INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2020, VOL. 66, NO. 3, PP. 465-472
Manuscript received February 28, 2020; revised July, 2020. DOI: 10.24425/ijet.2020.131900

Optimal Strategies for Computation of Degree ℓ^n Isogenies for SIDH

Michał Wroński, and Andrzej Chojnacki

***Abstract*—This article presents methods and algorithms for the computation of isogenies of degree ℓ^n. Some of these methods are obtained using recurrence equations and generating functions. A standard multiplication based algorithm for computation of isogeny of degree ℓ^n has time complexity equal to $O(n^2 M(n \log n))$, where $M(N)$ denotes the cost of integers of size N multiplication. The memory complexity of this algorithm is equal to $O(n \log(n \log(n)))$. In this article are presented algorithms for:**

- **determination of optimal strategy for computation of degree ℓ^n isogeny,**
- **determination of cost of optimal strategy of computation of ℓ^n isogeny using solutions of recurrence equations,**
- **determination of cost of optimal strategy of computation of ℓ^n isogeny using recurrence equations,**

where optimality in this context means that, for the given parameters, no other strategy exists that requires fewer operations for computation of isogeny.

Also this article presents a method using generating functions for obtaining the solutions of sequences (u_m) and (c_m) where c_m denotes the cost of computations of isogeny of degree ℓ^{u_m} for given costs p, q of ℓ-isogeny computation and ℓ-isogeny evaluation. These solutions are also used in the construction of the algorithms presented in this article.

***Keywords*—SIDH, Optimal strategies, Generating functions**

I. Introduction

NOWADAYS post-quantum cryptography is one of the most important areas in modern cryptography.

It is believed that the SIDH (Supersingular Isogeny Diffie-Hellman) algorithm is quantum resistant and therefore may be used in post-quantum cryptography protocols. One of the most time-consuming parts of SIDH is the computation of degree ℓ^n isogenies, where ℓ is a positive integer, most often equal to 2 or 3 and n is a positive integer, most often from the set $\{100, 101, \ldots, 1000\}$. This article will show how to efficiently determine the optimal strategy for degree ℓ^n computation, and its cost using linear recurrences and generating functions. In this article, strategy means the method of computation of something, and a strategy will be optimal if, and only if, for the given parameters no other strategy exists that allows one to compute something using a smaller number of operations. The number of operations required for computations using a given strategy is called the cost of the strategy.

M. Wroński is with Institute of Mathematics and Cryptology, Faculty of Cybernetics, Military University of Technology, Warsaw, Poland (e-mail: michal.wronski@wat.edu.pl).

A. Chojnacki is with Institute of Computer and Information Systems, Faculty of Cybernetics, Military University of Technology, Warsaw, Poland (e-mail: andrzej.chojnacki@wat.edu.pl).

The proof of the optimality of the strategy, and methods of determination of optimal strategy of degree ℓ^n computation using a dynamic programming approach was given for the first time in [1] by Jao, de Feo and Plut. In the same article, the authors also showed linear recurrence equations for the cost of this optimal strategy.

Instead of that, nowadays, in practical implementations of the SIDH algorithm and searching for the optimal strategy of degree ℓ^n computation, a dynamic programming approach is used which is inefficient.

This article will show how to use ideas from [1] for the construction of fast and computationally efficient algorithms for determining the optimal strategy of degree ℓ^n computation. In addition, how to compute the cost of such a strategy using the solutions of linear recurrences using generating functions will be shown.

II. SIDH

SIDH is a post-quantum algorithm developed by Jao, de Feo and Plut and first described in [1]. The algorithm is based on the difficulty of finding an isogeny of high degree between two known supersingular elliptic curves. It is believed that SIDH is quantum resistant because the endomorphism ring of supersingular elliptic curves is not commutative. A SIKE algorithm, which is SIDH with key encapsulation, is one of the candidates to become a post-quantum key agreement protocol.

A. SIDH algorithm

This description is based on the work of Costello and Hisil [2]. Let $r = f\,\ell_A^{e_A}\,\ell_B^{e_B} \pm 1$ be a large prime where ℓ_A is most often equal to 2, ℓ_B is most often equal to 3, $gcd(\ell_A, \ell_B) = 1$, $\ell_A^{e_A} \approx \ell_B^{e_B}$ and $e_A, e_B = 100, 101, \ldots, 1000$ in most practical solutions. Additionally, f is a small integer (compared to r) called a cofactor. The SIDH algorithm works in the isogeny class of supersingular elliptic curves over $\mathbb{F}_{r^2}$. Cardinality of all of these curves is equal to $(r \pm 1)^2 = (f\ell_A^{e_A}\ell_B^{e_B})^2$. If E is a public starting curve in this isogeny class, then to generate its public key, the first participant of the SIDH protocol (Alice) chooses a secret subgroup G_A of order $\ell_A^{e_A}$ on curve E and computes her public key E/G_A, which means that the elliptic curve E/G_A is isogenous to elliptic curve E and the kernel of this isogeny is the subgroup generated by point $P_A + [u_A]Q_A$ and then $G_A = \langle P_A + [u_A]Q_A \rangle$. Points P_A and Q_A are both of order $\ell_A^{e_A}$, but they belong to different torsion subgroups. In the

same manner, the second participant of the protocol (Bob) chooses a secret subgroup G_B of order $\ell_B^{e_B}$ and computes his public key E/G_B, where $G_B = \langle P_B + [u_B]Q_B \rangle$. Points P_B and Q_B are both of order $\ell_B^{e_B}$, but they belong to different torsion subgroups. After all these computations, the shared secret is $E/\langle G_A, G_B \rangle$, which means that the kernel of isogeny $\varphi : E \to E/\langle G_A, G_B \rangle$ is generated by both subgroups $G_A = \langle P_A + [u_A]Q_A \rangle, G_B = \langle P_B + [u_B]Q_B \rangle$. The security of SIDH is based on the fact that computation of the secret from E, E/G_A and E/G_B is difficult.

