VXEdDSA digital signature schemes and their specific elliptic curves Curve25519 and Curve448.

XEdDSA enables use of a single key pair format for both elliptic curve Diffie-Hellman and signatures. In some situations it enables using the same key pair for both algorithms.

VXEdDSA is an extension of XEdDSA to make it a *verifiable random function* (or VRF). Successful verification of a VXEdDSA signature returns a VRF output which is guaranteed to be unique for the message and public key. The VRF output for a given message and public key is indistinguishable from random to anyone who has not seen a VXEdDSA signature for that message and key.

2. Overview.

2.1. Elliptic curve parameters and the basic maths behind them.

In mathematics elliptic curves are plane algebraic curves, consisting of all points {x, y}, described by the equation:

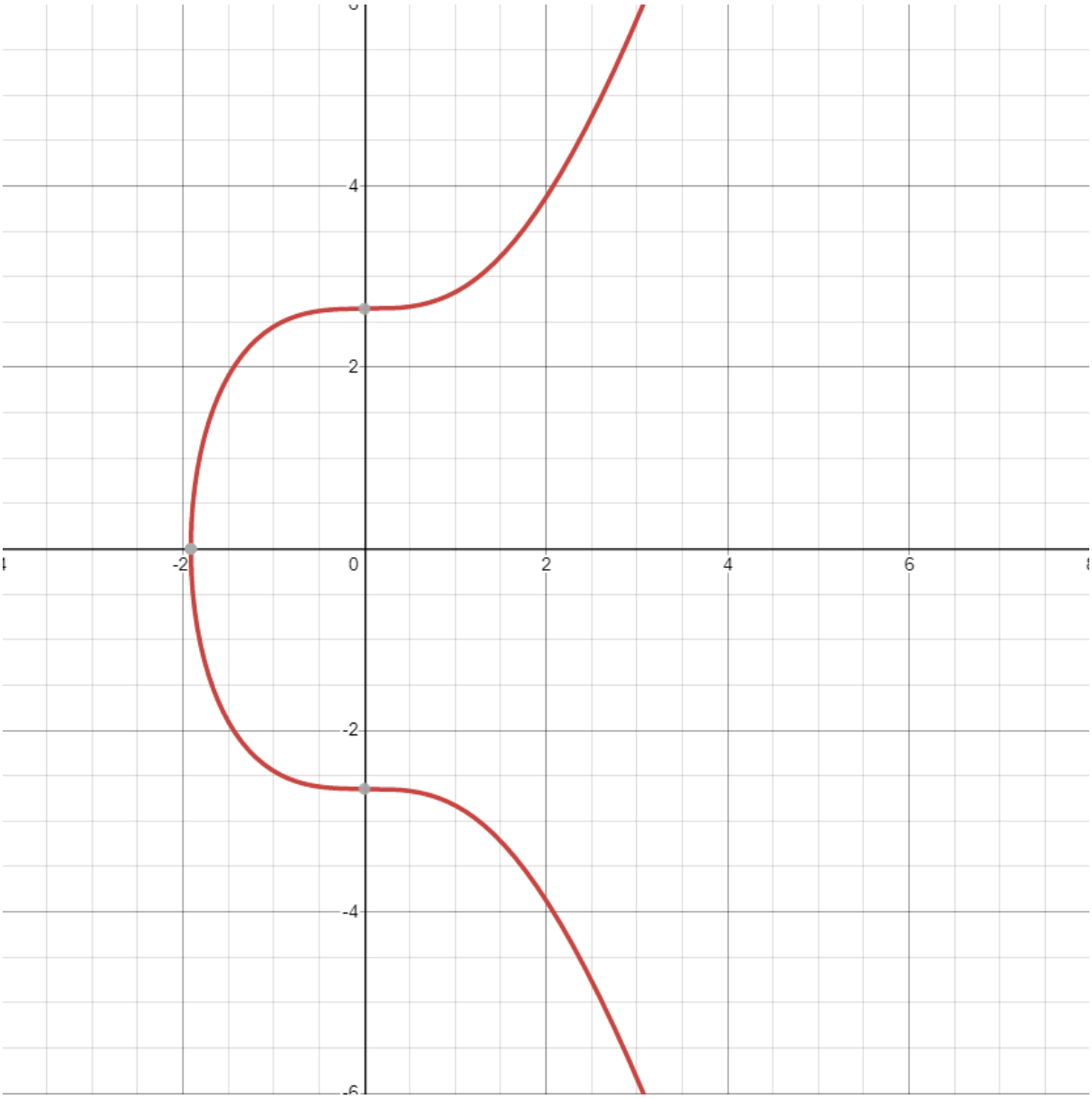
$$Ax^3 + Bx^2y + Cxy^2 + Dy^3 + Ex^2 + Fxy + Gy^2 + Hx + Iy + J = 0$$

