Optimization of the MOVA Undeniable Signature Scheme

Jean Monnerat[1⋆], Yvonne Anne Oswald[2], and Serge Vaudenay[1]

[1] EPFL, Switzerland
http://lasecwww.epfl.ch

[2] ETH Zürich, Switzerland

Abstract. This article presents optimization results on the recent MOVA undeniable signature scheme presented by Monnerat and Vaudenay at PKC '04 as well as its generalization proposed at Asiacrypt '04 which is based on a secret group homomorphism. The original MOVA scheme uses characters on $\mathbf{Z}_n^*$ and some additional candidates homomorphisms were proposed with its generalization. We give an overview of the expected performances of the MOVA scheme depending on the group homomorphism. Our optimizations focus on the quartic residue symbol and an homomorphism based on the computation of a discrete logarithm in a hidden subgroup of $\mathbf{Z}_n^*$. We demonstrate that the latter provides a signature generation which is three times faster than RSA.

Key words: Undeniable signatures, optimization.

1 Introduction

Undeniable signatures, which have been introduced by Chaum and van Antwerpen in [1], differ from classical digital signatures in the verification process. Contrary to classical digital signatures, where anyone holding the public key of the signer is able to verify whether a given signature is valid or not, one has to interact with the signer to be convinced of the validity or the invalidity of the signature. This interaction enables the signer to control the distribution of the signature verification. An undeniable signature scheme therefore consists of a key setup algorithm and a signature generation algorithm, as well as an interactive verification protocol. This protocol is composed of a confirmation and a denial protocol which allow to prove the validity resp. the invalidity of the signature.

In March 2004, a new undeniable signature scheme called MOVA was proposed by Monnerat and Vaudenay [12]. More recently, the same authors generalized this scheme to the more general framework of group homomorphisms [13]. Namely, the MOVA scheme can be seen as the particular case where the underlying homomorphism is a character on $\mathbf{Z}_n^*$. When the choice of the homomorphism

⋆ Supported by a grant of the Swiss National Science Foundation, 200021-101453/1.

is adequate (as for MOVA), this signature scheme allows signatures to be arbitrarily short (typically around 20–30 bits), depending on the required security level.

The goal of this paper is to optimize the signature generation algorithm of the generalized scheme based on group homomorphisms and to present a comparison of the signature generation efficiency between the group homomorphisms considered as potential candidates. In particular, we focus on the optimization of characters of order 4 which requires to deal with algorithms computing the quartic residue symbol. Moreover, one quartic residue symbol variant is of particular interest since it is the only homomorphism presenting the special property of having two levels of secret. We propose an application of this property where a delegate of a company needs to sign some pre-agreement of a transaction which will be finalized later by the company using an additional level of secret. We also analyze the case of a homomorphism proposed in [13] consisting of sending elements of $\mathbf{Z}_n^*$ to a cyclic subgroup followed by the computation of a discrete logarithm. We give details on an implementation using a precomputed table of discrete logarithms. A comparison with practical parameters (e.g., a modulus n of 1024 bits) with the Jacobi symbol as well as RSA using standard efficient methods is presented at the end of this article. Our implementations are done in C using the large numbers library GMP [6].

2 The MOVA Scheme

For the sake of simplicity, the generalized scheme [13] will be called MOVA as well. Below we review the main ideas and the signature generation algorithm of this undeniable signature scheme.

First, let us recall some basic definitions from [13] related to the interpolation of group homomorphisms.

Definition 1. *Let G and H be two Abelian groups.*

1. *Given $S := \{(x_1, y_1), \ldots, (x_s, y_s)\} \subseteq G \times H$, we say that the set of points S* interpolates in a group homomorphism *if there exists a group homomorphism $f : G \longrightarrow H$ such that $f(x_i) = y_i$ for $i = 1, \ldots, s$.*
2. *We say that a set of points $B \subseteq G \times H$* interpolates in a group homomorphism with another set of points $A \subseteq G \times H$ *if $A \cup B$ interpolates in a group homomorphism.*

