This is Chapter 21 by Roberto M. Avanzi and Nicolas Thériault of the Handbook of Elliptic and Hyperelliptic Curve Cryptography, Henri Cohen, Christophe Doche, and Gerhard Frey, Editors, CRC Press 2006.

Chapter 21

Index Calculus for Hyperelliptic Curves

Roberto M. Avanzi and Nicolas Thériault

Contents in Brief

21.1 General algorithm

As stated in Section 20.2.1, the index calculus algorithm can be applied to compute discrete logarithms in the Jacobian of hyperelliptic curves. For groups of this type, the set of primes are *prime divisors* or, in terms of the ideal class group, *prime ideals*. These divisors can be single $\mathbb{F}_q$-rational points or the sums of all Galois conjugates over $\mathbb{F}_q$ of a non-$\mathbb{F}_q$-rational point: In other words, if a divisor D has Mumford representation $[u(x), v(x)]$, D is prime if and only if the polynomial $u(x)$ is irreducible over $\mathbb{F}_q$.

Algorithm 21.1 Divisor decomposition

INPUT: A semi-reduced divisor $D = [u(x), v(x)]$.
OUTPUT: Prime divisors $P_1, \ldots, P_k$ and coefficients $e_1, \ldots, e_k$ such that $D = \sum_{j=1}^{k} e_j P_j$.

1. find the factorization $\prod_{i=1}^{k} u_i(x)^{e_i}$ of $u(x)$
2. $i \leftarrow 1$
3. **while** $i \leqslant k$ **do**
4. $\quad v_i \leftarrow v(x) \bmod u_i(x)$
5. $\quad P_i \leftarrow [u_i(x), v_i(x)]$

6. $i \leftarrow i + 1$
7. **return** $\sum_{j=1}^{k} e_j P_j$

As is common for index calculus algorithms, the prime decomposition does not require a divisor to be reduced and two divisors in the same class will have two different decompositions.

Remark 21.2 For the remainder of this chapter, once the prime decomposition of a divisor has been computed it will be assumed that both the Mumford representation and the prime decomposition can be used when required and the two representations will not be explicitly distinguished.

21.1.1 Hyperelliptic involution

As stated in Section 20.3.5, using group automorphisms can have a significant impact on the speed of the index calculus algorithm, although this impact is in the size of the constants involved and not in the asymptotic form of the resulting algorithm. In the case of hyperelliptic curves, the automorphism used is the hyperelliptic involution.

Definition 21.3 Let C be a hyperelliptic curve given by the equation $y^2 + h(x)y = f(x)$. The *hyperelliptic involution* of a divisor $D = [u(x), v(x)]$ is the divisor $[u(x), \tilde{v}(x)]$, denoted $-D$, where $\tilde{v}(x) := -v(x) - h(x) \pmod{u(x)}$.

Since the hyperelliptic involution is an automorphism of order two, it can be used to make the relation search twice as fast and the linear algebra four times faster (more if the sparse linear algebra algorithms of Section 20.3.3 cannot be used).

In order to take advantage of the hyperelliptic involution, we act as if for every prime $P \in \mathcal{P}_B$, its image under the involution were also in $\mathcal{P}_B$, but at most one copy of P and $-P$ is actually included in $\mathcal{P}_B$. The prime decomposition is rewritten to reflect that fact, i.e., if P is in the factor base and the prime decomposition of a divisor D would call for the prime $(-P)$, then we replace $e(-P)$ by $(-e)P$.

21.1.2 Adleman–DeMarrais–Huang

With the exception of some special cases, the first general description of an index calculus algorithm for Jacobians of hyperelliptic curves is due to Adleman, DeMarrais, and Huang [ADDE$^+$ 1999].

The algorithm can be used for curves over any finite field, but describing in full generality the various sub-algorithms would only make things unnecessarily confusing, so only the case curves defined over fields of odd characteristic and defined by equations of the form $y^2 = f(x)$ will be described in this section.

Factor base

Just as the factor base is chosen such that it contains primes with a small norm when we are dealing with the index calculus algorithm for finite fields, we attach a notion of "size" to prime divisors. For a given smoothness bound B, the factor base will then be composed of all the prime divisors of degree at most B, where the degree of a prime divisor is the degree of its polynomial $u(x)$ in the Mumford representation $[u(x), v(x)]$.

Definition 21.4 A divisor is said to be B-*smooth* if all the prime divisors in its decomposition have degree at most B.

To find all the prime divisors of a given degree, it suffices to test all the irreducible polynomials $u(x)$ of that degree, checking to see if there exists a polynomial $v(x)$ satisfying $v(x)^2 \equiv f(x) \pmod{u(x)}$. If we also take advantage of the hyperelliptic involution, we get the following method to find all the elements of the factor base:

Algorithm 21.5 Computing the factor base

INPUT: A hyperelliptic curve C and a smoothness bound B
OUTPUT: A set $\mathcal{P}_B$ of B-smooth prime divisors

1. $\mathcal{P}_B \leftarrow \{\}$
2. **for** every irreducible monic polynomial $u(x) \in \mathbb{F}_q[x]$, $\deg u \leqslant B$ **do**
3. **if** $u(x) \nmid f(x)$ and $f(x) \equiv$ square $\pmod{u(x)}$ **then**
4. find $v(x)$ such that $\deg v < \deg u$ and $v(x)^2 - f(x) \equiv 0 \pmod{u(x)}$
5. $P \leftarrow [u(x), v(x)]$
6. add P to $\mathcal{P}_B$
7. **return** $\mathcal{P}_B$

Relation search

Given a reduced divisor, the approach used by Adleman, DeMarrais, and Huang to obtain a smooth divisor consists of adding randomly chosen principal divisors (of degree at most d) to the reduced divisor until the semi-reduced divisor created is smooth. Because this can be applied separately to each of the prime divisors in the decomposition of a divisor, we consider only the case of unramified prime divisors and the divisor 0.