In the SIDH scheme all the public keys contain images of certain public points under the isogenies defined by their secret subgroups. During the key generation scheme, Alice not only sends Bob the curve E/G_A but also the image of isogeny ϕ_A at two points, P_B and Q_B. Such linear combinations of these points generate the set of subgroups chosen by Bob. So Alice's public key is $PK_A = (E/G_A, \phi_A(P_B), \phi_A(Q_B))$. In the same manner, the linear combinations of P_A and Q_A generate the set of subgroups chosen by Alice and Bob's public key is $PK_B = (E/G_B, \phi_B(P_A), \phi_B(Q_A))$.

During key generation Alice needs to randomly choose a secret integer $u_A \in \mathbb{Z}_{\ell_A{}^{e_A}}$, by computing $G_A = \langle P_A + [u_A]Q_A \rangle$. After receiving Bob's public key she is able to compute $E/\langle G_A, G_B \rangle = (E/G_B)/\langle \phi_B(P_A) + [u_A]\phi_B(Q_A) \rangle$. Bob does similar computations, and then both parties have the shared secret, which is the j-invariant of the elliptic curve $E/\langle G_A, G_B \rangle$.

III. Determination of optimal strategy of isogeny of degree ℓ^n with given kernel computation

Jao, de Feo and Plut in [1] showed how to determine the optimal strategy of degree ℓ^n computation. They showed how to compute such a strategy using a dynamic programming approach. The time complexity of this algorithm is equal to $O(n^2 M(n \log n))$ and memory complexity is equal to $O(n \log(n \log(n)))$, where $M(N)$ is the complexity of multiplication of two integers of size N. Also shown in [1] are recurrence equations for the cost of computations of isogeny of degree ℓ^n, where the given parameters are: cost of computation of point scalar multiplication by ℓ and cost of evaluation of isogeny of degree ℓ, which are denoted by p and q, where $p, q \in \mathbb{N}_+$.

First we shall describe an example of computing an isogeny for a given degree. Let's consider that one needs to compute an isogeny of degree ℓ^n with kernel R_0, where $n = 6$ and ℓ is a positive integer, for example $\ell = 2$. This example also comes from [1]. One method of computation of such isogeny may be the computation of all elements of the kernel, of which there are ℓ^6 such elements. If the exponent n is a large integer, then it will be difficult to compute all elements of the kernel and therefore this method of computation of isogeny of degree ℓ^n is inefficient.

The second method, which seems to be much more computationally efficient, is a multiplication based strategy (MBS). This method uses factorisation of isogeny ϕ of degree ℓ^n into isogenies of degree ℓ. In this example $\phi = \phi_5 \circ \phi_4 \circ \phi_3 \circ \phi_2 \circ \phi_1 \circ \phi_0$. The computation of isogeny of degree ℓ^6 is described below:

$$\phi_0 : E_0 \to E_1 = E_0/\langle [\ell^5]R_0 \rangle, R_1 = \phi(R_0),$$
$$\phi_1 : E_1 \to E_2 = E_1/\langle [\ell^4]R_1 \rangle, R_2 = \phi(R_1),$$
$$\phi_2 : E_2 \to E_3 = E_2/\langle [\ell^3]R_2 \rangle, R_3 = \phi(R_2),$$
$$\phi_3 : E_3 \to E_4 = E_3/\langle [\ell^2]R_3 \rangle, R_4 = \phi(R_3),$$
$$\phi_4 : E_4 \to E_5 = E_4/\langle [\ell]R_4 \rangle, R_5 = \phi(R_4),$$
$$\phi_5 : E_5 \to E_6 = E_5/\langle R_5 \rangle, R_6 = \phi(R_5).$$

The figure 1 shows lattice of computations for $n = 6$.

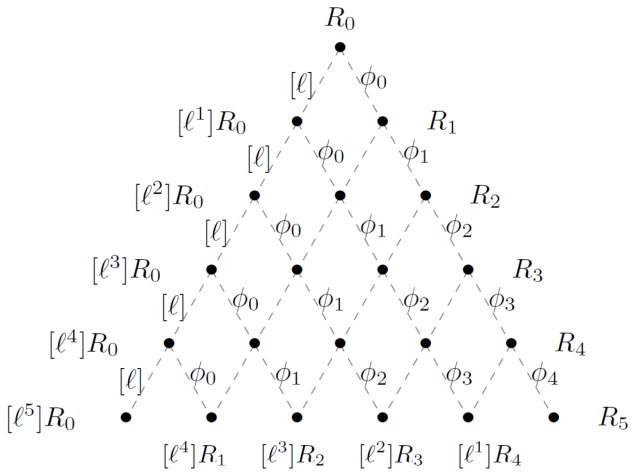

Fig. 1. Computational structure of the construction of isogeny ϕ which is given by $\phi = \phi_5 \circ \phi_4 \circ \phi_3 \circ \phi_2 \circ \phi_1 \circ \phi_0$. Figure comes from the [1].

A multiplication based strategy may be represented on the lattice presented in figure 1. In computations in a multiplication based strategy the most frequently used operation is the computation of point scalar multiplication, because the kernels of isogenies are the points $[\ell^5]R_0, [\ell^4]R_1, \ldots, [\ell]R_5$. The multiplication based strategy of computation of isogeny of degree ℓ^6 with given kernel may be interpreted as walking on the tree presented in the figure 2.

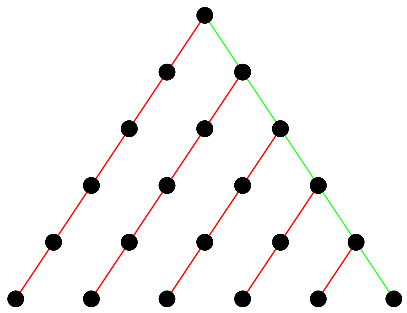

Fig. 2. Multiplication based strategy for isogeny of degree ℓ^6 with given kernel.

