How to Hash into Elliptic Curves*

Thomas Icart[1,2]

[1]Sagem Sécurité
[2]Université du Luxembourg
thomas.icart@m4x.org

Abstract. We describe a new explicit function that given an elliptic curve E defined over $\mathbb{F}_{p^n}$, maps elements of $\mathbb{F}_{p^n}$ into E in *deterministic* polynomial time and in a constant number of operations over $\mathbb{F}_{p^n}$. The function requires to compute a cube root. As an application we show how to hash *deterministically* into an elliptic curve.

1 Introduction

Some elliptic curve cryptosystems require to hash into an elliptic curve, for instance the Boneh-Franklin identity based encryption scheme [1]. In this scheme, a particular supersingular elliptic curve is used, for which there exists a one-to-one mapping f from the base field $\mathbb{F}_p$ to the curve. This enables to hash using $f(h(m))$ where h is a classical hash function.

Password-based authentication protocols give another context where hashing into an elliptic curve is sometimes required. For instance, the SPEKE (Simple Password Exponential Key Exchange) [6] and the PAK (Password Authenticated Key exchange) [4] protocols both require a hash algorithm to map the password into a point of the curve. However for ordinary curves the classical approach is inherently probabilistic; this implies that the number of operations required to hash the password may depend on the password itself. As shown in [3] this can lead to a timing attack. Therefore, it would be useful to be able to hash into a curve in a constant number of operations.

The first algorithm mapping $\mathbb{F}_{p^n}$ into an elliptic curve in *deterministic* polynomial time was published by Shallue and Woestijne in ANTS 2006 [9]. The algorithm is based on Skalba's equality [12] and uses a modification of the Tonelli-Shanks algorithm for computing square roots; the algorithm runs in time $\mathcal{O}(\log^4 q)$ for any field size $q = p^n$, and in time $\mathcal{O}(\log^3 q)$ when $q = 3 \mod 4$.

In this paper, we describe another algorithm that given any elliptic curve E defined over $\mathbb{F}_{p^n}$, maps elements of $\mathbb{F}_{p^n}$ into E in deterministic polynomial time, when $p^n = 2 \mod 3$. The new algorithm is based on a rational, explicit function from $\mathbb{F}_{p^n}$ to E, which can be implemented in $\mathcal{O}(\log^3 q)$ time and a constant number of operations over $\mathbb{F}_{p^n}$. Our technique is based on computing a cube root and is simpler than the Shallue and Woestijne algorithm.

As an application we show how to hash *deterministically* and efficiently into an elliptic curve. We provide two different constructions. Our first construction is one-way when the underlying hash function is one-way. The second construction additionally achieves collision resistance when the underlying hash function is collision resistant.

* An extended abstract of this paper will appear at CRYPTO 2009. This is the full version.

1.1 Related Works

We give a brief description of existing techniques to hash into elliptic curves. An elliptic curve over a field $\mathbb{F}_{p^n}$ where $p > 3$ is defined by a Weierstrass equation:

$$Y^2 = X^3 + aX + b \tag{1}$$

where a and b are elements of $\mathbb{F}_{p^n}$. Throughout this paper, we note $E_{a,b}$ the curve associated to these parameters. It is well known that the set of points forms a group; we denote by $E_{a,b}(\mathbb{F}_{p^n})$ this group and by N its order. We also note $q = p^n$: in particular $\mathbb{F}_q$ is the field of p^n elements.

'Try-and-Increment' Method. The algorithm is described in [2] and works as follows:

```
Input: $u$ an integer.
Output: $Q$, a point of $E_{a,b}(\mathbb{F}_q)$.
1. For $i = 0$ to $k - 1$
   (a) Set $x = u + i$
   (b) If $x^3 + ax + b$ is a quadratic residue in $\mathbb{F}_q$, then return $Q = (x, (x^3 + ax + b)^{1/2})$
2. end For
3. Return $\perp$
```

Heuristically, the algorithm fails to return a point for a fraction 2^{-k} of the inputs, where k is a security parameter. One drawback of the algorithm is that the number of operations is not constant. Indeed the number of steps of the algorithm depends on the input u: approximately half of the u are encoded within 1 step, one fourth within 2 steps, etc. In practice, if the input u has to remain secret, this can lead to a timing attack.

A simple countermeasure consists in outputting the point Q only after the end of the For loop so that the number of steps remains constant. However even with this countermeasure, the running time is not necessarily constant. Namely, if the Legendre symbol is used to determine whether $x^3 + ax + b$ is a quadratic residue, such operation takes $\mathcal{O}(\log^2 q)$ time using quadratic reciprocity laws but in general the number of operations is not constant and depends on the input u. Alternatively, one can compute the Legendre symbol using an exponentiation:

$$\left(\frac{z}{q}\right) = z^{(q-1)/2}.$$

Then the numbers of operations is constant but the running time for computing the Legendre symbol is now $\mathcal{O}(\log^3 q)$.

To summarize, if we do not use any countermeasure then the average running time is $\mathcal{O}(\log^3 q)$ due to the square root computation which takes $\mathcal{O}(\log^3 q)$ when $q = 3 \mod 4$. If we use a constant number of steps k with $k = \mathcal{O}(\log q)$, while computing the Legendre symbol efficiently, the running time is still $k \cdot \mathcal{O}(\log^2 q) + \mathcal{O}(\log^3 q) = \mathcal{O}(\log^3 q)$; however one might still be vulnerable to a timing attack. Finally, if we want to have a constant number of operations, one can use the exponentiation method to compute the Legendre symbol; however the running time becomes $k \cdot \mathcal{O}(\log^3 q) + \mathcal{O}(\log^3 q) = \mathcal{O}(\log^4 q)$. In this paper, we describe an algorithm with running time $\mathcal{O}(\log^3 q)$ and constant number of operations[1].