Algorithm 21.6 Smoothing a prime divisor

INPUT: An unramified prime divisor $D = [u(x), v(x)]$, a factor base $\mathcal{P}_B$, and a parameter d.
OUTPUT: The prime decomposition of an equivalent B-smooth divisor $\widetilde{D} = [\tilde{u}(x), \tilde{v}(x)]$.

1. **if** D is B-smooth **then**
2. **return** D
3. **else**
4. **repeat**
5. **repeat**
6. choose a random $A(x) \in \mathbb{F}_q[x]$ of degree $\leqslant d$
7. $B(x) \leftarrow -A(x)v(x) \bmod u(x)$
8. **until** $\gcd\big(A(x), B(x)\big) = 1$
9. $\tilde{u}(x) \leftarrow \big(B(x)^2 - A(x)^2 f(x)\big)/u(x)$
10. $\tilde{v}(x) \leftarrow B(x)A(x)^{-1} \bmod \tilde{u}(x)$
11. $\widetilde{D} \leftarrow [\tilde{u}(x), \tilde{v}(x)]$
12. $\widetilde{D} \leftarrow \sum_{i=1}^{r} e_i P_i$ $\quad\Big[$decomposition of $\widetilde{D}$ into prime divisors$\Big]$

13. **until** $\widetilde{D}$ is B-smooth
14. **return** $\widetilde{D}$

Algorithm 21.7 Finding smooth principal divisors

INPUT: A factor base $\mathcal{P}_B$ and a parameter d.
OUTPUT: The prime decomposition of a B-smooth principal divisor $\widetilde{D} = [\tilde{u}(x), \tilde{v}(x)]$.

1. **repeat**
2. **repeat**
3. choose random $A(x), B(x) \in \mathbb{F}_q[x]$ of degree $\leqslant d$
4. **until** $\gcd\big(A(x), B(x)\big) = 1$
5. $\tilde{u}(x) \leftarrow B(x)^2 - A(x)^2 f(x)$
6. $\tilde{v}(x) \leftarrow B(x)A(x)^{-1} \bmod \tilde{u}(x)$
7. $\widetilde{D} \leftarrow [\tilde{u}(x), \tilde{v}(x)]$
8. $\widetilde{D} \leftarrow \sum_{i=1}^{r} e_i P_i$ $\Big[$decomposition of $\widetilde{D}$ into prime divisors$\Big]$
9. **until** $\widetilde{D}$ is B-smooth
10. **return** $\widetilde{D}$

Since the smoothing algorithm does not work for ramified primes (i.e., primes of the form $[u(x), 0]$), it may be necessary to find an equivalent divisor that does not contain any ramified prime in its decomposition. This is done as follows:

Algorithm 21.8 Decomposition into unramified primes

INPUT: A semi-reduced divisor $D = [u(x), v(x)]$.
OUTPUT: The decomposition of an equivalent divisor $\widetilde{D}$ into unramified primes.

1. $d(x) \leftarrow \gcd\big(u(x), f(x)\big)$
2. **if** $d(x) = 1$ **then**
3. find the decomposition of $[u(x), v(x)]$ into prime divisors $\sum_{i=1}^{r} e_i P_i$
4. **return** $\widetilde{D} = \sum_{i=1}^{r} e_i P_i$
5. **else**
6. $u_1(x) \leftarrow u(x)/d(x)$
7. $v_1(x) \leftarrow v(x) \bmod u_1(x)$
8. $u_2(x) \leftarrow \big(d(x)^2 - f(x)\big)/d(x)$
9. $v_2(x) \leftarrow d(x) \bmod u_2(x)$
10. find the decomposition of $[u_1(x), v_1(x)]$ into prime divisors $\sum_{i=1}^{r} e_i P_i$
11. find the decomposition of $[u_2(x), v_2(x)]$ into prime divisors $\sum_{j=1}^{s} f_j Q_j$
12. **return** $\widetilde{D} = \sum_{i=1}^{r} e_i P_i - \sum_{j=1}^{s} f_j Q_j$

Algorithm

The index calculus algorithm presented by Adleman, DeMarrais, and Huang [ADDE$^+$ 1999] differs somewhat from Algorithm 20.3. The idea behind the algorithm is to consider the kernel of the map between the free abelian group $\mathbb{Z}^n$ generated by the factor base and the Jacobian of the curve. Since this kernel is a lattice of $\mathbb{Z}^n$, working out the structure of the lattice makes it relatively easy to find the relationship between two smooth divisors in the Jacobian.

We first find smooth representations of the pair of divisors for which we want to compute the discrete logarithm. To obtain the structure of the lattice, we must find enough smooth principal divisors (which correspond to points in the lattice) and decompose the linear system obtained from these relations. It may not be necessary to completely work out the structure of the lattice, and the search for smooth principal divisors is ended once the discrete logarithm is found.

