ON THE COMPLEXITY AND EFFICIENCY OF A NEW KEY EXCHANGE SYSTEM

Johannes A. Buchmann[1)] Stephan Düllmann[1)]
Hugh C. Williams[2)]

[1)] FB 10 Informatik
Universität des Saarlandes
D-6600 Saarbrücken
WEST GERMANY

[2)] Department of Computer Science
University of Manitoba
Winnipeg, Manitoba
CANADA R3T 2N2

ABSTRACT

In [2] Buchmann and Williams presented a new public key exchange system based on imaginary quadratic fields. While in that paper the system was described theoretically and its security was discussed in some detail nothing much was said about the practical implementation.

In this paper we discuss the practical aspects of the new system, its efficiency and implementation. In particular we study the crucial point of the method: ideal reduction. We suggest a refinement of the well known reduction method which has been implemented on a computer. We present extensive running time statistics and a detailed complexity analysis of the methods involved.

The implementation of the reduction procedure on chips is subject of future research.

I. THE DIFFIE-HELLMAN SCHEME

In their paper "New Directions in Cryptography" [3] Diffie and Hellman introduced in 1976 the idea of public key exchange. By this method it is

J.J. Quisquater and J. Vandewalle (Eds.): Advances in Cryptology - EUROCRYPT '89, LNCS 434, pp. 597-616, 1990.

possible to communicate a secret key for some cryptosystem over a public insecure channel. We briefly review the idea of Diffie and Hellman.

Suppose that Alice ($\mathcal{A}$) and Bob ($\mathcal{B}$) wish to secretly exchange a key.

(1) $\mathcal{A}$ chooses a finite group G and – at random – an element $\lambda \in G$. Both G and λ are sent to $\mathcal{B}$ over the public channel.

(2) Both $\mathcal{A}$ and $\mathcal{B}$ now select – at random – an integer a and b, respectively. These integers are kept secret.

(3) $\mathcal{A}$ computes $\alpha = \lambda^a$ and transmits the result to $\mathcal{B}$. In the same way $\mathcal{B}$ computes $\beta = \lambda^b$ and sends it to $\mathcal{A}$. (Note that the group elements α and β are public.)

(4) $\mathcal{A}$ computes $\gamma = \beta^a$, $\mathcal{B}$ computes $\gamma' = \alpha^b$.

Because $\gamma = \beta^a = (\lambda^b)^a = (\lambda^a)^b = \alpha^b = \gamma'$ both $\mathcal{A}$ and $\mathcal{B}$ are in possession of the same key γ without this key having been sent over the public channel.

In order to be able to apply this algorithm in practice it is necessary to have an efficient multiplication routine in G i.e., if all the elements are represented by numbers in $\{0, 1, \ldots, |G| - 1\}$ the representation of the product of two elements of G should be computable in time polynomial in $\log |G|$.

Then, using the method of binary shifting (see [6, p. 441 ff.]) powers of group elements can be computed very efficiently even for large exponents d, namely in $O(\log d)$ elementary operations in G.

The scheme is secure if the key cannot be guessed easiliy and cannot be determined easily from the public information.

To avoid the key beeing guessed easily it is necessary to choose a group G of very high order, to pick a starting element λ of high order (close to $|G|$) and to select large exponents a and b.

To make sure that the choice of G is adequate one has to analyse its arithmetic properties carefully. A neccessary condition is that the determination of discrete logarithms in G is difficult. Note, however, that this condition is not sufficient for the security of the system because the determination of λ^{ab} from λ, λ^a and λ^b might be easier than the calculation of a and b from λ, λ^a and λ^b.

So far the following groups have been suggested:

- The group $G = GL_n(\mathbb{Z}/p\mathbb{Z})$ of invertible $n \times n$-matrices over the finite field $(\mathbb{Z}/p\mathbb{Z})$. ([14])
- The group of points on an elliptic curve over a finite field. ([11])
- Several groups associated with higher dimensional varieties. ([7])
- The group $G = (\mathbb{Z}/n\mathbb{Z})$ where n is the product of two large primes. ([10])
- The class group of an imaginary quadratic field. ([2])

In this paper we continue the discussion of the case where G is the class group of an imaginary quadratic field.

II. THE CLASS GROUP OF AN IMAGINARY QUADRATIC FIELD

First we summarize the main facts concerning imaginary quadratic fields. All these facts are well known and therefore will be given without proof. Proofs of the statements made here and more detailed descriptions can be found in standard texts like Hua [5] or Narkiewicz [12], [13].