[1] In principle, it should be possible to implement the 'try-and-increment' method in constant time and complexity $\mathcal{O}(\log^3 q)$. For this, one should monitor the running time and eventually use dummy operations. However this could be cumbersome to implement in practice.

The 'Twisted' Curves. Another technique consists in using a curve and its twist as suggested in [5]. Given a curve defined by equation (1), one can define the twisted curve of equation

$$cY^2 = X^3 + aX + b$$

where c is a quadratic non-residue in $\mathbb{F}_q$. Then any $x \in \mathbb{F}_q$ is either the abscissa of a point of the original curve or its twist.

One drawback is that a modification of the cryptosystem is required to hide to the adversary which curve is used. In other words, this hashing technique cannot be used as a black box. Another drawback is that it doubles the computation time because the same computations must be performed separately on both curves.

Supersingular Curves. A curve $E_{a,b}$ is called supersingular when $N = q+1$. When $q \neq 1 \mod 3$, the map $x \mapsto x^3$ is a bijection, therefore the curves of equations

$$Y^2 = X^3 + b$$

are supersingular. One can then define the encoding

$$f : u \mapsto ((u^2 - b)^{1/3}, u) \tag{2}$$

and then the hash function

$$H : m \mapsto ((h(m)^2 - b)^{1/3}, h(m))$$

where h is a classical hash function.

However, the discrete logarithm on these curves is much easier than for ordinary curves. Indeed, such curves have an efficient computable pairing which enables to map the discrete logarithm problem onto a finite field; this is the MOV attack [7]. Therefore in order to avoid this attack, much larger parameters must be used. When no pairing operation is required, it is therefore more efficient to use ordinary curves.

The Shallue-Woestijne Algorithm. In ANTS 2006, Andrew Shallue and Christian van de Woestijne have proposed a new algorithm, that generates elliptic curve points in deterministic polynomial time [9].

Let $f(x) = x^3 + ax + b$. The algorithm is based on the Skalba's equality [12]: there exists four maps $X_1(t), X_2(t), X_3(t), X_4(t)$ such that

$$f(X_1(t)) \cdot f(X_2(t)) \cdot f(X_3(t)) = X_4(t)^2.$$

Then in a finite field, for a fixed parameter t, at least one of the $f(X_i(t))$ must be a quadratic residue, which implies that this $X_i(t)$ is an abscissa of a point of the elliptic curve $y^2 = f(x)$.

The computation of $X_1(t), X_2(t), X_3(t), X_4(t)$ and the choice amongst the $X_i(t)$ require to compute square roots in $\mathbb{F}_q$. Computing square roots in $\mathbb{F}_q$ can be done in probabilistic polynomial time using the Tonelli-Shanks algorithm. Thanks to the Skalba equality, the authors of [9] show how to do it deterministically using a modification of the Tonelli-Shanks algorithm, in time $\mathcal{O}(\log^4 q)$. We note that for $q = 3 \mod 4$, computing a square root is simply an exponentiation, which takes $\mathcal{O}(\log^3 q)$. Therefore the Shallue-Woestijne algorithm runs in time $\mathcal{O}(\log^4 q)$ for any field size $q = p^n$, and in time $\mathcal{O}(\log^3 q)$ when $q = 3 \mod 4$.

Using $\boldsymbol{H(m) = h(m).G}$ We note that for most protocols, it is not possible to hash using $H(m) = h(m).G$ where $h(m) \in \mathbb{Z}$ and G is a generator of the group of points of the elliptic curves. Namely in this case, the discrete logarithm of $H(m)$ with respect to G is known, which makes most protocols insecure. For example, it is easy to see that for Boneh-Franklin identity encryption scheme, the attacker can then decrypt any ciphertext. This remains true if we use $H(m) = h_1(m).G_1 + h_2(m).G_2$ or any such linear combination; in this case, the attacker can compute one Boneh-Franklin private key from a set of other private keys by solving a linear system.

2 An explicit Encoding from $\mathbb{F}_q$ to $E(\mathbb{F}_q)$

We consider the curve $E_{a,b} : Y^2 = X^3 + aX + b$ over the field $\mathbb{F}_{p^n}$ where $p > 3$ and $p^n = 2 \mod 3$. In these finite fields, the function

$$x \mapsto x^3$$

is a bijection with inverse function

$$x \mapsto x^{1/3} = x^{(2p^n-1)/3}.$$

This enables to create a simple parametrization of a subset of the elliptic-curve $E_{a,b}(\mathbb{F}_{p^n})$. To our knowledge, this parametrization is new. Let

$$\begin{aligned} f_{a,b} : \mathbb{F}_{p^n} &\mapsto E_{a,b} \\ u &\mapsto (x, y) \end{aligned}$$

where

$$\begin{aligned} x &= \left(v^2 - b - \frac{u^6}{27}\right)^{1/3} + \frac{u^2}{3} \\ y &= ux + v \end{aligned}$$

where

$$v = \frac{3a - u^4}{6u}.$$

For $u = 0$, we fix $f_{a,b}(0) = \mathcal{O}$, the neutral element of the elliptic curve.