For every optimal strategy of computation of isogeny of degree ℓ^n such strategy may be represented by the tree structure. This tree has special properties, because it always

has exactly n leaves and every path from the root to each leaf has length equal to $n-1$. Every tree having such properties and having exactly n leaves will be called a tree of size n. Additionally, every inner node of such tree has exactly one child or two children. The root of this tree, point R_0, is the kernel of the isogeny. The edge from the node to its left child illustrates the computation of point (node) multiplication by ℓ, and the path from the node to its right child illustrates the evaluation of the ℓ-isogeny in the given point (node).

However, MBS is more computationally efficient than computation of all elements of the kernel, it is not in general the optimal strategy.

Multiplication based strategy requires computation of $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$ multiplications by ℓ and $n-1$ evaluations of ℓ-isogenies. The next step to get a more computationally efficient strategy (requiring a smaller number of operations) may be to get a balance between the number of ℓ-multiplications and evaluations of ℓ-isogenies. Such strategy is called a balanced strategy (BS). This strategy is in general more computationally efficient than MBS. If the costs of computation of ℓ-multiplication and evaluation of ℓ-isogeny are equal, then BS is an optimal strategy for computation of a strategy of degree ℓ^n with a given kernel, because the tree representing this strategy has the smallest number of edges.

The costs of computation of point scalar multiplication by ℓ and evaluation of ℓ-isogeny are not in general the same, so (in general) BS is not the optimal strategy (OS). The method of searching for OS was described by De Feo, Jao and Plut in [1].

They showed that every OS for isogeny of degree ℓ^n may be built using optimal strategies of degree ℓ^k and ℓ^{n-k}, for some $k = 1, 2, \ldots, n-1$. The cost of OS is therefore equal to

$$C_{p,q}(n) = \min_{i=1,\ldots,n-1} \left(C_{p,q}(i) + C_{p,q}(n-i) \ + \ (n-i)p + iq\right),$$

where $C_{p,q}(n)$ denotes the cost of computation of ℓ^n-isogeny with cost of ℓ-multiplication equal to p and the cost of evaluation of ℓ-isogeny equal to q.

As was shown in [1], the tree representing the optimal strategy for n, p, q is built with two trees, being the optimal strategies for some k and $n-k$ with the same costs p and q. Therefore, the tree representing the optimal strategy has size n, and its left subtree (substrategy) has size k and is connected with the root (kernel of isogeny) by $n-k$ ℓ-multiplications. In the same way, the right subtree (substrategy) has size $n-k$ and is connected with root by k computations of ℓ-isogeny. If $p < q$, then the size of the left subtree will be smaller than the size of right subtree to minimize the cost of connections of these subtrees with the root (which means that every subtree is connected with the root, the first using only edges denoting ℓ-multiplication, the second, using only edges denoting the evaluation of ℓ-isogeny). If $p > q$, then the size of left subtree will be bigger. If $p = q$ one may use balanced strategy, then the size of the left and the right subtrees will be similar (no matter if the left or right subtree has a bigger value of n, and in some situations they will have the same size).

Figure 3 shows an example of such optimal strategy for $n = 16, p = 3, q = 7$.

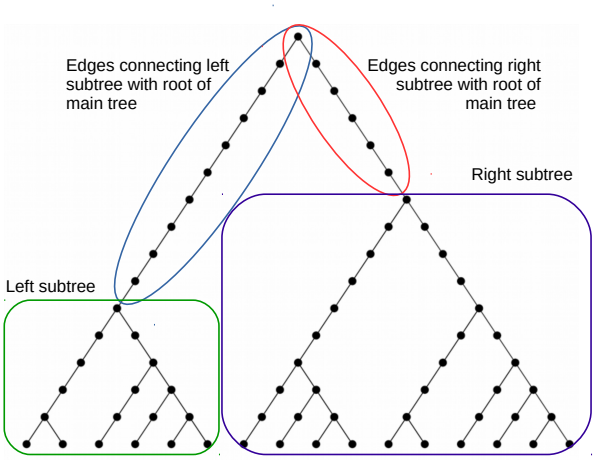

Fig. 3. Optimal strategy for $n = 16, p = 3, q = 7$.

Let's say that the tree being the optimal strategy for $n = 16, p = 3, q = 7$ is built using two trees, being optimal strategies for $n = 6$ and $n = 10$ with the same costs of p and q. The left subtree of size 6 (substrategy) is connected with the root (kernel of isogeny) by 10 ℓ-multiplications and the right subtree (substrategy) is connected with the kernel by 6 evaluations of ℓ-isogeny.

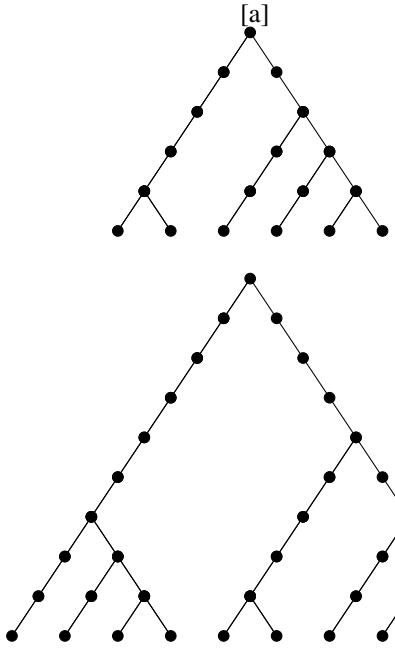

Fig. 4. Optimal strategies for $n = 6, p = 3, q = 7$ (fig. (a)) and $n = 10, p = 3, q = 7$ (fig. (b)).

