Batching Cipolla–Lehmer–Müller's square root algorithm with hashing to elliptic curves

Dimitri Koshelev

Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon

09/06/2023, CIRM, Marseille

Introduction

Let $\mathbb{F}_q$ be a finite field of characteristic > 3 and E be an elliptic $\mathbb{F}_q$-curve $y^2 = f(x) := x^3 + ax + b$.

Introduction

Let $\mathbb{F}_q$ be a finite field of characteristic > 3 and E be an elliptic $\mathbb{F}_q$-curve $y^2 = f(x) := x^3 + ax + b$.

Suppose there is a subgroup $G \subset E(\mathbb{F}_q)$ of a large prime order $r \mid N := \#E(\mathbb{F}_q)$. Besides, denote by $c := N/r$ the cofactor.

Introduction

Let $\mathbb{F}_q$ be a finite field of characteristic > 3 and E be an elliptic $\mathbb{F}_q$-curve $y^2 = f(x) := x^3 + ax + b$.

Suppose there is a subgroup $G \subset E(\mathbb{F}_q)$ of a large prime order $r \mid N := \#E(\mathbb{F}_q)$. Besides, denote by $c := N/r$ the cofactor.

Many protocols of elliptic cryptography need a *hash function* $\mathcal{H} \colon \{0,1\}^* \to G$, among which are

1. signatures (e.g., Boneh–Lynn–Shacham),
2. PAKE (Password-Authenticated Key Exchange) protocols,
3. OPRFs (Oblivious Pseudorandom Functions),
4. identity-based cryptography.

Introduction

Let $\mathbb{F}_q$ be a finite field of characteristic > 3 and E be an elliptic $\mathbb{F}_q$-curve $y^2 = f(x) := x^3 + ax + b$.

Suppose there is a subgroup $G \subset E(\mathbb{F}_q)$ of a large prime order $r \mid N := \#E(\mathbb{F}_q)$. Besides, denote by $c := N/r$ the cofactor.

Many protocols of elliptic cryptography need a *hash function* $\mathcal{H} \colon \{0,1\}^* \to G$, among which are

1. signatures (e.g., Boneh–Lynn–Shacham),
2. PAKE (Password-Authenticated Key Exchange) protocols,
3. OPRFs (Oblivious Pseudorandom Functions),
4. identity-based cryptography.

A naive method consists in the scalar multiplication $m \mapsto [m]P$ for a fixed non-zero point $P \in G$. However, it is insecure due to dependency on the group structure of E.

Components of $\mathcal{H}$

Almost all known hash functions to G are the compositions $\mathcal{H} = [c] \circ h \circ \eta$.

Components of $\mathcal{H}$

Almost all known hash functions to G are the compositions $\mathcal{H} = [c] \circ h \circ \eta$.

Here, $\eta\colon \{0,1\}^* \to S$ is a hash function to some finite set and $h\colon S \to E(\mathbb{F}_q)$ is just a map commonly called *encoding*.

Components of $\mathcal{H}$

Almost all known hash functions to G are the compositions $\mathcal{H} = [c] \circ h \circ \eta$.

Here, $\eta \colon \{0,1\}^* \to S$ is a hash function to some finite set and $h \colon S \to E(\mathbb{F}_q)$ is just a map commonly called *encoding*.

The scalar multiplication $[c]$ on E is said to be *clearing cofactor*.

Components of $\mathcal{H}$

Almost all known hash functions to G are the compositions $\mathcal{H} = [c] \circ h \circ \eta$.

Here, $\eta \colon \{0,1\}^* \to S$ is a hash function to some finite set and $h \colon S \to E(\mathbb{F}_q)$ is just a map commonly called *encoding*.

The scalar multiplication $[c]$ on E is said to be *clearing cofactor*.

The set S is usually very simple, hence it is easy to combine η from existing hash functions $\{0,1\}^* \to \{0,1\}^\ell$ for $\ell \in \mathbb{N}$.

Components of $\mathcal{H}$

Almost all known hash functions to G are the compositions $\mathcal{H} = [c] \circ h \circ \eta$.

Here, $\eta\colon \{0,1\}^* \to S$ is a hash function to some finite set and $h\colon S \to E(\mathbb{F}_q)$ is just a map commonly called *encoding*.

The scalar multiplication $[c]$ on E is said to be *clearing cofactor*.

The set S is usually very simple, hence it is easy to combine η from existing hash functions $\{0,1\}^* \to \{0,1\}^\ell$ for $\ell \in \mathbb{N}$.

The most complicated component of $\mathcal{H}$ is no doubt h, because it is based on high-dimensional algebraic geometry.

Components of $\mathcal{H}$

Almost all known hash functions to G are the compositions $\mathcal{H} = [c] \circ h \circ \eta$.

Here, $\eta\colon \{0,1\}^* \to S$ is a hash function to some finite set and $h\colon S \to E(\mathbb{F}_q)$ is just a map commonly called *encoding*.