The central idea of the generalized MOVA scheme is to consider a secret group homomorphism Hom between two publicly known Abelian groups Xgroup and Ygroup as the signer's secret key. The order of the group Ygroup is public and is denoted as d. The signer then chooses a set Skey $\subseteq$ Xgroup $\times$ Ygroup of Lkey points such that Skey interpolates in a unique homomorphism, namely Hom. The signer chooses Skey $:= \{(\mathrm{Xkey}_1, \mathrm{Ykey}_1), \ldots, (\mathrm{Xkey}_{\mathrm{Lkey}}, \mathrm{Ykey}_{\mathrm{Lkey}})\}$ in varying ways depending on the choice of one of the setup variants presented in [13]. The size of the parameter Lkey depends on the setup variant choice too. Then, to

sign a given message m the signer computes Lsig values $\mathrm{Xsig}_1, \ldots, \mathrm{Xsig}_{\mathrm{Lsig}} \in$ Xgroup from m by using a random oracle and computes $\mathrm{Hom}(\mathrm{Xsig}_i) := \mathrm{Ysig}_i$ for $1 \leq i \leq \mathrm{Lsig}$. Finally, the signature of m with respect to the secret key Hom is

$$\sigma := (\mathrm{Ysig}_1, \ldots, \mathrm{Ysig}_{\mathrm{Lsig}}).$$

In the verification step, the verifier will first send the message-signature pair (m, σ) he would like to verify. If the pair is valid, the signer launches a confirmation protocol with the verifier in which he proves that the set of points $\{(\mathrm{Xkey}_1, \mathrm{Ykey}_1), \ldots, (\mathrm{Xkey}_{\mathrm{Lkey}}, \mathrm{Ykey}_{\mathrm{Lkey}})\}$ interpolates in a group homomorphism (namely Hom) with the set $\{(\mathrm{Xsig}_1, \mathrm{Ysig}_1), \ldots, (\mathrm{Xsig}_{\mathrm{Lsig}}, \mathrm{Ysig}_{\mathrm{Lsig}})\}$. Otherwise, the signer launches a denial protocol in which he proves that the set of points $\{(\mathrm{Xkey}_1, \mathrm{Ykey}_1), \ldots, (\mathrm{Xkey}_{\mathrm{Lkey}}, \mathrm{Ykey}_{\mathrm{Lkey}})\}$ does not interpolate in a group homomorphism with the set $\{(\mathrm{Xsig}_1, \mathrm{Ysig}_1), \ldots, (\mathrm{Xsig}_{\mathrm{Lsig}}, \mathrm{Ysig}_{\mathrm{Lsig}})\}$. More details about the confirmation and denial protocols can be found in [13].

We state the so-called "Group Homomorphism Interpolation Problem" introduced in [13].

> **S-GHI Problem** (Group Homomorphism Interpolation Problem)
> **Parameters:** two Abelian groups G and H, a set of s points $S \subseteq G \times H$.
> **Input:** $x \in G$.
> **Problem:** find $y \in H$ such that (x, y) interpolates with S in a group homomorphism.

It is shown in [13] that the resistance of this scheme against existential forgery under a chosen-message attack relies on the hardness of the GHI problem with parameters $G =$ Xgroup, $H =$ Ygroup and $S =$ Skey. Hence, for a homomorphism (more formally a family of homomorphisms) for which the Skey-GHI problem is hard, we can assume that there is no easier method to forge a signature than performing an exhaustive search. Furthermore, if the homomorphism is such that it is hard to find any information bit on y in the Skey-GHI problem, the security level against an existential forgery attack depends exactly on the signature size which is $\mathrm{Lsig} \cdot \log_2(d)$.

3 Homomorphisms

In this section, we briefly describe some instances of the group homomorphism Hom considered in [13] such as characters on $\mathbf{Z}_n^*$ [12], the RSA encryption homomorphism [5, 14] or the discrete logarithm in a hidden subgroup [13].

3.1 Characters on $\mathbf{Z}_n^*$

Definition 2. *Let n be an integer. A character χ on $\mathbf{Z}_n^*$ is a group homomorphism from $\mathbf{Z}_n^*$ to $\mathbf{C} - \{0\}$ i.e.,*

$$\chi(ab) = \chi(a)\chi(b) \text{ for all } a, b \in \mathbf{Z}_n^*.$$

The characters on $\mathbf{Z}_n^*$ form a group with respect to the composition of functions. The order of a character χ is its order with respect to the group of characters. It is important to note that a character of order d maps any element to a dth root of the unity. In the MOVA scheme, the study focused on the characters of order 2, 3 and 4. In this article, we will not consider the case $d = 3$ since the algorithmic issues are similar to the case $d = 4$.

