Elligator: Elliptic-curve points indistinguishable from uniform random strings

Daniel J. Bernstein[1,3]
[1]Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607–7045
USA
djb@cr.yp.to

Anna Krasnova[2]
[2]Privacy & Identity lab
Institute for Computing and Information Sciences
Radboud University Nijmegen
Heyendaalseweg 135
6525 AJ Nijmegen
The Netherlands
anna@mechanical-mind.org

Tanja Lange[3]
[3]Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513
5600 MB Eindhoven
The Netherlands
tanja@hyperelliptic.org

ABSTRACT

Censorship-circumvention tools are in an arms race against censors. The censors study all traffic passing into and out of their controlled sphere, and try to disable censorship-circumvention tools without completely shutting down the Internet. Tools aim to shape their traffic patterns to match unblocked programs, so that simple traffic profiling cannot identify the tools within a reasonable number of traces; the censors respond by deploying firewalls with increasingly sophisticated deep-packet inspection.

Cryptography hides patterns in user data but does not evade censorship if the censor can recognize patterns in the cryptography itself. In particular, elliptic-curve cryptography often transmits points on known elliptic curves, and those points are easily distinguishable from uniform random strings of bits.

This paper introduces high-security high-speed elliptic-curve systems in which elliptic-curve points are encoded so as to be indistinguishable from uniform random strings.

1. INTRODUCTION

Elliptic-curve cryptography (ECC) is arguably the most important tool in modern public-key cryptography. It provides public-key signatures, public-key encryption, non-interactive key exchange, and many higher-level security features. It offers an attractive combination of high security, high speed, and (often critical for deployment) small space consumption.

However, for applications in censorship circumvention, ECC has a security problem. ECC protocols naturally send elliptic-curve points in the clear as long-term public keys, ephemeral public keys, ciphertext prefixes, challenges, etc. These points, even in compressed form, are obvious: they are easy to distinguish from uniform random strings.

There have been some ad-hoc workarounds for this problem, notably for ElGamal ciphertext prefixes, using a curve-or-twist technique introduced by Möller (see below). But each new ECC-based protocol faces the same problem. Protocol designers unaware of the issue continue building protocols that are trivially visible to attackers. Designers requiring keys and ciphertexts to be indistinguishable from random are forced to modify those protocols, hoping that the modifications do not compromise other forms of security.

The main goal of this paper is to eliminate this problem. The solution presented here works for a wide range of elliptic-curve protocols, essentially every protocol in which the transmitted curve points are generated at random. There is no longer any need for, e.g., ad-hoc handling of ciphertext prefixes; this paper's technique works for all of the types of elliptic-curve points mentioned above.

1.1 Distinguishers

We use the standard NIST P-256 elliptic curve as an example to illustrate the difficulties. A public key on the NIST P-256 elliptic curve is a pair (x, y) of integers satisfying the equation $y^2 = x^3 - 3x + b$ modulo the prime $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, where b is a standard constant. There are at least three obvious ways for an attacker to distinguish this public key from a uniform random string:

- Least severe: Normally x and y are represented as integers between 0 and $2^{256} - 2^{224} + 2^{192} + 2^{96} - 2$ inclusive, encoded as 256-bit strings. If the attacker sees a 256-bit string representing an integer larger than $2^{256} - 2^{224} + 2^{192} + 2^{96} - 2$ then the attacker knows that the string is not a valid value of x or of y. If the attacker sees a user sending a series of (e.g.) 2^{35} 256-bit strings, and the largest string is close to $2^{256} - 2^{224}$, then the attacker is reasonably confident that the user is sending elliptic-curve points.

  One can dismiss this attack as being too slow to be of interest, but we prefer more robust cryptographic primitives that maintain security with heavy use. The user can cover all integers between 0 and $2^{256} - 1$ by randomizing the representations of small integers x and y, but this adds very little security: the attacker easily collects statistics showing that some integers ap-

This work was supported by the National Science Foundation under grant 1018836, by the Netherlands Organisation for Scientific Research (NWO) under grants 639.073.005 and 040.09.003, by the European Commission under Contract ICT-2007-216676 ECRYPT II, and by SIDN.nl (`http://www.sidn.nl/`). Part of this work was done while the authors visited Academia Sinica; they wish to thank Bo-Yin Yang for his hospitality. Permanent ID of this document: `711e2f8601805e06e5fbb3c5953dd5d1`. Date: 2013.05.27.

pear half as often as others.

A secure solution is to randomly represent x and y as integers between 0 and, e.g., $2^{320} - 1$. Another secure solution, with smaller keys, is to switch to an elliptic curve using a prime much closer to a power of 2, such as NIST P-224 (prime $2^{224} - 2^{96} + 1$, lower security level) or Bernstein's Curve25519 from [4] (prime $2^{255} - 19$).

- More severe: The attacker simply checks the curve equation. If a 512-bit string has the form (x, y) where $y^2 = x^3 - 3x + b$ modulo this prime then the attacker is confident that the user is sending a public key.

  Many ECC systems save space by compressing y to a single (random-looking) bit, namely the sign in $\pm\sqrt{x^3 - 3x + b}$. Some ECC systems save space by eliminating y entirely. The cost for the legitimate user of computing a square root is almost always outweighed by the benefit of reducing keys to half size. Both of these mechanisms have the side effect of stopping this attack.

- Most severe: The attacker checks whether $x^3 - 3x + b$ is a square modulo this prime. This has chance 1/2 of occurring for a uniform random string, but if it occurs repeatedly then the attacker is reasonably confident that the user is sending public keys.

The third attack is quite difficult to stop. Our solution requires a quite drastic change in how curve points are represented as strings; this representation is the main topic of this paper.

Our solution is not limited to public keys: it also protects other randomly generated elliptic-curve points, such as the points appearing in ciphertexts in elliptic-curve versions of the ElGamal encryption system and points appearing in signature systems. See Section 2.

1.2 Previous work

Several years ago, in [35], Möller proposed a variant of the ElGamal encryption system that provides indistinguishability for ciphertexts as follows:

- Alice's public key is a pair $(aP, a'P')$ where a and a' are secret integers, P is a standard base point on an elliptic curve $E(\mathbf{F}_{2^n})$ over some binary field $\mathbf{F}_{2^n}$, and P' is a standard base point on a nontrivial quadratic twist $E'(\mathbf{F}_{2^n})$ of the same curve.

- To encrypt a message m using public key $(aP, a'P')$, Bob chooses a random integer b, chooses randomly between aP and $a'P'$, sends the x-coordinate of bP or bP' respectively, hashes $b(aP)$ or $b(a'P')$ respectively to obtain a secret key, and sends an encryption of m using this secret key.

- Alice recovers bP or bP' from the x-coordinate. Alice multiplies bP or bP' by a or a' to obtain abP or $a'bP'$, hashes it to obtain the same secret key, and decrypts the message. Möller requires Alice to perform tests to decide whether the input is on E or on E' and to recover the whole point bP or bP' . We comment that the "Montgomery ladder" handles both cases together if a' is chosen as a. (For background on the Montgomery ladder see [36], [13], and, for the binary case, [34].) This eliminates the need for such tests and offers efficient curve arithmetic.

The idea in Möller's proposal is that each element of $\mathbf{F}_{2^n}$ is a valid x-coordinate on E or its twist. The x-coordinate transmitted here can therefore be any element of $\mathbf{F}_{2^n}$, and in fact the distribution of the x-coordinates is indistinguishable from the uniform distribution of strings of length n. Möller slightly adjusts the choices so that the distribution is exactly uniform.

Our approach encodes points on a *single* curve as strings indistinguishable from uniform random strings. This has several obvious advantages over Möller's system. We obtain indistinguishability not just for ciphertexts but also for public keys. Möller's system has double-length public keys, while our public keys have minimal length. Möller's approach is limited to ElGamal encryption, while our approach handles a much wider range of ECC systems; see Section 2. Möller needs cryptographic security not just for E but also for E'; in our system twist security is not required, although we still suggest it as a desirable feature (see Section 4). Finally, Möller describes his system only for curves over binary fields $\mathbf{F}_{2^n}$, which are not as confidence-inspiring as curves over prime fields; see Section 4.

1.3 Application context

Möller's curve-or-twist approach is used in StegoTorus [42]. StegoTorus is an extension to Tor [16] for censorship circumvention. It makes Tor traffic resemble Skype, general HTTPS traffic, etc. and relies on secure communication with a StegoTorus server. Establishing a link with this server requires deriving temporary key material using the public key of the server. This key material is used to encrypt communication of a subsequent ephemeral DH key exchange. StegoTorus uses Möller's example parameters over $\mathbf{F}_{2^{163}}$.

For comparison, typical ECC protocols carry out a DH key exchange in the clear; see, e.g., the ntor example below. StegoTorus needs to encrypt its key exchange because the points sent in the key exchange are easily distinguishable from random data. This encryption adds the outer Möller layer, doubling the space used for the initial communication. This also slows down the client, slows down the server, doubles the size of server public keys (one point on the curve, one point on the twist), and requires implementations to handle computations on two curves.

We eliminate all of these issues by solving the underlying problem, namely the point distinguishability. The points that we send in a DH key exchange are indistinguishable from uniform random strings. See Sections 2 and 3.

Telex [43] is another censorship-circumvention tool. Telex messages pose as regular TLS messages to random uncensored sites; the only difference is that the nonce field contains a cryptographic value instead of a random value. The public key of the Telex server is a pair of points (aP, aP'), with P on an elliptic curve $E(\mathbf{F}_q)$ and P' on a nontrivial quadratic twist. Telex servers use deep-packet inspection on all traffic passing through them and identify Telex messages by checking whether the nonce field interpreted as $\alpha||\beta$ satisfies the following conditions. Interpret α as the x-coordinate of a point R on E or E' and compute aR on the appropriate curve. Identify the message as a Telex message if β matches a salted hash of aR, and route it according to some other, encrypted information.