The resulting algorithm can be written as follows:

Algorithm 21.9 Adleman–DeMarrais–Huang

INPUT: A hyperelliptic curve C, two divisors g and h with $h \in \langle g \rangle$, a smoothness bound B, and a parameter d.
OUTPUT: An integer t with $[t]g = h$.

1. compute the factor base $\mathcal{P}_B$ [use Algorithm 21.5]
2. $n \leftarrow |\mathcal{P}_B|$
3. $g \leftarrow \sum_{i=1}^{r} e_i P_i$ [decompose g into unramified primes]
4. $i \leftarrow 1$
5. **while** $i \leqslant r$ **do**
6. smooth the prime divisor P_i and put the result in $\widetilde{P}_i$ [use Algorithm 21.6]
7. $i \leftarrow i + 1$
8. $\tilde{g} \leftarrow \sum_{i=1}^{r} e_i \widetilde{P}_i$
9. write $\tilde{g}$ as a $1 \times n$ vector denoted by v_g
10. $h \leftarrow \sum_{j=1}^{s} f_j Q_j$ [decompose h into unramified primes]
11. $j \leftarrow 1$
12. **while** $j \leqslant s$ **do**
13. smooth the prime divisor Q_j and put the result in $\widetilde{Q}_j$ [use Algorithm 21.6]
14. $j \leftarrow j + 1$
15. $\tilde{h} \leftarrow \sum_{j=1}^{s} f_j \tilde{Q}_f$
16. write $\tilde{h}$ as a $1 \times n$ vector denoted by v_h
17. $M \leftarrow 0 \times n$ matrix
18. $j \leftarrow 0$
19. **repeat**
20. $j \leftarrow j + 1$
21. find a smooth principal divisor D_j [use Algorithm 21.7]
22. write D_j as a $1 \times n$ vector denoted by v_j
23. add v_j to M as the j-th row

24. find matrices $L \in GL(j,\mathbb{Z})$ and $R \in GL(n,\mathbb{Z})$ such that $LMR = C$ where C is a $m \times n$ matrix with all nonzero entries on the diagonal and with diagonal entries satisfying $c_i \mid c_{i+1}$ and $c_i > 0$
25. $c \leftarrow \prod_{i=1}^{j} c_i$
26. **if** $c > (\sqrt{q}+1)^{2g}$ **then** [upper bound of the group order]
27. $\tilde{v}_g \leftarrow v_g R$
28. $\tilde{v}_h \leftarrow v_h R$
29. **if** $\tilde{v}_h = t\tilde{v}_g$ for some $t \in \mathbb{Z}$ **then**
30. **return** t
31. **until** t is found

By setting $B = \lfloor \log_q L_{q^{2g+1}}(1/2, 1/\sqrt{2k}) \rfloor$ and $d = \lfloor \log_q L_{q^{2g+1}}(1/2, \sqrt{2k}) \rfloor$ where the constant k depends on the speed of the linear algebra, we get:

Theorem 21.10 [ADDE$^+$ 1999] Let C be a hyperelliptic curve of genus g over the finite field $\mathbb{F}_q$. If $\ln q \leqslant (2g+1)^{1-\epsilon}$, then there exists a constant $c \leqslant 2.181$ such that discrete logarithms in $J_C(\mathbb{F}_q)$ can be computed in expected time $L_{q^{2g+1}}(1/2, c)$.

The constant c is in fact dependent on the value of k coming from the linear algebra, with

$$c = \frac{k+1}{\sqrt{2k}}.$$

The value $c = 2.181$ is obtained when no sparse linear algebra arithmetic is assumed (with $k = 7.376$). Although the algorithms described in Section 20.3.3 cannot be applied in this context, sparse arithmetic might still be used, giving a value of $c = 4/\sqrt{6}$ (assuming $k = 3$ is possible).

21.1.3 Enge–Gaudry

The algorithm can be adapted to fit more closely with the index calculus as described in Chapter 20 (and used in the remainder of this chapter). Using results by Enge, Gaudry, and Stein [ENG 2002, ENGA 2002, ENST 2002] we get:

Theorem 21.11 [ENGA 2002, Theorem 1] Let C be a hyperelliptic curve of genus g defined over $\mathbb{F}_q$ and let $\log_g$ denote the logarithm in base g. For $g/\log_g q > t$, the discrete logarithm in the divisor class group of C can be computed with complexity bounded by

$$L_{q^g}\left(\frac{1}{2}, \sqrt{2}\left(\left(1+\frac{1}{2t}\right)^{1/2} + \left(\frac{1}{2t}\right)^{1/2}\right)\right).$$

21.2 Curves of small genus

Although Algorithm 21.9 is very efficient when $g > \log_g q$, it must be modified in order to be applied for hyperelliptic curves of "small" genus, i.e., such that $g < \log_g q$.

In the remainder of this chapter, we will assume that $q > g!$ (which implies $g < \log_g q$) and running times can be viewed in terms of a fixed genus and a varying field size.

21.2.1 Gaudry's algorithm

There are three major differences with the Algorithm 21.10 when adapting the index calculus algorithm to curves of small genus:

- The approach of Algorithm 20.3 is used.
- The only primes considered for the factor base are those coming from $\mathbb{F}_q$-rational points, i.e., we are looking for 1-smooth divisors.
- Rather than smoothing divisors, smooth relations are found by repeatedly looking at different reduced divisors until a smooth one is found.