Let $D < 0$ be a squarefree integer and let $K = \mathcal{Q}(\sqrt{D})$ be the quadratic field which is defined by adjoining $\sqrt{D}$ to the set of rational numbers $\mathcal{Q}$. If $\alpha \in K$ we denote by $\overline{\alpha}$ the complex conjugate of α, by $\mathrm{Tr}(\alpha)$ the *trace* of α, i.e. the value of $\alpha + \overline{\alpha}$, and by $N(\alpha)$ the *norm* of α, i.e. the value of $\alpha \cdot \overline{\alpha}$. Note that $N(\alpha) \geq 0$ for every $\alpha \in K$.

We define

$$r = \begin{cases} 1 & \text{if } D \equiv 2, 3 \bmod 4 \\ 2 & \text{if } D \equiv 1 \bmod 4. \end{cases}$$

Then the *discriminant* of K is given by

$$\Delta = \frac{4D}{r^2}.$$

For $\alpha, \beta \in K$ put $[\alpha, \beta] = \alpha\mathbb{Z} + \beta\mathbb{Z}$. Moreover, let

$$\omega = \frac{r - 1 + \sqrt{D}}{r}.$$

Then the *ring of algebraic integers* in K is given by

$$\mathcal{O}_K = [1, \omega].$$

The *fractional ideals* of $\mathcal{O}_K$ form a multiplicative abelian group denoted by $\mathcal{F}$. By *1-ideal* we denote the neutral element in this group. The *principal ideals* of $\mathcal{O}_K$ form a subgroup of $\mathcal{F}$ denoted by $\mathcal{P}$. The factor group $\mathcal{F}/\mathcal{P}$ is called the *class group* of $\mathcal{O}_K$ denoted by $\mathcal{C}_\Delta$. The class group is finite, its order is called the *class number* of $\mathcal{O}_K$ and is denoted by h. The elements of the class group are called the *ideal classes*. Two ideals of $\mathcal{O}_K$ are said to be *equivalent* if they are in the same ideal class.

Every integral ideal a of $\mathcal{O}_K$ can be written in the form

$$\mathrm{a} = [a, b + c\omega]$$

where $a, c \in \mathbb{Z}$ [1], $b \in \mathbb{Z}$ and

$$c|a,\ c|b,\ ac|N(b + c\omega).$$

For a given ideal a the integers a and c are uniquely determined and b is unique modulo a. a is the least positive rational integer in a, denoted by $L(\mathrm{a})$. If $c = 1$ then a is called *primitive*.

Every integral primitive ideal a of $\mathcal{O}_K$ can be uniquely presented in the form

$$\mathrm{a} = [L(\mathrm{a}), b + \omega]$$

with $b \in \mathbb{Z}$ and

$$-L(\mathrm{a}) < \mathrm{Tr}(b + \omega) \leq L(\mathrm{a}).$$

This is called the *normal presentation* of a.

An integral primitive ideal a whose normal presentation is $\mathrm{a} = [L(\mathrm{a}), b + \omega]$ is called *reduced* if

$$\begin{aligned} |b + \omega| &\geq L(\mathrm{a}), \\ \mathrm{Tr}(b + \omega) &> 0 \quad \text{if } |b + \omega| = L(\mathrm{a}). \end{aligned}$$

There are the following main facts concerning reduced ideals:

- For each reduced ideal we have $L(\mathrm{a}) < \sqrt{|\Delta|/3}$.
- Every ideal class of $\mathcal{O}_K$ contains exactly one reduced ideal.

Each ideal class can be represented by its reduced ideal. So the arithmetic in the class group of imaginary quadratic fields can be reduced to ideal-arithmetic: In order to determine the product of two ideal classes multiply their reduced ideals and compute the reduced ideal equivalent to this product.

Since every reduced ideal is presented by a pair of integers which are less than $\sqrt{|\Delta|/3}$ in absolute value class groups can be used in the Diffie-Hellman scheme if multiplication and reduction of ideals can be carried out efficiently. The secret key must be one of the integers of the normal presentation of the reduced ideal computed in the scheme.

III. THE ALGORITHMS

In [2] it is pointed out that D has to be of order of magnitude 10^{200} to guarantee high security of the scheme. Also the exponents should be of this order of magnitude. So first of all we need a multi-precision integer-arithmetic for implementing the scheme. For details see [6] or [1].

After the initialization all computations have the following structure: For an integer n and an ideal class Γ given by its reduced ideal $\mathbf{a}_\Gamma$ compute the reduced ideal in Γ^n, i.e. the reduced ideal equivalent to $(\mathbf{a}_\Gamma)^n$. This can be done by means of the well known fast exponentiation technique described in [6, p. 441].

Algorithm 3.1 (Exponentiation of ideal classes)
Input: *Reduced ideal* $\mathbf{a}$, *exponent* $n \in \mathbb{Z}^{>0}$.
Output: *Reduced ideal* $\mathbf{b}$ *equivalent to* $\mathbf{a}^n$.