The scalar multiplication $[c]$ on E is said to be *clearing cofactor*.

The set S is usually very simple, hence it is easy to combine η from existing hash functions $\{0,1\}^* \to \{0,1\}^\ell$ for $\ell \in \mathbb{N}$.

The most complicated component of $\mathcal{H}$ is no doubt h, because it is based on high-dimensional algebraic geometry.

When the input argument s of $\mathcal{H}$ is secret, its running time must depend only on the length of s to protect against *timing attacks*.

Components of $\mathcal{H}$

Almost all known hash functions to G are the compositions $\mathcal{H} = [c] \circ h \circ \eta$.

Here, $\eta\colon \{0,1\}^* \to S$ is a hash function to some finite set and $h\colon S \to E(\mathbb{F}_q)$ is just a map commonly called *encoding*.

The scalar multiplication $[c]$ on E is said to be *clearing cofactor*.

The set S is usually very simple, hence it is easy to combine η from existing hash functions $\{0,1\}^* \to \{0,1\}^\ell$ for $\ell \in \mathbb{N}$.

The most complicated component of $\mathcal{H}$ is no doubt h, because it is based on high-dimensional algebraic geometry.

When the input argument s of $\mathcal{H}$ is secret, its running time must depend only on the length of s to protect against *timing attacks*.

In particular, iterating $x \in \mathbb{F}_q$ as long as $\sqrt{f(x)} \notin \mathbb{F}_q$ is unsafe.

Simplified Shallue–van de Woestijne–Ulas map

It consists in a parametrization $\varphi\colon \mathbb{A}^1_t \to C$ of a (possibly singular) rational $\mathbb{F}_q$-curve C lying on the algebraic surface

$$y^2 = df(x_1)f(x_2) \quad \subset \quad \mathbb{A}^3_{(x_1,x_2,y)},$$

where $d \in (\mathbb{F}_q)^* \setminus (\mathbb{F}_q^*)^2$.

Simplified Shallue–van de Woestijne–Ulas map

It consists in a parametrization $\varphi\colon \mathbb{A}^1_t \to C$ of a (possibly singular) rational $\mathbb{F}_q$-curve C lying on the algebraic surface

$$y^2 = df(x_1)f(x_2) \quad \subset \quad \mathbb{A}^3_{(x_1,x_2,y)},$$

where $d \in (\mathbb{F}_q)^* \setminus (\mathbb{F}_q^*)^2$.

Given $t \in \mathbb{F}_q$, for exactly one coordinate $a_i := x_i\big(\varphi(t)\big)$ the value $f_i := f(a_i)$ is a quadratic residue in $\mathbb{F}_q$.

Simplified Shallue–van de Woestijne–Ulas map

It consists in a parametrization $\varphi\colon \mathbb{A}^1_t \to C$ of a (possibly singular) rational $\mathbb{F}_q$-curve C lying on the algebraic surface

$$y^2 = df(x_1)f(x_2) \quad \subset \quad \mathbb{A}^3_{(x_1,x_2,y)},$$

where $d \in (\mathbb{F}_q)^* \setminus (\mathbb{F}_q^*)^2$.

Given $t \in \mathbb{F}_q$, for exactly one coordinate $a_i := x_i\big(\varphi(t)\big)$ the value $f_i := f(a_i)$ is a quadratic residue in $\mathbb{F}_q$.

Therefore, we get the point $P = \big(a_i, \sqrt{f_i}\big) \in E(\mathbb{F}_q)$.

Simplified Shallue–van de Woestijne–Ulas map

It consists in a parametrization $\varphi\colon \mathbb{A}^1_t \to C$ of a (possibly singular) rational $\mathbb{F}_q$-curve C lying on the algebraic surface

$$y^2 = df(x_1)f(x_2) \quad \subset \quad \mathbb{A}^3_{(x_1,x_2,y)},$$

where $d \in (\mathbb{F}_q)^* \setminus (\mathbb{F}_q^*)^2$.

Given $t \in \mathbb{F}_q$, for exactly one coordinate $a_i := x_i\big(\varphi(t)\big)$ the value $f_i := f(a_i)$ is a quadratic residue in $\mathbb{F}_q$.

Therefore, we get the point $P = \left(a_i, \sqrt{f_i}\right) \in E(\mathbb{F}_q)$.

The simplified SWU map requires extracting one square root in $\mathbb{F}_q$.

Simplified Shallue–van de Woestijne–Ulas map

It consists in a parametrization $\varphi\colon \mathbb{A}^1_t \to C$ of a (possibly singular) rational $\mathbb{F}_q$-curve C lying on the algebraic surface

$$y^2 = df(x_1)f(x_2) \quad \subset \quad \mathbb{A}^3_{(x_1,x_2,y)},$$

where $d \in (\mathbb{F}_q)^* \setminus (\mathbb{F}_q^*)^2$.