Jao, de Feo and Plut also found that the cost of optimal strategies for isogeny of degree ℓ^n with cost of point ℓ-multiplication and cost of ℓ-isogeny evaluation being equal to p and q may be described using recurrence equations. Because OS for (p, q) are symmetric to the OS for (q, p), it is assumed that $p < q$. If $p = q$, then BS is OS. Additionally, p and q are positive integers. The theorem describing the cost of OS for a given n, p and q is described below [1].

Theorem 1: For given costs p and q, where $p < q$, let's (u_m) be the sequence which is defined by recurrence equation $u_0 = u_1 = \cdots = u_p = 0, u_{p+1} = \cdots = u_{p+q} = 1$ and $u_m = u_{m-p} + u_{m-q}$, for $m \geq p+q+1$. Let $f(n) : \mathbb{N} \to \mathbb{N}$ be the function of cost of OS for given n, p, q. Function f is given by recurrence equation $f(0) = f(1) = 0, f(n) = f(u_m) + m(n - u_m)$ for all $n = u_m, u_m + 1, \ldots, u_{m+1}$. Additionally, f may be described by the recurrence equation

$$f(u_m) = f(u_{m-p}) + f(u_{m-q}) + pu_{m-p} + qu_{m-q}, m \geq p+q+1.$$

In [1] it was also shown, that if $u_{m-2} \leq k$ and $u_{m-1} \leq n - k \leq u_m$, then

$$C_{p,q}(n) = C_{p,q}(k) + C_{p,q}(n-k) \ + \ (n-k)p + kq,$$

is always the cost of OS of computation of isogeny of degree ℓ^n for given costs p and q.

IV. Determination of the cost of the optimal strategy using generating functions

The content presented in this section is new and presents how the recurrence equations, given in section 1, may be solved.

The givens are $p, q \in \mathbb{N}_+$, where $q > p$ and the recurrence equation is

$$a_m = \begin{cases} 1, for\ m = 0, \\ 0, \ for\ 0 < m < p, \\ a_{m-p}, \ for\ p \leq m < q, \\ a_{m-p} + a_{m-q}, \ for\ m \geq q. \end{cases}$$

The generating function $A(z)$ of the sequence (a_m) is equal to

$$A(z) = 1 + \sum_{m=p}^{q-1} a_{m-p} z^m + \sum_{m \geq q} (a_{m-p} + a_{m-q}) z^m =$$

$$= 1 + \sum_{m \geq p} a_{m-p} z^m + \sum_{m \geq q} a_{m-q} z^m =$$

$$= 1 + z^p \sum_{m \geq 0} a_m z^m + z^q \sum_{m \geq 0} a_m z^m =$$

$$= 1 + z^p A(z) + z^q A(z).$$

Finally

$$A(z) = \frac{1}{1 - z^p - z^q}.$$

Using partial fractions representation of $A(z)$, one can find the solution to the sequence (a_m).

For the same numbers p, q, one can use generating functions to solve the recurrence equation for the sequence (w_m):

$$w_m = \begin{cases} 1, \ for\ m \leq q-1, \\ w_{m-p} + w_{m-q}, \ for\ m \geq q. \end{cases}$$

The generating function $W(z)$ of the sequence (w_m) is equal to:

$$W(z) = \sum_{m \geq 0} w_m z^m =$$

$$= \sum_{m=0}^{q-1} z^m + \sum_{m \geq q} w_{m-p} z^m + \sum_{m \geq q} w_{m-q} z^m =$$

$$= \frac{1 - z^q}{1 - z} + z^p \sum_{m \geq q} w_{m-p} z^{m-p} + z^q \sum_{m \geq q} w_{m-q} z^{m-q} =$$

$$= \frac{1 - z^q}{1 - z} + z^p \sum_{m \geq q-p} w_m z^m + z^q \sum_{m \geq 0} w_m z^m =$$

$$= \frac{1 - z^q}{1 - z} + z^p \sum_{m \geq 0} w_m z^m - \sum_{m=0}^{q-p-1} w_m z^m + z^q W(z) =$$

$$= \frac{1 - z^q}{1 - z} + (z^p + z^q) W(z) - \frac{z^p - z^q}{1 - z}.$$

Therefore $W(z) = \frac{1 - z^p}{(1-z)(1-z^p-z^q)} = \frac{1-z^p}{1-z} A(z)$.

Let's now consider the sequence (u_m), which was analyzed earlier, which is defined by the recurrence equation:

$$u_m = \begin{cases} 0, \ for\ m \leq p, \\ 1, \ for\ p < m \leq p+q, \\ u_{m-q} + u_{m-p}, \ for\ m > p+q. \end{cases}$$

The sequence (u_m) shifted $p+1$ times to the right is equal to the sequence (w_m). Therefore the generating function $U(z)$ of this sequence is equal to $U(z) = z^{p+1} W(z) = z^{p+1} \frac{1-z^p}{(1-z)(1-z^p-z^q)} = \frac{z^{p+1}(1-z^p)}{1-z} A(z)$.

Let c_m denote the cost of the optimal strategy whose size is equal to u_m. According to the previous considerations, the sequence (c_m) defines the recurrence equation

$$c_m = \begin{cases} 0, \ for\ m \leq p+q, \\ c_{m-p} + c_{m-q} + pu_{m-p} + qu_{m-q}, \ for\ m > p+q, \end{cases}$$

where the sequence (u_m) is defined as above. Let $C(z)$ denote the generating function of the sequence (c_m). Then

$$C(z) = \sum_{m \geq 0} c_m z^m =$$

$$= \sum_{m \geq p+q} c_{m-p} z^m + \sum_{m \geq p+q} c_{m-q} z^m + p \sum_{m \geq p+q} u_{m-p} z^m + q \sum_{m \geq p+q} u_{m-q} z^m =$$

$$= z^p \sum_{m \geq q+1} c_m z^m + z^q \sum_{m \geq p+1} c_m z^m + pz^p \sum_{m \geq q+1} u_m z^m + qz^q \sum_{m \geq p+1} u_m z^m =$$

$$= (z^p + z^q) C(z) + (pz^p + qz^q) U(z) - pz^p \sum_{m=0}^{q} c_m z^m - qz^q \sum_{m=0}^{p} u_m z^m.$$