In Cryptography, elliptic curves are described in Weierstrass form:

$$y^2 = x^3 + ax + b$$

For example the curve secp256k1 used by BitCoin has the form:

$$y^2 = x^3 + 7$$



The math above describe elliptic curve over infinite field of real number, while **Elliptic Curve Cryptography (ECC)** deals with elliptic curve over finite field $\mathbb{F}_p$.

An elliptic curve used with XEdDSA or VXEdDSDA has the following parameters:

Name	Definition
B	Base point
I	Identity point
p	Field prime
q	Order of base point

c	Cofactor
d	Twisted Edwards curve constant
A	Montgomery curve constant
n	Nonsquare integer modulo p
|p|	ceil(log2(p))
|q|	ceil(log2(q))
b	8 * (ceil((|p| + 1)/8))

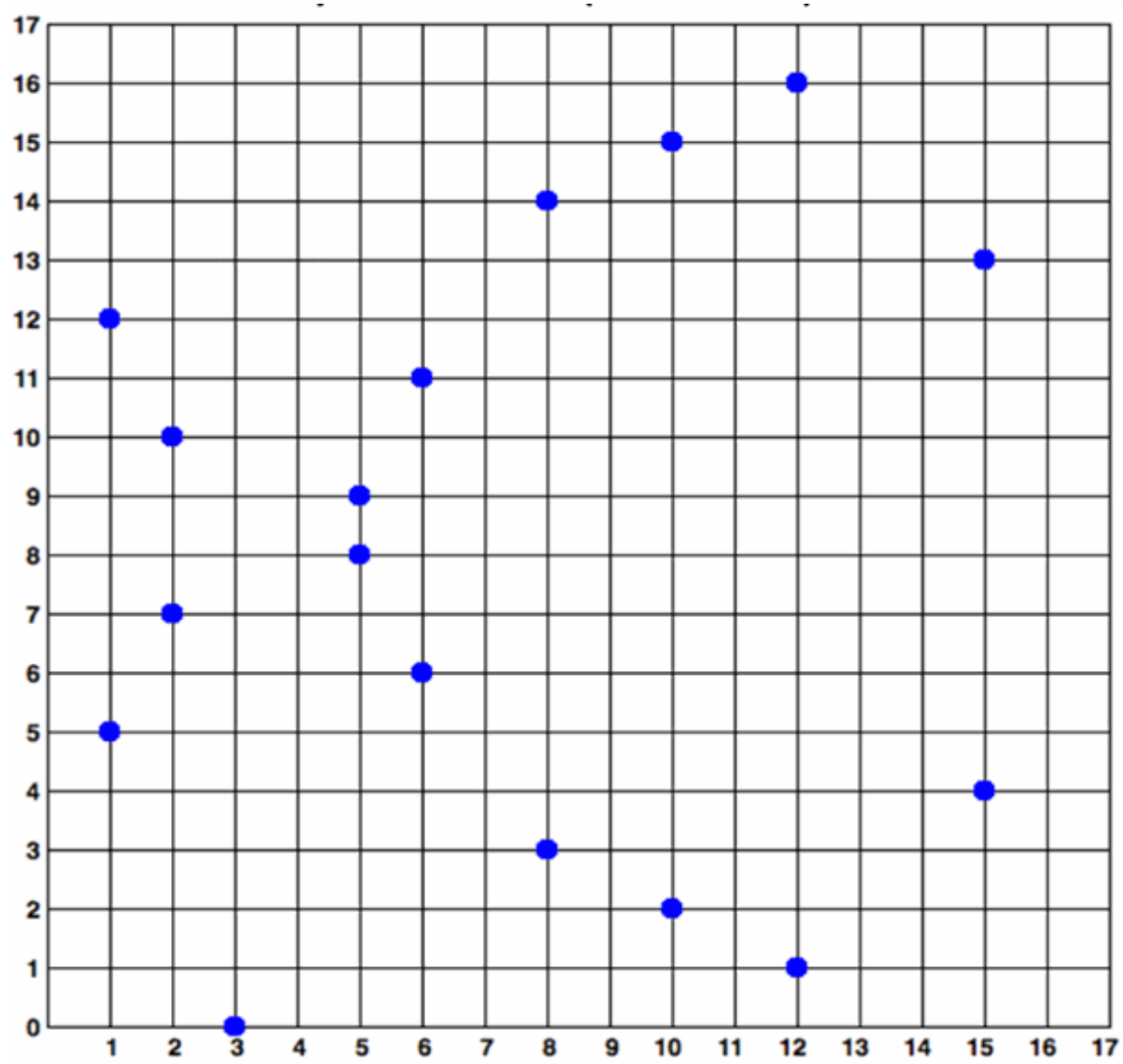
IMPORTANT PARAMETER 1: field prime p

An elliptic curve over finite field $\mathbb{F}_p$ is a set of points satisfied the equation:

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

p is called the **field prime.**

Example: the elliptic curve $y^2 \equiv x^3 + 7 \pmod{17}$ over the finite prime field $\mathbb{F}_{17}$:



IMPORTANT PARAMETER 2, 3 and 4: identity point I, order q and cofactor c

Elliptic curve is a group structure, so any addition between two points or scalar multiplication on a point of elliptic curve over finite field F_p will result in another point in F_p. Multiply a point with zero results in a special elliptic curve point is a **point at infinity** or **identity point,** called I.

Some curves form a single **cyclic group** of order n, or c non-overlapping **cyclic subgroup**, each of order q (the number of points in each subgroup), the order of the entire groups is $n \equiv c * q$, where c is the number of cyclic subgroup, or the **cofactor**

Note that order q of a certain elliptic curve's subgroup is the total number of all possible private keys for this curve.

IMPORTANT PARAMETER 5: generator point or base point $\mathbb{B}$

ECC cryptosystems define a special (constant) elliptic curve point called generator point $\mathbb{B}$, or **base point**, which can generate any other point in its subgroup over the elliptic curve by perform scalar multiplication between $\mathbb{B}$ and an integer in the range $[0, q]$.

