NICE - New Ideal Coset Encryption -

Michael Hartmann[1], Sachar Paulus[2], and Tsuyoshi Takagi[3]

[1] Darmstadt University of Technology,
Alexanderstr. 10, 64283 Darmstadt, Germany,
mhartman@cdc.informatik.tu-darmstadt.de,
[2] SECUDE Sicherheitstechnologie Informationssysteme GmbH,
Landwehrstr. 50a, 64283 Darmstadt, Germany,
paulus@secude.com,
[3] NTT Information Sharing Platform Laboratories,
Immermannstr. 40, D-40210 Düsseldorf, Germany,
ttakagi@ntt.de

Abstract. Recently, a novel public-key cryptosystem constructed on number fields is presented. The prominent theoretical property of the public-key cryptosystem is a quadratic decryption bit complexity of the public key, which consists of only simple fast arithmetical operations. We call the cryptosystem NICE (New Ideal Coset Encryption). In this paper, we consider practical aspects of the NICE cryptosystem. Our implementation in software shows that the decryption time of NICE is comparably as fast as the encryption time of the RSA cryptosystem with $e = 2^{16} + 1$. To show if existing smart cards can be used, we implemented the NICE cryptosystem using a smart card designed for the RSA cryptosystem. Our result shows that the decryption time of NICE is comparably as fast as the decryption time of RSA cryptosystem but not so fast as in software implementation. We discuss the reasons for this and indicate requirements for smartcard designers to achieve fast implementation on smartcards.
Key words: public-key cryptosystem, fast decryption, quadratic order, smart card implementation.

1 Why NICE?

Plenty of public-key cryptosystem not relying only on the RSA cryptosystem have been proposed. They are stemming from deep number theory (hyper- and superelliptic curves) to geometry and combinatorics (LLL-based systems). One major advantage of RSA is its simplicity: it can be easily implemented, one only needs a moderate background in mathematics to understand it. Moreover, RSA is quite fast - there exist public key cryptosystems which are much faster, but the combination of simplicity, speed and confidence in its security makes it the most practical cryptosystem. Until now, only one other system may be considered as equally interesting: ElGamal type systems on elliptic curves. Elliptic curves are somewhat more complicated, and the best known algorithms to break elliptic curve cryptosystems are much slower, in the order of exponential complexity.

Ç.K. Koç and C. Paar (Eds.): CHES'99, LNCS 1717, pp. 328–339, 1999.

So, both speed and security are worth being paid by the higher mathematical complexity.

But: both systems have one drawback in common: the decryption/signing time is of cubic complexity in the bit length of the public key for both systems because these steps consist of modular multiplication(s). This becomes even more important when thinking of smart cards. The smart card is considered to be the personal security computing device of tomorrow. It contains all personal secret information, especially the private keys for public key systems like decryption and signing. The complexity of PC operating systems is too high to be reliably secure; every relevant security operation should be effected by the smart card. Non-relevant operations like public key encryption and signature verification could be done by the PC. Operations which cannot be transferred to the PC at all due to security reasons are decryption and signing. Considering RSA, the task which has to be effected by a low power computing device is precisely the most complex task. Moreover, we can expect that the key length of the public key will increase with the progress of hardware technology. In addition, there are no guarantees that new sub-exponential attacks for the basic number theoretic problem will not be suddenly proposed. Therefore it would be better to have a more efficient public key cryptosystem.

As an alternative, we might use a new public-key cryptosystem constructed over number fields [18]. The cryptosystem has a theoretically fast decryption process such as a quadratic decryption complexity of bit-length of a public-key, which consists of only simple fast arithmetical operations. So even if the key length gets bigger in the future, there will be no great increase of the computational complexity. This becomes even more important when thinking of smart cards. In this paper, we call the new cryptosystem NICE, (New Ideal Coset Encryption). We focus on the practical aspects of NICE cryptosystem. We implement the NICE cryptosystem over different architectures, namely software on a standard PC and on a smartcard designed for the RSA cryptosystem. Our implementation in software shows NICE is as fast as the encryption time of the RSA cryptosystem with $e = 2^{16} + 1$. Implementation on a smartcard designed for the RSA cryptosystem is comparably as fast as the decryption time of the RSA cryptosystem but not so fast as in software implementation. We discuss the reasons for this and indicate requirements to achieve fast implementation on smartcards.

This paper is organized as follows: In section 2, we give several applications based on NICE cryptosystem. In section 3, we explain the details of the algorithms of the NICE cryptosystem. In section 4, we show timings of the implementation in software. In section 5, we discuss a smart card implementation and its problems.

