Schnorr$\mathbb{Q}$: Schnorr signatures on Four$\mathbb{Q}$

Craig Costello and Patrick Longa

Microsoft Research, USA

Schnorr$\mathbb{Q}$ is a digital signature scheme that is based on the well-known Schnorr signature scheme [6] combined with the use of the elliptic curve Four$\mathbb{Q}$ [3].

1 Rationale

Schnorr$\mathbb{Q}$ offers extremely fast, high-security digital signatures targeting the 128-bit security level. It was designed by instantiating (with minor modifications) the recent EdDSA [1] digital signature specifications [2,5] on a superior, state-of-the-art elliptic curve, Four$\mathbb{Q}$ [3]. Similar to Ed25519 [1], public keys are 32 bytes and signatures are 64 bytes.

2 Parameters

EdDSA has 11 parameters (see [2,5]). Below we specify the 11 parameters used to instantiate EdDSA on Four$\mathbb{Q}$, where we use an asterisk ($*$) to indicate that the specification differs from the requirement(s) in [2,5].

1. An odd prime power q.

$$q = p^2 \text{ with } p = 2^{127} - 1.$$

2. An integer b with $2^{b-1} > q$.

$$b = 256.$$

3. A $(b-1)$-bit encoding of the finite field $\mathbb{F}_q$.

   Here $\mathbb{F}_q = \mathbb{F}_{p^2} = \mathbb{F}_p(i)$ with $i^2 = -1$. Elements $x \in \mathbb{F}_q$ are written as $x = a + b \cdot i$ for $a, b \in \{0, 1, \ldots, 2^{127} - 1\}$, i.e., for $a = \sum_{i=0}^{126} a_i \cdot 2^i$ and $b = \sum_{i=0}^{126} b_i \cdot 2^i$ with $a_i, b_i \in \{0, 1\}$. The 255-bit encoding of $x \in \mathbb{F}_q$ is

$$\underline{x} = (a_0, a_1, \ldots, a_{126}, 0, b_0, b_1, \ldots b_{126}).$$

4. A cryptographic, collision-resistant hash function H producing $2b$-bit output.

5*. An integer $c \in \{2, 3\}$.

   Schnorr$\mathbb{Q}$ uses the stronger "cofactorless verification" equation [2], so the cofactor is irrelevant here. EdDSA specifies that secret keys are multiples of 2^c, and since Schnorr$\mathbb{Q}$ does not require this, here we implicitly have $c = 0$.

6*. An integer n with $c \leq n \leq b$.

Secret EdDSA scalars have exactly $n + 1$ bits, with the top bit always set and the bottom c bits always cleared. Schnorr$\mathbb{Q}$ secret scalars are all 256-bit strings, i.e., can be any of $\{0, 1, \ldots 2^{256} - 1\}$. Thus, we implicitly have $n = 255$, but note that the top bit of Schnorr$\mathbb{Q}$ secret scalars is not necessarily set.

7. A nonzero square element a of $\mathbb{F}_q$.

$$a = -1,$$

which is optimal in terms of performance when $q \equiv 1 (\mathrm{mod} 4)$.

8. A non-square element d of $\mathbb{F}_q$.

$$\begin{aligned} d &= d_a + d_b \cdot i; \\ d_a &= 4205857648805777768770; \\ d_b &= 125317048443780598345676279555970305165. \end{aligned}$$

9. An element $B \neq (0, 1)$ of the set $E = \{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q \colon ax^2 + y^2 = 1 + dx^2y^2\}$.

$$\begin{aligned} B &= (x_a + x_b \cdot i, y_a + y_b \cdot i)\,; \\ x_a &= 133173070547236760532149241662440243363; \\ x_b &= 72544766618652889802729346394492014752; \\ y_a &= 465; \\ y_b &= 0. \end{aligned}$$

10*. An odd prime ℓ such that $\ell B = 0$ and $2^c \cdot \ell = \#E$.

Here the 246-bit prime

$$\ell := 73846995687063900142583536357581573884798075859800097461294096333596429543$$

is such that $\ell B = 0$, but note that Four$\mathbb{Q}$ has $\#E = 2^3 \cdot 7^2 \cdot \ell$. The cofactor $2^3 \cdot 7^2$ is irrelevant in the cofactorless verification equation used in Schnorr$\mathbb{Q}$.

11. A "prehash" function H'.

Schnorr$\mathbb{Q}$ without prehashing means Schnorr$\mathbb{Q}$ where H' is the identity function, i.e., $H'(M) = M$. Schnorr$\mathbb{Q}$ with prehashing means Schnorr$\mathbb{Q}$ where H' generates a short output for a message of any length using a collision-resistant hash function; for example, $H'(M) = \text{SHA-512}(M)$. In this document, we refer to Schnorr$\mathbb{Q}$ without prehashing as simply "Schnorr$\mathbb{Q}$" and refer to Schnorr$\mathbb{Q}$ with prehashing as "Schnorr$\mathbb{Q}$ph".