The use of curves in Telex follows Möller's proposal except for the choice of finite field and the choice of equal secrets. The proposed parameters have $q = 2^{168} - 2^8 - 1$ so that the distribution of values in $\mathbf{F}_q$ is indistinguishable from that of length-168 strings. The attempted decryption performs a complete point recovery from x and uses standard Weierstrass-curve arithmetic to compute the scalar multiplications. We comment again that the "Montgomery ladder" is more efficient, handling both cases together. We also comment that the system could use somewhat larger finite fields, gaining security, without reducing the 56-bit hash size and without going beyond the standard TLS 224-bit nonce size: servers would allow only points R with a fixed number of implicit trailing zeros, and clients would repeatedly generate points until meeting that condition.

This paper's solution to the point-distinguishability problem would not save bandwidth for Telex connections but would still simplify implementations, removing any need to handle the twist, and would reduce public keys to half size, potentially a useful feature for small clients keeping track of many different Telex servers.

As a third protocol we consider ntor [27], a handshake protocol proposed for Tor achieving anonymity and one-way authentication with forward secrecy. This protocol assumes that Tor relays have public keys on an elliptic curve. The following is a simplified version of the ntor protocol, skipping certificates and saved session states, but presenting all parts relevant to the choice of curve group and representation. To extend a Tor circuit to a relay with public key $B = bP$ a client picks a random value x, computes $X = xP$, and sends X to the server. The server picks a random y, computes $Y = yP$, computes two secret keys as $k'||k = H(yX, bX, B, X, Y)$ for some hash function H, computes $t_B = \mathrm{MAC}_{k'}(B, X, Y)$ (an authenticator under key k'), and sends $Y||t_B$ to the client. The client computes $\bar{k}'||\bar{k} = H(xY, xB, B, X, Y)$ and verifies that $t_B = \mathrm{MAC}_{\bar{k}'}(B, X, Y)$. If the verification is successful, the client is convinced that it is communicating with the correct server: nobody other than the client and the relay could compute $xB = bX$. Both sides can use $k = \bar{k}$ for transmitting encrypted messages.

Tor currently uses SSL links to superencrypt its traffic. However, one can easily imagine the same ntor protocol being steganographically encoded inside a cover channel, with the goal of circumventing censorship, on top of the original goals of anonymity, one-way authentication, and forward secrecy. This raises the question of how the elliptic-curve points can be hidden. Using a pair of points xP, xP' in place of X does not work: it is even easier to distinguish from random than a single point. Möller's curve-or-twist approach, using a pair of points in place of the server's long-term key B, also does not work: the client is free to pick X on the curve or the twist, but the server is then forced to pick Y on the same curve as X, and this is something visible to a censor.

Our solution applies straightforwardly to this protocol. It does not need twists: it ensures that the encodings of points on a single curve are uniformly distributed. More generally, our solution can be used to encode as many points as desired during one session or across sessions.

1.4 Mapping strings to elliptic-curve points

Shallue and van de Woestijne at ANTS 2006 [40] gave a positive answer to the following question in pure algorithmic number theory: there is, provably, a (very easy) *probabilistic* polynomial-time algorithm that, given any elliptic curve E over any sufficiently large finite field $\mathbf{F}_q$, constructs a nonzero element of the group $E(\mathbf{F}_q)$; is there, provably, a *deterministic* polynomial-time algorithm for the same task? Shallue and van de Woestijne did more than construct a single point: they built a function $\phi : \mathbf{F}_q \to E(\mathbf{F}_q)$ such that each element of $E(\mathbf{F}_q)$ has at most C preimages, for a particular constant C; and they built a deterministic polynomial-time algorithm that computes $\phi(t)$ given q, E, and t. Trying $C + 1$ elements of $\mathbf{F}_q$ then produces a nonzero element of $E(\mathbf{F}_q)$.

Some cryptographic protocols rely on hashing strings to curve points. For example, in the Boneh–Franklin identity-based encryption scheme [10], public keys are points on pairing-friendly curves, and a user with identity i has public key $H(i)$. Boneh and Franklin constructed a suitable function H for certain supersingular curves. Icart at Crypto 2009 [29] pointed out that the Shallue–van de Woestijne function ϕ produced suitable functions H for any elliptic curve, allowing the Boneh–Franklin system to be adapted to much more efficient non-supersingular (but still pairing-friendly) curves. Icart and subsequent authors explored various replacements ϕ for the Shallue–van de Woestijne function; see [29], [12], [23], [22], [17], [24], [20], and [18].

The reader should be wondering why one cannot simply define $\phi(t)$ by trying consecutive field elements x starting from t (for example, trying $x = t$, $x = t + 1$, $x = t + 2$, etc. in the prime-field case) until finding a curve point (x, y). The answer for Shallue and van de Woestijne is that this is not *proven* to take polynomial time. On the other hand, there is overwhelming evidence for the conjecture that this takes polynomial time, and in cryptography there is no harm in making such a conjecture; cryptography is built on a foundation of conjectures that are much easier to question, such as the conjecture that there is no fast ECDL algorithm. This simple function from $\mathbf{F}_q$ to $E(\mathbf{F}_q)$ is exactly what is used in many papers on identity-based cryptography.

The real objection to this algorithm is that it does not take *constant* time. In many applications the time leaks secret information. One can compute the same function for almost all inputs in constant time by choosing an upper bound C on the number of consecutive field elements required with very high probability, say $C = 100$, and then always test exactly C values of x; but testing C times for squares turns out to be a serious performance problem.

It should be obvious that any of the functions ϕ mentioned above allows a public key $\phi(t) \in E(\mathbf{F}_q)$ to be represented as an element $t \in \mathbf{F}_q$, using a computation of $Q \mapsto \{t : \phi(t) = Q\}$ to encode a point and a computation of $t \mapsto \phi(t)$ to decode a point. However, this approach raises several important performance issues for key generation:

- The usual key-generation method produces a uniform random curve point by scalar multiplication, but there is no reason to think that this curve point can be expressed in the form $\phi(t)$. If it cannot then the user must try again, generating a new key.

- Even worse, to generate a *uniform* distribution of elements $t \in \mathbf{F}_q$ the user must generate a uniform random curve point Q, compute the number $k = \#\{t : \phi(t) = Q\}$ of preimages of the point, restart with probability $1 - k/C$, and finally select a uniform random preimage of the point. (This is what statisticians call "rejection sampling"; more advanced sampling methods are inapplicable since there is no way to generate a uniform random curve point with a specified value of k.) The average number of points generated is then $C\#E(\mathbf{F}_q)/q \approx C$. One must budget for even more repetitions in applications that need real-time guarantees.

- Each of these C points takes time not just for the computation of Q but also for the computation of $\{t : \phi(t) = Q\}$. This is the main bottleneck if ϕ is slow to invert.

In many protocols, ephemeral ECC keys are generated for every protocol run, and these computations can easily dominate the overall protocol performance. Of course, even in protocols where key generation is rare, each use of t incurs the cost of computing $\phi(t)$.

We minimize these performance problems by taking an encoding function ϕ that is implicit in the very recent paper [20] by Fouque, Joux, and Tibouchi. This function, in the cases we consider, maps $\{0, 1, 2, 3, \ldots, (q-1)/2\}$ *injectively* to $E(\mathbf{F}_q)$; see Theorem 4. Since the number of points on E over $\mathbf{F}_q$ is about q we have to try only about 2 points on average before finding a point of the form $\phi(t)$, and we do not need to randomize the preimage t. Not every elliptic curve is suitable for this function, but the curve requirements in this paper are compatible with state-of-the-art criteria for curve security and curve performance.

Fouque, Joux, and Tibouchi suggested that injectivity would be useful for a naive form of ElGamal encryption in which a message m is encoded injectively as an elliptic-curve point and then simply added to abP. We highlight the new anti-censorship application, namely encoding *points* as *strings* rather than encoding strings as points; we drastically simplify the definition of this function, while accelerating its forward and inverse computation; and we introduce a high-security high-speed elliptic curve that supports the function. See Section 4 for the curve, Section 3 for the function, and Section 2 for the cryptographic applications.

2. ELLIPTIC-CURVE PROTOCOLS

This section presents various elliptic-curve protocols in which public keys, ciphertexts, etc. are indistinguishable from uniform random strings. These protocols include all of the fundamental ECC constructions: static and ephemeral key exchange, encryption, and signatures.

The prerequisite for all of these protocols is an injective map ι from a set of strings $S \subseteq \{0,1\}^b$ to an elliptic-curve group $E(\mathbf{F}_q)$. We require $\#S$ to be extremely close to 2^b, so that a uniform random element of S is indistinguishable from a uniform random b-bit string. Section 3 explains how to construct ι and S for a broad class of elliptic curves; Section 4 presents a specific elliptic curve that supports the construction.

The idea, as explained in Section 1, is for each string $\tau \in S$ to represent the elliptic-curve point $\iota(\tau)$, i.e., for the point $\iota(\tau)$ to be encoded as the string τ. A uniform random element of $\iota(S)$, encoded in this way, is indistinguishable from a uniform random b-bit string. We do not require $\iota(S)$ to be all of $E(\mathbf{F}_q)$; our protocols compensate for this by repeatedly generating curve points until finding elements of $\iota(S)$. For our construction $\#\iota(S)$ is about $(1/2)\#E(\mathbf{F}_q)$, so only about 2 repetitions are required on average, as mentioned in Section 1.

Our primary objective in this section is to illustrate how easily ι can be used to systematically hide elliptic-curve points in a wide range of protocols. We do not mean to suggest that all of these protocols are being used in contexts where they need to be hidden from censors; but there is no obvious dividing line between protocols that would be useful for those contexts and protocols that would not.