Lemma 1. *Let $\mathbb{F}_{p^n}$ be a field where $p^n = 2 \mod 3$ and $p > 3$. For any $u \in \mathbb{F}_{p^n}$, $f_{a,b}(u)$ is a point of $E_{a,b}(\mathbb{F}_{p^n}) : Y^2 = X^3 + aX + b$.*

Proof. For $u \neq 0$, let $(x, y) = f_{a,b}(u)$. From the definition of x:

$$\left(x - \frac{u^2}{3}\right)^3 = v^2 - b - \frac{u^6}{27}.$$

This expands into:

$$x^3 - u^2x^2 + \frac{u^4}{3}x + b - v^2 = 0.$$

Since $u^4/3 = a - 2uv$, this can be rewritten into

$$x^3 - u^2x^2 + (a - 2uv)x + b - v^2 = 0$$

which leads to

$$x^3 + ax + b = u^2x^2 + 2uvx + v^2 = (ux + v)^2$$

and finally $x^3 + ax + b = y^2$. □

We present a similar result in characteristic 2 in appendix A.

Remark 1. We note that if $x \mapsto x^3$ is not a bijection, but $(v^2 - b - u^6/27)$ is a cube in $\mathbb{F}_q$, we can still use the formulas to compute $(x, y) \in E_{a,b}$.

3 Properties of our new Encoding $f_{a,b}$

Lemma 2. *The function $f_{a,b}$ can be implemented in deterministic polynomial time, with $\mathcal{O}(\log^3 q)$ running time and a constant number of operations over $\mathbb{F}_q$.*

Proof. When $q = 2 \mod 3$, computing $x \mapsto x^{1/3}$ is an exponentiation with exponent $(2q-1)/3$. This can be implemented in a constant number of operations over $\mathbb{F}_q$. We also need to compute $v = (3a - u^4)/6u$, which requires to compute $1/u = u^{q-2}$, which can also be done in a constant number of operations over $\mathbb{F}_q$. The total running time is then $\mathcal{O}(\log^3 q)$. □

In the following, we show how to compute $f_{a,b}^{-1}(P)$ given a point P. This will be used to show the one-wayness and collision resistance properties of the resulting hash function (see Section 4).

Lemma 3. *Let $P = (x, y)$ be a point on the curve $E_{a,b}$. The solutions u_s of $f_{a,b}(u_s) = P$ are the solutions of the polynomial equation:*

$$u^4 - 6u^2x + 6uy - 3a = 0. \tag{3}$$

Proof. The proof is very similar to the proof of Lemma 1. We write $v = \frac{3a-u^4}{6u}$. We show that the two systems are equivalent:

$$\begin{cases} y^2 = x^3 + ax + b \\ u^4 - 6u^2x + 6uy - 3a = 0 \end{cases} \Leftrightarrow \begin{cases} \left(x - \frac{u^2}{3}\right)^3 = v^2 - b - \frac{u^6}{27} \\ y = ux + v \end{cases}$$

From the definition of $f_{a,b}$, this proves the result of the Lemma.

We have:

$$\begin{cases} y^2 = x^3 + ax + b \\ u^4 - 6u^2x + 6uy - 3a = 0 \end{cases} \Leftrightarrow \begin{cases} y^2 = x^3 + ax + b \\ y = ux + v \end{cases}$$

$$\Leftrightarrow \begin{cases} u^2x^2 + 2uvx + v^2 = x^3 + ax + b \\ y = ux + v \end{cases} \Leftrightarrow \begin{cases} x^3 - u^2x^2 + (a - 2uv)x + b - v^2 = 0 \\ y = ux + v \end{cases}$$

$$\Leftrightarrow \begin{cases} x^3 - u^2x^2 + \frac{u^4}{3}x + b - v^2 = 0 \\ y = ux + v \end{cases} \Leftrightarrow \begin{cases} \left(x - \frac{u^2}{3}\right)^3 = v^2 - b - \frac{u^6}{27} \\ y = ux + v \end{cases}$$

□

Lemma 4. *$f_{a,b}^{-1}(P)$ is computable in polynomial time and $\left|f_{a,b}^{-1}(P)\right| \leq 4$, for all $P \in E_{a,b}$,*

Proof. Lemma 3 ensures that to compute $f_{a,b}^{-1}$, it is sufficient to solve a degree 4 equation over $\mathbb{F}_q$. Solving polynomial equations of degree d over a finite field can be solved in $O(d^2 \log^3 q)$ binary operations using the Berlekamp algorithm [11]. For this reason, $f_{a,b}^{-1}$ is computable in polynomial time. Furthermore, since the pre-images are solution of a degree 4 equation over $\mathbb{F}_q$, there are at most 4 solutions for any point P, which implies that $\left|f_{a,b}^{-1}(P)\right| \leq 4$. □

From $\left|f_{a,b}^{-1}(P)\right| \leq 4$ we obtain that our function $f_{a,b}$ generates at least a constant fraction of the elliptic-curve points:

Corollary 1. *Let $E_{a,b}$ be a curve over $\mathbb{F}_q$, where $q = p^n$ with $p > 3$ and $p^n = 2 \mod 3$. We have*

$$\frac{q}{4} \leq |\mathsf{Im}(f_{a,b})| \leq q$$

The bounds for $|\mathsf{Im}(f_{a,b})|$ are not tight. We make the following conjecture:

Conjecture 1. There exists a constant λ such that for any q, a, b

$$\left| |\mathsf{Im}(f_{a,b})| - \frac{5}{8} |E_{a,b}(\mathbb{F}_q)| \right| \leq \lambda\sqrt{q}$$