But $\sum_{m=0}^{q} c_m z^m = \sum_{m=0}^{p} u_m z^m = 0$. Therefore

$$C(z) = (z^p + z^q) C(z) + (pz^p + qz^q) U(z),$$

so

$$C(z) = \frac{(pz^p + qz^q) U(z)}{1 - z^p - z^q} =$$

$$= \frac{z^{p+1} (pz^p + qz^q)(1 - z^p)}{(1-z)(1 - z^p - z^q)} A(z) =$$

$$= \frac{z^{p+1} (pz^p + qz^q)(1 - z^p)}{1 - z} A^2(z) =$$

$$= z^{p+q+1} \frac{p(1 - z^q) + q(1 - z^p)}{1 - z} A^2(z).$$

Let's assume that the solution of the sequence (a_m) for which the generating function is equal to $A(z)$ is known. Function $A^2(z)$ is the generating function of the product of

generating functions of the sequence (a_m) by itself. The s-th element of such sequence is equal to $\sum_{j=0}^{s} a_j a_{s-j}$. After dividing the function $A^2(z)$ by $(1-z)$, one achieves the generating function of the series, whose m-th element is equal to $\sum_{s=0}^{m}\sum_{j=0}^{s} a_j a_{s-j}$.

After multiplication of the function $\frac{A^2(z)}{1-z}$ by $p(1-z^q)$, one achieves the generating function of subtraction of two sequences: the first one is the previously gained sequence multiplied by p, the second sequence is the sequence previously gained, shifted q times to the right. The final sequence is equal to:

$$\begin{cases} p\sum\limits_{r=0}^{m}\sum\limits_{j=0}^{r} a_j a_{r-j}, \ for\ m < q, \\ p\left(\sum\limits_{r=0}^{m}\sum\limits_{j=0}^{r} a_j a_{r-j} - \sum\limits_{r=0}^{m-q}\sum\limits_{j=0}^{r} a_j a_{r-j}\right), \ for\ m \geq q. \end{cases}$$

After performing analogous operations for the product of $\frac{A^2(z)}{1-z}$ and $q(1-z^p)$ one gets the sequence:

$$\begin{cases} q\sum\limits_{s=0}^{m}\sum\limits_{j=0}^{s} a_j a_{s-j}, \ for\ m < p, \\ q\left(\sum\limits_{s=0}^{m}\sum\limits_{j=0}^{s} a_j a_{s-j} - \sum\limits_{s=0}^{m-p}\sum\limits_{j=0}^{s} a_j a_{s-j}\right), \ for\ m \geq p, \end{cases}$$

and finally one gets the sequence, whose m-th element is equal to

$$\begin{cases} (p+q)\sum\limits_{s=0}^{m}\sum\limits_{j=0}^{s} a_j a_{s-j}, \ for\ m < p, \\ (p+q)\sum\limits_{s=0}^{m}\sum\limits_{j=0}^{s} a_j a_{s-j} \\ -q\sum\limits_{s=0}^{m-p}\sum\limits_{j=0}^{s} a_j a_{s-j}, \ for\ p \leq m < q, \\ (p+q)\sum\limits_{s=0}^{m}\sum\limits_{j=0}^{s} a_j a_{s-j} - q\sum\limits_{s=0}^{m-p}\sum\limits_{j=0}^{s} a_j a_{s-j} \\ -p\sum\limits_{s=0}^{m-q}\sum\limits_{j=0}^{s} a_j a_{s-j}, \ for\ m \geq q. \end{cases}$$

After multiplication of the function $\frac{p(1-z^q)+q(1-z^p)}{1-z}A^2(z)$ by z^{p+q+1}, one achieves the generating function of the last sequence, shifted $(p+q+1)$ places to the right, so one gets the generating function of the sequence, whose m-th element, which is also the m-th element of the sequence (c_m), may be represented as

$$c_m = \begin{cases} 0, \ for\ m \leq p+q, \\ (p+q)\sum\limits_{s=0}^{m-p-q-1}\sum\limits_{j=0}^{s} a_j a_{s-j}, \ for\ p+q < m \leq 2p+q, \\ (p+q)\sum\limits_{s=0}^{m-p-q-1}\sum\limits_{j=0}^{s} a_j a_{s-j} - q\sum\limits_{s=0}^{m-2p-q-1}\sum\limits_{j=0}^{s} a_j a_{s-j}, \\ \ for\ 2p+q < m \leq p+2q, \\ (p+q)\sum\limits_{s=0}^{m-p-q-1}\sum\limits_{j=0}^{s} a_j a_{s-j} - q\sum\limits_{s=0}^{m-2p-q-1}\sum\limits_{j=0}^{s} a_j a_{s-j} \\ -p\sum\limits_{s=0}^{m-p-2q-1}\sum\limits_{j=0}^{s} a_j a_{s-j}, \ for\ m \geq p+2q+1. \end{cases}$$

The formula presented above is the solution of the sequence (c_m). The elements of the sequence (a_m) may be given by symbolic or numeric representation according to which representation is more convenient for us.

However, if one would like to use the generating function $C(z)$ as a representation of the sequence (c_m) to solve this sequence numerically, then it is convenient to find a generating function representation of $C(z)$ which eliminates the existence of $A(z)$. One such representation is

$$C(z) = (qz^q + pz^p)\frac{\sum\limits_{i=p+1}^{2p} z^i}{\left(1-z^p-z^q\right)^2} - p\frac{\sum\limits_{i=2p+1}^{p+q} z^i}{1-z^p-z^q}.$$

Then, using for example MAPLE, one can find partial fractions for the representation of $C(z)$ and finally the solution to the sequence (c_m).