IMPORTANT PARAMETER 6 and 7: constant d of twisted Edwards curve and constant A of Montgomery curve

XEdDSA and VXEdDSA are defined for twisted Edwards curves consisting of points denoted (x, y). A twisted Edwards curve is birationally equivalent to some Montgomery curve consisting of points denoted (u, v), these two curves is described by the following equations

Twisted Edwards curve: $ax^2 + y^2 = 1 + dx^2y^2$

Montgomery curve: $By^2 = x^3 + Ax^2 + x$

When we discuss the base point $\mathbb{B}$ and identity point , we are referring to points on the twisted Edwards curve.

IMPORTANT PARAMETER 8: fixed non square constant integer $n \bmod p$ for Elligator2 (see 2.2.4)

IMPORTANT PARAMETER 9 and 10: $|p|$ and $|q|$

$|p|$ and $|q|$ are defined as:

$|p| = ceil(log_2(p))$

$|q| = ceil(log_2(q))$

IMPORTANT PARAMETER 11: bitlength b

XEdDSA and VXEdDSA ECC has a fixed bitlength b for encoding point or integer, defined as:

$b = 8 * (ceil((|p| + 1)/8))$

FROM THIS POINT ONWARD, BOLD AND NON-HEADERS CHARACTERS DENOTE BYTES SEQUENCE.

2.2. Basic functions in XEdDSA and VXEdDSA.

2.2.1. Elliptic curve conversion.

Elliptic curve Diffie-Hellman is calculated using the Montgomery ladder. This algorithm is simple and naturally resistant to timing side channels attack. The Montgomery ladder also allows each party's public key to be a Montgomery u-coordinate. Using a single coordinate instead of the whole point allows for smaller public keys.

However, EdDSa signatures are defined on twisted Edwards curve, as a compressed point of twisted Edwards y-coordinate and a sign bit s. This is an alternate representation of a proper twisted Edward point and determine the x-coordinate as specified in [1] and [2]. Luckily, a twisted Edwards curve is birationally equivalent to some Montgomery curve, so it is possible to convert a Montgomery point to a twisted Edwards point.

Function $on\_curve(P)$ returns true if *P* if the point *P* satisfies the curve equation.