Given $t \in \mathbb{F}_q$, for exactly one coordinate $a_i := x_i\big(\varphi(t)\big)$ the value $f_i := f(a_i)$ is a quadratic residue in $\mathbb{F}_q$.

Therefore, we get the point $P = \left(a_i, \sqrt{f_i}\right) \in E(\mathbb{F}_q)$.

The simplified SWU map requires extracting one square root in $\mathbb{F}_q$.

As is known, for $q \not\equiv 1 \pmod 8$ this is equivalent to raising f_i to some fixed power $n \in \mathbb{Z}/(q-1)$.

Simplified Shallue–van de Woestijne–Ulas map

It consists in a parametrization $\varphi\colon \mathbb{A}^1_t \to C$ of a (possibly singular) rational $\mathbb{F}_q$-curve C lying on the algebraic surface

$$y^2 = df(x_1)f(x_2) \quad \subset \quad \mathbb{A}^3_{(x_1,x_2,y)},$$

where $d \in (\mathbb{F}_q)^* \setminus (\mathbb{F}_q^*)^2$.

Given $t \in \mathbb{F}_q$, for exactly one coordinate $a_i := x_i\big(\varphi(t)\big)$ the value $f_i := f(a_i)$ is a quadratic residue in $\mathbb{F}_q$.

Therefore, we get the point $P = \left(a_i, \sqrt{f_i}\right) \in E(\mathbb{F}_q)$.

The simplified SWU map requires extracting one square root in $\mathbb{F}_q$.

As is known, for $q \not\equiv 1 \pmod 8$ this is equivalent to raising f_i to some fixed power $n \in \mathbb{Z}/(q-1)$.

There are explicit elementary formulas of φ at least for j-invariant $\neq 0, 1728$, or equivalently, for the coefficients $a, b \neq 0$.

Highly 2-adic fields and table lookups

Hereafter, $\mathbb{F}_q$ is assumed to be a *highly* 2*-adic field*. By definition, $q - 1 = 2^\nu m$ for $\nu, m \in \mathbb{N}$ and ν is quite large. Lots of modern elliptic curves are defined over such fields to be able to use FFT.

Highly 2-adic fields and table lookups

Hereafter, $\mathbb{F}_q$ is assumed to be a *highly 2-adic field*. By definition, $q - 1 = 2^\nu m$ for $\nu, m \in \mathbb{N}$ and ν is quite large. Lots of modern elliptic curves are defined over such fields to be able to use FFT.

Undoubtedly, $\sqrt{\cdot} \in \mathbb{F}_q$ can be found through (constant-time) *Tonelli–Shanks's algorithm*. However, it requires $O(\log(q) + \nu^2)$ operations in $\mathbb{F}_q$, namely $O(\log^2(q))$ for ν close to $\log_2(q)$.

Highly 2-adic fields and table lookups

Hereafter, $\mathbb{F}_q$ is assumed to be a *highly 2-adic field*. By definition, $q - 1 = 2^\nu m$ for $\nu, m \in \mathbb{N}$ and ν is quite large. Lots of modern elliptic curves are defined over such fields to be able to use FFT.

Undoubtedly, $\sqrt{\cdot} \in \mathbb{F}_q$ can be found through (constant-time) *Tonelli–Shanks's algorithm*. However, it requires $O(\log(q) + \nu^2)$ operations in $\mathbb{F}_q$, namely $O(\log^2(q))$ for ν close to $\log_2(q)$.

The given drawback can be mitigated to $O\big(\log(q) + (\nu/\mu)^2\big)$ ones with $O(2^\mu \nu/\mu)$ storage (where μ is a parameter) by means of *Bernstein's table-lookup variant*.

Highly 2-adic fields and table lookups

Hereafter, $\mathbb{F}_q$ is assumed to be a *highly 2-adic field*. By definition, $q - 1 = 2^\nu m$ for $\nu, m \in \mathbb{N}$ and ν is quite large. Lots of modern elliptic curves are defined over such fields to be able to use FFT.

Undoubtedly, $\sqrt{\cdot} \in \mathbb{F}_q$ can be found through (constant-time) *Tonelli–Shanks's algorithm*. However, it requires $O(\log(q) + \nu^2)$ operations in $\mathbb{F}_q$, namely $O(\log^2(q))$ for ν close to $\log_2(q)$.

The given drawback can be mitigated to $O\big(\log(q) + (\nu/\mu)^2\big)$ ones with $O(2^\mu \nu/\mu)$ storage (where μ is a parameter) by means of *Bernstein's table-lookup variant*.

In practice, this memory overhead is not significant for moderate 2-adicities such as $\nu \approx 32$. However, assuming a secret input, Bernstein's approach is vulnerable to *cache-timing attacks*.

Highly 2-adic fields and table lookups

Hereafter, $\mathbb{F}_q$ is assumed to be a *highly 2-adic field*. By definition, $q - 1 = 2^\nu m$ for $\nu, m \in \mathbb{N}$ and ν is quite large. Lots of modern elliptic curves are defined over such fields to be able to use FFT.