Consider, for example, static ECDH public keys. A static public key might be distributed openly, as part of a cryptographic software package, in which case it does not add any new risk of censorship; changing the encoding of the public key does nothing to hide the package from a censor. On the other hand, if a client already has all necessary cryptographic software, and a server broadcasts a series of static public keys by encoding those keys inside otherwise innocent web pages or other cover traffic, then indistinguishability from random is exactly the tool needed to defend those public keys against a censor with an accurate model of the cover traffic. A censored Tor client should be able to see frequent updates of public keys for Tor bridges, for example; see [1] for a detailed discussion of the speed at which various governments detect and suppress access to Tor bridges.

2.1 Notation and domain parameters

To simplify the protocol statements we assume that the elliptic-curve group $E(\mathbf{F}_q)$ is cyclic: specifically, that it is generated by a standard base point P of order n. In the case that n is not prime (for example, n is 4 times a prime in Section 4) we do *not* restrict points to a prime-order subgroup: any proper subgroup is trivially distinguishable from the full group. For non-cyclic groups the protocols would have to be modified to handle multiple generators.

Let H denote a hash function, and let $||$ denote concatenation of strings. Symmetric authenticated encryption of a message M using a secret key k is denoted as $c = \mathrm{Enc}_k(m)$, decryption as $m = \mathrm{Dec}_k(c)$. A standard security requirement for symmetric encryption is that the ciphertexts are indistinguishable from random (see, e.g., [3]); one can, for example, safely use AES in counter mode. For authentication there is a split in the literature between authenticators aiming at the "PRF" property, which guarantees indistinguishability, and authenticators aiming merely at the "MAC" property, which guarantees unforgeability but does not guarantee indistinguishability; we require the PRF property. See [39] for a unified security notion for authenticated encryption that guarantees indistinguishability from random bit strings.

2.2 Long-term Diffie–Hellman keys

Each user U sets up a public key:

1. U generates a random integer u.
2. U computes $P_U = uP$. If $P_U \notin \iota(S)$ then U repeats from step 1.
3. U publishes $\iota^{-1}(P_U) = \tau_U$ and keeps u secret.

The encoding τ_U of the public key is a string which can be broadcast; it is indistinguishable from a random string.

Without further communication users Alice and Bob can compute a shared secret from their public strings τ_A and τ_B. Alice can compute this key as follows and then use it to authenticate and encrypt a message to Bob:

1. Alice computes $\iota(\tau_B) = P_B$.
2. Alice computes $k = H(u_A P_B)$.

Upon receiving an encrypted message from Alice, Bob can likewise compute the same shared secret and then decrypt the message and verify the authenticator:

1. Bob computes $\iota(\tau_A) = P_A$.
2. Bob computes $k = H(u_B P_A)$.

2.3 ElGamal encryption

Assume u_B and $P_B = u_B P$ is a static key pair of Bob and that Bob has published $\tau_B = \iota^{-1}(P_B)$. Alice wants to send Bob a message M. To encrypt the message she performs the following steps:

1. She generates a random integer r.
2. She computes the point $R = rP$. If $R \notin \iota(S)$ she repeats from step 1.
3. She computes $\iota^{-1}(R) = \tau_R$.
4. She computes $P_B = \iota(\tau_B)$.
5. She computes the shared key value $k = H(rP_B)$.
6. She encrypts message m using key k: $c = \text{Enc}_k(m)$.
7. She sends the tuple (τ_R, c) as an encryption of m.

To decrypt the received message, Bob:

1. computes $\iota(\tau_R) = R$,
2. computes the same shared key value: $k = H(u_B R)$, and
3. decrypts the message: $m = \text{Dec}_k(c)$.

ElGamal encryption also appears in [20] as an application of an injective map between strings and curve points, but our application is completely different from the application in [20]. There is a critical difference in the underlying encryption methods: we use symmetric cryptography to encrypt the message m using a key derived from rP_B, whereas [20] adds rP_B to a curve point M that represents m. This is exactly where [20] uses an injective map, namely to encode the string m as a curve point M. Note, however, that this also (1) prevents [20] from encrypting long messages and (2) allows malleability, in violation of the basic security standards for public-key encryption. We instead use the standard "KEM/DEM" structure to provide secure public-key encryption, and as a consequence do not need to encode strings as curve points. We use an injective map for a completely different reason: we encode curve points as strings, preventing those points from being recognized by censors.

2.4 Short-term Diffie–Hellman keys

In this protocol, Alice and Bob agree on a shared secret without using long-term keys. This protocol is important as a way to provide forward secrecy. Of course, this provides no authentication, but authentication can be added as a subsequent layer.

1. Alice and Bob generate short-term keys as follows:
   (a) Generate a random integer r.
   (b) Compute point $R = rP$. If $R \notin \iota(S)$ repeat from step 1a.
2. Alice sends $\mu_A = \iota^{-1}(R_A)$ to Bob; Bob sends $\mu_B = \iota^{-1}(R_B)$ to Alice.
3. Alice decodes $R_B = \iota(\mu_B)$; Bob decodes $R_A = \iota(\mu_A)$.
4. Alice computes the shared key value $k = H(r_A R_B)$; Bob computes the shared key value as $k = H(r_B R_A)$.

2.5 Schnorr signatures

Assume that u_B and $P_B = u_B P$ are respectively private and public keys of Bob, where $P_B \in \iota(S)$ and Bob has published $\tau_B = \iota^{-1}(P_B)$. To sign a message m, Bob performs the following steps:

1. Choose a random integer r .
2. Compute $R = rP$. If $R \notin \iota(S)$, repeat from step 1.
3. Compute $\tau = \iota^{-1}(R)$.
4. Compute $h = H(R||m)$.
5. Compute $s = r + hu_B \pmod{n}$.
6. The signature is the tuple $\psi = (\tau, s)$.

To have this signature resemble a random string the distribution of s must be indistinguishable from random, too. This means that also n must be close to a power of 2. By Hasse's theorem, the number of points on $E(\mathbf{F}_q)$ is within $2\sqrt{q}$ of q. To detect a difference from random strings of the same length about $\sqrt{q}$ strings need to be inspected. Since the security of the discrete logarithm on E is about $\sqrt{q}$ this does not pose any risk.

To verify the signature (τ, s), Alice has to perform the following steps:

1. Compute $P_B = \iota(\tau_B)$.
2. Compute $R = \iota(\tau)$.
3. Compute $h = H(R||m)$.
4. Compare $R + hP_B$ and sP. If they are equal, accept. Otherwise, reject.

Schnorr's original signature system actually sent (h, s). We follow "EdDSA" from [6] in sending an encoding of (R, s); security is the same, since one can reconstruct (h, s) from (R, s) and vice versa. The advantage of (R, s) is that it allows batching, making signature verification about twice as fast, as explained in [6]. Our use of ι makes this signature indistinguishable from a uniform random string.

3. THE INJECTIVE MAP

Section 2 needs an injective map ι from a large set S of strings to $E(\mathbf{F}_q)$. This section presents the mathematical details: the construction of ι, how to compute ι, how to test whether a curve point is in the image of ι, and how to invert ι on curve points in the image. To help the reader visualize the mathematical structure of ι we include a picture as Figure 1.

We impose certain requirements on q and E: we consider only primes q; we require q to be congruent to 3 modulo 4;

we require E to be a complete Edwards curve; and we impose an extra algebraic requirement ($c = 2/s^2$ in Theorem 1) that allows only half of all complete Edwards curves. Approximately 1/16 of all isomorphism classes of elliptic curves over all finite fields satisfy these requirements. (Asymptotically 100% of all finite fields, ordered by size, are prime fields; 50% of those primes are congruent to 3 modulo 4; 25% of elliptic curves over those fields are complete Edwards curves; 50% of those satisfy the extra algebraic requirement.) See Section 4 for specific choices of q and E.

The heart of ι is a function $\phi : \mathbf{F}_q \to E(\mathbf{F}_q)$ defined in Theorem 1 and Definition 2. This function satisfies $\phi(t) = \phi(-t)$ for each $t \in \mathbf{F}_q$ but has no other collisions (see Theorem 3), so its restriction to $\{0, 1, 2, \ldots, (q-1)/2\}$ is injective. Theorem 4 simply defines S as $\{0, 1, 2, \ldots, (q-1)/2\}$ represented in little-endian form as b-bit strings, where $b = \lfloor \log_2 q \rfloor$, and defines ι as the corresponding representation of ϕ. For indistinguishability we add the requirement that $(q+1)/2$ be extremely close to 2^b.

As mentioned in Section 1, this function ϕ was introduced by Fouque, Joux, and Tibouchi in [20]. Our main contributions in this section are a much more concise definition of ϕ; much more direct proofs of the relevant properties of ϕ; a simple method to invert ϕ; and a simple test for whether a curve point is in the image of ϕ.

The function $-\phi$ has the same useful properties as ϕ. Our choice of sign is not the same as the choice in [20]: in the notation below, the ratio is $\chi(c)$, i.e., $\chi(2)$. This choice of sign slightly simplifies our formulas for the forward and inverse maps, although it is not the main simplification compared to [20]. We also comment that $\phi(t) = -\phi(1/t)$ for $t \neq 0$.

Computing ι in a sensible way is almost as fast as traditional point decompression, not a serious bottleneck compared to scalar multiplication. Inverting ι is slightly slower, but testing whether a curve point is in the image of ι is very fast. See Section 3.5 for further performance analysis.

3.1 Squares, square roots, and χ

Before proving theorems we review the relevant properties of squares in the finite field $\mathbf{F}_q$, where q is a prime power congruent to 3 modulo 4. In defining ι we consider only primes q for simplicity (so that $0, 1, 2, \ldots, (q-1)/2$ are distinct field elements) but our theorems about ϕ also apply to prime powers.