For more details about characters, we refer to the article on the MOVA scheme [12] and the textbook of Ireland and Rosen [7].

Jacobi Symbol We consider a public modulus $n = pq$ where p, q are two large secret primes. From the theory of characters, it directly follows that there exist exactly 4 characters of order 2 on $\mathbf{Z}_n^*$, namely the Jacobi symbols $(\cdot/n)_2$, $(\cdot/p)_2$, $(\cdot/q)_2$ and the trivial character. Note that the Jacobi symbol $(\cdot/n)_2$ and the trivial character are not suitable for our purpose since they can be efficiently computed without the knowledge of the factorization of n.

Quartic Characters The theory of the characters of order 4 naturally occurs in the context of Gaussian integers. We recall the required background related to our study. Most of these results are taken from [7].

The ring of the Gaussian integers is defined as

$$\mathbf{Z}[i] := \{a + bi \mid a, b \in \mathbf{Z}\}.$$

The norm of an element $\alpha = a + bi$ is defined as $N(\alpha) = \alpha \cdot \bar{\alpha} = a^2 + b^2$, where $\bar{\alpha}$ denotes the complex conjugate of α. $\mathbf{Z}[i]$ is well known to be Euclidean which implies that we can talk about the gcd of two Gaussian integers and there is an Euclidean division: given α, $\beta \in \mathbf{Z}[i]$ with $\beta \neq 0$, there exists γ, $\delta \in \mathbf{Z}[i]$ s.t. $\alpha = \gamma\beta + \delta$ and $N(\delta) < N(\beta)$. Note that γ and δ are not necessarily unique. The units (invertible elements) of $\mathbf{Z}[i]$ are ± 1, $\pm i$. We say that two elements α, β are associate if and only if $\alpha = u\beta$ for a unit u. The gcd of two Gaussian integers is uniquely defined up to an associate. Moreover, we say that two Gaussian integers α and β are relatively prime iff the only common divisors are units, i.e., their gcd is a unit. In this case we will use the notation $\gcd(\alpha, \beta) \sim 1$. Any prime element of $\mathbf{Z}[i]$ is of the following form or the associate of an element of this form:

1. $1 + i$
2. $q \equiv 3 \pmod 4$ a prime in $\mathbf{Z}$
3. π such that $N(\pi) \equiv 1 \pmod 4$ is a prime in $\mathbf{Z}$

Any Gaussian integer has a unique decomposition into primes up to a unit. For any prime $\sigma \in \mathbf{Z}[i]$, the quotient $\mathbf{Z}[i]/(\sigma)$ is a field with $N(\sigma)$ element. This allows to define the *quartic residue symbol*.

Definition 3. *Let α, $\beta \in \mathbf{Z}[i]$ be such that $(1 + i) \nmid \beta$ and $\gcd(\beta, \alpha) \sim 1$. The* quartic residue symbol *is defined as $\chi_\beta : \mathbf{Z}[i] \to \{\pm 1, \pm i\}$*

$$\chi_\beta(\alpha) = \begin{cases} u \text{ such that } \alpha^{\frac{N(\beta)-1}{4}} \equiv u \pmod \beta, u \in \{\pm 1, \pm i\}, & \text{if } \beta \text{ is prime} \\ \prod_i \chi_{\beta_i}(\alpha), & \text{if } \beta = \prod_i \beta_i,\ \beta_i \text{ prime} \end{cases}$$

The quartic residue symbols which are considered for MOVA [12] are chosen as follows. Let p, q be two rational primes such that $p \equiv q \equiv 1 \pmod 4$. There exist π, σ such that $p = \pi\bar{\pi}$ and $q = \sigma\bar{\sigma}$. π and σ can be computed with the help of the algorithms of Tonelli and Cornacchia (for more details see [3]). Then, we choose Hom $= \chi_\beta$ with $\beta = \pi$ or $\beta = \pi\sigma$. Moreover, we take Xgroup $:= \mathbf{Z}_n^*$ which is a natural choice since $\mathbf{Z}[i]/(\pi\sigma) \simeq \mathbf{Z}_n$ and $\mathbf{Z}[i]/(\pi) \simeq \mathbf{Z}_p$.