(1) $N \leftarrow n$; $\mathbf{b} \leftarrow 1$*-ideal*; $\mathbf{c} \leftarrow \mathbf{a}$;

(2) if N *is even then goto (5)*;

(3) $\mathbf{b} \leftarrow \mathbf{c} \cdot \mathbf{b}$;

(4) reduce $\mathbf{b}$;

(5) $N \leftarrow \lfloor N/2 \rfloor$;

(6) if $N = 0$ *terminate.*

(7) $\mathbf{c} \leftarrow \mathbf{c} \cdot \mathbf{c}$;

(8) reduce $\mathbf{c}$ *and goto (2)*;

For practical purposes it is more convenient to represent an integral primitive normally presented ideal $\mathbf{a} = [L(\mathbf{a}), b+\omega]$ by $(A, B) \in \mathbb{Z}^2$ with

$A = rL(\mathrm{a})$ and $B = rb + r - 1$. Note that the new presentation is well defined and unique.

For the multiplication we use an algorithm based on Shanks [15] which computes a primitive ideal equivalent to the product of two primitive ideals:

Algorithm 3.2 (Multiplication of ideals)
Input: *Two primitive ideals* (A_1, B_1), (A_2, B_2).
Output: *Primitive ideal* (A_3, B_3) *equivalent to the product of the input ideals.*

(1) $G' \leftarrow \gcd(A_1, A_2)$;
compute the coefficient V' *in* $A_1V' + A_2W' = G'$;

(2) $B_3 \leftarrow V'A_1(B_2 - B_1)$;

(3) $A_3 \leftarrow rA_1A_2$;

(4) if $G' = 1$ *then goto (8);*

(5) $G \leftarrow \gcd(G', (B_1 + B_2))$;
compute the coefficients U', U *in* $U'G' + U(B_1 + B_2) = G$;

(6) $B_3 \leftarrow [B_3U' + U(D - B_1^2)]/G$;

(7) $A_3 \leftarrow A_3/G^2$;

(8) $B_3 \leftarrow B_1 + B_3$;

If the ideals to be multiplied are equal then step 1 of the multiplication algorithm can be replaced by

$$G = A_1,\ W' = 1,\ V' = 0$$

because $A_1 = A_2$. This simplification leads to the following algorithm for squaring an ideal:

Algorithm 3.3 (Squaring of an ideal)
Input: *Primitive ideal* (A_1, B_1).
Output: *Primitive ideal* (A_3, B_3) *equivalent to the square of the input ideal.*

(1) $G \leftarrow \gcd(A_1, 2B_1)$;
compute the coefficient U *in* $U'A_1 + 2UB_1 = G$;

(2) $B_3 \leftarrow B_1 + [U(D - B_1^2)]/G$;

(3) $A_3 \leftarrow r(A_1/G)^2$;

(4) terminate.

Now we discuss ideal reduction. In order to decrease the size of the coefficients of the ideal representation whenever it is possible the ideals are reduced after each multiplication in Algorithm 3.1. So the reduction algorithm is applied very frequently and therefore it is necessary to have a fast reduction method. Our method is a refinement of the following well known algorithm (see [8] or [9]).

Algorithm 3.4 (Reduction of ideals, classical version)
Input: *Primitive ideal* (A', B').
Output: *Reduced ideal* (A, B) *(in normal presentation) equivalent to the input ideal.*

(1) $A \leftarrow A'$; $B \leftarrow B'$;

(2) $B \leftarrow round(B/A) \cdot A - B$;

(3) $A_N \leftarrow (B^2 - D)/A$;

(4) if $A_N < A$ *then* $A \leftarrow A_N$ *and goto (2);*

(5) if $(2B < A)$ *and* $(B < 0$ *or* $A < A_N)$ *then* $B \leftarrow -B$;

In step 2 of the algorithms "round" means a multi precision integer function which computes the rounded quotient of two integers. This can be done by the following algorithm:

Algorithm 3.5 (Rounded quotient of two integers)
Input: *Integers* A *and* B, $B \neq 0$.
Output: *Integer* C *with* $C = round(A/B)$.

(1) $C \leftarrow A/B$; $R \leftarrow A \bmod B$;

(2) if $2|R| \geq |B|$ *then* [*if* $A > 0$ *then* $C \leftarrow C + 1$ *else* $C \leftarrow C - 1$];

Now we present the refinement of the algorithm which was theoretically described in [2].

Algorithm 3.6 (Reduction of ideals, optimized version)

Input: *Primitive ideal* (A', B').