Define $\chi : \mathbf{F}_q \to \mathbf{F}_q$ by $\chi(a) = a^{(q-1)/2}$. If a is a non-zero square then $\chi(a) = 1$; if a is a non-square then $\chi(a) = -1$; if $a = 0$ then $\chi(a) = 0$. Note that $(q-1)/2$ is odd since $q \equiv 3 \pmod 4$, so $\chi(-1) = -1$, so -1 is not a square. More generally, $\chi(\chi(a)) = \chi(a)$. There are several easy ways to manipulate χ arguments: for example, $\chi(ab) = \chi(a)\chi(b)$, $\chi(1/a) = \chi(a) = 1/\chi(a)$ if $a \neq 0$, and $\chi(a^2) = 1$ if $a \neq 0$.

If a is a square then $a^{(q+1)/4}$ is a square root of a: its square is $a^{(q+1)/2} = \chi(a)a = a$. More precisely, it is the **principal** square root of a: the unique square root that is itself a square. Any square root b of a satisfies $b = \chi(b)a^{(q+1)/4}$.

The function χ is called a **quadratic character**. See [33] for further background on finite fields.

3.2 The map

THEOREM 1. *Let q be a prime power congruent to* 3 *modulo* 4*. Let s be a nonzero element of $\mathbf{F}_q$ with $(s^2-2)(s^2+2) \neq 0$. Define $c = 2/s^2$. Then $c(c-1)(c+1) \neq 0$.*

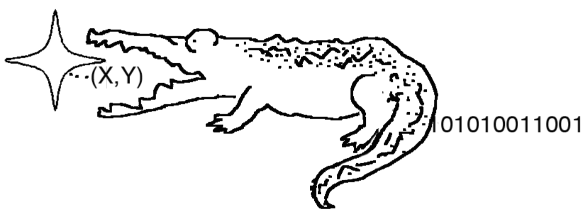

Figure 1: The structure of our encoding function ι^{-1}, a bijection from a large subset of $E(\mathbf{F}_q)$ to a large set S of b-bit strings. The elliptic curve E is required to be a complete Edwards curve, shown on the left together with a sample element $P = (x, y)$ of $E(\mathbf{F}_q)$. A sample b-bit output is shown on the right. See Theorem 1, Definition 2, Theorem 3, and Theorem 4 for further details regarding the function in the middle.

Define $r = c + 1/c$ and $d = -(c+1)^2/(c-1)^2$. Then $r \neq 0$, and d is not a square.

The following elements of $\mathbf{F}_q$ are defined for each $t \in \mathbf{F}_q \setminus \{\pm 1\}$:

$$\begin{aligned} u &= (1-t)/(1+t),\\ v &= u^5 + (r^2-2)u^3 + u,\\ X &= \chi(v)u,\\ Y &= (\chi(v)v)^{(q+1)/4}\chi(v)\chi(u^2 + 1/c^2),\\ x &= (c-1)sX(1+X)/Y,\\ y &= (rX - (1+X)^2)/(rX + (1+X)^2). \end{aligned}$$

Furthermore $x^2 + y^2 = 1 + dx^2y^2$; $uvXYx(y+1) \neq 0$; and $Y^2 = X^5 + (r^2-2)X^3 + X$.

PROOF. $\boldsymbol{c(c-1)(c+1) \neq 0}$**:** By definition $c = 2/s^2$ so $c \neq 0$. By hypothesis $s^2 \neq 2$ and $s^2 \neq -2$ so $c \neq 1$ and $c \neq -1$.

$\boldsymbol{r \neq 0}$**:** If $r = 0$ then $c = -1/c$ so $c^2 = -1$, contradiction.

$\boldsymbol{d}$ **is not a square:** Otherwise $-1 = d(c-1)^2/(c+1)^2$ is a square, contradiction.

$\boldsymbol{u}$ **is defined and** $\boldsymbol{u \neq 0}$**:** By hypothesis $1 + t \neq 0$ and $1 - t \neq 0$.

$\boldsymbol{v \neq 0}$**:** $r^2 - 2 = c^2 + 1/c^2$ so $v = u(u^2 + c^2)(u^2 + 1/c^2)$. If $v = 0$ then there are three possibilities: $u = 0$, contradiction; or $u^2 + c^2 = 0$ so $-1 = (u/c)^2$, contradiction; or $u^2 + 1/c^2 = 0$ so $-1 = (uc)^2$, contradiction.

$\boldsymbol{XY \neq 0}$**, so** $\boldsymbol{x}$ **is defined:** As above $u^2 + 1/c^2 \neq 0$ so all factors in X and Y are nonzero.

$\boldsymbol{1 + X \neq 0}$**, so** $\boldsymbol{x \neq 0}$**:** If $X = -1$ then $u = -\chi(v)$ so $v = -\chi(v)(1 + r^2 - 2 + 1) = -\chi(v)r^2$ so $\chi(v) = -\chi(v)$, contradiction.

$\boldsymbol{(X, Y)}$ **satisfies** $\boldsymbol{Y^2 = X^5 + (r^2-2)X^3 + X}$**:** $X = \chi(v)u$ so $X^5 + (r^2-2)X^3 + X = \chi(v)(u^5 + (r^2-2)u^3 + u) = \chi(v)v$. Also $\chi(v)v$ is a square so $(\chi(v)v)^{(q+1)/2} = \chi(v)v$ so $Y^2 = \chi(v)v$.

$\boldsymbol{rX + (1+X)^2 \neq 0}$**, so** $\boldsymbol{y}$ **is defined:** If $rX = -(1+X)^2$ then $(r^2 + 4r)X^2 = X^4 - 2X^2 + 1$ so

$$\begin{aligned} Y^2 &= X(X^4 + (r^2-2)X^2 + 1) = X^3(2r^2 + 4r)\\ &= rX \cdot X^2(2r + 4) = -(1+X)^2X^2(s + 2/s)^2 \end{aligned}$$

so -1 is a square, contradiction.

$\boldsymbol{y + 1 \neq 0}$**:** If $y = -1$ then $(rX - (1+X)^2)/(rX + (1+X)^2) = -1$ so $rX - (1+X)^2 = -(rX + (1+X)^2)$ so $rX = 0$, contradiction.

$\boldsymbol{x^2+y^2=1+dx^2y^2}$**:** First $(c-1)^2s^2=(c-1)^2(2/c)=2(r-2)$ so

$$\begin{aligned}Y^2(1-x^2)&=Y^2-(c-1)^2s^2X^2(1+X)^2\\&=X^5+(r^2-2)X^3+X-2(r-2)X^2(1+X)^2\\&=X(rX-(1+X)^2)^2.\end{aligned}$$

Similarly $-d=(c+2+1/c)/(c-2+1/c)=(r+2)/(r-2)$ so $-d(c-1)^2s^2=2(r+2)$ so

$$\begin{aligned}Y^2(1-dx^2)&=Y^2-d(c-1)^2s^2X^2(1+X)^2\\&=X^5+(r^2-2)X^3+X+2(r+2)X^2(1+X)^2\\&=X(rX+(1+X)^2)^2.\end{aligned}$$

Note that $Y^2(1-dx^2)\neq 0$, and divide: $(1-x^2)/(1-dx^2)=(rX-(1+X)^2)^2/(rX+(1+X)^2)^2=y^2$; i.e., $x^2+y^2=1+dx^2y^2$. $\square$

Definition 2. *In the situation of Theorem 1, the* **decoding function** *for the complete Edwards curve* $E:x^2+y^2=1+dx^2y^2$ *is the function* $\phi:\mathbf{F}_q\to E(\mathbf{F}_q)$ *defined as follows:* $\phi(\pm1)=(0,1)$*; if* $t\notin\{\pm1\}$ *then* $\phi(t)=(x,y)$.

3.3 Inverting the map

Theorem 3. *In the situation of Definition 2:*

1. *If* $t\in\mathbf{F}_q$ *then the set of preimages of* $\phi(t)$ *under* ϕ *is* $\{t,-t\}$.
2. $\phi(\mathbf{F}_q)$ *is the set of* $(x,y)\in E(\mathbf{F}_q)$ *such that*
   - $y+1\neq 0$*;*
   - $(1+\eta r)^2-1$ *is a square, where* $\eta=\dfrac{y-1}{2(y+1)}$*; and*
   - *if* $\eta r=-2$ *then* $x=2s(c-1)\chi(c)/r$.
3. *If* $(x,y)\in\phi(\mathbf{F}_q)$ *then the following elements* $\bar{X},z,\bar{u},\bar{t}$ *of* $\mathbf{F}_q$ *are defined and* $\phi(\bar{t})=(x,y)$*:*

$$\begin{aligned}\bar{X}&=-(1+\eta r)+((1+\eta r)^2-1)^{(q+1)/4},\\z&=\chi\big((c-1)s\bar{X}(1+\bar{X})x(\bar{X}^2+1/c^2)\big),\\\bar{u}&=z\bar{X},\\\bar{t}&=(1-\bar{u})/(1+\bar{u}).\end{aligned}$$

Proof. Statement 1 of the theorem has two parts: a forward statement $\phi(t)=\phi(-t)$, and a reverse statement that there are no other preimages. Statement 2 also has two parts: a forward statement that any $(x,y)\in\phi(\mathbf{F}_q)$ satisfies certain conditions, and a reverse statement that any element of $E(\mathbf{F}_q)$ satisfying those conditions is in $\phi(\mathbf{F}_q)$. We organize the proof as (A) forward 1; (B) 3, forward 2, and reverse 1; (C) reverse 2.

A. Fix $t\in\mathbf{F}_q$. We now show that $\phi(t)=\phi(-t)$. This is the forward part of statement 1 in the theorem.

If $t\in\{\pm1\}$ then $\phi(t)=(0,1)$ and $\phi(-t)=(0,1)$ by definition, so $\phi(t)=\phi(-t)$. Assume from now on that $t\notin\{\pm1\}$.