Using these ideas, Gaudry [GAU 2000b] obtained an algorithm that is more efficient when the genus of the hyperelliptic curve is small. The second step of Algorithm 20.3 is done as follows:

Algorithm 21.12 Relation search

INPUT: A hyperelliptic curve C, two divisors g and h with $h \in \langle g \rangle$, and a factor base $\mathcal{P}_1$.
OUTPUT: A system of r 1-smooth divisors of the form $[\alpha_i]g \oplus [\beta_i]h$.

1. choose random $\alpha, \beta \in \{1, \ldots, |J_C(\mathbb{F}_q)|\}$
2. $D \leftarrow [\alpha]g \oplus [\beta]h$
3. $i \leftarrow 1$
4. **while** $i \leqslant r$ **do**
5. $\quad D \leftarrow \Phi(D)$
6. $\quad$ update α and β
7. $\quad$ decompose D into prime divisors
8. $\quad$ **if** D is 1-smooth **then**
9. $\quad\quad D_i \leftarrow D$
10. $\quad\quad i \leftarrow i + 1$
11. **return** $\{D_1, \ldots, D_r\}$

Remark 21.13 To take into account the various linear algebra methods of Section 20.3.3, the desired number of smooth relations produced by Algorithm 21.12 (and by the remaining algorithms in this chapter) is denoted by r. In most cases, r can be assumed to be close to the size of the factor base: If Wiedemann's method is used, we need $r = |\mathcal{B}| + 1$, while if Lanczos' method is preferred, r may be chosen somewhat larger (to get a system with higher rank).

Since approximately 1 in $g!$ divisors in the Jacobian is 1-smooth, obtaining enough relations is quite fast and takes time $O\left(g^2 g!\, q^{1+\epsilon}\right)$. The other steps of Algorithm 20.3 are dominated by the linear algebra, which has running time $O\left(g^3 q^{2+\epsilon}\right)$. These running times combine to give the result:

Theorem 21.14 [GAU 2000b] Let C be a hyperelliptic curve of genus g over the finite field $\mathbb{F}_q$. If $q > g!$ then discrete logarithms in $J_C(\mathbb{F}_q)$ can be computed in expected time $O(g^3 q^{2+\epsilon})$.

21.2.2 Refined factor base

Because the running time for Gaudry's algorithm is dominated by the cost of solving the linear algebra, a natural approach to improving the speed of the algorithm is to reduce the cost of the

linear algebra part. Unless new sparse linear algebra algorithms are developed, the only option is to reduce the size of the system, which means reducing the size of the factor base (since we need at least as many relations as factor base elements).

To do this, we choose the factor base $\mathcal{B}$ as a subset of $\mathcal{P}_1$. Since reducing the size of the factor base makes it less likely that a random reduced divisor is smooth (or *$\mathcal{B}$-smooth*), this increases the cost of the relation search. If the factor base is reduced too much, the relation search will eventually become the dominant part of the algorithm as smooth relations become rarer. This brings us back to the usual situation for index calculus algorithms, where optimizing the running time is a balancing act between the relation search and the linear algebra.

By choosing the factor base $\mathcal{B}$ such that $|\mathcal{B}| = O(g^2 q^{g/(g+1)+\epsilon})$, we get the following result:

Theorem 21.15 [THÉ 2003a, Theorem 2] Let C be a hyperelliptic curve of genus g over the finite field $\mathbb{F}_q$. If $q > g!$ then discrete logarithms in $J_C(\mathbb{F}_q)$ can be computed in expected time $O(g^5 q^{2-\frac{2}{g+1}+\epsilon})$.

21.2.3 Harvesting

For the index calculus of hyperelliptic curves, Avanzi and Thériault [AVTH 2004] introduce a "drastic" extension of pruning they call harvesting. The idea behind harvesting is to start with a very overdetermined system, in fact of k times as many relations as variables for k relatively large (even $k \geqslant 100$ can be useful), and then remove as many relations as possible while at the same time reducing as much as possible the numbers of variables present in the system, in order to find a small subsystem that is still overdetermined.

How are we going to choose the equations that can be removed to obtain the smallest possible system after the filtering? We remove all relations that contain "rare" variables, i.e., the variables that appear less frequently with nonzero coefficients in the equations of the system. If this process is repeated until the system has only a few more relations than variables, we can reasonably expect that the result is almost as small as possible.

At the end, we can obtain a system that is still overdetermined, but with much less variables than the elements in the original factor base. Of course there might exist a smaller subsystem of the original system, but that subsystem cannot be found efficiently, hence the construction by an approximation method.

Such a harvesting step, just like pruning, has the nice side effect of decreasing the multiplicities of all other variables that appeared in the removed relations. This means that one can see if the new system can be harvested again, until it reaches the desired size or it can no longer be harvested.

After harvesting has been performed and the desired level of overdetermination has been reached, we can still perform merging. In fact, merging after harvesting will be, in comparison, *more efficient* than if it had been performed before.

Harvesting can also be viewed as a redefinition (reduction) of the factor base *a posteriori* (i.e., after the relation search has been done). Since harvesting requires more smooth relations but produces a smaller linear system, which leaves the relation search and linear algebra unbalanced (compared to an optimized algorithm with $k \sim 1$), it is only natural to readjust the original size of the factor base to re-balance the algorithm. If, for a given k, the reduction factor of harvesting is large enough, the running time for the new optimized algorithm will be smaller than the running time with $k \sim 1$.