In the following, we motivate our conjecture. From lemma 3, the size of $\mathsf{Im}(f_{a,b})$ depends on the existence of a solution of the equation $u^4 - 6u^2x + 6uy - 3a = 0$ for a given point (x, y) of $E_{a,b}(\mathbb{F}_q)$. A degree 4 polynomial has no root if and only if it is irreducible or if it is the product of two degree 2 irreducible polynomials. Over any finite field $\mathbb{F}_q$ with large q, it is known that random polynomials of degree d are irreducible with asymptotic probability $1/d$ as q goes to infinity [8]. For this reason, there exist approximately $q^2/2$ irreducible degree 2 monic polynomials. Hence, there exist $\binom{q^2/2}{2} \approx q^4/8$ products of two irreducible degree 2 polynomials. This implies that there exist approximately $q^4/4 + q^4/8 = 3q^4/8$ degree 4 monic polynomials with no root in $\mathbb{F}_q$. For this reason, we can estimate that a fraction 5/8 of random monic degree 4 polynomials have roots. As a consequence the size of $\mathsf{Im}(f_{a,b})$ should be approximately 5/8 of the size of $E_{a,b}$. Our conjecture is made by analogy of the Hasse bound.

Theorem 1 (Hasse Bound). $||E_{a,b}(\mathbb{F}_q)| - q - 1| \leq 2\sqrt{q}$

We have tested our conjecture for all curves $E_{a,b}$ over base field $\mathbb{F}_p$ such that $p = 2 \mod 3$ with $p < 10000$. For all these curves, we have computed the number of points of the curve and we also have computed the number of points in $\mathsf{Im}(f_{a,b})$. After this computation, we found a lower bound for λ as 2.3114.

From this conjecture, we have the following corollary, which gives a deterministic, surjective function onto $E_{a,b}(\mathbb{F}_q)$.

Corollary 2. *If Conjecture 1 is true with $\lambda \leq 3$, if $q = 2 \mod 3$ and $q > 1070$, then:*

$$\begin{aligned} F : (\mathbb{F}_q)^2 &\mapsto E_{a,b}(\mathbb{F}_q) \\ (u_1, u_2) &\mapsto f_{a,b}(u_1) + f_{a,b}(u_2) \end{aligned}$$

is a surjective map.

Proof. To prove that F is surjective, we use the drawer principle. Given a point $P \in E_{a,b}(\mathbb{F}_q)$, the set $S_1 = \{P - f_{a,b}(u)\}_{u \in \mathbb{F}_q}$ is made of at least $5q/8 - 3\sqrt{q+1+2\sqrt{q}}$ points. The set $S_2 = \{f_{a,b}(u)\}_{u \in \mathbb{F}_q}$ is also made of the same number of points. This implies that the set $S_1 \cap S_2$ is not empty if $|S_1| + |S_2| > |E_{a,b}(\mathbb{F}_q)|$. This is always true when

$$2\left(\frac{5q}{8} - 3\sqrt{q+1+2\sqrt{q}}\right) > q + 1 + 2\sqrt{q}$$

which leads to $q > 1070$. □

Finally, we note that computing discrete logarithms of $f_{a,b}(u)$ is hard if computing discrete logarithms in $E_{a,b}(\mathbb{F}_q)$ is hard. This is because the function $f_{a,b}$ is efficiently invertible and $\left|f_{a,b}^{-1}(P)\right| \leq 4$ for any P. Let G be a generator of $E_{a,b}(\mathbb{F}_q)$. If we are given as input a random point P, with probability at least $1/4$ we have that $P \in \mathsf{Im}(f_{a,b})$, so we can compute $u \in \mathbb{F}_q$ such that $P = f_{a,b}(u)$. Then if an algorithm can compute x such that $f_{a,b}(u) = x.G$, this gives x such that $P = x.G$. This shows that if an algorithm can compute the discrete logarithm of $f_{a,b}(u)$, then such algorithm can be used to compute the discrete logarithm in $E_{a,b}(\mathbb{F}_q)$. The same argument applies to any encoding function f which is polynomially invertible on its outputs and with a polynomially bounded pre-image size. The argument can be easily extended to show that for any generator base $(G_1, ..., G_n)$, computing $x_1, \ldots, x_n$ such that $f_{a,b}(x) = \sum_i x_i.G_i$ is hard if computing discrete logarithms in $E_{a,b}(\mathbb{F}_q)$ is hard.

4 How to Hash onto Elliptic Curves

Given a function f into an elliptic curve E, we describe two constructions of hash functions into E. We define L as the maximal size of $f^{-1}(P)$ where P is any point on E:

$$L = \max_{P \in E}(\left|f^{-1}(P)\right|)$$

For our encoding function $f_{a,b}$, we have $L \leq 4$ (see lemma 4). We note that if we work in a subgroup of E of order n with cofactor r, we can use the encoding function $f'_{a,b} = r.f_{a,b}$. If r is relatively prime to n, then we must have $L \leq 4r$.

Our first construction is as follows: given a hash function $h : \{0,1\}^* \mapsto \mathbb{F}_q$, we define

$$H(m) = f(h(m))$$

as a hash function into the curve $E_{a,b}(\mathbb{F}_q)$. In the following, we show that H is one-way if h is one way.

4.1 One-Wayness

Definition 1. *A hash function is (t, ε)-one-way, if any algorithm running in time t, when given a random $y \in \mathit{Im}(h)$ as input, outputs m such that $h(m) = y$ with probability at most ε. A hash function is one-way if ε is negligible for any polynomial t in the security parameter.*

Lemma 5. *If h is a (t, ε)-one-way hash function then H is (t', ε')-one-way where $\varepsilon' = L^2\varepsilon$, where $L = \max_{P \in E}(\left|f^{-1}(P)\right|)$. Therefore, if L is polynomial in the security parameter and h is one-way, then H is one-way.*

Proof. We receive $y = h(m)$ as input and we must compute m' such that $y = h(m')$, using an algorithm $\mathcal{A}$ that breaks the one-wayness of H. Given $y = h(m)$, we first compute $f(y) = f(h(m)) = H(m)$ and we use $\mathcal{A}$ to get a preimage of $H(m)$. While using this way of proceedings, two problems occur:

1. $f(y)$ does not follow the uniform distribution even if y does,
2. a preimage m' to $H(m)$ is not necessarily a preimage of y.