Define u,v,X,Y,x,y from t as in Theorem 1. Then $\phi(t)=(x,y)$ by definition.

Put $t'=-t$, and define u',v',X',Y',x',y' the same way from t'. Then $\phi(t')=(x',y')$. The proof strategy is to compare successively u' to u, v' to v, etc., concluding that $x'=x$ and $y'=y$.

$\boldsymbol{u'}=(1-t')/(1+t')=(1+t)/(1-t)=1/u$.

$\boldsymbol{v'}=u'^5+(r^2-2)u'^3+u'=\frac{1}{u^5}+(r^2-2)\frac{1}{u^3}+\frac{1}{u}$, so $v'u^6=u+(r^2-2)u^3+u^5=v$; i.e., $v'=v/u^6$. Note that $\chi(v')=\chi(v)$ since $\chi(u^6)=1$.

$\boldsymbol{X'}=\chi(v')u'=\chi(v)/u=1/(\chi(v)u)=1/X$ since $\chi(v)=1/\chi(v)$.

$$\boldsymbol{y'}=\frac{rX'-(1+X')^2}{rX'+(1+X')^2}=\frac{r\frac{1}{X}-(1+\frac{1}{X})^2}{r\frac{1}{X}+(1+\frac{1}{X})^2}=\frac{rX-(X+1)^2}{rX+(X+1)^2}=y.$$

$\boldsymbol{Y'}=(\chi(v')v')^{(q+1)/4}\chi(v')\chi\big(u'^2+1/c^2\big)$. This takes the most work; the first and third factors each need careful analyses. The second factor is easy: $\chi(v')=\chi(v)$ as above.

First factor: $\chi(v')v'=\chi(v)v/u^6$. Note that the product $\chi(u)u^3$ is a square and is therefore the principal square root of u^6; i.e., $(u^6)^{(q+1)/4}=\chi(u)u^3$. Hence $(\chi(v')v')^{(q+1)/4}=(\chi(v)v/u^6)^{(q+1)/4}=(\chi(v)v)^{(q+1)/4}\chi(u)/u^3$.

Third factor: Recall that $v=u(u^2+c^2)(u^2+1/c^2)$ and that $\chi(a)=\chi(ab^2)$ for any $b\neq 0$. Thus

$$\begin{aligned}\chi\big(u'^2+1/c^2\big)&=\chi\big(c^2u^4(u'^2+1/c^2)(u^2+1/c^2)^2\big)\\&=\chi\big(u^2(c^2+u^2)(u^2+1/c^2)^2\big)\\&=\chi\big(uv(u^2+1/c^2)\big).\end{aligned}$$

Now multiply to obtain $Y'=Y\chi(u)\chi(uv)/u^3=Y/(\chi(v)u)^3=Y/X^3$.

Finally

$$\begin{aligned}\boldsymbol{x'}&=(c-1)sX'(1+X')/Y'=(c-1)s\frac{1}{X}\left(1+\frac{1}{X}\right)\bigg/\frac{Y}{X^3}\\&=(c-1)sX(1+X)/Y=x.\end{aligned}$$

Hence $\phi(-t)=(x',y')=(x,y)=\phi(t)$ as claimed.

B. Fix $t\in\mathbf{F}_q$, and define $(x,y)=\phi(t)$. We show that $\bar{X},z,\bar{u},\bar{t}$ in the theorem are defined and that $\bar{t}\in\{t,-t\}$, so $\phi(\bar{t})=(x,y)$; this is statement 3 in the theorem. We also show the forward part of statement 2: namely, $y+1\neq 0$; $(1+\eta r)^2-1$ is a square, where $\eta=(y-1)/(2(y+1))$; and if $\eta r=-2$ then $x=2s(c-1)\chi(c)/r$. We also show the reverse part of statement 1: there are no preimages of $\phi(t)$ other than t and $-t$.

The definition of ϕ has two cases: if $t\in\{1,-1\}$ then $(x,y)=(0,1)$; if $t\notin\{1,-1\}$ then u,v,X,Y,x,y are defined in Theorem 1. Note that in the second case $x\neq 0$ by Theorem 1, so in particular t is not a preimage of $(0,1)$.

In the first case $y+1=2\neq 0$; $\eta=0$; $(1+\eta r)^2-1=0$ is a square; $\bar{X}=-1$; $z=0$; $\bar{u}=0$; and $\bar{t}=1\in\{t,-t\}$. As noted above, 1 and -1 are the only preimages of $(0,1)$.

What remains is the second case. Here $y+1\neq 0$ by Theorem 1. The equation $y=(rX-(1+X)^2)/(rX+(1+X)^2)$ implies $X^2+(2+r(y-1)/(y+1))X+1=0$, i.e., $X^2+2(1+\eta r)X+1=0$. Note that this forces the discriminant $4(1+\eta r)^2-4$ to be a square; i.e., $(1+\eta r)^2-1$ is a square. Divide by X to see that $X+1/X=-2(1+\eta r)$.

If $\eta r=-2$ then $(X-1)^2=0$ so $X=1$ so $u\in\{\pm1\}$; the case $u=-1$ forces $1-t=-(1+t)$, contradiction, so $u=1$ and $t=0$; so $v=r^2$, so $Y=(r^2)^{(q+1)/4}\chi(1+1/c^2)=\chi(r)r\chi(r/c)=r\chi(c)$, so $x=2(c-1)s\chi(c)/r$ as claimed; also note for future reference that $y=(r-4)/(r+4)$, i.e., $\phi(0)=(2(c-1)s\chi(c)/r,(r-4)/(r+4))$.

Define $t'=-t$, and define u',v',X',Y',x',y' as in Part A of this proof. Then $X'=1/X$, so $X+X'=-2(1+\eta r)$.

By construction $1+\eta r+\bar{X}$ is a square root of $(1+\eta r)^2-1$; i.e., $\bar{X}^2+2(1+\eta r)\bar{X}+1=0$. Now $(\bar{X}-X)(\bar{X}-X')=\bar{X}^2-(X+X')\bar{X}+XX'=\bar{X}^2+2(1+\eta r)\bar{X}+1=0$ so

$\bar{X} = X$ or $\bar{X} = X'$. This forces $\bar{u} = u$ or $\bar{u} = u'$, since the definition of z turns out to match $\chi(v)$ and $\chi(v')$:

- If $\bar{X} = X$ then $(c-1)s\bar{X}(1+\bar{X}) = xY$ so $z = \chi\big(x^2Y(X^2+1/c^2)\big) = \chi(Y)\chi\big(X^2+1/c^2\big)$. Note that $(\chi(v)v)^{(q+1)/4}$ is a square and $\chi\big(u^2+1/c^2\big) = \chi\big(X^2+1/c^2\big)$, so $\chi(Y) = \chi(v)\chi(X^2+1/c^2)$, so $z = \chi(v)$, so $\bar{u} = \chi(v)X = u$, so $\bar{t} = t$.
- If $\bar{X} = X'$ then similarly $z = \chi(v')$, $\bar{u} = u'$, and $\bar{t} = t' = -t$.

To summarize, $\bar{t} \in \{t, -t\}$, so $\phi(\bar{t}) = (x,y)$.

The same logic also shows that there are no preimages p of (x,y) except for t and $-t$. Indeed, if $(x,y) = \phi(p)$ then substituting p for t in the same proof shows that $\bar{t} \in \{p, -p\}$, so $p \in \{\bar{t}, -\bar{t}\} = \{t, -t\}$.

C. Fix $(x,y) \in E(\mathbf{F}_q)$. Assume that $y+1 \neq 0$; that $(1+\eta r)^2 - 1$ is a square, where $\eta = (y-1)/(2(y+1))$; and that if $\eta r = -2$ then $x = 2s(c-1)\chi(c)/r$. We now show that $(x,y) \in \phi(\mathbf{F}_q)$. This is the reverse part of statement 2 of the theorem.

If $x = 0$ then $(x,y) = (0, \pm 1)$ from the curve equation; but $y+1 \neq 0$, so $(x,y) = (0,1) = \phi(1) \in \phi(\mathbf{F}_q)$ as claimed. Assume from now on that $x \neq 0$.

If $y = 1$ then $x = 0$ from the curve equation, contradiction. Hence $y \neq 1$; i.e., $\eta \neq 0$.

Define $X = -(1+\eta r) + ((1+\eta r)^2 - 1)^{(q+1)/4}$. As above $1+\eta r+X$ is a square root of $(1+\eta r)^2-1$, so $X^2+2(1+\eta r)X+1 = 0$. This quadratic equation has several consequences. First, $X \neq 0$. Second, $rX + (1+X)^2 \neq 0$: otherwise subtract to see that $(1-2\eta)rX = 0$, so $1 = 2\eta$, so $y-1 = y+1$, contradiction. Third, $X \neq -1$: otherwise $\eta = 0$, contradiction. Fourth, $y = (rX - (1+X)^2)/(rX + (1+X)^2)$.

If $X = 1$ then $y = (r-4)/(r+4)$; also $\eta r = -2$ so by assumption $x = 2s(c-1)\chi(c)/r$ so $(x,y) = \phi(0) \in \phi(\mathbf{F}_q)$. Assume from now on that $X \neq 1$.