Undoubtedly, $\sqrt{\cdot} \in \mathbb{F}_q$ can be found through (constant-time) *Tonelli–Shanks's algorithm*. However, it requires $O(\log(q) + \nu^2)$ operations in $\mathbb{F}_q$, namely $O(\log^2(q))$ for ν close to $\log_2(q)$.

The given drawback can be mitigated to $O\big(\log(q) + (\nu/\mu)^2\big)$ ones with $O(2^\mu \nu/\mu)$ storage (where μ is a parameter) by means of *Bernstein's table-lookup variant*.

In practice, this memory overhead is not significant for moderate 2-adicities such as $\nu \approx 32$. However, assuming a secret input, Bernstein's approach is vulnerable to *cache-timing attacks*.

There is a vast literature about secure table lookups. Nevertheless, it is desirable to completely avoid them if possible.

(Cipolla–Lehmer–)Müller's algorithm

For square root extraction in highly 2-adic fields there is also *Müller's algorithm*, which is an enhancement of *Cipolla–Lehmer's algorithm*.

(Cipolla–Lehmer–)Müller's algorithm

For square root extraction in highly 2-adic fields there is also *Müller's algorithm*, which is an enhancement of *Cipolla–Lehmer's algorithm*.

Unfortunately, in contrast to the subsequent stage of the algorithm, its first one is not deterministic. Hence, the algorithm is not so popular among cryptographers.

(Cipolla–Lehmer–)Müller's algorithm

For square root extraction in highly 2-adic fields there is also *Müller's algorithm*, which is an enhancement of *Cipolla–Lehmer's algorithm*.

Unfortunately, in contrast to the subsequent stage of the algorithm, its first one is not deterministic. Hence, the algorithm is not so popular among cryptographers.

More concretely, let's pick once and for all any quadratic non-residue $v \in \mathbb{F}_q$. Suppose that we possess a quadratic residue z^2 with the unknown $z \in \mathbb{F}_q$.

(Cipolla–Lehmer–)Müller's algorithm

For square root extraction in highly 2-adic fields there is also *Müller's algorithm*, which is an enhancement of *Cipolla–Lehmer's algorithm*.

Unfortunately, in contrast to the subsequent stage of the algorithm, its first one is not deterministic. Hence, the algorithm is not so popular among cryptographers.

More concretely, let's pick once and for all any quadratic non-residue $v \in \mathbb{F}_q$. Suppose that we possess a quadratic residue z^2 with the unknown $z \in \mathbb{F}_q$.

Cipolla–Lehmer–Müller's algorithm for determining z starts with searching for an element $x \in \mathbb{F}_q$ such that $x^2 - z^2$ is a non-square in $\mathbb{F}_q$. Put another way, $x^2 - z^2 = vy^2$ for some $y \in \mathbb{F}_q$.

(Cipolla–Lehmer–)Müller's algorithm

For square root extraction in highly 2-adic fields there is also *Müller's algorithm*, which is an enhancement of *Cipolla–Lehmer's algorithm*.

Unfortunately, in contrast to the subsequent stage of the algorithm, its first one is not deterministic. Hence, the algorithm is not so popular among cryptographers.

More concretely, let's pick once and for all any quadratic non-residue $v \in \mathbb{F}_q$. Suppose that we possess a quadratic residue z^2 with the unknown $z \in \mathbb{F}_q$.

Cipolla–Lehmer–Müller's algorithm for determining z starts with searching for an element $x \in \mathbb{F}_q$ such that $x^2 - z^2$ is a non-square in $\mathbb{F}_q$. Put another way, $x^2 - z^2 = vy^2$ for some $y \in \mathbb{F}_q$.

There is a long-standing open problem about how to find x in constant polynomial time and without assuming unproven conjectures of number theory.

Châtelet surface

Substituting the separable cubic $\mathbb{F}_q$-polynomial $f(t)$ to the place of z^2, we get the so-called *Châtelet surface* $S_f : x^2 - vy^2 = f(t)$.

Châtelet surface

Substituting the separable cubic $\mathbb{F}_q$-polynomial $f(t)$ to the place of z^2, we get the so-called *Châtelet surface* $S_f : x^2 - vy^2 = f(t)$.

It is well known that S_f is $\mathbb{F}_q$-*unirational*, that is, there is a rational (not necessarily proper) $\mathbb{F}_q$-parametrization $\pi : \mathbb{A}^2 \to S_f$.

Châtelet surface

Substituting the separable cubic $\mathbb{F}_q$-polynomial $f(t)$ to the place of z^2, we get the so-called *Châtelet surface* $S_f : x^2 - vy^2 = f(t)$.

It is well known that S_f is $\mathbb{F}_q$-*unirational*, that is, there is a rational (not necessarily proper) $\mathbb{F}_q$-parametrization $\pi : \mathbb{A}^2 \to S_f$.