Let $\mathcal{A}$ be an algorithm, which computes preimages of H in time t with a probability at least ε'. Let S be the event that $\mathcal{A}$ succeeds. When $\mathcal{A}$ receives an input P which is uniformly distributed in $\mathsf{Im}(H)$, we have:

$$\Pr[S] = \sum_{p \in \mathsf{Im}(H)} \Pr[S|P = p] \Pr[P = p] = \sum_{p \in \mathsf{Im}(H)} \Pr[S|P = p] \frac{1}{|\mathsf{Im}(H)|} \geq \varepsilon'$$

However, we use $f(y)$ as input to $\mathcal{A}$, which is not uniformly distributed even if y is uniformly distributed. We thus need to compute the success probability of $\mathcal{A}$ given P computed as $f(y)$. Let $\Pr(S')$ be this success probability. We have:

$$\begin{aligned}\Pr\left[S'\right] &= \sum_{y \in \mathsf{Im}(h)} \Pr[S|P = f(y)] \Pr[P = f(y)] \\ &= \sum_{p \in \mathsf{Im}(H)} \Pr[S|P = p] \left(\sum_{y \in f^{-1}(p)} \frac{1}{|\mathsf{Im}(h)|} \right)\end{aligned}$$

By definition of L, $|\mathsf{Im}(H)| \geq |\mathsf{Im}(h)|/L$. This leads to:

$$\Pr\left[S'\right] \geq \sum_{p \in \mathsf{Im}(H)} \Pr[S|P = p] \frac{1}{|\mathsf{Im}(h)|} \geq \sum_{p \in \mathsf{Im}(H)} \Pr[S|P = p] \frac{1}{L\,|\mathsf{Im}(H)|} \geq \frac{\varepsilon'}{L}.$$

This proves that $\mathcal{A}$ has a probability at least ε'/L to successfully compute a preimage to $f(y) = H(m)$. Assume that $\mathcal{A}$ has successfully computed one preimage of $f(y)$. Since each point has possibly L different preimages through f, this preimage is the preimage of $y = h(m)$ with probability $1/L$. This implies that this algorithm has a probability at least ε'/L^2 to succeed. This gives $\varepsilon'/L^2 \leq \varepsilon$. □

4.2 Collision Resistance

Definition 2. *A family $\mathcal{H}$ of hash functions is (t, ε)-collision-resistant, if any algorithm running in time t, when given a random $h \in \mathcal{H}$, outputs (m, m') such that $h(m) = h(m')$ with probability at most ε.*

Our first construction is easily extended to hash function families: given a family $\mathcal{H}$ of hash functions, we define for each $h \in \mathcal{H}$ the function $H = f \circ h$. We then study whether the family of hash functions formed by the H is collision resistant.

A collision to one H occurs if and only if:

1. there exists m and m' such that $h(m) = h(m')$; this is a collision for h,
2. or $f(u) = f(u')$ for $u = h(m), u' = h(m')$ and $u \neq u'$; this is a collision for f.

In the following, we argue that we cannot prove the collision resistance of H based on the collision resistance of h only. Namely, we note that given a hash function h, it is easy to construct an elliptic curve with collisions on $H = f_{a,b} \circ h$. Indeed, given (m, m'), let $u = h(m)$ and $u' = h(m')$. From this couple (u, u'), we compute the degree 4 polynomial:

$$(X - u)(X - u')(X^2 + (u + u')X - w) \tag{4}$$

where w is a randomly chosen element in $\mathbb{F}_q$. This polynomial is equal to:

$$X^4 - 6xX^2 + 6yX - 3a$$

where

$$x = -\frac{uu' + w - (u + u')^2}{6}, \;\; y = \frac{(u + u')(uu' - w)}{6}, \;\; a = -\frac{uu'w}{3}.$$

Let $b = y^2 - x^3 - ax$. Hence (x, y) is a point on the elliptic curve $E_{a,b}$ by definition of b. For this reason, a preimage of (x, y) through $f_{a,b}$ is a solution of the equation:

$$X^4 - 6xX^2 + 6yX - 3a = 0 \tag{5}$$

which is exactly the polynomial (4) by definition of x, y and a. For this reason, u and u' are solutions of the equations (4) and are preimages of (x, y). Hence (m, m') is a collision for $H = f_{a,b} \circ h$.

However, if $E_{a,b}$ is defined independently from h, it seems difficult to find (m, m') such that $f_{a,b}(y) = f_{a,b}(y')$ where $y = h(m)$ and $y' = h(m')$. In this case, H should be collision resistant. We cannot prove that H is collision resistant based only on the collision resistance of h, and we clearly need some additional properties on h. In the next section, we provide a different construction for which collision resistance can be proved based only on the collision resistance of h.

4.3 Making f Collision Free

In this section, we show how to construct a family $\mathcal{G}$ of functions derived from an encoding function f, which is collision free except with negligible probability. Then given a hash function h and given $g \in \mathcal{G}$, $H'(m) = g(h(m))$ will be collision resistant assuming that h is collision resistant.

Definition 3. *A family $\mathcal{G}$ of functions is ε-collision-free if the probability that $g \in \mathcal{G}$ has a collision is at most ε.*

In other words, a collision-free family of functions is a family in which most functions are injective. To construct such collision free family, we use the notion of family of pair-wise independent functions.