Output: *Reduced ideal* (A, B) *(in normal presentation) equivalent to the input ideal.*

(1) $A \leftarrow A'$;
if $B' < 0$ *then* $s \leftarrow -1$ *else* $s \leftarrow 1$;
$B \leftarrow |B'|$;

(2) $Q \leftarrow B/A$; $R \leftarrow B \bmod A$; $M \leftarrow A - 2R$;
if $M \geq 0$ *then* $B \leftarrow R$ *and* $s \leftarrow -s$ *else* $B \leftarrow R + M$;
$A_N \leftarrow (B^2 - D)/A$;

(3) if $A_N < A$ *then* $A_O \leftarrow A$ *and* $A \leftarrow A_N$ *else goto (6);*

(4) $Q \leftarrow B/A$; $R \leftarrow B \bmod A$; $M \leftarrow A - 2R$;

(5) $A_N \leftarrow A_O - (R + B) \cdot Q$;
If $M \geq 0$ *then* $B \leftarrow R$ *and* $s \leftarrow -s$ *else* $B \leftarrow R + M$ *and*
$A_N \leftarrow A_N + M$;
goto (3);

(6) if $s < 0$ *then* $B \leftarrow -B$;
if $(2B < A)$ *and* $(B < 0$ *or* $A < A_N)$ *then* $B \leftarrow -B$;

Comparing the two versions of the algorithm we see that the number of iterations in both algorithms and also the sequences of the values for A and B computed in both algorithms are equal. There are two main differences between the two versions:

The computation of the new value of A in each iteration in the optimized version needs one division of multi precision integers less than in the first version because the division in step 3 is avoided. Instead of this the value of A from the preceeding iteration is used. Therefore this simplification cannot be made up in the first iteration. So step 2 of Algorithm 3.6 contains the first iteration of Algorithm 3.4.

The computation of the rounded quotient of A and B is avoided in the optimized version. Moreover the sign of B is stored in a seperate variable s. In this way one multiplication of multi precision integers in each iteration is avoided.

The following theorem makes sure that algorithm 3.6 indeed computes the reduced ideal equivalent to the input ideal in finitly many

iterations. Moreover an estimation for the number of iterations is given. In [2] this theorem was proved for Algorithm 3.4 which requires one more iteration.

Theorem 3.1 *Let (A, B) represent a primitive ideal. Algorithm 3.4 finds the reduced ideal equivalent to (A, B) in at most i iterations of the steps 3,4 and 5 where*

$$i = \max\left\{0, \left\lfloor \frac{1}{2}\log\left(\frac{3A}{5\sqrt{|D|}}\right)\right\rfloor + 1\right\}.$$

Finally we describe the initialization where we have to find an appropriate starting ideal class for a given value of D. We chose – at random – a prime number q ($q \equiv 3 \bmod 4$) such that D is quadratic residue modulo q. Then

$$\begin{aligned} A &= r \cdot q \\ B &= D^{\frac{q+1}{4}} \bmod rq \end{aligned}$$

satisfy the condition

$$rA \mid B^2 - D.$$

So (A, B) represents a primitive ideal. Reduction of this ideal gives the starting ideal class.

IV. THE COMPLEXITY

First we compute the size of the integers occuring in the computations of our key exchange system for a fixed value of D.

Theorem 4.1

(a) If (A, B) is a reduced ideal in normal presentation then

$$\begin{aligned} A &\leq 2\sqrt{\frac{|D|}{3}}, \\ |B| &\leq \sqrt{\frac{|D|}{3}}. \end{aligned}$$

(b) If (A, B) is the primitive ideal computed by Algorithm 3.2 or 3.3 then

$$A \leq \frac{8|D|}{3},$$
$$|B| \leq \frac{16|D|}{3}\sqrt{\frac{|D|}{3}}$$

provided that the input ideals were reduced and given in normal presentation.

(c) The size of the integers occuring in the computations of our key exchange system is bounded by

$$M = 3|D|\sqrt{|D|}.$$

Proof:

(a) Let $\mathfrak{a} = [L(\mathfrak{a}), b+\omega] = (A, B)$. Then, according to the definition of A and B in section 4, we have $A = rL(\mathfrak{a})$ and $B = rb+r-1$. Moreover

$$b + \omega = \frac{B+1}{r} - 1 + \frac{r-1+\sqrt{D}}{r} = \frac{B+\sqrt{D}}{r}$$

and therefore

$$\mathrm{Tr}(b+\omega) = \frac{2B}{r} \leq 2B.$$

Since $\mathfrak{a}$ is reduced we have

$$A = rL(\mathfrak{a}) < r\sqrt{\frac{|\Delta|}{3}} = r\sqrt{\frac{4|D|}{3r^2}} = 2\sqrt{\frac{|D|}{3}}.$$

Since $\mathfrak{a}$ is in normal presentation we have $|\mathrm{Tr}(b+\omega)| \leq L(\mathfrak{a})$ and therefore

$$|B| \leq \frac{L(\mathfrak{a})}{2} = \sqrt{\frac{|D|}{3}}.$$

(b) In Algorithm 3.2 the new value for A is computed in steps 3 and 7. From these steps and part (a) of this theorem we get

$$A \leq 2 \cdot \left(2\sqrt{\frac{|D|}{3}}\right)^2 = \frac{8|D|}{3}.$$

The new value for B is computed in steps 2, 4 and 8. The coefficients V', V, U' in step 2 and 5 can be choosen less than $2\sqrt{\frac{|D|}{3}}$. So

$$B \leq 2\sqrt{\frac{|D|}{3}} \cdot 2\sqrt{\frac{|D|}{3}} \cdot \sqrt{\frac{|D|}{3}}.$$

The same arguments apply to Algorithm 3.3.