2 Applications

In section 3, we will present NICE in the formulation of an encryption scheme. An immediate application is therefore session key distribution from a powerful server

to a device which has a limited computing power or where time is important. An example for such a device is e.g. a mobile phone.

Another application of NICE is the use as an authentication scheme. The usual protocols (2-way and 3-way) can be adapted to use NICE as encryption component. Of course, it could be combined with RSA, where the modulus is the absolute value of the discriminant of the non-maximal order, i.e. Δ_q. In that case, the 3-way protocol can be realized in such a way that the client (= the low computing power device) only effects the fast components of both algorithms.

As a last application, we propose an undeniable signature scheme. NICE itself cannot be used as a classical signature scheme; undeniable signature schemes have their own use e.g. in online transactions. A signature of this kind cannot be verified without the interaction of the signer. The standard example for its application is its use by a software development company: the distributed software is signed by means of an undeniable signature of the company to allow legal users to ensure themselves that they use unmodified software. Since interaction with the seller is needed to check the signature, illegal users either cannot check and risk to use some virus-infected software or will be traced by the software-seller as soon as they ask for interactive verification. Details can be found in [4]. Again, the low computing power device only effectuates the NICE decryption steps, so smart cards can be used for assuring the security of online transactions.

3 The NICE Cryptosystem

In this section, we present an overview of the NICE cryptosystem. Details can be found in [18]. The idea of NICE is roughly as follows: consider two finite abelian groups G and H which are related by a surjective map $\pi : G \to H$. Moreover, there exists a well-defined bijective mapping of sets $\phi : H \to U$ of H onto a subset of U of G such that $\pi(\phi(M))) = M$ for all $M \in H$. The representation of elements of G and the group operation algorithm of G are publicly known, as well as an element h of the kernel of π. U is chosen such that a consecutive set of representations of elements of G are representations of elements of U. This information is publicly known. Assume that you know the group H (i.e. representation of group elements and group operation) and how to compute π, but no one else does. The message space consists of the publicly known elements of U. Now, a message m is probabilistically encrypted by randomly multiplying an element h^r of Kerπ onto it: the ciphertext is $c = m * h^r$. Decryption simply works as follows: compute $\phi(\pi(m * h^r))$.

This is a secure cryptosystem if the computation of the map π cannot be deduced from the given information, namely the group G, the kernel element h and the test for U. There exist some constructions of this scheme using number theoretic problems, e.g. [17]. An overview can be found in [19].

The following implementation of this scheme is especially interesting: Generate two random primes $p, q > 4$ such that $p \equiv 3 \pmod 4$ and let $\Delta_1 = -p$. Let $H = Cl(\Delta_1)$ be the ideal class group of the maximal order with discriminant Δ_1 and $G = Cl(\Delta_q)$ be the ideal class group of the non-maximal order with

conductor q. Δ_q will be public, whilst its factorization into Δ_1 and q will be kept private.

Any element of G is given by a pair of numbers (a, b) such that $0 < a \leq \sqrt{|\Delta_q|/3}$, $-a < b \leq a$ and $b^2 \equiv \Delta_q \mathrm{mod} 4a$ (and some other minor requirements). Elements of H are represented in the same way with Δ_1 instead of Δ_q. The group operation works as follows:

Composition in $\mathrm{Cl}(\Delta_q)$
Input: $(a_1, b_1), (a_2, b_2) \in \mathrm{Cl}(\Delta_q)$, the discriminant Δ_q
Output: $(a, b) = (a_1, b_1) * (a_2, b_2)$.

1. */* Multiplication step */*
   1.1. Solve $d = ua_1 + va_2 + w(b_1 + b_2)/2$ for $d, u, v, w \in \mathbb{Z}$ using the extended Euclidean algorithm
   1.2. $a \leftarrow a_1 a_2/d^2$
   1.3. $b \leftarrow b_2 + (va_2(b_1 - b_2) + w(\Delta_q - b_2^2)/2)/d \mathrm{mod} 2a$
2. */* Reduction step */*
   2.1. $c \leftarrow (\Delta_q - b^2)/4a$
   2.2. WHILE $\{-a < b \leq a < c\}$ or $\{0 \leq b \leq a = c\}$ DO
      2.2.1. Find $\lambda, \mu \in \mathbb{Z}$ s.t. $-a \leq \mu = b + 2\lambda a < a$ using division with remainder
      2.2.2. $(a, b, c) \leftarrow (c - \lambda \frac{b+\mu}{2}, -\mu, a)$
   2.3. IF $a = c$ AND $b < 0$ THEN $b \leftarrow -b$
   2.4. RETURN (a, b)

This algorithm has quadratic bit complexity $O((\log_2 \Delta_q)^2)$ and is only needed for the encryption step. For the decryption step, we need only to compute π. The computation of the map π works as follows:

Computation of π
Input: $(a, b) \in Cl_{\Delta_q}$, the fundamental discriminant Δ_1, the discriminant Δ_q and the conductor q
Output: $(A, B) = \pi((a, b))$.