The MAPLE code of the algorithm for searching for numerical solutions of (u_m) and (c_m) is presented below.

Algorithm 1 Algorithm for searching for the solutions (u_m) and (c_n).

Input: $p, q \in \mathbb{N}_+, p < q$
Output: u, c: solutions of $(u_m), (c_m)$
$l := sum(z^i, i = p+1..2*p)$;
$d := 1 - z^p - z^q$;
$f := l/d$;
$g := convert(f, parfrac, z, complex)$;
$with(genfunc)$:
$n := integer \to integer$;
$u := evalf(rgf\_expand(g, z, n))$;
$l3 := sum(z^j, j = 2*p+1..p+q)$;
$l2 := p*z^p + q*z^q$;
$f2 := l2*l/d^2 - p*l3/d$;
$g2 := convert(f2, parfrac, z, complex)$;
$c := evalf(rgf\_expand(g2, z, n))$;
return u, c;

V. Computationally efficient algorithms for searching for the cost of optimal strategy and form of optimal strategy for large values of n

In [3] an algorithm is presented for finding the optimal strategy using the dynamic programming approach. In this algorithm one can use p and q, being positive rational numbers, but unfortunately, for large sizes of n ($n > 10000$) this algorithm is computationally inefficient. Additionally, computational efficiency of this algorithm in practice does not depend on the costs p and q.

In this section an algorithm will be presented which allows one to find the exact value of the cost of the optimal strategy, even if n is a very large number. Additionally, it will be shown how to find substrategies (left and right subtrees of the tree presenting the strategy) of the optimal strategy. It should be assumed that costs p and q are positive integers. For computational efficiency of this algorithm the numbers p and q should not be too large. In most cases p and q may be estimated using the number of multiplications necessary for a given ℓ-multiplication and evaluation of ℓ-isogeny. Other operations, like addition and subtraction have very little influence on the cost of computations, especially for large fields.

The algorithm which has been designed by us uses the asymptotic behavior of the sequence (u_m), which allows us to find m_0, such that for a tree of size n holds $n \approx u_{m_0}$. In most cases, the given value will not be exact and it will be necessary to round this number to the nearest integer.

First, we show how one can find $n \approx u_{m_0}$.

Function $Q(z) = 1 - z^p - z^q$ has precisely q roots. If some $x \in \mathbb{C}$ were the multiple root of this function, then its derivative would be equal to zero at the point $x \in \mathbb{C}$, which means that $Q'(x) = -px^{p-1} - qx^{q-1} = 0$. $x = 0$ is not the root of the function $Q(z)$ because $Q(0) = 1$, so $Q'(x) = 0$ which means that $x^{q-p} = -\frac{p}{q}$ and then the roots of the polynomial $Q'(x)$ need to be equal to $x_k = \sqrt[q-p]{\frac{p}{q}}\left(\cos\frac{\pi+2k\pi}{q-p} + i\sin\frac{\pi+2k\pi}{q-p}\right)$ for $k = 0, 1, \ldots, p-q-1$.

Let's assume that x_k for some k is the root of the polynomial $Q(x)$. Then

$$Q(x_k) =$$

$$= 1 - \left(\sqrt[q-p]{\frac{p}{q}}\left(\cos\frac{\pi+2k\pi}{q-p} + i\sin\frac{\pi+2k\pi}{q-p}\right)\right)^p$$

$$- \left(\sqrt[q-p]{\frac{p}{q}}\left(\cos\frac{\pi+2k\pi}{q-p} + i\sin\frac{\pi+2k\pi}{q-p}\right)\right)^q =$$

$$= 1 - \left(\frac{p}{q}\right)^{\frac{p}{q-p}}\left(\cos\left((\pi+2k\pi)\frac{p}{q-p}\right) + i\sin\left((\pi+2k\pi)\frac{p}{q-p}\right)\right)$$

$$- \left(\frac{p}{q}\right)^{\frac{q}{q-p}}\left(\cos\left((\pi+2k\pi)\frac{q}{q-p}\right) + i\sin\left((\pi+2k\pi)\frac{q}{q-p}\right)\right) =$$

$$= 1 - \left(-\frac{p}{q}\right)^{\frac{p}{q-p}} - \left(-\frac{p}{q}\right)^{\frac{q}{q-p}} = 1 - \left(-\frac{p}{q}\right)^{\frac{p}{q-p}}\left(1 + \left(-\frac{p}{q}\right)^{\frac{q-p}{q-p}}\right) =$$

$$= 1 + \left(-\frac{p}{q}\right)^{\frac{p}{q-p}}\frac{q-p}{q}.$$

But if $Q(x) = 0$, then $\left(-\frac{p}{q}\right)^{\frac{p}{q-p}} = -\frac{q}{q-p}$. Because $p < q$, then $-\frac{q}{q-p} < -1$ and $\left|\left(-\frac{p}{q}\right)^{\frac{p}{q-p}}\right| < 1$, so the equation $\left(-\frac{p}{q}\right)^{\frac{p}{q-p}} = -\frac{q}{q-p}$ does not hold for any values of $p, q \in \mathbb{N}_+$ for which $p < q$. So, finally, there is no value x_k which is the root of the polynomial $Q(z) = 1 - z^p - z^q$. This means that all roots of the function $Q(z) = 1 - z^p - z^q$ are distinct. Let $R(z) = \frac{1}{Q(z)} = \frac{1}{\left(1-\frac{z}{z_1}\right)\left(1-\frac{z}{z_2}\right)\cdots\left(1-\frac{z}{z_q}\right)}$, where $z_1, z_2, \ldots, z_q$ are distinct roots of the polynomial $Q(z)$. The function $R(z)$ is then the generating function of the sequence (a_m), where $a_m = b_1\frac{1}{z_1^m} + b_2\frac{1}{z_2^m} + s + b_l\frac{1}{z_q^m}$ and $b_j = \frac{-1}{z_j Q'(z_j)}$ for $j = 1, \ldots, q$.