Definition 4 (Pair-wise Independence). *A family $\mathcal{V}$ of functions $v : R \mapsto S$ is ε-pair-wise independent if given any couple $(r_1, r_2) \in R^2$ with $r_1 \neq r_2$ and any couple $(s_1, s_2) \in S^2$:*

$$\Pr_{v \in \mathcal{V}} [v(r_1) = s_1 \wedge v(r_2) = s_2] \leq \varepsilon.$$

The following theorem shows that the family $\mathcal{G} = \{f \circ v\}_{v \in \mathcal{V}}$ is collision free.

Theorem 2. *Let $f : S \mapsto T$ be a function such that $\left|f^{-1}(t)\right| \leq L$ for all $t \in T$. Let $\mathcal{V}$ be a family of ε-pair-wise independent functions from R to S. Then the family $\mathcal{G} = (f \circ v)_{v\in\mathcal{V}}$ is ε'-collision-free where*

$$\varepsilon' = |R|^2 \cdot |S| \cdot L \cdot \varepsilon.$$

Proof. We consider the number of collisions for f. A collision is an unordered pair (s, s') with $s \neq s'$ such that $f(s) = f(s')$. If $\left|f^{-1}(t)\right| = L$ for all $t \in T$, then the number of collisions is exactly $|S|(L-1)/2$. Namely, every preimage gives $L(L-1)/2$ collisions and they are $|S|/L$ such preimages. The following lemma shows this is actually the maximum number of collisions that can be obtained.

Lemma 6. *If $\left|f^{-1}(t)\right| \leq L$ for all $t \in T$, then the number of collisions N_{col} of f verifies $N_{col} \leq |S|(L-1)/2$.*

Proof. The proof of this lemma is given in Appendix B.

We consider a particular collision (s_1, s_2) for f. We consider $(r_1, r_2) \in R^2$ with $r_1 \neq r_2$, hence by definition of $\mathcal{V}$:

$$\Pr\left[v(r_1) = s_1 \wedge v(r_2) = s_2\right] \leq \varepsilon.$$

Since there are at most $|R|^2$ such couples (r_1, r_2), the probability that there exists at least one couple (r_1, r_2) such that $v(r_1) = s_1$ and $v(r_2) = s_2$ is at most $|R|^2 \cdot \varepsilon$.

Given a collision s_1, s_2 for f, we must consider both couples (s_1, s_2) and (s_2, s_1). Moreover we must also consider couples of the form (s_1, s_1) for all $s_1 \in S$, since these couples lead to collisions for v. Therefore the probability to have a collision for $f \circ v$ is at most

$$(2N_{col} + |S|) \cdot |R|^2 \cdot \varepsilon \leq |R|^2 \cdot |S| \cdot L \cdot \varepsilon.$$

which terminates the proof. □

Therefore, our second construction is as follows. Given a security parameter k and an integer $q = p^n$ with $q \geq 2^k$, we consider the following family of functions:

$$\begin{aligned}(v_{c,d})_{c,d\in\mathbb{F}_q} : \{0,1\}^k &\mapsto \mathbb{F}_q \\ x &\mapsto c \cdot x + d\end{aligned}$$

where x is seen as an element in $\mathbb{F}_q$. It is easy to see that this family is $1/q^2$-pair-wise independent.

Given an elliptic curve E, we combine the encoding function f with the functions in the $v_{c,d}$ family to get a collision-free family $\mathcal{G}$:

$$\begin{aligned}\mathcal{G} = (f \circ v_{c,d})_{c,d\in\mathbb{F}_q} : \{0,1\}^k &\mapsto E(\mathbb{F}_q) \\ x &\mapsto f(c \cdot x + d)\end{aligned}$$

Finally, given a family $\mathcal{H}$ of collision-resistant functions $h : \{0,1\}^* \mapsto \{0,1\}^k$, we construct the following family of hash functions into the curve E:

$$\begin{aligned}\mathcal{H}_E = (f \circ v_{c,d} \circ h)_{\substack{c,\, d \,\in\, \mathbb{F}_q \\ h \in \mathcal{H}}} : \{0,1\}^* &\mapsto E(\mathbb{F}_q) \\ m &\mapsto f(c \cdot h(m) + d)\end{aligned}$$

Theorem 3. *If $\mathcal{H}$ is a (t, ε)-collision resistant family of hash functions, then $\mathcal{H}_E$ is a (t', ε')-collision resistant family of hash functions where*

$$\varepsilon' = \varepsilon + L\frac{2^{2k}}{q}.$$

Proof. Theorem 2 ensures that $\mathcal{G}$ is a ε_1-collision free family where:

$$\varepsilon_1 = \frac{4 \cdot 2^{2k} \cdot q}{q^2} = \frac{L2^{2k}}{q}.$$

For this reason, given a random $g \in \mathcal{G}$, g does not have any collision except with probability at most $\frac{L2^{2k}}{q}$. In this case, a collision to $h_E \in \mathcal{H}_E$ is exactly a collision of $h \in \mathcal{H}$, the underlying hash function.