1. $b_{\mathcal{O}} \leftarrow \Delta_q \bmod 2$
2. Solve $1 = uq + va$ for $u, v \in \mathbb{Z}$ using the extended Euclidean algorithm
3. $B \leftarrow bu + ab_{\mathcal{O}} v \bmod 2a$
4. $C \leftarrow (\Delta_1 - B^2)/4A$
5. WHILE NOT $(\{-A < B \leq A < C\}$ or $\{0 \leq B \leq A = C\})$ DO
   5.1 Find $\lambda, \mu \in \mathbb{Z}$ s.t. $-A \leq \mu = B + 2\lambda A < A$ using division with remainder
   5.2 $(A, B, C) \leftarrow (C - \lambda \frac{B+\mu}{2}, -\mu, A)$
6. IF $A = C$ AND $B < 0$ THEN $B \leftarrow -B$
7. RETURN (A, B)

This algorithm π has quadratic bit complexity $O((\log_2 \Delta_q)^2)$ ([18]). Moreover, only simple well-known operations are needed, thus this algorithm can easily be implemented on an existing smart card.

Finally, the test belonging to U is simply whether for an element (a, b) a is smaller than $[\sqrt{|\Delta_1|/4}]$ (or a lower bound thereof of the form 2^k). The computation of the map ϕ will not be needed in that implementation.

We explain how the NICE encrytion scheme works: In the key generation, we choose an element (h, b_h) from the kernel Kerπ and make (h, b_h) public. The message m is embedded in an element (m, b_m) of $Cl(\Delta_q)$ with m smaller than $[\sqrt{|\Delta_1|/4}]$. Encryption is done in the class group $Cl(\Delta_q)$ by computing $(c, b_c) = ((m, b_m) * (h, b_h)^r)$, where r is a random integer smaller than 2^l with $l = \log_2\left(q - \left(\frac{\Delta_1}{q}\right)\right)$. Then, having the secret information, namely the knowledge of the conductor q, one can go to the maximal order and the image of the message (m, b_m) in the maximal order is revealed, since $\pi(c, b_c) = \pi(m, b_m)$ and m can be recovered without computing ϕ.

The NICE encryption protocol

1. **Key generation:** Generate two random primes $p, q > 4$ with $p \equiv 3 \pmod 4$ and $\sqrt{p/4} < q$. Let $\Delta_1 = -p$ and $\Delta_q = \Delta_1 q^2$. Let k and l be the bit lengths of $[\sqrt{|\Delta_1|/4}]$ and $q - \left(\frac{\Delta_1}{q}\right)$ respectively. Choose an element (h, b_h) in $Cl(\Delta_q)$, where
$$\pi((h, b_h)) = (1, 0) \tag{1}$$
Then $((h, b_h), \Delta_q, k, l)$ are the system parameters, and q is the secret key.
2. **Encryption:** Let (m, b_m) be the plaintext, in $Cl(\Delta_q)$ with $\log_2 m < k$. Pick up a random $l - 1$ bit integer and we encrypt the plaintext as follows using binary exponentiation and precomputation techniques:
$$(c, b_c) = (m, b_m) * (h, b_h)^r \tag{2}$$
Then (c, b_c) is the ciphertext.
3. **Decryption:** Using the secret key q, we compute $(d, b_d) = \pi((c, b_c))$. The plaintext is then $m = d$.

A message embedding technique and security aspects of this cryptosystem, we refer to [18]. Again, please note that this cryptosystem can easily be implemented using well-known techniques and existing smart cards. This will be shown in the next section.