(c) The input ideals for the multiplication and squaring algorithm are always reduced ideals in normal presentation. The input ideals for the reduction algorithm are always results of multiplication and squaring. Moreover the successive values of A in the reduction algorithm are strictly decreasing with the exeption of the last one and therefore the size of the input value of A is an upper bound for all values of A and B occuring in the algorithm. So by (a) and (b) we see the maximum integer occuring in the system is bounded by

$$\frac{16|D|}{3}\sqrt{\frac{|D|}{3}} = \frac{16\sqrt{3}}{9}|D|\sqrt{|D|} < 3|D|\sqrt{|D|}.$$

□

Note that by part (c) of this theorem we also can estimate the number of computer words necessary to store the integers used in the algorithm. If for example D has 200 decimal digits then our multi precision arithmetic package must be able to handle numbers of 301 decimal digits.

Next we list the number of elementary operations (additions, multiplications and divisions) with multi precision integers for each of our algorithms. Multiplication by r is considered as an addition because r is either 1 or 2.

To analyze the multiplication and squaring algorithm we first need to know the number of elementary operations necessary to compute the gcd's. This computation can be done by the euclidian algorithm [6, p. 325]. If one has to compute a gcd and both coefficients of its representation the euclidian algorithm needs one division, three additions and three multiplications in each division step. This case will be called "gcd2". If one has to compute a gcd and only one of the coefficients of its representation one can avoid some computations. Then the euclidian algorithm only needs one division, two additions and two multiplications in each division step. This case will be called "gcd1".

Theorem 4.2

(a) The maximum number of division steps in the euclidian algorithm (see [6, p. 325]) used for the computation of the gcd*'s in the multiplication and squaring algorithm is*

$$M' \leq \frac{3}{4}\log|D| + 1.$$

(b) In case $G' = 1$ Algorithm 3.2 requires at most $(3 + 2M')$ additions, $(3 + 2M')$ multiplications and M' divisions of multi precision integers.

(c) In case $G' > 1$ Algorithm 3.2 requires at most $(6 + 5M')$ additions, $(7 + 5M')$ multiplications and $(2 + 2M')$ divisions of multi precision integers.

(d) Algorithm 3.3 requires at most $(3 + 2M')$ additions, $(3 + 2M')$ multiplications and $(2 + M')$ divisions of multi precision integers.

Proof:

(a) From the bounds given in Theorem 4.1 (a) for the input values of the multiplication and squaring algorithm and by the listings of these algorithms in the previous section we see that the input values for the gcd's are bounded by $2\sqrt{\frac{|D|}{3}}$. By Corollary L in [6, p. 343] we can conclude that the number of division steps required when the euclidian algorithm is applied to numbers u and v with $0 \leq u, v < N$ is at most $4.8 \log_{10} N + 0.68$. So in our special case we have

$$\begin{aligned} M' &\leq 4.8 \log_{10} \left(2\sqrt{\frac{|D|}{3}} \right) + 0.68 \\ &= 4.8 \log_{10} \frac{2}{\sqrt{3}} + \frac{4.8}{\log 10} \log \sqrt{|D|} + 0.68 \\ &\leq \frac{3}{2} \log \sqrt{|D|} + 1 \\ &= \frac{3}{4} \log |D| + 1. \end{aligned}$$

(b) For $G' = 1$ Algorithm 3.2 requires three additions, three multiplications and one gcd1.

(c) For $G' > 1$ Algorithm 3.2 requires three additions, seven multiplications, two divisions, one gcd1 and one gcd2.

(d) Algorithm 3.3 requires three additions, three multiplications, two divisions and one gcd1.

□

Now we turn to the reduction algorithm. Here we first have to compute the maximum number of iterations.