Thereby, we are able to generate for free points $(x, y, t) \in S_f(\mathbb{F}_q)$ to execute Cipolla–Lehmer–Müller's algorithm of finding $\sqrt{f(t)}$.

Châtelet surface

Substituting the separable cubic $\mathbb{F}_q$-polynomial $f(t)$ to the place of z^2, we get the so-called *Châtelet surface* $S_f : x^2 - vy^2 = f(t)$.

It is well known that S_f is $\mathbb{F}_q$-*unirational*, that is, there is a rational (not necessarily proper) $\mathbb{F}_q$-parametrization $\pi : \mathbb{A}^2 \to S_f$.

Thereby, we are able to generate for free points $(x, y, t) \in S_f(\mathbb{F}_q)$ to execute Cipolla–Lehmer–Müller's algorithm of finding $\sqrt{f(t)}$.

Only the element x is essentially necessary in the algorithm, but y will not hurt in its low-level optimizations.

Châtelet surface

Substituting the separable cubic $\mathbb{F}_q$-polynomial $f(t)$ to the place of z^2, we get the so-called *Châtelet surface* $S_f : x^2 - vy^2 = f(t)$.

It is well known that S_f is $\mathbb{F}_q$-*unirational*, that is, there is a rational (not necessarily proper) $\mathbb{F}_q$-parametrization $\pi : \mathbb{A}^2 \to S_f$.

Thereby, we are able to generate for free points $(x, y, t) \in S_f(\mathbb{F}_q)$ to execute Cipolla–Lehmer–Müller's algorithm of finding $\sqrt{f(t)}$.

Only the element x is essentially necessary in the algorithm, but y will not hurt in its low-level optimizations.

The trouble is that $f(t)$ may be a non-square in $\mathbb{F}_q$. So, the parametrization π does not give a hash function to $E(\mathbb{F}_q)$, but just to $E(\mathbb{F}_q) \cup E^T(\mathbb{F}_q)$, where $E^T : vy^2 = f(x)$ is the (unique) quadratic twist of E.

Generalized Châtelet surface

To fix the given imperfection, it is suggested to consider a genus 2 curve $H\colon s^2 = h(t)$ having two (quadratic) $\mathbb{F}_q$-covers $\varphi\colon H \to E$ and $\varphi^T\colon H \to E^T$.

Generalized Châtelet surface

To fix the given imperfection, it is suggested to consider a genus 2 curve $H\colon s^2 = h(t)$ having two (quadratic) $\mathbb{F}_q$-covers $\varphi\colon H \to E$ and $\varphi^T\colon H \to E^T$.

The desired H is derived for the curve E of any j-invariant $\neq 0, 1728$ by gluing somehow the curves E, E^T along their 2-torsion subgroups.

Generalized Châtelet surface

To fix the given imperfection, it is suggested to consider a genus 2 curve $H\colon s^2 = h(t)$ having two (quadratic) $\mathbb{F}_q$-covers $\varphi\colon H \to E$ and $\varphi^T\colon H \to E^T$.

The desired H is derived for the curve E of any j-invariant $\neq 0, 1728$ by gluing somehow the curves E, E^T along their 2-torsion subgroups.

Introduce $H^T\colon vs^2 = h(t)$, the quadratic hyperelliptic twist of H.

Generalized Châtelet surface

To fix the given imperfection, it is suggested to consider a genus 2 curve $H\colon s^2 = h(t)$ having two (quadratic) $\mathbb{F}_q$-covers $\varphi\colon H \to E$ and $\varphi^T\colon H \to E^T$.

The desired H is derived for the curve E of any j-invariant $\neq 0, 1728$ by gluing somehow the curves E, E^T along their 2-torsion subgroups.

Introduce $H^T\colon vs^2 = h(t)$, the quadratic hyperelliptic twist of H.

Up to the isomorphism $(x, y) \mapsto (x, vy)$, formulas of φ^T equally define an $\mathbb{F}_q$-cover $H^T \to E$. By abuse of notation, it will be also denoted by φ^T.

Generalized Châtelet surface

To fix the given imperfection, it is suggested to consider a genus 2 curve $H\colon s^2 = h(t)$ having two (quadratic) $\mathbb{F}_q$-covers $\varphi\colon H \to E$ and $\varphi^T\colon H \to E^T$.

The desired H is derived for the curve E of any j-invariant $\neq 0, 1728$ by gluing somehow the curves E, E^T along their 2-torsion subgroups.

Introduce $H^T\colon vs^2 = h(t)$, the quadratic hyperelliptic twist of H.

Up to the isomorphism $(x, y) \mapsto (x, vy)$, formulas of φ^T equally define an $\mathbb{F}_q$-cover $H^T \to E$. By abuse of notation, it will be also denoted by φ^T.