4 NICE Running Times in Software

The prominent property of the proposed cryptosystem is the running time of the decryption. Most prominent cryptosystems require decryption time $O((\log_2 n)^3)$, where n is the size of the public key. The total running time of the decryption process of our cryptosystem is $O((\log_2 \Delta_q)^2)$ bit operations. In order to demonstrate the improved efficiency of our decryption, we implemented our scheme using the LiDIA library [2]. It should be emphasized here that our implementation was not optimized for cryptographic purposes — it is only intended to

provide a comparison between RSA and NICE. The results are shown in table 1. In these tests, we did choose $p \approx q \approx n^{1/3}$, so that breaking RSA and NICE by factoring is approximately equally hard. Other variations and a discussion of the variants can be found in [18].

Table 1. Average timings for the new cryptosystem with 80-bit encryption exponent r compared to RSA with encryption exponent $2^{16}+1$ over 100 randomly chosen pairs of primes of the specified size on a Pentium 266 Mhz using the LiDIA library

$\log_2(\Delta_q)$	1024	1536	2048	3072
RSA encryption	2.2 ms	4.8 ms	7.5 ms	15.2 ms
RSA classical decryption	259.2 ms	751.3 ms	1643.9 ms	4975.6 ms
RSA decryption with CRT	110.5 ms	291.7 ms	629.7 ms	1855.5 ms
NICE encryption	602.9 ms	1180.1 ms	1902.0 ms	3933.5 ms
NICE decryption	3.8 ms	6.2 ms	10.0 ms	19.3 ms

Observe that one can separate the fast exponentiation step of the encryption as a "precomputation" stage. Indeed, if we can securely store the values $(p, b_p)^r$, then the actual encryption can be effected very rapidly, since it requires only one ideal multiplication and one ideal reduction. Moreover, using well-known techniques for randomized encryption, we can even reduce the encryption time much more. Note that no square root technique like the Pollard-rho method or Shanks' algorithm are directly applicable to the ciphertext (c, b_c), because the encryption consists of $(c, b_c) = (m, b_m)(p, b_p)^r$ where r is a random exponent and (m, b_m) is the secret plaintext. This means that we can use a very short random exponent r having e.g. about 80 bits.

It should be mentioned that the size of a message for our cryptosystem is significantly smaller than the size of a message for the RSA encryption (e.g. 256 bit vs. 768 bit, or 341 bit vs. 1024 bit). In connection with the very fast decryption time, an excellent purpose for our cryptosystem could be (symmetric) key distribution. In that setting, the short message length is not a real drawback. On the other hand, the message length is longer than for ElGamal encryption on "comparably" secure elliptic curves (e.g. 341 bit vs. 180 bit).

Table 2. Rate of the speed increasing when the bit-length of a public-key becomes larger

$\log_2(n)$	1024	1536	2048	3072
RSA encryption	1	2.18	3.41	6.91
RSA classical decryption	1	2.90	6.34	19.20
RSA decryption with CRT	1	2.64	5.70	16.79
NICE encryption	1	1.96	3.15	6.52
NICE decryption	1	1.63	2.63	5.08

Note that even if a public-key becomes large, the rate at which the speed of decryption of NICE increases is not so large as that of the RSA cryptosystem. This shows the effectiveness of a quadratic decryption time of NICE cryptosystem. The ratio is given in table 2.

5 A Smart Card Implementation and Its Problems

Moreover, we implemented the NICE decryption on a smart card. More precisely, we implemented the NICE decryption algorithm using the Siemens development kit for chip card controller ICs based on Keil PK51 for Windows. As assembler we used A51, as linker L51 to generate code for the 8051 microcontroller family. The software simulation were made using dScope-51 for Windows and the drivers for SLE 66CX160S. Thus, we realized a software emulation of an assembler implementation of the NICE decryption algorithm to be run on the existing Siemens SLE 66CX160S. Unfortunately, the timings of this software simulation were unrealistic. So, we did run some timings on a hardware simulator for SLE 66CX160S. Thanks to Deutsche Telekom AG, Produktzentrum Telesec in Siegen and Infineon/Siemens in Munich for letting us use their hardware simulator. See the timings for decryption in table 3 for a smart card running at 4.915 MHz.

Table 3. Timings for the decryption of the new cryptosystem compared to RSA using the hardware simulator of Siemens 66CX160S at 4.915 MHz

$\log_2(n)$	1024
RSA decryption with CRT	490 ms
New CS decryption for $p \approx q \approx \Delta_q^{1/3}$	1242 ms
Improved version	1035 ms

The very first implementation was very inefficient; the straightforward algorithms used in the software comparison proved to be much slower on the smart card than the existing RSA on the card. This was surprising, but after a while this could be easily explained: the cryptographic coprocessor has been optimized for modular exponentiation. On the other side, NICE uses mostly divisions with remainder and comparisons. These operations are slow on the coprocessor, so we had to modify the decryption algorithm to speed it up in hardware. We describe here two significant changes:

The computation of the inverse of a modulo q (step 3 in the computation of π) using the extended Euclidean algorithm took (with $q \approx p \approx 341$ bit) about 9 seconds (!), whereas computing the inverse using Fermat's little theorem - by using fast exponentiation mod q - took less than 1 second. Note that the decryption time using this method is no longer of quadratic complexity.