Theorem 4.3

(a) The maximum number of iterations in the optimized version of the reduction algorithm is

$$M'' \leq \frac{1}{4}\log|D| + 2.$$

(b) Algorithm 3.4 requires $(1+4(M''+1))$ *additions,* $2(M''+1)$ *multiplications and* $2(M''+1)$ *divisions of multi precision integers. Here the operations caused by the seperate " round"-algorithm are included.*

(c) Algorithm 3.6 requires $(5+6M'')$ *additions,* $(1+M'')$ *multiplications and* $(2+M'')$ *divisions of multi precision integers.*

Proof:

(a) Using the bound given in Theorem 4.1 (b) we have

$$A \leq \frac{8|D|}{3}$$

for the input ideal of the reduction algorithm. From Theorem 3.1 we conclude that the number of iterations in Algorithm 3.6 is bounded by

$$\begin{aligned} M'' &\leq \frac{1}{2}\log\left(\frac{3^{\frac{8|D|}{3}}}{5\sqrt{|D|}}\right) + 1 \\ &= \frac{1}{2}\log\left(\frac{8}{5}\sqrt{|D|}\right) + 1 \\ &= \frac{1}{2}\log\frac{8}{5} + \frac{1}{4}\log|D| + 1 \\ &\leq \frac{1}{4}\log|D| + 2. \end{aligned}$$

(b) The number of iterations in Algorithm 3.4 is one more than in Algorithm 3.6. Each iteration (steps 2, 3 and 4) requires two additions, two multiplications, one division and one call of the round procedure in which there are two additions and one division. Step 5 of the reduction algorithm requires one additional addition.

(c) Each iteration (steps 3, 4 and 5) in Algorithm 3.6 requires six additions, one multiplication and one division. Additionally, steps 2 and 6 require five additions, one multiplication and two divisions.

□

Now we have to examine Algorithm 3.1 for the exponentiation of ideal classes.

Theorem 4.4 *If the exponents a and b are bounded by $a \cdot b < \sqrt{|D|}$ then the total number of iterations performed in all calls of the exponentiation algorithm in the key exchange system is at most*

$$M''' \leq \log|D| + 4.$$

Proof: The number of iterations in algorithm 3.1 for an arbitrary exponent n is lower than $\lfloor \log n \rfloor + 1$ (see [6, 443]). Each exponent is used twice in the scheme. Therefore the total number of iterations performed in the four calls of the exponentiation algorithm is bounded by

$$\begin{aligned} M''' &\leq 2(\log a + 1 + \log b + 1) \\ &\leq 2\log(a \cdot b) + 4 \\ &\leq 2\log\sqrt{|D|} + 4 \\ &= \log|D| + 4. \end{aligned}$$

□

Finally we give an upper bound for the number of elementary operations performed in the key exchange system which follows from Theorems 4.2, 4.3 and 4.4.

Theorem 4.5 *Using the optimized version of the reduction algorithm our key exchange system takes at most*

$$\begin{aligned} 217 + \frac{169}{2}\log|D| + \frac{33}{4}\log^2|D| \quad & \textit{additions,} \\ 95 + \frac{185}{4}\log|D| + \frac{23}{4}\log^2|D| \quad & \textit{multiplications,} \\ 64 + \frac{105}{4}\log|D| + \frac{11}{4}\log^2|D| \quad & \textit{divisions} \end{aligned}$$

of multi-precicison integers.

Now we give the asymptotical bit complexity for our algorithms and for the whole method. Clearly the complexity depends on the implementation of the multi precision routines. Using the so called "Classical Algorithms" (see [6, p. 250 ff.]) multiplication and divsion of n-bit numbers require $O(n^2)$ bit operations. Using the so-called "Fast Multiplication Techniques" these operations numbers require only $O(n \log n \log\log n)$ bit-operations (see [1, p. 272, 286]).

Theorem 4.6

(a) The algorithms for squaring and multiplication of ideals and both versions of the reduction algorithm have bit complexity

(i) $O(\log^3 |D|)$ *if the classical algorithms are used,*

(ii) $O(\log^2 |D| \log\log |D| \log\log\log |D|)$ *if fast multiplication technique is used.*

(b) The method for public key exchange has bit complexity

(i) $O(\log^4 |D|)$ *if the classical algorithms are used,*

(ii) $O(\log^3 |D| \log\log |D| \log\log\log |D|)$ *if fast multiplication technique is used.*

Proof: Both statements follow immediately from the previous results. Note that according to Theorem 4.1 (c) the binary length of the integers occuring in our scheme is $O(\log |D|)$. □

The theorem shows that both versions of the reduction algorithm have the same asymptotical complexity. The improvement in the optimized version only affects the O-constant as can be seen from Theorem 4.3.

We see that using fast multiplication techniques the running time of our method for public key exchange is a cubic polynomial in the length of the input data. Therefore it is executable for big exponents and big discriminants.

V. RUNNING TIME STATISTICS

The method for public key exchange has been implemented on a SIEMENS 7580-S computer of the University of Düsseldorf [4]. The programming language was FORTRAN-77. For the multi-precision arithmetic we used the classical algorithms. These and some procedures to get the running time statistics presented below were taken from the number theoretic subroutine library KANT.