The *generalized Châtelet surface* $S_h\colon x^2 - vy^2 = h(t)$ turns out to be still $\mathbb{F}_q$-unirational for our degree 6 polynomials $h(t)$. And the corresponding formulas are elementary.

Generalized Châtelet surface

To fix the given imperfection, it is suggested to consider a genus 2 curve $H\colon s^2 = h(t)$ having two (quadratic) $\mathbb{F}_q$-covers $\varphi\colon H \to E$ and $\varphi^T\colon H \to E^T$.

The desired H is derived for the curve E of any j-invariant $\neq 0, 1728$ by gluing somehow the curves E, E^T along their 2-torsion subgroups.

Introduce $H^T\colon vs^2 = h(t)$, the quadratic hyperelliptic twist of H.

Up to the isomorphism $(x, y) \mapsto (x, vy)$, formulas of φ^T equally define an $\mathbb{F}_q$-cover $H^T \to E$. By abuse of notation, it will be also denoted by φ^T.

The *generalized Châtelet surface* $S_h\colon x^2 - vy^2 = h(t)$ turns out to be still $\mathbb{F}_q$-unirational for our degree 6 polynomials $h(t)$. And the corresponding formulas are elementary.

Thus, we get a map into $E(\mathbb{F}_q)$ through $H(\mathbb{F}_q) \cup H^T(\mathbb{F}_q)$.

Running time of Müller's algorithm

The unique bottleneck of Müller's algorithm is computing the n-th element of the non-full *Lucas sequence*

$$V_i := aV_{i-1} - V_{i-2}, \qquad V_0 := 2, \qquad V_1 := a$$

for a certain $a \in \mathbb{F}_q$ and $n := (q-1)/4 \in \mathbb{N}$.

Running time of Müller's algorithm

The unique bottleneck of Müller's algorithm is computing the n-th element of the non-full *Lucas sequence*

$$V_i := aV_{i-1} - V_{i-2}, \qquad V_0 := 2, \qquad V_1 := a$$

for a certain $a \in \mathbb{F}_q$ and $n := (q-1)/4 \in \mathbb{N}$.

For determining V_n we possess *Postl's algorithm*, which performs $\approx 2\log_2(q)$ multiplications in $\mathbb{F}_q$. In fact, there is a folklore trick reducing the running time to $\approx 2\log_2(q) - \nu$ ones.

Running time of Müller's algorithm

The unique bottleneck of Müller's algorithm is computing the n-th element of the non-full *Lucas sequence*

$$V_i := aV_{i-1} - V_{i-2}, \qquad V_0 := 2, \qquad V_1 := a$$

for a certain $a \in \mathbb{F}_q$ and $n := (q-1)/4 \in \mathbb{N}$.

For determining V_n we possess *Postl's algorithm*, which performs $\approx 2\log_2(q)$ multiplications in $\mathbb{F}_q$. In fact, there is a folklore trick reducing the running time to $\approx 2\log_2(q) - \nu$ ones.

Taking it into account, Postl's algorithm may be faster than (or at least of the same performance as) the exponentiation in $\mathbb{F}_q$ to some power m of length $\approx \log_2(q)$ and of Hamming weight $\omega \lesssim \log_2(q)$.

Running time of Müller's algorithm

The unique bottleneck of Müller's algorithm is computing the n-th element of the non-full *Lucas sequence*

$$V_i := aV_{i-1} - V_{i-2}, \qquad V_0 := 2, \qquad V_1 := a$$

for a certain $a \in \mathbb{F}_q$ and $n := (q-1)/4 \in \mathbb{N}$.

For determining V_n we possess *Postl's algorithm*, which performs $\approx 2\log_2(q)$ multiplications in $\mathbb{F}_q$. In fact, there is a folklore trick reducing the running time to $\approx 2\log_2(q) - \nu$ ones.

Taking it into account, Postl's algorithm may be faster than (or at least of the same performance as) the exponentiation in $\mathbb{F}_q$ to some power m of length $\approx \log_2(q)$ and of Hamming weight $\omega \lesssim \log_2(q)$.

This happens when

$$2\log_2(q) - \nu \lesssim \log_2(q) + \omega \qquad \Leftrightarrow \qquad \log_2(q) \lesssim \nu + \omega$$

if the conventional binary exponentiation method is applied.