With the approach of Section 21.2.2, harvesting can be used to reduce the running time of the index calculus algorithm by a non-negligible factor (although only the constant term in the asymptotic running time is affected). In Jacobians of genus 6 curves for example, a total saving of close to 45% can be obtained using $k = 100$. Harvesting can also be applied to the algorithm in Section 21.3.1 and Section 21.2.1 (although there is no re-balancing of the choice of the factor base in that case),

but it is not yet known how well it can be adapted to the other algorithms of this chapter and of Chapter 20.

Parallelization

Although it is not clear how to perform pruning and merging in parallel with a non shared-memory computer, such as a cluster, some distribution is possible (and relatively easy) for harvesting.

We proceed by repeated steps as follows: Each node keeps track of its own part of the system and lets the server know how many times each variable is used. The server can then decide which variables will be removed from the system and tells the nodes. The nodes then look for the relations containing the variables removed (and remove those equations from the system) and update the number of times variables are used.

In order to reduce latency, variables are not removed one by one but in groups (still with the "rarest" variables first). Since removing "blocks" of variables together is likely to have a detrimental impact on the effectiveness of harvesting, one must be careful and choose these blocks as small as possible without getting too much latency.

Once the system is reduced to the point where harvesting is no longer possible or distributing the work is no longer advantageous, the remaining relations are collected and the filtering is completed on a single processor. If this is implemented carefully, the total amount of information transmitted is not much more than what would be transmitted to collect the full system (certainly much less than twice that amount) while most of the work is distributed among the nodes. A major advantage of this approach is the distribution of the system pre-harvesting, potentially making it possible to work with systems that would be too big to be handled by a single processor.

If the expansion factor k is *very* large (and larger than the number of client computers involved in the filtering), then some harvesting may be done locally by each node (and independently of other nodes), reducing the number of relations but not the number of variables (since a variable removed on one node can still be used on another). The harvesting can then be done distributively on the reduced system or the relations can be collected together and the filtering (including harvesting) can be done on a single computer. This will reduce network traffic (as no information is communicated on the relations that were removed locally) but will also (presumably) reduce the effectiveness of the filtering.

21.3 Large prime methods

As was described in Section 20.5, using large primes to decrease the time required to find relations can have a significant impact on the running time of the index calculus algorithm.

The main difference when using large primes for Jacobians of hyperelliptic curves is in their definition. Whereas large primes are usually defined as all the primes not included in the factor base, in this situation we only consider all the primes in $\mathcal{P}_1$ not included in the factor base (in other words, we ignore large primes of degree greater than one). This restriction is not strictly speaking a necessary one, but more one of convenience.

In essence, this is because a large prime must appear in at least two different divisors found during the relation search before it can be of any use in building smooth relations. Although almost-smooth divisors with a large prime of degree at least 2 are more common than almost-smooth divisors with a linear large prime, the low probability that a specific prime of higher degree appears more than once far offsets the large number of these primes.

Even if all primes outside the factor base were to be considered, the number of smooth relations obtained due to the use of the primes of degree larger than one would be almost insignificant compared to what is obtained using only $\mathcal{P}_1$. In practice, restricting the definition of large primes

does increase the running time of the relation search, but not in any meaningful way. The main reason to restrict the definition of large prime is to reduce the amount of memory required to run the algorithm, the set $\mathcal{P}_1$ being much more manageable than the set of all primes.

At the time this book was written, methods using relations with one or two large primes had been described, but no effective way had yet been found to use relations with more than two large primes. The algorithms described here will therefore be limited to single and double large prime variants.

21.3.1 Single large prime

In order to take advantage of the high number of almost-smooth divisors, we must find pairs of these divisors with the same large prime (up to sign, using the hyperelliptic involution). The approach relies on the birthday paradox, since the large primes in almost-smooth divisor are uniformly distributed among the set of linear large primes. In order to identify these pairs, we use a list $\mathcal{P}$ of the large primes encountered during the search and a list $\mathcal{R}$ of the almost-smooth divisors in which they appeared. To avoid producing redundant relations, only the first copy of a large prime is included in the list, and subsequent copies are used to produce smooth relations (using the corresponding divisors to cancel the large primes).

Algorithm 21.16 Single large prime

INPUT: A hyperelliptic curve C, two divisors g and h with $h \in \langle g \rangle$, and a factor base $\mathcal{B}$.
OUTPUT: A system of r $\mathcal{B}$-smooth divisors of the form $D_i = [\alpha_i]g \oplus [\beta_i]h$.

1. $\mathcal{P} \leftarrow \{\}$ and $\mathcal{R} \leftarrow \{\}$
2. choose random $\alpha, \beta \in \{1, \ldots, |J_C(\mathbb{F}_q)|\}$
3. $D \leftarrow [\alpha]g \oplus [\beta]h$
4. $i \leftarrow 1$
5. **while** $i \leqslant r$ **do**
6. $\quad D \leftarrow \Phi(D)$, update α and β
7. $\quad$ decompose D into prime divisors
8. $\quad$ **if** D is $\mathcal{B}$-smooth **then**
9. $\quad\quad D_i \leftarrow D$
10. $\quad\quad i \leftarrow i + 1$
11. $\quad$ **if** D is almost-smooth with large prime P **then**
12. $\quad\quad$ **if** $P = P_j \in \mathcal{P}$ **then**
13. $\quad\quad\quad R \leftarrow$ almost-smooth divisor containing P_j
14. $\quad\quad\quad D_i \leftarrow D \ominus R$
15. $\quad\quad\quad i \leftarrow i + 1$
16. $\quad\quad$ **else if** $P = -P_j \in \mathcal{P}$ **then**
17. $\quad\quad\quad R \leftarrow$ almost-smooth divisor containing P_j
18. $\quad\quad\quad D_i \leftarrow D \oplus R$
19. $\quad\quad\quad i \leftarrow i + 1$
20. $\quad\quad$ **else**
21. $\quad\quad\quad$ add P to $\mathcal{P}$