Function $u\_to\_y(u)$ applies a curve-specific birational map to convert the u-coordinate of a point on the Montgomery curve to the y-coordinate of the equivalent point on the twisted Edwards curve.

Function $convert\_mont(u)$ converts a Montgomery u-coordinate to a twisted Edwards point P

```
convert_mont(u):
  u_masked = u (mod 2|p|) # mask the excessive high bits of u, standard
                          # practice for Curve25519 Montgomery public keys,
                          # and is specified in [4]
  P.y = u_to_y(u_masked) # convert u to twisted Edward's y
  P.s = 0 # set sign bit to 0
  return P
```

2.2.2. Calculate key pair.

To make private keys compatible with the conversion in 2.2.1, a twisted Edwards private key is defined as a scalar a where the twisted Edwards public key $A = aB$. A Montgomery private key can be any scalar.

Function $calculate\_key\_pair(k)$ converts a Montgomery private key k to a pair of twisted Edwards public key and private key (A, a) (Note that A here is the public key, not the Montgomery curve constant). then adjusts the private key if necessary to produce a sign bit of zero.

```
calculate_key_pair(k):
  E = kB # multiply the Montgomery private key k by the twisted Edwards base point B
  A.y = E.y # set y-coordinate of twisted Edwards public key
  A.s = 0 # set sign bit of twisted Edwards public key
  if E.s == 1: # force a sign bit of 0 for twisted Edwards private key
    a = -k (mod q)
  else:
    a = k (mod q)
  return A, a
```

2.2.3. Bytes sequences.

In this document, the concatenation of byte sequences $\boldsymbol{x}$ and $\boldsymbol{P}$ is $\boldsymbol{x} \| \boldsymbol{P}$.

Checking byte sequences $\boldsymbol{X}$ and $\boldsymbol{Y}$ for equality is done with $bytes\_equal(\boldsymbol{X}, \boldsymbol{Y})$.

An integer in bold represents a byte sequence of b bits encoding that integer in little-endian form. An elliptic curve point in bold (e.g. $\boldsymbol{P}$) encodes $\boldsymbol{P}.y$ as an integer in little-endian form of $b - 1$ bits in length, followed by a bit for $\boldsymbol{P}.s$.

2.2.4. Hash functions.

XEdDSA and VXEdDSA require a cryptographic hash function. The default hash function is SHA-512 [3]. (**NOTE: dùng hash nào để sửa thẳng vào đây)**

$hash$ is defined as a function that applies the cryptographic hash to an input byte sequence, and returns an integer which is the output from the cryptographic hash parsed in little-endian form.

Function $hash_i(\boldsymbol{X})$ is indexed by non-negative integer i such that $2^{|p|} - 1 - i > p$.

```
hash_i(X):
  return hash(2^b - 1 - i || X)
```

Different $hash_i$ will be used for different purposes, to provide cryptographic domain separation. Note that $hash_i$ will never call hash with the first b bits encoding a valid scalar or elliptic curve point, since the first $|p|$ bits encode an integer greater than p.

Note also that $hash_0$ is reserved for use by other specifications, and is not used in this document.

2.2.5. Hashing a bytes sequences to an elliptic curve point with Elligator2 .

VXEdDSA requires mapping an input message to an elliptic curve point, which is done using the Elligator2 map [5].

Lemma (Bernstein, Hamburg, Krasnova, Lange): If u_1 and u_2 are integers modulo p such that $u_2 = -A - u_1$ and $u_2/u_1 = nr^2$ for any r and fixed nonsquare n, then the Montgomery curve equation $v^2 = u(u^2 + Au + 1)$ has a solution for $u = u_1$ or $u = u_2$, or both.

From the lemma it follows that $u_1 = -A/(1 + nr^2)$ and $u_2 = -Anr^2/(1 + nr2)$. So given r, we can easily calculate u_1 and u_2 and use the Legendre symbol [6] to choose whichever value gives a square w.