It should be noted that $Q(z) = \frac{1}{A(z)}$ and $R(z) = A(z)$. Now if z_1 is the least positive root of the polynomial $Q(z)$ and $R(z)$ is the generating function of sequence (a_m) then $a_m \sim \frac{D}{z_1^m}$ for some $D \in \mathbb{R}_+$. Because $U(z) = \frac{z^{p+1}(1-z^p)}{1-z}A(z)$ then u_m has similar asymptotic behavior as a_m and $u_m \sim \frac{E}{z_1^m}$ for some $E \in \mathbb{R}_+$. Finally, m_0 for which $u_{m_0} \approx n$ is equal to $\log_{z_1}\frac{E}{n}$.

Using the methods presented in this article, one can construct several algorithms for computing the cost of the optimal strategy. However, these methods also allow to construct algorithms to determine the structure of the tree presenting the optimal strategy, and therefore the structure of optimal strategy itself. Below are presented the algorithms constructed by us. The complexity of these algorithms is analyzed in section VI.

First, we presented the algorithm which allows one to find the structure of the tree presenting the optimal strategy.

Algorithm 2 Algorithm of searching for subtrees of the optimal strategy tree.

Input: Costs $p, q \in \mathbb{N}_+, p \le q$, size of strategy n
Output: Table A containing size of substrategies

```
j := p + q + 1;
if Cost of ℓ-isogeny is greater than the cost of point multiplication by ℓ then
    for i := 1 to n by 1 do
        if u_j ≤ i and u_{j+1} > i then
            A_i := u_{j-p};
        end
        else
            j := j + 1; A_i := u_{j-p};
        end
    end
end
else
    for i := 1 to n by 1 do
        if u_j ≤ i and u_{j+1} > i then
            A_i := i - u_{j-p};
        end
        else
            j := j + 1; A_i := i - u_{j-p};
        end
    end
end
return A;
```

In algorithm 2 it is necessary to find substrategies of the optimal strategy. Using the fact that for $n = u_m, u_m + 1, \ldots, u_{m+1}$ every optimal strategy tree consists of subtrees of size k and l, where $k \leq l, n = k + l, k = u_{m-q}, u_{m-q} + 1 \ldots, u_{m-q+1} - 1$, $l = u_{m-q}, u_{m-q} + 1, \ldots, u_{m-q+1} - 1$ we find that the optimal strategy of size n is generated by substrategies of size $n - u_{m-p}$ and u_{m-p}.

The next algorithm allows us to search for the cost of the optimal strategy using generating functions.

Algorithm 3 Algorithm of searching for the cost of optimal strategy using generating functions.

Input: $n, p, q \in \mathbb{N}_+, p \leq q$, formulas for (u_m) and (c_m), function $\overline{u^{-1}}(n)$ which is asymptotically equal to $u^{-1}(n)$. This function allows to compute such m_0, that $n \approx u_{m_0}$
Output: Cost $Cn = C_{p,q}(n)$
$m_0 := Round(\overline{u^{-1}}(n))$;
if $u_{m_0} \leq n$ **then**
  while $u_{m_0} \leq n$ **do**
    $m_0 := m_0 + 1$;
  end
  $m_0 := m_0 - 1$;
end
else
  while $u_{m_0} > n$ **do**
    $m_0 := m_0 - 1$;
  end
end
$Cn := C(m_0) + m_0(n - u_{m_0})$;
return Cn;

In algorithm 3 if $u_{m_0} > n$, then m_0 should be decreased until $u_{m_0} \leq n$. If $u_{m_0} \leq n$, then it should be checked if $u_{m_0+1} > n$. If yes, then m_0 is the value which is being searched for. If not, then m_0 should be increased until $u_{m_0} \leq n$ and $u_{m_0+1} > n$.

When the value of m_0 is found, one can find the cost $C_{p,q}(n)$ of using the formula for c_{m_0}, because $C_{p,q}(u_{m_0}) = f(u_{m_0})$. So $f(n) = f(u_{m_0}) + m_0(n - u_{m_0})$.

In algorithm 3 the function $\overline{u^{-1}}(x)$ is used, which is asymptotically equal to $u^{-1}(x)$. In general, in the while loop of the algorithm (instead of cases where n is very small) an m_0 will be found such that $m_0 = \lfloor \overline{u^{-1}}(n) \rceil$, where $\lfloor x \rceil$ is a rounding of the number x to the nearest integer. This means that in general it will not be necessary to make any steps of while loop, or only one such step. For very small values of n it is likely that one will make more steps of while loop.

Algorithm 4, allows us to search for the cost of the optimal strategy using a recurrence equation for the cost of optimal strategy, but does not require a solution to the sequence (a_m) nor the generating function of this sequence.

Algorithm 4 Algorithm for searching for the cost of optimal strategy using a recurrence equation for the cost of optimal strategy.

Input: $n, p, q \in \mathbb{N}_+, p \leq q$
Output: Cost $C_n = C_{p,q}(n)$
for $i := p + 1$ *to* $p + q$ *by* 1 **do**
  $u_i := 1$;
end
while $u_i < n$ **do**
  $i := i + 1$; $u_i := u_{i-p} + u_{i-q}$;
end
$s := i - 1$;
for $i := p + 1$ *to* $p + q$ *by* 1 **do**
  $c_i := 0$;
end
for $i := p + q + 1$ *to* s *by* 1 **do**
  $c_i := c_{i-p} + c_{i-q} + pu_{i-p} + qu_{i-q}$;
end
$c_n := c_s + s(n - u_s)$;
return C_n;

Algorithm 4 only uses a recurrence representation of the sequences (u_m) and (c_m) and returns the cost of the optimal strategy.