22. add D to $\mathcal{R}$ [almost-smooth divisor containing P]
23. **return** $\{D_1, \ldots, D_r\}$

By choosing the factor base $\mathcal{B}$ such that $|\mathcal{B}| = O(g^2 q^{(g-\frac{1}{2})/(g+\frac{1}{2})+\epsilon})$, we get the following result:

Theorem 21.17 [THÉ 2003a, Theorem 3] Let C is a hyperelliptic curve of genus g over the finite field $\mathbb{F}_q$. If $q > g!$ then discrete logarithms in $J_C(\mathbb{F}_q)$ can be computed in expected time $O(g^5 q^{2-\frac{4}{2g+1}+\epsilon})$.

In practice, smooth relations coming from smooth divisors are used, but they are not taken into account to obtain this result. Since only a small number of smooth divisors are found during the relation search, they do not have a significant impact on the time required for the relation search.

21.3.2 Double large primes

Since almost-smooth divisors can be used to produce relations so much faster, it is natural to also consider 2-almost-smooth divisors. The idea is to produce chains of 2-almost-smooth (and almost-smooth) divisors where two consecutive divisors have a common large prime. To each such chain, we associate a relation where the divisors of the chain are added to the first one in such a way that the common large primes between consecutive divisors are canceled.

The chain produces a smooth relation if all large primes are canceled. This can happen in two ways: The first and last divisors in the chain are almost-smooth, or the first and last divisors in the chain are 2-almost-smooth with a common large prime and the cancellation of the other large primes in the chain also cancels that one. It is important to notice that even if the divisors at the extremities of the chain are 2-almost-smooth with a common large prime, that large prime need not be canceled in the relation associated to that chain. On average, the large prime common to the first and last divisor will only cancel out in half of these chains.

In order to find chains of divisors producing smooth relations, we use a graph G where the vertices are the large primes and an edge between two vertices correspond to a 2-almost-smooth divisor containing those two large primes. Since we also want to take advantage of almost-smooth divisors, the graph G contains an extra vertex denoted 1 and an almost-smooth divisor is associated to the edge between its large prime and 1. The search for chains of divisors producing smooth relations can then be viewed as a search for cycles in the graph G corresponding to the random walk.

The general algorithm for the relation search using divisors with up to two large primes can be given as follows:

Algorithm 21.18 Double large primes

INPUT: A hyperelliptic curve C, two divisors g and h with $h \in \langle g \rangle$, and a factor base $\mathcal{B}$.
OUTPUT: A system of r $\mathcal{B}$-smooth divisors of the form $D_i = [\alpha_i]g \oplus [\beta_i]h$.

1. $G \leftarrow$ empty graph
2. choose random $\alpha, \beta \in \{1, \ldots, |J_C(\mathbb{F}_q)|\}$
3. $D \leftarrow [\alpha]g \oplus [\beta]h$
4. $i \leftarrow 1$
5. **while** $i \leqslant r$ **do**
6. $\quad D \leftarrow \Phi(D)$, update α and β

```
7.        decompose D into prime divisors
8.        if D is B-smooth then
9.            D_i ← D
10.           i ← i + 1
11.       if D is almost-smooth or 2-almost-smooth then
12.           update G
13.           if a smooth divisor S is created then
14.               D_i ← S
15.               i ← i + 1
16.   return {D_1, ..., D_r}
```

Obviously, the graph must be updated in such a way that no redundant relations are created. For example, it may be possible to break down a cycle into distinct subcycles, but even if all the relations obtained from each subcycle and from the original cycle itself are smooth, it would be undesirable to use them all since they are clearly not linearly independent.

The main difference between the various double large prime algorithms is how the graph G is updated, i.e., how edges are added to the graph. In the following sections, we will describe three ways of constructing the graph and produce only good relations. We will call these the *full graph*, *simplified graph*, and *concentric circles* method.

21.3.2.a Full graph method

The first, and more natural, approach to building the graph is to use every almost-smooth and 2-almost-smooth divisor to add an edge to the graph, giving us the *full graph*.

To avoid producing redundant relations, edges that would close a cycle are used to produce relations but are not added to the graph, which means that the graph G never actually contains any cycles. Since not all cycles produce smooth relations, the non-smooth relations that may be obtained from these cycles (which are in fact almost-smooth relations) are used to produce an edge between 1 and the non-canceled large prime.