Function $elligator2(u)$ implements this map from an integer r to an integer u.

```
elligator2(r):
  u1 = -A * inv(1 + nr^2) (mod p)
  w1 = u1(u1^2 + A*u1 + 1) (mod p)
  if w1^((p-1)/2) == -1 (mod p):
    u2 = -A - u1 (mod p)
    return u2
  return u1
```

Function $hash\_to\_point(\boldsymbol{X})$ map a bytes sequences to an Edwards points.

```
hash_to_point(X):
  h = hash_2(X)
  r = h (mod 2^(|p|))
  s = floor((h mod 2^b) / 2^(b-1))
  u = elligator2(r)
  P.y = u_to_y(u)
  P.s = s
  return cP
```

3. XEdDSA.

The XEdDSA signing algorithm requires the following inputs:

Name	Definition
k	Montgomery private key (integer mod q)
M	Message to sign (byte sequence)
Z	64 bytes secure random data (byte sequence)

The output is a signature $(R||s)$ where a byte sequence of length $2b$ bits, where R encodes a point and s encodes an integer modulo q.

```
xeddsa_sign(k, M, Z):
  A, a = calculate_key_pair(k)
  r = hash_1(a || M || Z) (mod q)
  R = rB
  h = hash(R || A || M) (mod q)
  s = r + ha (mod q)
  return R || s
```

The XEdDSA verification algorithm requires the following inputs:

Name	Definition
u	Montgomery public key (byte sequence of b bits)
M	Message to sign (byte sequence)
R || **s**	Signature to verify (byte sequence of 2b bits

If XEdDSA verification is successful it returns true, otherwise it returns false.

```
xeddsa_verify(u, M, (R || s)):
if u >= p or R.y >= 2^(|p|) or s >= 2^(|q|):
  return false
A = convert_mont(u)
if not on_curve(A):
  return false
h = hash(R || A || M) (mod q)
R_check = s*B - h*A
if bytes_equal(R, R_check):
  return true
return false
```

4. VXEdDSA.

VXEdDSA signing algorithm takes the same input as XEdDSA.

The output is a pair of values. First, a signature $(\boldsymbol{V}||\boldsymbol{h}||\boldsymbol{s})$, which is a byte sequence of length $3b$ bits, where $\boldsymbol{V}$ encodes a point and $\boldsymbol{h}$ and $\boldsymbol{s}$ encode integers modulo q. Second, a VRF output byte sequence y of length b bits, formed by multiplying the $\boldsymbol{V}$ output by the cofactor c.

```
vxeddsa_sign(k, M, Z):
  A, a = calculate_key_pair(k)
  B_v = hash_to_point(A || M)
  V = a*B_v
  r = hash_3(a || V || Z) (mod q)
  R = r*B
  R_v = r*B_v
  h = hash_4(A || V || R || Rv || M) (mod q)
  s = r + h*a (mod q)
  v = hash_5(c*V) (mod 2^b)
  return (V || h || s), v
```

VXEdDSA verification algorithm takes the same inputs as XEdDSA, except with a VXEdDSA signature instead of an XEdDSA signature.

If VXEdDSA verification is successful, it returns a VRF output byte sequence y of length
equal to b bits; otherwise it returns false.

```
vxeddsa_verify(u, M, (V || h || s)):
  if u >= p or V.y >= 2^(|p|) or h >= 2^(|q|) or s >= 2^(|q|):
    return false
  A = convert_mont(u)
  B_v = hash_to_point(A || M)
  if not on_curve(A) or not on_curve(V):
    return false
  if c*A == I or c*V == I or B*v == I:
    return false
  R = s*B - h*A
  R_v = s*B_v - h*V
  h_check = hash_4(A || V || R || R_v || M) (mod q)
  if bytes_equal(h, h_check):
    v = hash_5(c*V) (mod 2^b)
    return v
  return false
```

5. Curve25519.

The Curve25519 elliptic curve specified in [4] can be used with XEdDSA and VXEdDSA, giving XEd25519 and VXEd25519. This curve defines the following parameters:

Name	Definition
B	convert_mont(9)
I	(x=0, y=1)
p	2^255 - 19
q	2^252 + 27742317777372353535851937790883648493
c	8
d	-121665 / 121666 (mod p)
A	486662
n	2
|p|	255
|q|	253
b	256

The u_to_y function implements the birational map from [4] by calculating:

$$y = (u - 1) * inv(u + 1)(mod\ p)$$

XEd25519 signatures are valid Ed25519 signatures [1] and vice versa, provided the public keys are converted with the birational map.