VI. Analysis of the complexity of the presented algorithms

The algorithm using the dynamic programming approach presented in [1] and implemented in [3] has computational complexity equal to $O(n^2)$, if the complexity of multiplication is not considered. Using the asymptotic formula from [1] for cost of optimal strategy equal to $C_{p,q}(n) \sim -\frac{1}{\log z} n \log n$ and if the integers which are multiplied have size at most $-\frac{1}{\log z} n \log n$ then the cost of the algorithm may be estimated as $O\left(n^2 M\left(n \log n\right)\right)$, where $M(N)$ is the function returning the cost of multiplication of integers of size N. Additionally, this algorithm has memory complexity equal to $O\left(n \log\left(n \log n\right)\right)$.

Algorithm 2, used for finding substrategies of the optimal strategy, has time complexity equal to $O(n)$ and memory complexity equal to $O(n \log n)$. If not all values need to be stored in arrays, then this algorithm may be modified, if one notes that for the set $\{u_m, u_m + 1, \ldots u_{m+1}\}$ substrategies of optimal strategy may be found if one knows the value u_{m-p}. In this case, the time complexity of the algorithm of finding subtrees of the optimal strategy may be reduced to $O(m_0)$, where $m_0 \approx u^{-1}(n)$. The memory complexity of such algorithm is equal to $O(m_0)$. Unfortunately, every access to the size of substrategy for some $k < n$ will require searching for the proper value in the given sorted array which costs $\log m_0$ operations.

Algorithm 3, which uses generating functions, has time complexity equal to $O\left(m_0 M\left(n \log n\right)\right)$. This complexity is the result of powering of real numbers. Additionally, $u_{m_0} \approx n$ and $m_0 << n$. There are some exceptional cases, where m_0 is very small (this depends of the values of p and q). Memory complexity depends on the final result and is equal to $O\left(\log\left(n \log n\right)\right)$. The algorithm is numerically unstable

and requires precomputations, where one solves recurrence equations. For smaller values of p and q (lower than 20) and n less than 100000, this algorithm is numerically stable and may be used.

Algorithm 4, which uses recurrence equations, has the same time complexity $O\left(m_0 M\left(n\log n\right)\right)$, as algorithm 3. The memory complexity of this algorithm is equal to $O\left(m_0\log\left(n\log n\right)\right)$ but it is possible to reduce this complexity to $O\left(\log\left(n\log n\right)\right)$, if one remembers only the last q numbers. Therefore this algorithm seems to be the most suitable for practical implementations. Algorithm 4 does not require many precomputations and is not as sensitive for large costs of p and q as algorithm 3. Additionally, in algorithm 4 computations may be made using only integers and therefore this algorithm is numerically stable.

VII. Conclusion

In this article we have presented algorithms which allow us to find the optimal strategy for computation of isogeny of degree ℓ^n for costs of computation of ℓ-point multiplication and evaluation of ℓ-isogeny equal to p and q, where $p, q \in \mathbb{N}_+$. However, the costs of point ℓ-multiplication and evaluation of ℓ-isogeny do not need to be integers. To get equivalent costs, the best for computations, for given rational costs (for example time of execution instead of number of operations) $\frac{a}{b}$ of point ℓ-multiplication and $\frac{c}{d}$ for computation of ℓ-isogeny, where $a, b, c, d \in \mathbb{N}_+$, additional transformations can be made. The ratio of these numbers is equal to $\frac{ad}{bc}$. Then p and q may be natural numbers such that $p = min\{\frac{ad}{NWD\{ad,bc\}}, \frac{bc}{NWD\{ad,bc\}}\}$ and $q = max\{\frac{ad}{NWD\{ad,bc\}}, \frac{bc}{NWD\{ad,bc\}}\}$.

We have also shown how to solve recurrence equations for the cost of optimal strategy of size n. These methods allow us to construct computationally efficient algorithms for computing the cost of the optimal strategy, similarly to the structure of the optimal strategy. Presented below algorithms have improved time and memory complexity than the algorithms currently being used:

- determination of optimal strategy for computation of degree ℓ^n isogeny whose time complexity is equal to $O\left(n\right)$ and memory complexity is equal to $O\left(n\log n\right)$,
- determination of cost of optimal strategy of computation ℓ^n isogeny using solutions of recurrence equations, whose time complexity is equal to $O\left(m_0 M\left(n\log\left(n\right)\right)\right)$ and memory complexity is equal to $O\left(\log\left(n\log n\right)\right)$.
- determination of cost of optimal strategy of computation ℓ^n isogeny using recurrence equations whose time complexity is equal to $O\left(m_0 M\left(n\log\left(n\right)\right)\right)$ and memory complexity equal to $O\left(m_0\log\left(n\log n\right)\right)$ or $O\left(\log\left(n\log n\right)\right)$, if one remembers only the last q numbers.

Time of execution of the algorithm using dynamic programming approach [3] implemented in MAGMA, for $n = 100,000$ and costs of ℓ-point multiplication equal to $p = 3$ and cost of ℓ-isogeny evaluation equal to $q = 7$ returned a solution in $5473.694s$. By contrast, the implementation in MAGMA of joint algorithms 2 and 4, returned the optimal strategy in only $0.641s$.

References

[1] L. D. Feo, D. Jao, and J. Plût, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," Cryptology ePrint Archive, Report 2011/506, 2011, https://eprint.iacr.org/2011/506.

[2] C. Costello and H. Hisil, "A simple and compact algorithm for sidh with arbitrary degree isogenies," in *Advances in Cryptology – ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds. Cham: Springer International Publishing, 2017, pp. 303–329.

[3] C. Costello, P. Longa, and M. Naehrig, "Efficient algorithms for supersingular isogeny diffie-hellman," in *Advances in Cryptology – CRYPTO 2016*, M. Robshaw and J. Katz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 572–601.