Observe that

$$\begin{aligned}&(rX + (1+X)^2)^2(1-y^2)\\ &\quad = (rX + (1+X)^2)^2 - (rX - (1+X)^2)^2\\ &\quad = 4rX(1+X)^2.\end{aligned}$$

Recall that $-d = (r+2)/(r-2)$ and similarly observe that

$$\begin{aligned}&(rX + (1+X)^2)^2(1-dy^2)\\ &\quad = (rX + (1+X)^2)^2 + \frac{r+2}{r-2}(rX - (1+X)^2)^2\\ &\quad = (2r/(r-2))(X^4 + (r^2-2)X^2 + 1).\end{aligned}$$

Note that $1 - dy^2 \neq 0$ since d is not a square. Divide:

$$x^2 = \frac{1-y^2}{1-dy^2} = \frac{2(r-2)X^2(1+X)^2}{X^5 + (r^2-2)X^3 + X}.$$

Define $Y = (c-1)sX(1+X)/x$. Then

$$\begin{aligned}Y^2 &= (c-1)^2s^2X^2(1+X)^2/x^2\\ &= 2(r-2)X^2(1+X)^2/x^2\\ &= X^5 + (r^2-2)X^3 + X.\end{aligned}$$

Define $z = \chi\big(Y(X^2+1/c^2)\big)$. Both Y and X^2+1/c^2 are nonzero, so $z \in \{\pm 1\}$.

Define $u = zX$. Then $u \in \{\pm X\}$. Note that $u \neq -1$, since $X \notin \{\pm 1\}$.

Define $v = u^5 + (r^2-2)u^3 + u$. Then $v = z(X^5 + (r^2-2)X^3 + X) = zY^2$, so $\chi(v) = \chi(z) = z$. Hence $X = \chi(v)u$ and $Y^2 = \chi(v)v$. Furthermore $\chi(v) = z = \chi\big(Y(X^2+1/c^2)\big) = \chi\big(Y(u^2+1/c^2)\big)$, so $\chi(Y) = \chi(v)\chi\big(u^2+1/c^2\big)$, so $Y = (\chi(v)v)^{(q+1)/4}\chi(v)\chi\big(u^2+1/c^2\big)$.

Finally define $t = (1-u)/(1+u)$. Then $t \notin \{\pm 1\}$ and $u = (1-t)/(1+t)$. The formulas for u, v, X, Y, x, y in Theorem 1 are all satisfied, so $(x,y) = \phi(t) \in \phi(\mathbf{F}_q)$ as claimed. $\square$

3.4 Encoding as strings

THEOREM 4. *In the situation of Definition 2, assume that q is prime, and define $b = \lfloor \log_2 q \rfloor$. Define $\sigma : \{0,1\}^b \to \mathbf{F}_q$ by $\sigma(\tau_0, \tau_1, \ldots, \tau_{b-1}) = \sum_i \tau_i 2^i$. Define $S = \sigma^{-1}(\{0,1,2,\ldots,(q-1)/2\})$. Define $\iota : S \to E(\mathbf{F}_q)$ as follows: $\iota(\tau) = \phi(\sigma(\tau))$. Then $\#S = (q+1)/2$; ι is an injective map from S to $E(\mathbf{F}_q)$; and $\iota(S) = \phi(\mathbf{F}_q)$.*

PROOF. First $2^b \leq q$ so the integers $0, 1, \ldots, 2^b - 1$ are distinct in $\mathbf{F}_q$; hence σ is injective. Furthermore $2^b > q/2$ so $\{0,1,\ldots,(q-1)/2\}$ is a subset of $\{0,1,\ldots,2^b-1\}$; hence each of $0,1,\ldots,(q-1)/2$ has a preimage under σ, and S has exactly $(q+1)/2$ elements.

If $\iota(\tau) = \iota(\tau')$ then $\phi(\sigma(\tau)) = \phi(\sigma(\tau'))$, so $\sigma(\tau) = \pm\sigma(\tau')$ by Theorem 3; but $\sigma(\tau)$ and $\sigma(\tau')$ are both in $\{0,1,\ldots,(q-1)/2\}$, so $\sigma(\tau) = \sigma(\tau')$, so $\tau = \tau'$. Hence ι is injective.

Each element of $\iota(S)$ has the form $\phi(\sigma(\tau))$ and is therefore in $\phi(\mathbf{F}_q)$. Conversely, if $P \in \phi(\mathbf{F}_q)$ then $P = \phi(t)$ for some $t \in \mathbf{F}_q$, so also $P = \phi(-t)$ by Theorem 3. At least one of $t, -t$ is in $\{0,1,\ldots,(q-1)/2\}$, i.e., in $\sigma(S)$, so P is in $\phi(\sigma(S)) = \iota(S)$. $\square$

3.5 Performance analysis

The definitions of u, v, X, Y, x, y in Theorem 1 involve divisions by $1+t$, c, Y, and $rX + (1+X)^2$. The reciprocal of c is trivially precomputed, and the other divisions are easily replaced by a few multiplications: one simply stores field elements as fractions, i.e., works in projective coordinates. There are several easy ways to reduce the number of multiplications: for example, factor $u^5 + (r^2-2)u^3 + u$ as $u(u^2+c^2)(u^2+1/c^2)$, and reuse u^2+1/c^2 in computing Y.

The main bottlenecks are then the following exponentiations: computing $\chi(v)$ (used repeatedly); computing $\chi\big(u^2+1/c^2\big)$; computing $(\chi(v)v)^{(q+1)/4}$, the principal square root of $\chi(v)v$; and computing a final division if the output (x,y) is needed in affine coordinates instead of projective coordinates. The only essential exponentiation is for the square-root computation: the χ computations and division can use Euclid's algorithm (blinded to protect against timing attacks) rather than exponentiation.

Similar comments apply to inverting ϕ (or ι). There is one essential exponentiation, the square-root computation to obtain $\bar{X}$ in Theorem 3. There are also two exponentiations that can be replaced by Euclidean computations: one χ computation to obtain z, and one division to obtain the final output $\bar{t}$. Fractions eliminate the initial division by $2(y+1)$, but fractions cannot be used for $\bar{t}$, since the goal is to obtain the unique string representing $\bar{t}$.

It is easier to test, given $(x,y) \in \mathbf{F}_q \times \mathbf{F}_q$, whether $(x,y) \in \phi(\mathbf{F}_q)$ (i.e., whether $(x,y) \in \iota(S)$) without inverting ϕ. One first checks $x^2 + y^2 = 1 + dx^2y^2$ to verify $(x,y) \in E(\mathbf{F}_q)$, if this is not already known. Then, by Theorem 3, $(x,y) \in \phi(\mathbf{F}_q)$ if and only if the following three conditions are satisfied:

- $y + 1 \neq 0$;
- $(1 + \eta r)^2 - 1$ is a square, where $\eta = (y-1)/(2(y+1))$; i.e., $r(y-1)(r(y-1) + 4(y+1))$ is a square;
- if $\eta r = -2$ (equivalently, if $r(y-1) = -4(y+1)$) then $x = 2s(c-1)\chi(c)/r$.

This requires a few multiplications and one χ computation.

4. CONSTRUCTION OF A SUITABLE ELLIPTIC CURVE

This section introduces Curve1174, a high-security high-speed elliptic curve that supports the injective map presented in Section 3. In particular, this section specifies Curve1174; presents the criteria that we used to construct Curve1174; and analyzes the extent to which various previous curves meet the same criteria.

4.1 The curve

Curve1174 is the Edwards curve $x^2 + y^2 = 1 - 1174x^2y^2$ over the field $\mathbf{F}_q$, where q is the prime number $2^{251} - 9$. The coefficient -1174 is a non-square in $\mathbf{F}_q$, so Curve1174 is a complete Edwards curve by [7, Theorem 3.3]: the sum of any two points (x_1, y_1) and (x_2, y_2) in Curve1174$(\mathbf{F}_q)$ is

$$\left(\frac{x_1y_2 + y_1x_2}{1 - 1174x_1x_2y_1y_2}, \frac{y_1y_2 - x_1x_2}{1 + 1174x_1x_2y_1y_2} \right),$$

with no divisions by 0 and no exceptional cases. The neutral element of the curve is $(0, 1)$.

To see that we have the desired injective map, note that q is congruent to 3 modulo 4; define s as the element

$$1806494121122717992522804053500797229648438766985538871240722010849934886421$$

of $\mathbf{F}_q$ (split onto two lines here for readability); define $c = 2/s^2$; and define $d = -(c+1)^2/(c-1)^2$. Then $d = -1174$. The Edwards curve in Theorem 1 and Definition 2, for this choice of (q, s), is exactly Curve1174.

Curve1174 is birationally equivalent to the Montgomery curve $(4/1175)V^2 = U^3 + (4/1175-2)U^2 + U$ by [7, Theorem 3.2]. The leading coefficient $4/1175$ is a non-square in $\mathbf{F}_q$, so $V^2 = U^3 + (4/1175 - 2)U^2 + U$ is a nontrivial quadratic twist of this curve. The Sage computer-algebra system [41] counts points on this curve in under 10 seconds on a laptop using the following script:

```
q=2^251-9
E=EllipticCurve(GF(q),[0,4/1175-2,0,1,0])
print E.trace_of_frobenius()
# output -45330879683285730139092453152713398836
```

The number of points on the twist is thus $q + 1 + t$, and the number of points on Curve1174 is $q + 1 - t$, where $t =$ 45330879683285730139092453152713398836. These integers $q+1+t, q+1-t$ have the form $4p_0, 4p_1$ respectively, where p_0 and p_1 are primes close to 2^{249}. Generic methods to compute a discrete logarithm on Curve1174 or its twist take approximately $\sqrt{\pi 2^{247}} \approx 2^{124.3}$ group operations on average.

The point $(U, V) = (4, 1922577764211167023040871244220551478340301270840905838377461328496334409 6)$ on the Montgomery curve $(4/1175)V^2 = U^3 + (4/1175-2)U^2 + U$ has order $4p_1$. The corresponding point on Curve1174 is $(x, y) = (4/V, 3/5)$.