We computed many examples where the value of D was the product of two prime numbers each of size up to 10^{100}. This choice of D was done according to the remarks in [2] concerning the security of the method. Both cases $r = 1$ and $r = 2$ occured equally often.

In the table below we present the statistics of 32 of these examples. Here the size of D varies between 10^{120} and 10^{200}. For each size of D we took several exponents of different size. The columns of the table have the following contents:

(1) Number of the example

(2) Size of D (number of decimals)

(3) Size of the product of the exponents (number of decimals)

(4) Number of calls of the squaring algorithm

(5) Number of calls of the multiplication algorithm

(6) Number of calls of the reduction algorithm

(7) Average number of iterations in the reduction algorithm

(8) Maximum number of iterations in the reduction algorithm

(9) Theoretical bound for the number of iterations in the reduction algorithm

(10) Average running time of the squaring algorithm

(11) Average running time of the multiplication algorithm

(12) Average running time of the classical version of the reduction algorithm

(13) Average running time of the optimized version of the reduction algorithm

(14) Total running time of the program with the optimized version of the reduction algorithm

All times are given in seconds (CPU).

From the structure of the exponentiation algorithm we know the connection between the exponents and the number of calls of our algorithms: The number of squarings (col. 4) is exactly twice the sum of the logarithms of the two exponents. The number of multiplications (col. 5) is twice the number of One's in the binary representation of the exponents. The number of reductions (col. 6) minus 1 is exactly the sum of

the columns 4 and 5 because there is a reduction after each squaring or multiplication and one reduction in the initialization.

The theoretical bound for the number of iterations in the reduction algorithm is computed from the formula in Theorem 4.3 (a) with a value of D rounded to the next power of ten.

Table (Running time statistics)

	Decimals		Subroutine Calls			Iterations in reduction			Average running time				Total running time
											Reduction		
	D	$a \cdot b$	Squ	Mul	Red	aver	max	theo	Squ	Mul	Cla	Opt	
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)
1	198	10	62	30	93	57	80	166	0.23	0.23	1.60	0.27	47.11
2	198	30	192	88	281	64	79	166	0.26	0.28	1.80	0.30	159.83
3	197	50	324	162	487	65	77	165	0.27	0.28	1.81	0.30	280.79
4	197	71	468	236	705	66	82	165	0.27	0.28	1.82	0.30	408.24
5	197	97	640	294	935	66	80	165	0.27	0.27	1.84	0.30	540.30
6	182	11	66	32	99	51	71	153	0.20	0.21	1.25	0.22	42.54
7	182	30	190	82	273	59	79	153	0.23	0.24	1.44	0.26	133.59
8	182	48	310	140	451	60	75	153	0.23	0.24	1.48	0.27	225.44
9	182	71	466	238	705	61	74	153	0.24	0.24	1.50	0.27	356.13
10	182	93	608	286	895	62	74	153	0.24	0.24	1.51	0.27	454.34
11	178	19	118	62	181	54	72	148	0.21	0.21	1.28	0.23	80.75
12	177	40	246	132	379	57	74	148	0.22	0.22	1.34	0.25	177.06
13	177	60	394	202	597	59	75	148	0.23	0.23	1.39	0.25	288.64
14	177	79	518	238	757	59	75	148	0.23	0.23	1.40	0.25	363.59
15	167	10	58	26	85	48	72	140	0.18	0.18	1.03	0.20	32.25
16	167	40	256	122	379	55	69	140	0.20	0.21	1.18	0.22	162.92
17	166	60	394	202	597	55	70	139	0.20	0.20	1.20	0.22	255.87
18	166	79	518	238	757	55	70	139	0.20	0.21	1.22	0.23	326.39
19	157	10	62	40	103	46	64	132	0.17	0.17	0.89	0.18	46.50
20	157	30	196	98	295	51	66	132	0.18	0.19	0.99	0.20	113.45
21	152	51	334	182	517	50	63	128	0.18	0.18	0.94	0.19	193.32
22	158	71	466	228	695	53	67	133	0.19	0.19	1.04	0.21	276.74
23	146	10	62	30	93	42	59	123	0.14	0.16	0.72	0.16	28.40
24	147	30	190	84	275	47	60	124	0.16	0.16	0.83	0.18	93.22
25	136	50	328	188	517	46	59	115	0.15	0.15	0.72	0.16	161.26
26	142	71	466	238	705	48	62	119	0.16	0.17	0.81	0.18	240.87
27	127	20	126	44	171	40	51	107	0.13	0.13	0.58	0.14	45.33
28	133	39	246	132	379	44	56	112	0.14	0.14	0.67	0.15	111.96
29	127	60	394	202	597	43	53	107	0.13	0.14	0.61	0.14	167.23
30	123	20	128	54	183	40	54	104	0.12	0.13	0.54	0.13	47.35
31	129	41	258	140	399	43	55	109	0.13	0.14	0.62	0.15	113.18
32	124	59	380	176	557	42	52	104	0.13	0.13	0.58	0.14	149.86

The comparison of the values in column 8 and 9 shows that the maximum number of iterations in the reduction algorithms is about half of the theoretical bound.