In the reduction process the quotient in the division with remainder step is most of the time very small (say ≤ 10, see Appendix A); to effect a division is

this case is much more time consuming than subsequent subtractions. We did replace reduction step 5

5.1 Find $\lambda, \mu \in \mathbb{Z}$ s.t. $-A \leq \mu = B + 2\lambda A < A$ using division with remainder
5.2 $(A, B, C) \leftarrow (C - \lambda\frac{B+\mu}{2}, -\mu, A)$

by the following algorithm.

5.1 WHILE $B \leq -A$ OR $B > A$ DO
  5.1.1 IF $B < 0$ THEN $B \leftarrow B + 2A$; $C \leftarrow C - (B + A)$;
  5.1.2 ELSE $B \leftarrow B - 2A$; $C \leftarrow C - (B - A)$;

Every time that the bitlength of B was exceeding the bitlength of variant B by at least 3. Using this improvement, we could decrease the running time from about 1.8 s to 1.2 s. This is already faster than a 1024 bit RSA decryption without Chinese remainder theorem (approx. 1.6 ms).

The timings in table 3 were made including these two improvements. Moreover, a detailed timing analysis in Siegen showed that both our static memory management and as well as the cryptographic coprocessor are not optimal for this algorithm. We discuss this in the sequel. An average overview of the most time consuming parts is given in table 4.

Table 4. Detailed timings for different functions in the decryption of the new cryptosystem

Function	Average time over the whole computation
`mul`, multiplications on the coprocessor	170 ms
`div`, divisions on the coprocessor	231 ms
`left_adjust`, length correction of the variables	376 ms
`C2XL`, moving numbers into the coprocessor	118 ms
`XL2C`, moving numbers out of the coprocessor	223 ms
others (comparisons, small operations)	114 ms
Overall time	1242 ms

One major difference between RSA and NICE is the number of variables needed during the computation of the decryption algorithm. In our implementation, we need to store 11 variables of length at most 2048 bit. Computations of the cryptographic coprocessor are shortening these variables. Thus, we had to adjust the length of the variables after each important operation. This was done by moving the top nonzero bytes of the number to the fixed address of the number and so "erasing" leading zero bytes. To do this, we used the cryptographic coprocessor. Now the exact timings showed that about 33 % of the running time is spent by the function `left_adjust`, which effects this correction.

Now changing the memory management from static to dynamic (i.e. in the `XL2C` and `C2XL` functions making the appropriate changes and having additionally some registers holding the starting address of the numbers), we got an

improvement to **637 ms** using the software simulator. Both Infineon/Siemens and Deutsche Telekom reported the overall time of the hardware simulator now to be **1035 ms**. Note that the amount of memory required is **965 Bytes** and thus fits into a real SLE 66CX160S.

As one can see from table 4, another important time consuming operation is to move numbers into and out of the coprocessor. At this point, we would get a speedup of about 350 ms if we could leave the numbers in registers inside the coprocessor. It is clear that the currently used processor is not prepared for such operations, since it is optimized for RSA, thus operations with very few variables. At this point we ask the hardware community to present solutions to this problem.

6 Conclusion and Acknowledgements

The NICE cryptosystem is fast and well suited for software implementation. To get an equally fast speedup compared to RSA on a smart card, we think that the underlying hardware must be developed adequately. Nevertheless, if this is done, the think that NICE can be a competitor to RSA whenever fast decryption is needed.

We thank Deutsche Telekom AG, Produktzentrum Telesec for letting us testing NICE on the hardware simulator and Siemens AG/Infineon GmbH for their valuable help concerning the use of the development kit as well as running our code on their hardware simulator.