From the properties of the quartic residue symbol and the Jacobi symbol, we can show that $(\chi_{\pi\sigma}(a))^2 = (a/n)_2$ for any $a \in \mathbf{Z}$. Therefore, without the knowledge of the factorization of n we can easily deduce one bit of $\chi_{\pi\sigma}(a)$. In practice, we will compress this quartic residue symbol to one bit sending $1, i$ to the bit 0 and $-1, -i$ to the bit 1. To decompress, it suffices to compute the Jacobi symbol to retrieve the right quartic residue symbol. Hence, with this quartic residue symbol we have to perform two times more evaluations than with χ_π for the same level of security against an existential forgery. This shows that the signature generation will be anyway less efficient for $\chi_{\pi\sigma}$ than for χ_π.

A motivation for using $\chi_{\pi\sigma}$ is, that this character has two levels of secret, namely the secret key $\pi\sigma$ does not allow to factorize n. As mentioned in [13] an expert group knowledge of the group $\mathbf{Z}_n^*$ is required in order to convert a signature into an ordinary one. Here, this expert group knowledge corresponds to the ability to factorize. Hence, we can imagine an application where a mobile delegate of a company is able to sign some pre-agreement of some contracts or transactions using $\chi_{\pi\sigma}$ which can be confirmed by a server of the company. Later, the delegate sends a report to his company, which then can issue an ordinary signature for a final agreement by converting the signature of the delegate. In such a scenario, even if the delegate loses his key or it is stolen, he can contact his company before a confirmation of the signature is performed. In any case, the company never converts a signature before it is convinced that the delegate key was not lost or stolen.

3.2 RSA

Following the long tradition of the RSA based cryptography, an undeniable signature scheme based on RSA [14] was proposed in 1997 by Gennaro et al.[5]. This scheme can be seen as a special case of the generalized MOVA scheme when the homomorphism is the RSA encryption function defined on a modulus of safe primes. So, the signature is generated as for the regular RSA signature scheme.

3.3 Discrete Logarithm in a Hidden Subgroup

Another homomorphism suitable for the generalized MOVA scheme is based on the discrete logarithm in a hidden subgroup.

Let n be such that $n = pq$ with $p = rd + 1$, q, d prime, $\gcd(q - 1, d) = 1$, $\gcd(r, d) = 1$ and g generating a subgroup of $\mathbf{Z}_p^*$. We obtain g by choosing a random element $h \in \mathbf{Z}_n^*$ until h satisfies $h^r \bmod p \neq 1$ and we set $g = h^r \bmod p$. Like this we find a homomorphism by "sending" the input in a hidden cyclic

subgroup of order d and then computing its discrete logarithm with respect to the generator g,

$$\begin{aligned} \phi : \mathbf{Z}_n^* &\longrightarrow \mathbf{Z}_d \\ x &\longmapsto \log_g(x^r \bmod p). \end{aligned}$$

4 Quartic Residue Symbol

4.1 Background

We recall some properties of the quartic residue symbol which play a crucial role in the algorithms we will consider. To this end, we introduce the notion of "primarity".

We say that a Gaussian integer $\alpha = a + bi$ is primary if and only if either $a \equiv 1 \pmod 4$, $b \equiv 0 \pmod 4$ or $a \equiv 3 \pmod 4$, $b \equiv 2 \pmod 4$. It can be shown that for any nonunit $\alpha \in \mathbf{Z}[i]$ with $(1+i) \nmid \alpha$, there is a unique associate of α which is primary.

Theorem 4. *Let $\beta = a + bi$, α and α' be some Gaussian integers such that $\gcd(\beta, \alpha) \sim \gcd(\beta, \alpha') \sim 1$ and $(1+i) \nmid \beta$. The following properties hold.*

1. Modularity: *If $\alpha \equiv \alpha' \pmod \beta$ then $\chi_\beta(\alpha) = \chi_\beta(\alpha')$.*
2. Multiplicativity: $\chi_\beta(\alpha\alpha') = \chi_\beta(\alpha)\chi_\beta(\alpha')$.
3. Quartic Reciprocity Law: *If α and β are primary,*

$$\chi_\alpha(\beta) = \chi_\beta(\alpha) \cdot (-1)^{\frac{N(\alpha)-1}{4} \cdot \frac{N(\beta)-1}{4}}.$$

4. Complementary Reciprocity Laws: *If β is primary,*

$$\chi_\beta(i) = i^{\frac{N(\beta)-1}{4}} \text{ and } \chi_\beta(1+i) = i^{\frac{a-b-b^2-1}{4}}.$$