Prehashing. As is described in [5] for the two analogous EdDSA options, choosing between Schnorr$\mathbb{Q}$ and Schnorr$\mathbb{Q}$ph depends on which feature is more important for a given application: collision resistance or a single-pass interface for generating signatures. Schnorr$\mathbb{Q}$ is resilient to collisions in the hash function but requires two passes over the input message to generate a signature, whereas Schnorr$\mathbb{Q}$ph is not resilient to collisions in the hash function H' but supports interfaces that perform a single pass over the input message to generate a signature. Refer to [2,5] for more details about the security of prehashing.

Encoding and parsing integers. The integer $S \in \{0, 1, \ldots, \ell - 1\}$ below is encoded in little-endian form as a 256-bit string $\underline{S}$. The bit string $\underline{S} = (S_0, S_1, \ldots, S_{255})$ is parsed to the integer $S = S_0 + 2S_1 + \cdots + 2^{255} S_{255}$.

Encoding and parsing curve points. An element $x = a + b \cdot i \in \mathbb{F}_q$ encoded as $\underline{x} = (a_0, \ldots, a_{126}, 0, b_0, \ldots b_{126})$ is defined as "negative" if only if $a_{126} = 1$ and $a \neq 0$, or if $b_{126} = 1$ and $a = 0$. The point $(x, y) \in E$ is encoded as the 256-bit string $\underline{(x, y)}$, which is the 255-bit encoding of y followed by a sign bit; this sign bit is 1 if and only if x is negative. A parser recovers (x, y) from a 256-bit string as follows: parse the first 255 bits as y; compute $u/v = (y^2 - 1)/(dy^2 + 1)$; compute $\pm x = \sqrt{u/v}$, where the $\pm$ is chosen so that the sign of x matches the b-th bit of the string. Low-level details for performing this decompression efficiently are in Appendix §A.

Secret keys and public keys. A secret key is a 256-bit string k. The hash $H(k) = (h_0, h_1, \ldots, h_{511})$ determines an integer $s = \sum_{i=0}^{255} h_i \cdot 2^i$, which in turn determines the multiple $A = [s]B$. The corresponding public key is $\underline{A}$. The bits $h_{256}, h_{257}, \ldots, h_{511}$ are used below during signing.

Signing. The Schnorr$\mathbb{Q}$ signature of a message M under a secret key k is defined as follows. Define $r = H(h_{256}, \ldots, h_{511}, M) \in \{0, 1, \ldots, 2^{512} - 1\}$. Define $R = [r]B$ and $S = (r - s \cdot H(\underline{R}, \underline{A}, M)) \bmod \ell$. The signature of M under k is the 512-bit string $(\underline{R}, \underline{S})$.

(Implementation note: for efficiency, reduce r and $H(\underline{R}, \underline{A}, M)$ modulo ℓ before the computation of R and S, respectively.)

Schnorr$\mathbb{Q}$ph simply uses Schnorr$\mathbb{Q}$ to sign $H'(M)$.

Verification. "Cofactorless" verification of an alleged Schnorr$\mathbb{Q}$ signature of a message M under a public key $\underline{A}$ works as follows. The verifier parses the inputs so that A and R are elements in E and S is an integer in the set $\{0, 1, \ldots, l - 1\}$, then computes $R' = [S]B + [H(\underline{R}, \underline{A}, M)]A$ and finally checks the verification equation $\underline{R'} = \underline{R}$. The signature is rejected if parsing (i.e., any decoding) fails, if S is not in the range $\{0, 1, \ldots, l - 1\}$, or if the verification equation does not hold.

Schnorr$\mathbb{Q}$ph simply uses Schnorr$\mathbb{Q}$ to verify a signature for $H'(M)$.

Examples: the following instances use SHA-512, from the SHA-2 hash family [7], and SHA3-512, from the recently standardized SHA-3 hash family [8]. Both options produce digests of 512 bits in size and provide 256 bits of collision-resistant security.

- Schnorr$\mathbb{Q}$-SHA-512 is Schnorr$\mathbb{Q}$ with $H =$ SHA-512.
- SHA-512-Schnorr$\mathbb{Q}$-SHA-512 is Schnorr$\mathbb{Q}$ph with $H = H' =$ SHA-512.
- Schnorr$\mathbb{Q}$-SHA3-512 is Schnorr$\mathbb{Q}$ with $H =$ SHA3-512.
- SHA3-512-Schnorr$\mathbb{Q}$-SHA3-512 is Schnorr$\mathbb{Q}$ph with $H = H' =$ SHA3-512.