Other hash functions $\mathcal{H}_k$ suitable for large ν

Whenever $q \equiv 1 \pmod 3$, almost all previous hash functions to elliptic curves need to extract a square root in $\mathbb{F}_q$.

k	Year	Author	Bottleneck	Conditions
1	2009	Icart	$\sqrt[3]{\cdot}$	$q \equiv 2 \pmod 3$
2	2022	K.	$\sqrt[3]{\cdot}$	$q \equiv 1 \pmod 3$, $a = 0$, $\sqrt{b} \in \mathbb{F}_q$
3	2023	K.	$\sqrt[7]{\cdot}$	$q \equiv 2, 4 \pmod 7$, j-invariant $-3^3 5^3$

Other hash functions $\mathcal{H}_k$ suitable for large ν

Whenever $q \equiv 1 \pmod 3$, almost all previous hash functions to elliptic curves need to extract a square root in $\mathbb{F}_q$.

k	Year	Author	Bottleneck	Conditions
1	2009	Icart	$\sqrt[3]{\cdot}$	$q \equiv 2 \pmod 3$
2	2022	K.		$q \equiv 1 \pmod 3$, $a = 0$, $\sqrt{b} \in \mathbb{F}_q$
3	2023		$\sqrt[7]{\cdot}$	$q \equiv 2, 4 \pmod 7$, j-invariant $-3^3 5^3$

Icart constructs (by elementary reasoning) a cyclic trigonal $\mathbb{F}_q$-curve $T : y^3 = g(x)$ and an $\mathbb{F}_q$-cover $T \to E$. At the same time, when $q \equiv 2 \pmod 3$, we have the bijection $\sqrt[3]{\cdot} : \mathbb{F}_q \to \mathbb{F}_q$.

Other hash functions $\mathcal{H}_k$ suitable for large ν

Whenever $q \equiv 1 \pmod 3$, almost all previous hash functions to elliptic curves need to extract a square root in $\mathbb{F}_q$.

k	Year	Author	Bottleneck	Conditions
1	2009	Icart	$\sqrt[3]{\cdot}$	$q \equiv 2 \pmod 3$
2	2022	K.		$q \equiv 1 \pmod 3$, $a = 0$, $\sqrt{b} \in \mathbb{F}_q$
3	2023		$\sqrt[7]{\cdot}$	$q \equiv 2, 4 \pmod 7$, j-invariant $-3^3 5^3$

Icart constructs (by elementary reasoning) a cyclic trigonal $\mathbb{F}_q$-curve $T : y^3 = g(x)$ and an $\mathbb{F}_q$-cover $T \to E$. At the same time, when $q \equiv 2 \pmod 3$, we have the bijection $\sqrt[3]{\cdot} : \mathbb{F}_q \to \mathbb{F}_q$.

In turn, $\mathcal{H}_2$, $\mathcal{H}_3$ are also based on interesting algebraic geometry, namely on some Calabi–Yau threefold and the Klein quartic.

Standardized curve NIST P-224

As far as the speaker knows, the 2-adicity $\nu = 96$ is maximal among the basic fields of standardized elliptic curves. It is attained by the curve NIST P-224 from the American standard recently updated.

Standardized curve NIST P-224

As far as the speaker knows, the 2-adicity $\nu = 96$ is maximal among the basic fields of standardized elliptic curves. It is attained by the curve NIST P-224 from the American standard recently updated.

The curve is defined over a field $\mathbb{F}_q$ of length $\lceil \log_2(q) \rceil = 224$. Its order $q \equiv 1 \pmod 3$, hence Icart's hash function $\mathcal{H}_1$ is not applicable to the curve as opposed to the new one $\mathcal{H}_{new}$.

Standardized curve NIST P-224

As far as the speaker knows, the 2-adicity $\nu = 96$ is maximal among the basic fields of standardized elliptic curves. It is attained by the curve NIST P-224 from the American standard recently updated.

The curve is defined over a field $\mathbb{F}_q$ of length $\lceil \log_2(q) \rceil = 224$. Its order $q \equiv 1 \pmod 3$, hence Icart's hash function $\mathcal{H}_1$ is not applicable to the curve as opposed to the new one $\mathcal{H}_{new}$.

Before it, the simplified SWU hash function $\mathcal{H}_{sSWU}$ was the best for NIST P-224.

Standardized curve NIST P-224

As far as the speaker knows, the 2-adicity $\nu = 96$ is maximal among the basic fields of standardized elliptic curves. It is attained by the curve NIST P-224 from the American standard recently updated.

The curve is defined over a field $\mathbb{F}_q$ of length $\lceil \log_2(q) \rceil = 224$. Its order $q \equiv 1 \pmod 3$, hence Icart's hash function $\mathcal{H}_1$ is not applicable to the curve as opposed to the new one $\mathcal{H}_{new}$.

Before it, the simplified SWU hash function $\mathcal{H}_{sSWU}$ was the best for NIST P-224.

A detailed analysis shows that Tonelli–Shanks's algorithm requires ≈ 5009 multiplications in the field $\mathbb{F}_q$ under consideration. In turn, Müller's algorithm performs $\approx 2 \cdot 224 - 96 = 352$ ones.

Standardized curve NIST P-224

As far as the speaker knows, the 2-adicity $\nu = 96$ is maximal among the basic fields of standardized elliptic curves. It is attained by the curve NIST P-224 from the American standard recently updated.

The curve is defined over a field $\mathbb{F}_q$ of length $\lceil \log_2(q) \rceil = 224$. Its order $q \equiv 1 \pmod 3$, hence Icart's hash function $\mathcal{H}_1$ is not applicable to the curve as opposed to the new one $\mathcal{H}_{new}$.