Curve1174 and its twist do not have any structure allowing fast pairings or other special approaches to computing discrete logarithms. The primes p_0 and p_1 do not equal the field characteristic q. The order of q modulo p_0 is not small: it is $(p_0 - 1)/2$. The order of q modulo p_1 is not small: it is $p_1 - 1$. The endomorphism ring of Curve1174 has a large discriminant: up to squares this discriminant equals $t^2 - 4q$, which is divisible once by the prime 1615674151140249923338703492557 99, so the discriminant must be a multiple of this prime.

4.2 Design criteria

We consider only prime fields. Bernstein, citing subfield attacks from [25] and [15], wrote in [4] that prime fields "have the virtue of minimizing the number of security concerns for elliptic-curve cryptography"; see [19] and [38] for recent developments of the attack strategy from [15]. Similarly, the Brainpool standard [11] and NSA's Suite B standards [32] require prime-field ECC. There is general agreement that prime fields are the safe, conservative choice for ECC. Prime fields also perform very well across a wide range of processors; the current ECC speed records on high-end Intel processors take advantage of special Intel support for binary fields (see [37]), but most CPUs do not have any comparable support.

We consider only primes q congruent to 3 modulo 4. This is required for the injective map. These primes also have the well-known benefit of allowing very simple square-root computations; most other primes allow square-root computations at about the same *speed* but with more complicated methods.

We consider only complete Edwards curves, i.e., curves $x^2 + y^2 = 1 + dx^2y^2$ where d is not a square. This is required for the injective map. About 25% of all elliptic curves over $\mathbf{F}_q$ are expressible as complete Edwards curves, as mentioned in [7, Abstract] and experimentally verified in [5, Section 4]. Complete Edwards curves also have the advantages of being extremely fast and of allowing a single addition formula with no exceptions. Complete Edwards curves are also expressible as Montgomery curves supporting very fast and uniform Montgomery-ladder computations.

To protect against generic discrete-logarithm algorithms we impose the standard requirement of a large prime dividing the number of curve points. This forces q to be even larger, where the gap accounts for the cofactor: the number of curve points divided by this prime. To minimize the performance problems of a large q we consider only Edwards curves with minimal cofactor, namely 4. The number of curve points is 4 times a prime for slightly below 1% of all choices of d, for the size of q that we consider below.

We also impose the requirement of "twist security": a large prime dividing the number of points on the quadratic twist of the curve. This prevents "twist attacks" against protocols that use the "Montgomery ladder" without checking that incoming points are on the curve; [4, Section 3] says that this defense was proposed by Bernstein in 2001. For $q \equiv 3 \pmod 4$ roughly 1/10000 of all choices of d have the number of points on the curve and the number of points on the twist each being 4 times a prime.

We require d to have the form $-(c+1)^2/(c-1)^2$ with $c = 2/s^2$. This is required for the injective map. For $q \equiv 3 \pmod 4$ about half of all non-squares d are expressible in this way.

For standard performance reasons we take q very close to, but not above, a power of 2. The primes $q \equiv 3 \pmod 4$ within 32 of 2^e for $200 \le e \le 300$ are $2^{206} - 5$, $2^{212} - 29$, $2^{226} - 5$, $2^{243} - 9$, $2^{251} - 9$, and $2^{285} - 9$. Note that these fields ensure that σ^{-1} covers nearly all of $\{0, 1, \ldots, 2^b - 1\}$ giving a very close to uniform distribution of the encoding function. We focus on the last two of these primes as providing quantitatively safe security levels, and choose $2^{251} - 9$ as being obviously faster.

Some Edwards-curve operations involve multiplications by d. To speed up these multiplications we take the smallest possible d in absolute value, subject to the other requirements. The choice $d = -1174$ for $q = 2^{251} - 9$ is smaller than expected.

4.3 Previous curves over prime fields

There is a long history of specific elliptic curves being designed to meet various security and performance criteria. For example, almost fifteen years ago the IEEE P1363 standard [30, Sections A.9–A.12]

- specified curves $y^2 = x^3 - 3x + b$ to "provide the fastest arithmetic on elliptic curves";
- imposed various further conditions upon these curves, with the security goal of making discrete logarithms difficult to compute; and
- specified a procedure to generate "verifiably pseudo-random" curves meeting these conditions.

NIST's standard curves P-192, P-224, P-256, P-384, and P-521 were generated as follows: five particular prime fields were chosen with the goal of maximizing performance; the IEEE P1363 procedure was used to generate one curve over each of those fields.

Subsequent research developed new security and performance criteria for curves over prime fields: twist security, Montgomery compatibility, Edwards compatibility, and completeness. The NIST curves, unsurprisingly, flunk these criteria: choosing cofactor 1 for $y^2 = x^3 - 3x + b$ is incompatible with both Montgomery and Edwards, and one cannot expect twist security if it is not demanded in advance. Newer curves meet all of these criteria: for example, Curve25519 was explicitly designed for twist security and Montgomery compatibility, and was shown in [7] to also be expressible as a complete Edwards curve.

These extra criteria do not improve discrete-logarithm security, but they do improve real-world security. These criteria allow the *simplest* implementations to be *correct* implementations, whereas for other curves the simplest implementations that seem to work actually have hidden flaws that compromise security. See, e.g., [9, Section 4.1], [31], and [21].

We are imposing a new security condition to support censorship circumvention: namely, an efficient way to encode a large fraction of all curve points as strings indistinguishable from uniform random strings. It is easy to imagine how this condition could be accidentally met by previously generated curves:

- The advantages of complete Edwards curves have been well known for five years.
- The advantages of Montgomery curves have been well known for even longer. A random Montgomery curve has a good chance of being expressible as a complete Edwards curve; see [5, Section 4].
- All complete Edwards curves over $\mathbf{F}_q$ for $q \equiv 3 \pmod 4$ meet the new security condition. Half of these curves are within the streamlined case expressed by Theorem 1.

Given the amount of performance optimization of Curve25519 (see [4], [26], [14], [6], and [8]) and the wide deployment of Curve25519 in several applications (see, e.g., [2]) we were hoping that this condition would be met by Curve25519. However, our approach to encoding is clearly limited to $q \equiv 3 \pmod 4$, while Curve25519 is defined over $\mathbf{F}_q$ with $q \equiv 1 \pmod 4$.

We would expect serious implementations of Curve1174 to be competitive in speed with Curve25519. (Curve1174 has some small advantages: for example, $2^{251} - 9$ is closer to a power of 2 than $2^{255} - 19$ is, and 1174 is considerably smaller than 486662. On the other hand, Curve25519 also has a small advantage: it is expressible in "-1-twisted Edwards form", allowing the speedup explained in [28].) The critical advantage of Curve1174 is the injective map presented in Section 3.

5. REFERENCES

[1] Appelbaum, J., and Dingledine, R. How governments have tried to block Tor, 2011. `http://ftp.ccc.de/congress/28C3/mp4-h264-HQ/28c3-4800-en-how_governments_have_tried_to_block_tor_h264.mp4`.

[2] Apple. iOS security, 2012. `http://images.apple.com/iphone/business/docs/iOS_Security_Oct12.pdf`.

[3] Bellare, M., Desai, A., Jokipii, E., and Rogaway, P. A concrete security treatment of symmetric encryption. In *FOCS* (1997), IEEE Computer Society, pp. 394–403.

[4] Bernstein, D. J. Curve25519: New Diffie-Hellman speed records. In *Public Key Cryptography* (2006), M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958 of *Lecture Notes in Computer Science*, Springer, pp. 207–228.

[5] Bernstein, D. J., Birkner, P., Joye, M., Lange, T., and Peters, C. Twisted Edwards curves. In *AFRICACRYPT* (2008), S. Vaudenay, Ed., vol. 5023 of *Lecture Notes in Computer Science*, Springer, pp. 389–405.

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

[7] Bernstein, D. J., and Lange, T. Faster addition and doubling on elliptic curves. In *ASIACRYPT* (2007), K. Kurosawa, Ed., vol. 4833 of *Lecture Notes in Computer Science*, Springer, pp. 29–50.

[8] Bernstein, D. J., and Schwabe, P. NEON crypto. In *CHES* (2012), E. Prouff and P. Schaumont, Eds., vol. 7428 of *Lecture Notes in Computer Science*, Springer, pp. 320–339.

[9] Biehl, I., Meyer, B., and Müller, V. Differential fault attacks on elliptic curve cryptosystems. In

CRYPTO (2000), M. Bellare, Ed., vol. 1880 of *Lecture Notes in Computer Science*, Springer, pp. 131–146.

[10] Boneh, D., and Franklin, M. K. Identity-based encryption from the Weil pairing. In *CRYPTO* (2001), J. Kilian, Ed., vol. 2139 of *Lecture Notes in Computer Science*, Springer, pp. 213–229.

[11] Brainpool. ECC Brainpool standard curves and curve generation, v. 1.0, 2005. `http://www.ecc-brainpool.org/download/Domain-parameters.pdf`.

[12] Brier, E., Coron, J.-S., Icart, T., Madore, D., Randriam, H., and Tibouchi, M. Efficient indifferentiable hashing into ordinary elliptic curves. In *CRYPTO* (2010), T. Rabin, Ed., vol. 6223 of *Lecture Notes in Computer Science*, Springer, pp. 237–254.

[13] Brier, E., and Joye, M. Weierstraß elliptic curves and side-channel attacks. In *Public Key Cryptography* (2002), D. Naccache and P. Paillier, Eds., vol. 2274 of *Lecture Notes in Computer Science*, Springer, pp. 335–345.

[14] Costigan, N., and Schwabe, P. Fast elliptic-curve cryptography on the Cell Broadband Engine. In *AFRICACRYPT* (2009), B. Preneel, Ed., vol. 5580 of *Lecture Notes in Computer Science*, Springer, pp. 368–385.