References

1. Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *J. Cryptographic Engineering*, 2(2):77–89, 2012. 1
2. Daniel J. Bernstein, Simon Josefsson, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. EdDSA for more curves. Cryptology ePrint Archive, Report 2015/677, 2015. http://eprint.iacr.org/2015/677. 1, 2
3. Craig Costello and Patrick Longa. Four$\mathbb{Q}$: Four-dimensional decompositions on a $\mathbb{Q}$-curve over the Mersenne prime. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015*, volume 9452 of *Lecture Notes in Computer Science*, pages 214–235. Springer, 2015. Full version: https://eprint.iacr.org/2015/565. 1
4. Mike Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. http://eprint.iacr.org/2012/309. 4
5. Ilari Liusvaara and Simon Josefsson. Edwards-curve Digital Signature Algorithm (EdDSA). Internet-Draft draft-irtf-cfrg-eddsa-05, Internet Engineering Task Force, 2016. Work in Progress. Available at: https://tools.ietf.org/html/draft-irtf-cfrg-eddsa-05. 1, 2
6. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1990. 1
7. U.S. Department of Commerce/National Institute of Standards and Technology. Secure Hash Standard (SHS). FIPS-180-4, 2015. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf. 3
8. U.S. Department of Commerce/National Institute of Standards and Technology. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. FIPS-202, 2015. http://www.nist.gov/customcf/get_pdf.cfm?pub_id=919061. 3

A Fast decompression

Point decompression is required during signature verification in order to recover coordinate x from a 256-bit string $\underline{R} = \underline{(x, y)}$. Decompression computes $u/v = (y^2 - 1)/(dy^2 + 1)$ and then $x = \pm\sqrt{u/v}$. Write $u = u_0 + u_1 \cdot i$, $v = v_0 + v_1 \cdot i$ and $x = x_0 + x_1 \cdot i$ for $u_0, u_1, v_0, v_1, x_0, x_1 \in \mathbb{F}_p$. Our goal is to compute x_0 and x_1 from u_0, u_1, v_0, v_1. Equating coefficients in

$$(x_0 + x_1 \cdot i)^2 = \frac{u_0 + u_1 \cdot i}{v_0 + v_1 \cdot i}$$

yields two quadratic equations in x_0^2 and x_1^2 over $\mathbb{F}_p$, the solutions of which are

$$x_0^2 = \frac{2\alpha \pm 2\sqrt{\alpha^2 + \gamma^2}}{4\beta} \qquad \text{and} \qquad x_1^2 = \frac{-2\alpha \pm 2\sqrt{\alpha^2 + \gamma^2}}{4\beta}, \tag{1}$$

where $\alpha = u_0v_0 + u_1v_1$, $\beta = v_0^2 + v_1^2$, $\gamma = u_1v_0 - u_0v_1$.

First, we compute $t = 2(\alpha + \sqrt{\alpha^2 + \gamma^2}) = 2(\alpha + (\alpha^2 + \gamma^2)^{2^{125}})$. If $t = 0$, then compute $t = 2(\alpha - (\alpha^2 + \gamma^2)^{2^{125}})$. Up to the sign in front of $\sqrt{\alpha^2 + \gamma^2}$ (which will be resolved in a moment), we now have $t = 4\beta x_0^2$.

Observe that $\pm x_0 x_1 = \gamma/(2\beta)$. Following [4], we compute β^{-1} and recover x_0 and x_1 using one exponentiation as follows. We first compute $\pm r = \sqrt{1/(t \cdot \beta^3)} = (t \cdot \beta^3)^{2^{125}-1}$, and then recover $\pm x_0 = (r \cdot \beta \cdot t)/2$ and $\pm x_1 = r \cdot \beta \cdot \gamma$.

The sign ambiguities are resolved as follows. The sign in front of $\sqrt{\alpha^2 + \gamma^2}$ is checked by computing $\beta \cdot (2x_0)^2$ and comparing against t; if these are not equal then x_0 and x_1 are swapped. Set $x := x_0 + x_1 \cdot i$ and if the sign of x does not match the 256-th bit in the public key, compute $x = -x$. Finally, the sign of x_1 is resolved by checking the curve equation: if $-x^2 + y^2 \neq 1 + dx^2y^2$, then we take $x_1 := -x_1$ and reset $x := x_0 + x_1 \cdot i$.

Summary. On top of a few multiplications, squarings and additions, decompression takes only two exponentiations in $\mathbb{F}_p$: one has exponent 2^{125} and the other has exponent $2^{125} - 1$. This is highly convenient since the first case only requires an easy "squares-only" addition chain and the second case requires an addition chain that is already present in the addition chain for inversions.