Before it, the simplified SWU hash function $\mathcal{H}_{sSWU}$ was the best for NIST P-224.

A detailed analysis shows that Tonelli–Shanks's algorithm requires ≈ 5009 multiplications in the field $\mathbb{F}_q$ under consideration. In turn, Müller's algorithm performs $\approx 2 \cdot 224 - 96 = 352$ ones.

To sum up, $\mathcal{H}_{new}$ carries out ≈ 4657 fewer multiplications than $\mathcal{H}_{sSWU}$. In other terms, there is an acceleration of about 14 times.

Conclusions

Seemingly, the hashing approach of the present talk can be extended to elliptic $\mathbb{F}_q$-curves E of $j = 1728$ and to those of $j = 0$ whose Frobenius trace has a small divisor d.

Conclusions

Seemingly, the hashing approach of the present talk can be extended to elliptic $\mathbb{F}_q$-curves E of $j = 1728$ and to those of $j = 0$ whose Frobenius trace has a small divisor d.

For such curves, we are still able to glue E, E^T along their 4-torsion and d-torsion subgroups, respectively, obtaining desired covers $\varphi\colon H \to E$ and $\varphi^T\colon H \to E^T$ from a genus 2 curve H.

Conclusions

Seemingly, the hashing approach of the present talk can be extended to elliptic $\mathbb{F}_q$-curves E of $j = 1728$ and to those of $j = 0$ whose Frobenius trace has a small divisor d.

For such curves, we are still able to glue E, E^T along their 4-torsion and d-torsion subgroups, respectively, obtaining desired covers $\varphi\colon H \to E$ and $\varphi^T\colon H \to E^T$ from a genus 2 curve H.

The only potential obstacle on the path is non-unirationality over $\mathbb{F}_q$ of the new generalized Châtelet surface S_h.

Conclusions

Seemingly, the hashing approach of the present talk can be extended to elliptic $\mathbb{F}_q$-curves E of $j = 1728$ and to those of $j = 0$ whose Frobenius trace has a small divisor d.

For such curves, we are still able to glue E, E^T along their 4-torsion and d-torsion subgroups, respectively, obtaining desired covers $\varphi\colon H \to E$ and $\varphi^T\colon H \to E^T$ from a genus 2 curve H.

The only potential obstacle on the path is non-unirationality over $\mathbb{F}_q$ of the new generalized Châtelet surface S_h.

Meanwhile, (most) modern curves of $j = 0, 1728$ over highly 2-adic fields are initially equipped with an $\mathbb{F}_q$-isogeny χ of small degree from another elliptic curve. Therefore, indirect hashing via χ takes place.

Conclusions

Seemingly, the hashing approach of the present talk can be extended to elliptic $\mathbb{F}_q$-curves E of $j = 1728$ and to those of $j = 0$ whose Frobenius trace has a small divisor d.

For such curves, we are still able to glue E, E^T along their 4-torsion and d-torsion subgroups, respectively, obtaining desired covers $\varphi\colon H \to E$ and $\varphi^T\colon H \to E^T$ from a genus 2 curve H.

The only potential obstacle on the path is non-unirationality over $\mathbb{F}_q$ of the new generalized Châtelet surface S_h.

Meanwhile, (most) modern curves of $j = 0, 1728$ over highly 2-adic fields are initially equipped with an $\mathbb{F}_q$-isogeny χ of small degree from another elliptic curve. Therefore, indirect hashing via χ takes place.

At the moment, we hereby handled (almost) all real-world elliptic curves over fields of large 2-adicity ν.

Main references on the topic of other authors

1. Müller S., *On the computation of square roots in finite fields*, Designs, Codes and Cryptography, 2004.
2. Shallue A., van de Woestijne C. E., *Construction of rational points on elliptic curves over finite fields*, ANTS 2006.
3. Icart T., *How to hash into elliptic curves*, CRYPTO 2009.
4. Brier E., Coron J.-S., Icart T., Madore D., Randriam H., Tibouchi M., *Efficient indifferentiable hashing into ordinary elliptic curves*, CRYPTO 2010.
5. Farashahi R. R., Fouque P.-A., Shparlinski I. E., Tibouchi M., Voloch J. F., *Indifferentiable deterministic hashing to elliptic and hyperelliptic curves*, Mathematics of Computation, 2013.
6. Chávez-Saab J., Rodriguez-Henriquez F., Tibouchi M., *SwiftEC: Shallue–van de Woestijne indifferentiable function to elliptic curves. Faster indifferentiable hashing to most elliptic curves*, ASIACRYPT 2022.
7. Faz-Hernandez A., Scott S., Sullivan N., Wahby R. S., Wood C. A., *Hashing to elliptic curves*, Internet-draft (CFRG), 2022.

Merci de votre attention !