Let $\mathcal{A}$ be an algorithm which attacks the collision resistance of $\mathcal{H}_E$ with a success probability ε'. For a given $h_E \in \mathcal{H}_E$, let $g \in \mathcal{G}$ and $h \in \mathcal{H}$ be such that $h_E = g \circ h$. In the case where g is collision free and when $\mathcal{A}$ succeeds to attack $g \circ h$, it indeed finds a collision to h. Let S be the event that $\mathcal{A}$ succeeds to attack the collision resistance of $\mathcal{H}_E$. We have

$$\begin{aligned}\Pr\left[S\right] &= \Pr\left[S|\text{g is CF}\right] \cdot \Pr\left[\text{g is CF}\right] + \Pr\left[S|\text{g is not CF}\right] \cdot \Pr\left[\text{g is not CF}\right] = \varepsilon' \\ &\le \Pr\left[S|\text{g is CF}\right] \cdot \Pr\left[\text{g is CF}\right] + \frac{L2^{2k}}{q} \le \varepsilon + \frac{L2^{2k}}{q}\end{aligned}$$

□

Note that if we take q of size $5k/2$ bits, we obtain $\varepsilon' \le \varepsilon + 2^{-k/2}$. Therefore if we have a family $\mathcal{H}$ of k-bit ε-collision resistant hash functions where $\varepsilon = 2^{-k/2}$, we obtain the same security level for $\mathcal{H}_E$, namely $\varepsilon' = 2^{-k/2+1}$.

In practice, given a curve E defined modulo a prime p, we randomly select $c, d \in \mathbb{F}_p$ and a function $h \in \mathcal{H}$; this defines a hash function:

$$\begin{aligned}H_E : \{0,1\}^* &\mapsto E(\mathbb{F}_p) \\ m &\mapsto f(c \cdot h(m) + d)\end{aligned}$$

Finally, we have that H_E is a one way hash function when $c \neq 0$:

Lemma 7. *If h is a (t, ε)-one-way hash function, then for any c, d with $c \neq 0$, $H_E = f \circ v_{c,d} \circ h$ is a (t', ε')-one way hash function ,with $\varepsilon' = L^2 \cdot \varepsilon$.*

Proof. Given $y = h(m)$, we want to compute m' such that $y = h(m')$. Let $\mathcal{A}$ be an algorithm that solves the one-wayness of H_E with a success probability ε'. Since $c \neq 0$, $v_{c,d}$ is an injective map from $\{0,1\}^k$ to $\mathbb{F}_p$. For this reason, $f \circ v_{c,d}$ has at most L preimages for a given point on E.

Furthermore, we want to use $\mathcal{A}$ on the input $H_E(m)$ where $H_E(m)$ is computed from $f(v_{c,d}(y))$. As in the proof of Lemma 6, f introduces a bias on the distribution of $f(v_{c,d}(y))$ compared to the uniform distribution on the image of H_E. However, this bias can be bounded by the same factor $1/L$ as in Lemma 6. For these reasons, $H_E = f \circ v_{c,d} \circ h$ is $L^2\varepsilon$-one-way. □

We note that this second construction requires a much larger q than the previous construction. For example, for a 160-bit hash function h, the first construction requires a 160-bit integer q, whereas our second construction requires q to be of size $5 \cdot 160/2 = 400$ bits.

5 Practical Implementations

In this section we compare the running time needed by various encodings into elliptic-curves. We first consider our function $f_{a,b}$ with Euclide's algorithm to implement the inversion in $\mathbb{F}_p$; we also consider $f_{a,b}$ (v2) with an exponentiation instead in order to have a constant number of operations over $\mathbb{F}_p$.

We have also implemented the various 'try-and-increment' algorithms: the classic algorithm, the algorithm with a constant number of steps but with fast Legendre symbol (v2) and the algorithm with a constant number of operations using an exponentiation for the Legendre symbol (v3). We have also implemented the encoding defined by equation (2) for a supersingular elliptic-curve; in this case we have used a finite field ensuring the same security level as for ordinary elliptic curves.

The implementation has been done on 10 different 160-bit primes, randomly chosen such that $p = 2 \mod 3$ and $p = 3 \mod 4$. For every prime, 10 different couples of parameters (a, b) have been randomly chosen. And on these 10 different curves, we runned every algorithm 1000 times on random inputs. We used a 512-bit prime for the supersingular curves case.

Algorithm	Constant Running Time	Running Time
$f_{a,b}$	No	0.22 ms
Try and Increment	No	0.24 ms
Try and Increment v2	No	1.86 ms
$f_{a,b}$ v2	Yes	0.40 ms
Try and Increment v3	Yes	16.1 ms
Supersingular Curves	Yes	3.67 ms

Table 1. Average Time of each Algorithms using the Number Theory Library (NTL) [10] and running on a laptop using the Intel®Core$^{\text{TM}}$2 Duo T7100 chip at a frequency of $1,80$ Ghz.

We obtain that when a constant running time is required, our method performs much better than the 'try-and-increment' algorithm and the algorithm for supersingular curves. It also performs slightly better even when a constant running time is not required.

6 Conclusion

We have provided a new algorithm that encodes an integer into an elliptic curve point in a constant number of field operations. This encoding exists for any curve under the condition that the map $x \mapsto x^3$ is a bijection on the base field. This encoding is efficiently computable with the same complexity as one exponentiation and one inversion on the base field.

From our encoding, we have defined two constructions, which enable to hash into an elliptic curve. The first construction is provably one-way and the second is provably one-way and collision resistant in the standard model. Our algorithm can be used for password based authentication protocol over elliptic curves. Indeed, it enables to efficiently encode passwords or PIN-codes into points of the curve in a constant number of field operations.

We also note that our encoding enables to compute points of elliptic curves over RSA rings without knowing the factorization of $N = pq$. Consider the following problem: given $N = pq$ where p and q are prime integers and a, b in $\mathbb{Z}_N$, find (x, y) such that $y^2 = x^3 + ax + b \mod N$. Previously, factoring N was required to compute such (x, y). Our function $f_{a,b}$ proves that a cube root oracle is actually sufficient.