The comparision of the average running time for the two versions of the reduction algorithm shows that there is indeed an important speeding up. The optimized version only requires 20% or less of the running time of the classical version. Most of this gain is caused by avoiding one multi precision division in each iteration. Note that the optimized version of the reduction algorithm is nearly as fast as the multiplication of ideals.

Note also that there is almost no difference in the average running time of the multiplication and the squaring algorithm. Here the theoretical improvement has no practical effect.

The examination of the total running time of the program confirms the complexity result given in Theorem 4.6.

Now we give the data computed by our program for the first example mentioned in the table above. This example has the highest values of D.

We list the information to be transmitted over the public channel (D, the starting ideal class Λ and the ideal classes $\Phi = \Lambda^a$ and $\Psi = \Lambda^b$) and also the secret information (the exponents a and b and the ideal class $\Gamma = \Lambda^{ab}$ which gives the secret key).

Note that even if we use low exponents like in the first example the secret key has a reasonable size of about $\sqrt{|D|}$. Nevertheless one should choose larger exponents like in the fifth example to have greater security.

Example

$$
\begin{aligned}
D \;=\; & -11980374469127450695419647802766644830542210733866131993089788267603365417346323605263646114675501\\
& 72246998555004267772913478851528257403104704701886\\
& 44025223079332669700563976732928049691985604910619
\end{aligned}
$$

Secret exponents: $a = 543210, \quad b = 7980$

Starting ideal class Λ: $A = 982, \quad B = 31$

Ideal class $\Phi = \Lambda^a$:

$$A = 97142961303637398608244575991720552258806601494913086285698000466252477914793685446384554925704750$$

$$B = 39599343846494648599463917438519954544524065268375336467884301137146841919182233472651118628470641$$

Ideal class $\Psi = \Lambda^b$:

$$A = 32451842285914702948500775856419391223263538507688024167263208946226417449854469021281560392496561 0$$

$$B = 13464677655162601698578228403004470869914739828741163081794853734026579982862374446145573388814867 1$$

Ideal class $\Gamma = \Lambda^{ab}$:

$$A = 33264343635181781616896753284870110832510161684174460645002006317124780344636958567041199986601143 4$$

$$B = -97577809448926539679909701428640738988823144173532031955109450292182506204157234412540729934371 05$$

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.

[2] J. Buchmann and H.C. Williams, *A key exchange system based on imaginary quadratic fields*, Journal of Cryptology 1 (1989), to appear.

[3] W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Transactions on Information theory 22 (1976), 472 – 492.

[4] S. Düllmann, *Ein neues Verfahren zum öffentlichen Schlüsselaustausch*, Staatsexamensarbeit, Universität Düsseldorf, 1988.

[5] Hua Loo Keng, *Introduction to number theory*, Springer Verlag, Berlin and New York, 1982.

[6] D.E. Knuth, *The art of computer programming*, vol. 2: *Seminumerical algorithms*, Addison-Wesley, Reading, Massachusetts, 2. Auflage, 1981.

[7] N. Koblitz, *A Family of Jacobians Suitable for Discrete Log Cryptosystems*, to appear in: Proceedings of Crypto'88, Lecture Notes of Computer Science, Springer-Verlag.

[8] J.C. Lagarias, *Worst-Case Complexity Bounds for Algorithms in the Theory of Integral Quadratic Forms*, Journal of Algorithms 1 (1980), 142 – 186.

[9] H.W Lenstra, Jr., *On the calculation of regulators and class numbers of quadratic fields*, London Math. Soc. Lecture Notes **56** (1982), 123 – 150.

[10] K.S. McCurley, *A Key Distribution System equivalent to Factoring*, preprint, 1987.

[11] V. Miller, *Use of Elliptic Curves in Cryptography*, Advances in cryptology (Proceedings of Crypto'85), Lecture Notes in Computer Science **218** (1986), Springer-Verlag, NY, 417 - 426.

[12] W. Narkiewicz, *Elementary and analytic theory of algebraic numbers*, Warszawa, 1974.

[13] W. Narkiewicz, *Number theory*, Warszawa, 1977, engl. edition 1983.

[14] R.W.K. Odoni, V. Varadharajan and P.W. Sanders, *Public Key Distribution in Matrix Rings*, Electronic Letters **20** (1984), 386 – 387.

[15] D. Shanks, *Class number, a theory of factorization and genera*, Proc. Symposia in Pure Mathematics **20** (1971), 415 – 440.