References

1. L. M. Adleman and K. S. McCurley, "Open problems in number theoretic complexity, II" proceedings of ANTS-I, LNCS 877, (1994), pp.291-322.
2. I. Biehl, J. Buchmann, and T. Papanikolaou. *LiDIA - A library for computational number theory.* The LiDIA Group, Universität des Saarlandes, Saarbrücken, Germany, 1995.
3. I. Biehl and J. Buchmann; "An analysis of the reduction algorithms for binary quadratic forms," Technical Report No. TI-26/97, Technische Universität Darmstadt, (1997).
4. I. Biehl and S. Paulus and T. Takagi, "Efficient Undeniable Signature Schemes based on Ideal Arithmetic in Quadratic Orders," in preparation.
5. J. Buchmann and H. C. Williams; "A key-exchange system based on imaginary quadratic fields," Journal of Cryptology, 1, (1988), pp.107-118.
6. J. Buchmann and H. C. Williams; "Quadratic fields and cryptography," London Math. Soc. Lecture Note Series 154, (1990), pp.9-26.
7. J. Buchmann, S. Düllmann, and H. C. Williams. On the complexity and efficiency of a new key exchange system. In *Advances in Cryptology - EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, pages 597–616, 1990.
8. J. Cowie, B. Dodson, R. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery, J. Zayer; "A world wide number field sieve factoring record: on to 512 bits," Advances in Cryptology – ASIACRYPT '96, LNCS 1163, (1996), pp.382-394.

9. D. A. Cox: *Primes of the form* $x^2 + ny^2$, John Wiley & Sons, New York, 1989
10. W. Diffie and M. Hellman, "New direction in cryptography," IEEE Transactions on Information Theory, 22, (1976), pp.472-492.
11. ECMNET Project; `http://www.loria.fr/~zimmerma/records/ecmnet.html`
12. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithm in $GF(p)$," IEEE Transactions on Information Theory, 31, (1985), pp.469-472.
13. D. Hühnlein, M. J. Jacobson, Jr., S. Paulus, and T. Takagi; "A cryptosystem based on non-maximal imaginary quadratic orders with fast decryption," Advances in Cryptology – EUROCRYPT '98, LNCS 1403, (1998), pp.294-307.
14. H. W. Lenstra, Jr., "Factoring integers with elliptic curves", Annals of Mathematics, 126, (1987), pp.649-673.
15. A. K. Lenstra and H. W. Lenstra, Jr. (Eds.), *The development of the number field sieve*. Lecture Notes in Mathematics, 1554, Springer, (1991).
16. A. J. Menezes, P. van Oorschot, S. Vastone: *Handbook of applied crytpography*. CRC Press, 1996.
17. T. Okamoto and S. Uchiyama, "A new public-key cryptosystem as secure as factoring," Advances in Cryptology – EUROCRYPT '98, LNCS 1403, (1998), pp.308-318.
18. S. Paulus and T. Takagi, "A new public-key cryptosystem over the quadratic order with quadratic decryption time," to appear in Journal of Cryptology.
19. S. Paulus and T. Takagi, "A generalization of the Diffie-Hellman problem based on the coset problem allowing fast decryption ," Proceedings of ICISC'98, Seoul, Korea, 1998.
20. R. Peralta and E. Okamoto, "Faster factoring of integers of a special form," IEICE Trans. Fundamentals, Vol.E79-A, No.4, (1996), pp.489-493.
21. R. J. Schoof: *Quadratic Fields and Factorization*. In: H.W. Lenstra, R. Tijdeman, (eds.): *Computational Methods in Number Theory*. Math. Centrum Tracts **155**. Part II. Amsterdam, 1983. pp. 235-286.

A An Example of the Reduction Step

Input: discriminant Δ_1, an ideal $\mathfrak{A} = (A, B)$ before the reduction step

```
D1 = -391955329881115736847552399099477748671287962016063668607498092668663577456528165441653561322531245266 3
A  =  14709196987236217470310344790336554582213450995580439354777614852261315535437072993917565044745312551892647190947573
      65980819130165941950615573544141741449 1
B  = -16120786611173332936175433825318881027235314017722179961013037966005831645873381632911088594573172128409548203242276
      66120696958777850232980376265572115021 3
```