Algorithm 21.19 Full graph method

INPUT: A graph G, a set of relations $\mathcal{R} = \{R_e\}$ corresponding to the edges of G, and a new almost-smooth or 2-almost-smooth relation R.
OUTPUT: Updated G and $\mathcal{R}$ and (if possible) a smooth relation S.

```
1.  if R is almost-smooth then
2.      P ← large prime in R
3.      if P is connected to 1 then
4.          (e_1, ..., e_i) ← path from 1 to P in G
5.          use R and R_{e_1}, ..., R_{e_i} to cancel the large primes
6.          S ← smooth divisor obtained
7.          leave G and R untouched
8.      else
9.          add the edge (1, P) to G
```

```
10.              add R_(1,P) = R to R
11.  if R is 2-almost-smooth then
12.      P_1, P_2 ← large primes in R
13.      if (P_1, P_2) would create a loop in G then
14.          if the loop would contain the vertex 1 then
15.              (e_1, ..., e_i) ← path from 1 to P_1 in G
16.              (f_1, ..., f_j) ← path from 1 to P_2 in G
17.              use R and R_{e_1}, ..., R_{e_i} and R_{f_1}, ..., R_{f_j} to cancel the large primes
18.              S ← smooth divisor obtained
19.              leave G and R untouched
20.          else
21.              (e_1, ..., e_i) ← path from P_1 to P_2 in G
22.              use R and R_{e_1}, ..., R_{e_i} to cancel the large primes other than P_1
23.              if P_1 is also canceled then                          [smooth relation]
24.                  S ← smooth divisor obtained
25.                  leave G and R untouched
26.              else                                                  [almost-smooth relation]
27.                  R_(1,P_1) ← almost-smooth relation obtained
28.                  add the edge (1, P_1) to G
29.                  add R_(1,P_1) to R
30.      else
31.          add the edge (P_1, P_2) to G
32.          add R_(P_1,P_2) = R to R
33.  return G, R and if possible S
```

Remark 21.20 This method is heuristically faster than the methods in next sections but its running time is (at the time of this writing) still unproven. Although the relation search is faster with the full graph method, there is no proven bound on the weight of the system generated using this method, hence the cost of solving the linear system is unclear. If the system can be proven to be sparse enough (and its weight is not significantly greater than with the other graph methods), this is the method that should be favored.

The other methods described in the next two sections look for cycles in specific subgraphs of G instead of the entire full graph and limit themselves to the connected component of the vertex 1.

21.3.2.b Simplified graph method

In order to make the analysis more accessible, Gaudry, Thériault, and Thomé [GATH+ 2004] introduce a more restrictive approach to the graph construction. The idea is to discard edges that fall completely outside of the connected component of the graph containing the vertex 1.

Algorithm 21.21 Simplified graph method

INPUT: A graph G, a set of relations $\mathcal{R} = \{R_e\}$ corresponding to the edges of G, and a new almost-smooth or 2-almost-smooth relation R.
OUTPUT: Updated G and $\mathcal{R}$ and (if possible) a smooth relation S.

1. **if** R is almost-smooth **then**
2. $P \leftarrow$ large prime in R
3. **if** P is connected to 1 **then**
4. $(e_1, \ldots, e_i) \leftarrow$ path from 1 to P in G
5. use R and $R_{e_1}, \ldots, R_{e_i}$ to cancel the large primes
6. $S \leftarrow$ smooth divisor obtained
7. leave G and $\mathcal{R}$ untouched
8. **else**
9. add the edge $(1, P)$ to G
10. add $R_{(1,P)} = R$ to $\mathcal{R}$
11. **if** R is 2-almost-smooth **then**
12. $P_1, P_2 \leftarrow$ large primes in R
13. **if** P_1 and P_2 are both connected to 1 **then**
14. $(e_1, \ldots, e_i) \leftarrow$ path from 1 to P_1 in G
15. $(f_1, \ldots, f_j) \leftarrow$ path from 1 to P_2 in G
16. use R and $R_{e_1}, \ldots, R_{e_i}$ and $R_{f_1}, \ldots, R_{f_j}$ to cancel the large primes
17. $S \leftarrow$ smooth divisor obtained
18. leave G and $\mathcal{R}$ untouched
19. **else if** one of P_1 or P_2 is connected to 1 **then**
20. add the edge (P_1, P_2) to G
21. add $R_{(P_1,P_2)} = R$ to $\mathcal{R}$
22. **else**
23. leave G and $\mathcal{R}$ untouched
24. **return** G, $\mathcal{R}$ and if possible S

This approach has the following advantages compared to the full graph method:

- The algorithm as proven bounds on both the relation search and the linear algebra.
- All the cycles found in the graph produce a smooth relation since they correspond to chains of divisors with first and last divisors almost-smooth.

Obviously the bound on the relation search with the simplified graph methods automatically gives an upper bound on the relation search for the full graph method. However, the bound on the linear algebra may not hold true for Algorithm 21.19, which is why Algorithm 21.21 is still important.

There are some significant disadvantages, however:

- Cycles in the full graph that are not connected to 1 cannot be found.

- When looking at an edge during the search, if neither of the vertices have already been connected to 1 then the edge is not used. But it can happen that the edge would be in the connected component of 1 in the full graph (because of edges that are found later in the search or other edges that were not used). This will most likely destroy some cycles.
- The relation search is slower (because of the previous points).

By choosing the factor base $\mathcal{P}_B$ such that $|\mathcal{P}_B| = O(g^2 q^{(g-1)/g+\epsilon})$, we get the following result:

Theorem 21.22 [GATH$^+$ 2004, Theorem 2] Let C be a hyperelliptic curve of genus g over the finite field $\mathbb{F}_q$. If $q > g!$ then discrete logarithms in $J_C(\mathbb{F}_q)$ can be computed in expected time $O(g^5 q^{2-\frac{2}{g}+\epsilon})$.