Acknowledgments: I wish to thank Jean-Sébastien Coron for the time he spent to help me write this paper. I also thank Julien Bringer, Hervé Chabanne, Bruno Kindarji and the anonymous referees for their helpful comments.

References

1. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.
2. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptology*, 17(4):297–319, 2004.
3. Colin Boyd, Paul Montague, and Khanh Quoc Nguyen. Elliptic curve based password authenticated key exchange protocols. In Vijay Varadharajan and Yi Mu, editors, *ACISP*, volume 2119 of *Lecture Notes in Computer Science*, pages 487–501. Springer, 2001.
4. Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *EUROCRYPT*, pages 156–171, 2000.
5. Olivier Chevassut, Pierre-Alain Fouque, Pierrick Gaudry, and David Pointcheval. The twist-augmented technique for key exchange. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 410–426. Springer, 2006.
6. David P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Comput. Commun. Rev.*, 26(5):5–26, 1996.
7. Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.
8. Robert Sedgewick and Philippe Flajolet. *An Introduction to the Analysis of Algorithms.* Addison-Wesley Publishing Company, 1996. 512 pages. (ISBN 0-201-40009-X).
9. Andrew Shallue and Christiaan van de Woestijne. Construction of rational points on elliptic curves over finite fields. In Florian Hess, Sebastian Pauli, and Michael E. Pohst, editors, *ANTS*, volume 4076 of *Lecture Notes in Computer Science*, pages 510–524. Springer, 2006.
10. Victor Shoup. Ntl, Number Theory C++ Library. `http://www.shoup.net/ntl/`.
11. Victor Shoup. A new polynomial factorization algorithm and its implementation. *J. Symb. Comput.*, 20(4):363–397, 1995.
12. M. Skalba. Points on elliptic curves over finite fields. *Acta Arith*, 117:293–301, 2005.

A An explicit Encoding from $\mathbb{F}_{2^n}$ to $E(\mathbb{F}_{2^n})$

The equations which define elliptic curves in characteristic 2 are somehow different from the Weierstrass equation:

$$Y^2 + XY = X^3 + aX^2 + b$$

where a and b are elements of $\mathbb{F}_{2^n}$. For an odd n, the map $x \mapsto x^3$ is a bijection. Let

$$\begin{aligned} f_{a,b} : \mathbb{F}_{2^n} &\mapsto (\mathbb{F}_{2^n})^2 \\ u &\mapsto (x, ux + v^2) \end{aligned}$$

where

$$v = a + u + u^2, x = (v^4 + v^3 + b)^{1/3} + v.$$

Lemma 8. *Let $\mathbb{F}_{2^n}$ be a field with n odd. For any $u \in \mathbb{F}_{2^n}$, $f_{a,b}(u)$ is a point of $E_{a,b} : Y^2 + XY = X^3 + aX^2 + b$.*

Proof. Given a parameter u, let (x, y) be $f_{a,b}(u)$. We have the following equations for x, u and v:

$$\begin{aligned} 0 &= (x+v)^3 + b + v^3 + v^4 \\ &= x^3 + vx^2 + v^2x + b + v^4. \end{aligned}$$

Since $v = a + u + u^2$, this can be rewritten into:

$$\begin{aligned} x^3 + ax^2 + b &= ux^2 + u^2x^2 + v^2x + v^4 \\ &= (ux + v^2)((u+1)x + v^2) = y(x+y) \end{aligned}$$

Hence, $(x, y) = f_{a,b}(u)$ is a point of $E_{a,b}$. □

A.1 Cardinality of $\mathsf{Im}(f_{a,b})$ in Characteristic 2

As in the case of the characteristic p, it is possible to bound the $|\mathsf{Im}(f_{a,b})|$.

Theorem 4. $2^{n-2} < |\mathsf{Im}(f_{a,b})| \leq 2^n$

Proof. The inequality $|\mathsf{Im}(f_{a,b})| \leq 2^n$ is the consequence that $f_{a,b}$ is a function.

The other side of the inequality $2^{n-2} < |\mathsf{Im}(f_{a,b})|$ can be explained thanks to the equation $y = ux + a^2 + u^2 + u^4$. As for the characteristic p, the second equation of the Lemma 9 is enough to inverse $f_{a,b}$. This equation can be rewritten as

$$0 = y + a + ux + u^2 + u^4$$

Given a point (x, y), if u is a solution of this equation then $f_{a,b}(u) = (x, y)$. Since the equation is of degree 4, there are at most 4 different u for each point image of $f_{a,b}$. For this reason, there are at least $2^n/4 = 2^{n-2}$ points in $\mathsf{Im}(f_{a,b})$. □

B Proof of Lemma 7

Proof. We want to bound the number N_{col} of collisions of f. We have that:

$$N_{col} = \sum_{i=2}^{L} \left(\sum_{x \in \mathsf{Im}(f) || f^{-1}(x)| = i} \binom{i}{2} \right).$$

Indeed a t collision is in fact $\binom{t}{2}$ collisions. Let $\lambda_i = \sum_{x \in \mathsf{Im}(f) || f^{-1}(x)| = i} 1$. Since $\sum_{i=2}^{B} i\lambda_i \leq |S|$, this implies that:

$$N_{col} = \sum_{i=2}^{B} \lambda_i \frac{i(i-1)}{2} \leq \frac{B-1}{2} \sum_{i=2}^{B} i\lambda_i \leq \frac{|S|\,(B-1)}{2}$$

This inequality about $Ncol$ is tight because in the case where f has only B-collisions, $Ncol$ is equal to $|S|\,(B-1)/2$. □