Reduction step 5.1.: quotient λ, remainder μ such that $B = 2A\lambda + \mu$

```
(quotient, remainder) =
(-1,   13297607363299102004445255755354228137191587973438698748542191738516799425000764354924041494917452975375146178652876584094130155403366825077082271136787 69)
(-2,  -12761243590327174728420394096256304728831977979402420516873134568804685699281859725103664466875812641319914710949015928355646393488678950389641306963210 5)
(5,   -78528290011262457388456843941329248503454444407474349379378311176000683515304069232120458471520185971710394090308261868661757907073551937411599271404065)
(3,    10244818933510891787370810612951117737019488809536816914420866112436574171352321538119941956085359074784832443404744032975209160972759289660851787546 69)
(-2,  -10224129845590147501382365118494266252021893092244629712061058909068680852197387614386894226954222864793077231352357416590511833810655070898694329260 5)
(5,   -11105296432786254324801445410868102997262161184053297146125015494247765187360954811606805469093183807780471092809093473562855601861999949481990650315)
(2,    22472176875075749176609304144458704509990853674790256950472930566758992023189046610292013458995337873101127300307126536968520513812928743328161111 51)
(-24, -19399768262238616074750925620345945700130240074364394015776939186812798812752098861599659305585438681964815901740646853690299529528949871546723550 3)
(5,   -87214024755816152729750202158525445031303615368933392297827179212874361504535183941307866086816850522516735441849 1)
```