6. Curve448.

The Curve448elliptic curve specified in [4] can be used with XEdDSA and VXEdDSA, giving XEd448 and VXEd448. This curve defines the following parameters:

Name	Definition
B	convert_mont(5)
I	(x=0, y=1)
p	2^448 - 2^224 - 1
q	2^446 - 138180668098951153520073867485154268803366924748821786098945475038 85

c	4
d	39082 / 39081 (mod p)
A	1563261
n	1
|p|	448
|q|	446
b	456

The u_to_y function implements the birational map from [4] by calculating:

$$y = (u - 1) * inv(u + 1)(mod\, p)$$

XEd25519 signatures are valid Ed25519 signatures [1] and vice versa, provided the public keys are converted with the birational map.

XEd448 differs from EdDSA [2] in choice of hash function. XEd448 uses SHA-512, whereas [2] recommends a 912-bit hash (912 = 2b). If the hash function is secure, outputs larger than 512 bits don't add security with Curve448, so XEd448 makes a simpler choice.

7. Performance considerations.

This section contains an incomplete list of performance considerations.

Signing time: Calling calculate_key_pair for every XEdDSA signature roughly doubles signing time compared to EdDSA, because of an additional scalar multiplication $E = kB$. To avoid this cost signers may cache the (non-secret) point E.

Pre-hashing: Except for XEdDSA verification, the signing and verification algorithms hash the input message twice. For large messages this could be expensive, and would require either large buffers or more complicated APIs. To prevent this, APIs may wish to specify a maximum message size that all implementations must be capable of buffering. Protocol designers can specify "pre-hashing" of message fields to fit within this. Designers are encouraged to use pre-hashing selectively, so as to limit the potential impact from collision attacks.

8. Security considerations.

This section contains an incomplete list of security considerations.

Random secret inputs: XEdDSA and VXEdDSA signatures are randomized, they are not deterministic in the sense of [1] or [7]. The caller must pass in a new secret and random 64 byte value each time the signing function is called.

Constant time: The signing algorithms must not perform different memory accesses or take different amounts of time depending on secret information. This is typically achieved by "constant time" implementations that execute a fixed sequence of instructions and memory accesses, regardless of secret keys or message contents. Particular care should be taken with the $calculate\_key\_pair$ and $hash\_to\_point$ (within $elligator2$) functions due to their use of conditional branching.

Key reuse: It is safe to use the same key pair to produce XEdDSA and VXEdDSA signatures.
In theory, under some circumstances it is safe to use a key pair to produce signatures and also use the same key pair within certain Diffie-Hellman based protocols [8]. In practice this is a complicated topic requiring careful analysis, and is outside the scope of the current document.

9. Reference.

[1] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," Journal of Cryptographic Engineering, vol. 2, no. 2, 2012. https://ed25519.cr.yp.to/ed25519-20110705.pdf

[2] D. J. Bernstein, S. Josefsson, T. Lange, P. Schwabe, and B.-Y. Yang, "EdDSA for more curves." Cryptology ePrint Archive, Report 2015/677, 2015. http://eprint.iacr.org/2015/677

[3] NIST, "FIPS 180-4. Secure Hash Standard (SHS)," National Institute of Standards & Technology, Gaithersburg, MD, United States, 2012. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[4] A. Langley, M. Hamburg, and S. Turner, "Elliptic Curves for Security." Internet Engineering Task Force; RFC 7748 (Informational); IETF, Jan-2016. http://www.ietf.org/rfc/rfc7748.txt

[5] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange, "Elligator: Elliptic-curve points indistinguishable from uniform random strings." Cryptology ePrint Archive, Report

2013/325, 2013. http://eprint.iacr.org/2013/325

[6] Wikipedia, “Legendre symbol — Wikipedia, The Free Encyclopedia.” 2016. https://en.wikipedia.org/w/index.php?title=Legendre_symbol

[7] T. Pornin, “Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA).” Internet Engineering Task Force; RFC 6979 (Informational); IETF, Aug-2013. http://www.ietf.org/rfc/rfc6979.txt

[8] J. P. Degabriele, A. Lehmann, K. G. Paterson, N. P. Smart, and M. Strefler, “On the Joint Security of Encryption and Signature in EMV.” Cryptology ePrint Archive, Report 2011/615, 2011. http://eprint.iacr.org/2011/615

[9] Svetlin Nakov, Practical Cryptography for developers, Elliptic Curve Cryptography (ECC) https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc#the-generator-point-in-ecc

Note (For senpai Khai)

Đoạn code thì cứ để pseudocode hay viết code thật của mình vào?

Mình dùng thuật toán/curve nào có phải thì xóa cái còn lại không?