[15] Diem, C. The GHS attack in odd characteristic. *Journal of the Ramanujan Mathematical Society 18* (2003), 1–32. `http://www.math.uni-leipzig.de/~diem/preprints`.

[16] Dingledine, R., Mathewson, N., and Syverson, P. F. Tor: The second-generation onion router. In *USENIX Security Symposium* (2004), USENIX, pp. 303–320.

[17] Farashahi, R. R. Hashing into Hessian curves. In *AFRICACRYPT* (2011), A. Nitaj and D. Pointcheval, Eds., vol. 6737 of *Lecture Notes in Computer Science*, Springer, pp. 278–289.

[18] Farashahi, R. R., Fouque, P.-A., Shparlinski, I., Tibouchi, M., and Voloch, J. F. Indifferentiable deterministic hashing to elliptic and hyperelliptic curves. *Math. Comput. 82*, 281 (2013).

[19] Faugère, J.-C., Perret, L., Petit, C., and Renault, G. Improving the complexity of index calculus algorithms in elliptic curves over binary fields. In *EUROCRYPT* (2012), D. Pointcheval and T. Johansson, Eds., vol. 7237 of *Lecture Notes in Computer Science*, Springer, pp. 27–44.

[20] Fouque, P.-A., Joux, A., and Tibouchi, M. Injective encodings to elliptic curves. `http://www.di.ens.fr/~fouque/pub/injenc.pdf`, 2012.

[21] Fouque, P.-A., Lercier, R., Réal, D., and Valette, F. Fault attack on elliptic curve Montgomery ladder implementation. In *FDTC* (2008), L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, Eds., IEEE Computer Society, pp. 92–98.

[22] Fouque, P.-A., and Tibouchi, M. Deterministic encoding and hashing to odd hyperelliptic curves. In *Pairing* (2010), M. Joye, A. Miyaji, and A. Otsuka, Eds., vol. 6487 of *Lecture Notes in Computer Science*, Springer, pp. 265–277.

[23] Fouque, P.-A., and Tibouchi, M. Estimating the size of the image of deterministic hash functions to elliptic curves. In *LATINCRYPT* (2010), M. Abdalla and P. S. L. M. Barreto, Eds., vol. 6212 of *Lecture Notes in Computer Science*, Springer, pp. 81–91.

[24] Fouque, P.-A., and Tibouchi, M. Indifferentiable hashing to Barreto-Naehrig curves. In *LATINCRYPT* (2012), A. Hevia and G. Neven, Eds., vol. 7533 of *Lecture Notes in Computer Science*, Springer, pp. 1–17.

[25] Frey, G. How to disguise an elliptic curve (Weil descent), 1998. `http://www.cacr.math.uwaterloo.ca/conferences/1998/ecc98/slides.html`.

[26] Gaudry, P., and Thomé, E. The mpFq library and implementing curve-based key exchanges. In *SPEED: software performance enhancement for encryption and decryption* (2007), pp. 49–64.

[27] Goldberg, I., Stebila, D., and Ustaoglu, B. Anonymity and one-way authentication in key exchange protocols. *Des. Codes Cryptography 67*, 2 (2013), 245–269.

[28] Hisil, H., Wong, K. K.-H., Carter, G., and Dawson, E. Twisted Edwards curves revisited. In *ASIACRYPT* (2008), J. Pieprzyk, Ed., vol. 5350 of *Lecture Notes in Computer Science*, Springer, pp. 326–343.

[29] Icart, T. How to hash into elliptic curves. In *CRYPTO* (2009), S. Halevi, Ed., vol. 5677 of *Lecture Notes in Computer Science*, Springer, pp. 303–316.

[30] Institute of Electrical and Electronics Engineers. *P1363 Draft Standard Specifications for Public Key Cryptography.* IEEE, 1999.

[31] Izu, T., and Takagi, T. Exceptional procedure attack on elliptic curve cryptosystems. In *Public Key Cryptography* (2003), Y. Desmedt, Ed., vol. 2567 of *Lecture Notes in Computer Science*, Springer, pp. 224–239.

[32] Law, L. E., and Solinas, J. A. Suite B cryptographic suites for IPsec, 2011. `https://tools.ietf.org/html/rfc6379`.

[33] Lidl, R., and Niederreiter, H. *Finite Fields.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1997.

[34] López, J., and Dahab, R. Fast multiplication on elliptic curves over GF(2^{m}) without precomputation. In *CHES* (1999), Çetin Kaya Koç and C. Paar, Eds., vol. 1717 of *Lecture Notes in Computer Science*, Springer, pp. 316–327.

[35] Möller, B. A public-key encryption scheme with pseudo-random ciphertexts. In *Computer Security—ESORICS 2004*, P. Samarati, P. Ryan, D. Gollmann, and R. Molva, Eds., vol. 3193 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, pp. 335–351.

[36] Montgomery, P. L. Speeding the Pollard and elliptic curve methods of factorization. *Math. Comp. 48*, 177 (1987), 243–264.

[37] Oliveira, T., López, J., Aranha, D. F., and Rodríguez-Henríquez, F. Two is the fastest prime. *IACR Cryptology ePrint Archive 2013* (2013), 131. `http://eprint.iacr.org/2013/131`.

[38] Petit, C., and Quisquater, J.-J. On polynomial systems arising from a Weil descent. In *ASIACRYPT* (2012), X. Wang and K. Sako, Eds., vol. 7658 of *Lecture Notes in Computer Science*, Springer, pp. 451–466.

[39] Rogaway, P., and Shrimpton, T. A provable-security treatment of the key-wrap problem. In *EUROCRYPT* (2006), S. Vaudenay, Ed., vol. 4004 of *Lecture Notes in Computer Science*, Springer, pp. 373–390.

[40] Shallue, A., and van de Woestijne, C. Construction of rational points on elliptic curves over finite fields. In *ANTS* (2006), F. Hess, S. Pauli, and M. E. Pohst, Eds., vol. 4076 of *Lecture Notes in Computer Science*, Springer, pp. 510–524.

[41] Stein, W., et al. *Sage Mathematics Software (Version 2.9)*. The Sage Development Team, 2013. http://www.sagemath.org.

[42] Weinberg, Z., Wang, J., Yegneswaran, V., Briesemeister, L., Cheung, S., Wang, F., and Boneh, D. StegoTorus: a camouflage proxy for the Tor anonymity system. In *ACM Conference on Computer and Communications Security* (2012), T. Yu, G. Danezis, and V. D. Gligor, Eds., ACM, pp. 109–120.

[43] Wustrow, E., Wolchok, S., Goldberg, I., and Halderman, J. A. Telex: Anticensorship in the network infrastructure. In *USENIX Security Symposium* (2011), USENIX Association.

APPENDIX

A. PYTHON IMPLEMENTATION

The following Python script contains reference implementations of ι and ι^{-1}, illustrated for a 6-bit encoding of 64 different points on a complete Edwards curve over $\mathbf{F}_{127}$.

```
q = 127
s = 37

def expmod(b,e,m):
  if e == 0: return 1
  t = expmod(b,e/2,m)**2 % m
  if e & 1: t = (t*b) % m
  return t

def inv(x):
  return expmod(x,q-2,q)

def chi(x):
  return expmod(x,(q-1)/2,q)

def issquare(x):
  return chi(x) == 1 or chi(x) == 0

def sqrt(x):
  return expmod(x,(q+1)/4,q)

b = 1
while 1<<(b+1) <= q: b = b + 1
c = (2*inv(s*s)) % q
r = (c+inv(c)) % q
d = (-(c+1)*(c+1)*inv((c-1)*(c-1))) % q

def decode(tau):
  # simple reference implementation of iota
  t = sum(tau[i]<<i for i in range(b))
  if t > (q-1)/2: return
  if t == 1: return (0,1)
  u = ((1-t)*inv(1+t)) % q
  v = (u*(u*u+c*c)*(u*u+inv(c*c))) % q
  X = (chi(v)*u) % q
  Y = (sqrt(chi(v)*v)*chi(v)*chi(u*u+inv(c*c))) % q
  x = ((c-1)*s*X*(1+X)*inv(Y)) % q
  y = ((r*X-(1+X)*(1+X))*inv(r*X+(1+X)*(1+X))) % q
  return (x,y)

def encode(P):
  # simple reference implementation of iota^(-1)
  x,y = P
  if (x*x+y*y-1-d*x*x*y*y) % q != 0: return
  if (y+1) % q == 0: return
  eta = ((y-1)*inv(2*(y+1))) % q
  if not issquare((1+eta*r)*(1+eta*r)-1): return
  if (eta*r+2) % q == 0:
    if (x-2*s*(c-1)*chi(c)*inv(r)) % q != 0: return
  X = (-(1+eta*r)+sqrt((1+eta*r)*(1+eta*r)-1)) % q
  z = chi((c-1)*s*X*(1+X)*x*(X*X+inv(c*c)))
  u = (z*X) % q
  t = ((1-u)*inv(1+u)) % q
  if t > (q-1)/2: t = q - t
  return tuple([(t >> i) & 1 for i in range(b)])

def checkencoding():
  numpoints = 0
  for t in range(1<<b):
    tau = tuple([(t >> i) & 1 for i in range(b)])
    P = decode(tau)
    if P:
      numpoints += 1
      if encode(P) != tau:
        raise Exception("encoding fails")
  if numpoints != (q+1)/2:
    raise Exception("wrong number of points")
  for x in range(q):
    for y in range(q):
      if (x*x+y*y-1-d*x*x*y*y) % q == 0:
        P = (x,y)
        tau = encode(P)
        if tau:
          if decode(tau) != P:
            raise Exception("decoding fails")

def printencoding():
  for t in range(1<<b):
    tau = tuple([(t >> i) & 1 for i in range(b)])
    print tau,decode(tau)

checkencoding()
printencoding()
```