910488305041328185564437639 27)
(5, -6589256154220428348763484804804029082058570540526813862090115099621050606775639083942326160598304048110107273013593512069684208466784180774933)
(3, -3537254096503620593414134057544050452333307165981859271665819380718315946782301163388529333399300671894655640007692872338965796469950209889815)
(7, -2472222842945937271424896215402386502429325414491606332343370688814610711133379933847844061543273235463853475378203764748268929369094978771111)
(2, 2112389608236713424619032162896477405216890407360964289710918331154798745302770489601942799572676509170820009529975660819814689202686945135)
(-5, -1388284996108376306829982546506800877297853993441043568602812670369274619799575723119384100819083300454158126117796116538719307405505)
(3, -7658114509885474231440468833038946684623106412514495155907595574141977825664772294590168889402849912338147698286083138580548663993732859)
(6, 4517159759733376362399236556227045130183695609000864138692049628351526155596718588532429655796773572510798393925384463045341798439293 1)
(-3, 5794939929115365995173503218660834384356867425629734633724798264593924802267798569392261604601174789678479246717539296859648971048121 3)
(-3, 1350677525258851054839945596647219385591771756805416894334501304553049196927373740657200062584045458093714655933189094973877882883289)
(-15, 3371933193122954317459601666464315138791547976559206823565738527321556044311098226988811614870540130943134313822596913769977980851)
(-3, 3238784738313438459689607455120743637545067827608086766817740807231823823229426993321003368051506422845515923690442961002663734 1)
(-4, 1671441973814644319675282257387185423140516601720995276908313731235903162398085717823307898310219398918074555980967325428760202 43)
(-5, -3093900115975595931597610737455819706682001946354992923221164365960676797968302039424049370634009318226407024312653906939082883)
(11, -1153106858058216736045054364752637655451729341586832024627696861235896647014417982287287152417816575504218178089201218624241 93)
(3, -2141647788492943418138863192402374136126459880981687307633113383031860259633208806973393304746388429069572783831215217010945 3)
(2, 1288852359116033410613791243994164241591905355073735634247165392805618377946014905015701508681255525992478185310146905642061)
(-8, 3163357815943921306495379915269786320411592267672643410223358096222549433470096155720769765128564997634716268578637456771)
(-5, -1321001889705188081349892737410199533959581699250834689584960915480292822075788101381291893508778863463935154716199449891)
(5, -1182076788551730602734619766770475116661074953562382333258507467774484339528870419622234444687351217744611329878842257829)
(2, 2111927946171327759620701789350226653831314331196303085750441510781171013182825382234563175842409351590856581943897569 7)
(-2, -2743285305114172425541110540802561105720020141938918426600397745755784059219379623215801476055360582029554774547226713)
(3, 2860748158891985401474429713136592782274727989378104287097773627056634739169980867676818174655427818291322493595179 75)
(-3, 1367532997912052465443644868991541513477548820779851985467270422519783676891382207571396439942634426120110281740330 9)
(-7, -5733925658197840890157616477656092654752132021299440670896385313615178397931288947596754116299993867692880971575736)
(3, 4642014124720289963275873549242830849165566501873785964876948824665393672719043131032957884408990381414512601481 1)
(-4, 2648752977947845396817832353867014934646503864836612394943244361752704526516616369491852849332139457426460748557 3)
(-5, 2108559245863855227022919968753755644328349091487882633136164649958232365711910457701858011575588866966405003 7)
(-3, 2235894385477513774702853297152645208875741821470379502651453995593835452771449594075431944337959109115485481 1)
(-3, -3066428918730050632325097660734163737851572230223614656315369265785635066318979992409078183621174963776878039)
(2, 1420865989874072755665282230298476831106299907933671932074879616218452312494744507627168783334527681703713 07)
(-10, -4013833186275051054261529002448266001854462865736283000730209262386311300176819714480584897953649188768987)
(3, 5133187827798145400999334148526572649747134934693745854533795436478351854566215608625239085628765532177 09)
(-2, -6167743688256320381536911317831344419466606317713806120336190796093010939774300327391806056220720881742 1)
(4, -5705343812063530411004779368065864956629980006638652578479437486951568472760468845862750562481954230324 3)
(3, -9155818781739316346966742047891535700816738572323029795737035526883331905337014079460712753247612705 3)
(21, 4538685336850697187842499653670888781357885670866161805360478300528997110425684029899747272018435 7)
(-10, 2096400963120895452091855125365379202225167263766964847877535223464862248383150866105858048576688 3)
(-2, -2452654164796263512959165563775971471231657933925013542908668110590277226101198472717238287967 71)
(4, -1470527936734615380865065306683575610942226478246315871086537382991040364168167464991647297381 65)
(4, 1396379881291112013027970946380117830029928835625361305509205527869349306688873128639700073563 7)
(-2, -2243059469029860341733582498093820347780557194768743665298129222007894230642566854465652585 85)
(3, -3325369847370702933567816339534141091828891003805965525320260390920031496638268930678355279 79)
(3, -5386967244970895407120103734398017221619479235338351726393439969384585546901687694925011245 1)
(2, 8809243662589899724441862302427489374063110417101583652380728682887905485821254651162082773)
(-3, 1523916850396224929683411116520822483473365693967880645688939842558533917993725763450387843)
(-2, -1754489364392193514749431904081409848801520964881650742860018735768133417632767916325280 51)
(4, -7172119525392604758586912757900411673673729560882300073557646023277852673433205207722 85)
(4, 4119835862177913257337980256296728495137802867857784860484730448194124783997117493312 93)
(-3, 3995237825404633164098524535985314848043967467204685269347794088687337119079150377955 5)
(-4, 2430418964222694480671212916136321892771426164807894584775183594905868264571397226917)
(-4, -2004816389506792126513706962375499698367870035307726680846526580047822967554312546 61)
(3, -1580903667060605871996966226809045877298060404628075628715566840184055884802721257)
(5, -1523585718315717333817603063481778938222355282742962613553158680390832265726743003)
(2, 1843114936956169477779561058082915459597712000852831017060382904728609896638743 91)
(-4, 1860872135112964248738280512893474954558440050772629678474760840270585563475300 5)
(-3, 1869347445173273085208023229014781878050128548111311960144791632389450110085991)
(-4, 1783807300968834504033504909171901616894208529518268605707305999380528892537777)
(8, 8087485289366444463103610663066729692803032028198886352664396077222417226 51)
(-8, 3309898705186043303215014982616604211164204855720333399793893848761439618 61)
(-3, -1866453661596902048360161861366941356077297843206608429615574493135596 03)
(6, -1412504875497260093376398626643530047287435788085760178403662982434702419)
(2, 2199964927348448841457853722934605034894275587147913593955307296446379 5)
(-3, 2351604360566553921011925424106798286299309046100721528085455153587032 7)
(-3, -3077511624656829785194435187496372328805308338329375129763801878536 03)
(2, 2971521129174281864734874025301780049383610478829534623568875299744 35)
(-5, 2043047968784113838865178649033621466031951358193306410738753843972 5)

```
(-3,  -715710522544461579764349638671583037687849172262961920648424641623)
(9,    1417951347087597514948488495652560626163289875991704192430469253 1)
(-5,  -128252533327126479297702878036187960221874533930004646526839929 1)
(2,    7133580613529858737331990335086812687125553016512866750439935)
(-89, -31402743133518053709238680774128360213498381554795257802491)
(3,   -567001434728009782779912454379629768945775967984631840880 1)
(2,    4668843591720874170487843007845306559025103686844156105 49)
(-6,   3584612499799271885192020009484500568653151822992857983 5)
(-2,  -517494697911868646394916535930577472302841252187015015 1)
(3,   -638632432529495911868923554773736141329464245095651 25)
(27,   7425935781990473220367357982454273737732519281514 97)
(0,   -7425935781990473220367357982454273737732519281514 97)
```

Output: the reduced ideal $(A1, B1)$ equivalent to $\mathfrak{A}$

```
A1 = 9562398414327226521335535763343291861775258207781 49
B1 = -7425935781990473220367357982454273737732519281514 97
```