Remark 21.23 The ϵ in the exponents of Theorems 21.14, 21.15, 21.17, and 21.22 are due to $\ln q$ factor in the running times. These ϵ hide the fact that the double large prime algorithm contains an extra $\ln q$ factors compared to the three other variants.

21.3.2.c Concentric circles method

The approach described in this section is a modification of an algorithm by Nagao [NAG 2004].

Like the simplified graph method, we are looking for cycles in the connected component of the full graph containing 1. Instead of building the graph bit by bit as almost-smooth and 2-almost-smooth relations are found, the random walk is completely done first and then the graph is constructed.

To make it possible to keep the weight of the smooth relations low enough (and give a bound on the linear algebra), the random walk is prolonged so that the connected component of 1 contains more cycles than necessary. The shortest relations are found by looking at the large primes in concentric circles centered at 1, i.e., in terms of their distance to the vertex 1.

In order to reduce the total cost of the index calculus algorithm as much as possible, the factor base is again chosen with size $O(g^2 q^{(g-1)/g+\epsilon})$, giving the same running time of $O(g^5 q^{2(g-1)/g+\epsilon})$ as the simplified graph method (possibly with a slightly different constant). To make notation easier, three sets are constructed during the random walk:

- the set $\mathcal{F}$ contains the smooth divisors found,
- the set $\mathcal{G}$ contains the almost-smooth divisors, and
- the set $\mathcal{H}$ contains the 2-almost-smooth divisors.

For each concentric circle, we build a list $\mathcal{L}_n$ of the large primes at distance n from the vertex 1 (and their corresponding almost-smooth relations) and a subset $\mathcal{H}_n$ of the divisors in $\mathcal{H}$ that have not yet been used to built smooth or almost-smooth relations.

Algorithm 21.24 Concentric circles method

INPUT: Sets $\mathcal{F}, \mathcal{G}, \mathcal{H}$ of respectively smooth, almost-smooth, and 2-almost-smooth divisors.
OUTPUT: A system of r smooth relations.

1. $i \leftarrow 1$
2. **while** $i \leqslant |\mathcal{F}|$ **and** $i \leqslant r$ **do**
3. $\quad D_i \leftarrow i$-th divisor in $\mathcal{F}$
4. $\mathcal{L}_1 \leftarrow \{\}$
5. $j \leftarrow 1$
6. **while** $j \leqslant |\mathcal{G}|$ **and** $i \leqslant r$ **do**

```
7.        R ← j-th divisor in 𝒢
8.        P ← large prime in R
9.        if ± P ∈ ℒ_1 then
10.           R_0 ← almost-smooth relation for ± P
11.           use R and R_1 to cancel the large prime
12.           D_i ← smooth divisor obtained
13.           i ← i + 1
14.           leave ℒ_1 untouched
15.       else
16.           almost-smooth divisor for ± P ← R
17.           add P to ℒ_1
18.       j ← j + 1
19.   ℋ_1 ← ℋ
20.   n ← 1
21.   while i ⩽ r do
22.       ℒ_{n+1} ← {}
23.       ℋ_{n+1} ← {}
24.       j ← 1
25.       while j ⩽ |ℋ_n| and i ⩽ r do
26.           R ← j-th divisor in ℋ_n
27.           P_1, P_2 ← large primes in R
28.           if ± P_1 and ± P_2 ∈ ℒ_n ∪ ℒ_{n+1} then
29.               R_1 ← almost-smooth relation for ± P_1
30.               R_2 ← almost-smooth relation for ± P_2
31.               use R, R_1 and R_2 to cancel the large primes
32.               D_i ← smooth divisor obtained
33.               i ← i + 1
34.               leave ℒ_{n+1} and ℋ_{n+1} untouched
35.           else if ± P_1 ∈ ℒ_n, ± P_2 ∉ ℒ_n ∪ ℒ_{n+1} then
36.               R_1 ← almost-smooth relation for ± P_1
37.               use R and R_1 to cancel P_1
38.               almost-smooth divisor for ± P_2 ← almost-smooth divisor obtained
39.               add P_2 to ℒ_{n+1}
40.               leave ℋ_{n+1} untouched
41.           else if ± P_2 ∈ ℒ_n, ± P_1 ∉ ℒ_n ∪ ℒ_{n+1} then
42.               R_2 ← almost-smooth relation for ± P_2
43.               use R and R_2 to cancel P_2
44.               almost-smooth divisor for ± P_1 ← almost-smooth divisor obtained
```

```
45.                 add P_1 to L_{n+1}
46.                 leave H_{n+1} untouched
47.             else
48.                 add R to H_{n+1}
49.                 leave L_{n+1} untouched
50.             j ← j + 1
51.         n ← n + 1
52.     return {D_1, ..., D_r}
```

Parallelization

Contrary to Algorithms 21.19 and 21.21, Algorithm 21.24 can be easily adapted to work distributively even on non-shared memory computers. The sets $\mathcal{F}, \mathcal{G}$, and $\mathcal{H}$ are constructed independently on each node. At the beginning of every layer of concentric circles, each node has its own $\mathcal{H}_n$ and the server sends $\mathcal{L}_n$ and the corresponding partial relations to the nodes. Each node computes its $\mathcal{H}_{n+1}$ and its part of $\mathcal{L}_{n+1}$. All the partial $\mathcal{L}_{n+1}$'s and corresponding partial relations (as well as the smooth relations produced) are collected by the server and combined, keeping at most one copy of each large prime (and using supplementary appearances to produce more smooth relations).