4.2 Basic Algorithm

Description To compute the quartic residue symbol $\chi_\beta(\alpha)$ directly, one has to know the factorization of β into primes over $\mathbf{Z}[i]$ and the computation contains an exponentiation. To avoid this factorization as well as the costly exponentiation we apply the properties of the quartic residue symbol iteratively. First we reduce α to an element $\hat{\alpha}$ equivalent to α modulo β and that satisfies $N(\hat{\alpha}) < N(\beta)$. From now on, such a reduction of an element α modulo β will be denoted $\mathrm{Red}_\beta(\alpha)$. Note that the obtained $\hat{\alpha} \leftarrow \mathrm{Red}_\beta(\alpha)$ is not necessarily unique. Then, we find the unique representation $\hat{\alpha} = i^j \cdot (1+i)^k \cdot \alpha'$ with α' primary and employ the multiplicativity property and the complementary laws of the quartic residue symbol. Next, we interchange α and β according to the law of reciprocity and start again. Hence, the size of both α and β decrease progressively. We stop the iteration process when α or β is a unit. The detailed algorithm is described in Algorithm 1.

Algorithm 1 Basic Algorithm Quartic Residuosity in $\mathbf{Z}[i]$

Require: $\alpha, \beta \in \mathbf{Z}[i] \setminus \{0\}$, $\gcd(\alpha, \beta) \sim 1$ and $(1+i) \nmid \beta$
Ensure: $c = \chi_\beta(\alpha)$ $(c = 0 \Leftrightarrow \chi_\beta(\alpha)$ is not defined)
1: $\alpha \leftarrow \mathrm{Red}_\beta(\alpha)$
2: **if** $\alpha = 0$ **then** $c = 0$ **end if**
3: let primary $\alpha_1, \beta_1 \in \mathbf{Z}[i]$ be defined by
$\alpha = (i)^{i_1} \cdot (1+i)^{j_1} \cdot \alpha_1$ and
$\beta = (i)^{i_2} \cdot \beta_1$
4: let $m, n \in \mathbf{Z}$ be defined by $\beta_1 = m + ni$
5: $t \leftarrow \frac{m-n-n^2-1}{4} j_1 + \frac{m^2+n^2-1}{4} i_1 \bmod 4$
6: replace α with β_1, β with α_1
7: $t \leftarrow t + \frac{(N(\alpha)-1)(N(\beta)-1)}{8} \bmod 4$
8: **while** $N(\alpha) > 1$ **do**
9: (LOOP INVARIANT: α, β are primary)
10: $\alpha \leftarrow \mathrm{Red}_\beta(\alpha)$
11: let primary α_1 be defined by $\alpha = (i)^{i_1} \cdot (1+i)^{j_1} \cdot \alpha_1$
12: let $m, n \in \mathbf{Z}$ be defined by $\beta = m + ni$
13: $t \leftarrow t + \frac{m-n-n^2-1}{4} j_1 + \frac{m^2+n^2-1}{4} i_1 \bmod 4$
14: replace α with β, β with α_1
15: $t \leftarrow t + \frac{(N(\alpha)-1)(N(\beta)-1)}{8} \bmod 4$
16: **end while**
17: **if** $N(\alpha) \neq 1$ **then** $c \leftarrow 0$ **else** $c \leftarrow i^t$ **end if**

Computation of Related Subfunctions For this algorithm we have to implement a few functions for calculating basic operations in the ring of Gaussian integers (let $\alpha, \beta \in \mathbf{Z}[i]$):

1. Multiplication: $\alpha \cdot \beta$
2. Modular reduction: $\mathrm{Red}_\beta(\alpha)$
3. Norm: $N(\alpha)$
4. Division by $(1+i)^r$
5. Primarisation: transforms α into its primary associate if possible

The multiplication and the norm are trivially implemented by performing integer multiplications between the appropriate integer components.

The division of α by $(1+i)^r$ can be done by first raising $(1+i)$ to the power of r and then dividing α by the result. We propose a way of achieving the same by only using shift operations, additions, and interchanging the imaginary and real part if necessary. The following equations demonstrate our procedure. Let $\alpha = a + bi$,

$$\frac{\alpha}{(1+i)} = \frac{a+b}{2} + \frac{b-a}{2} i,$$

$$\frac{\alpha}{(1+i)^r} = \frac{i^{3k} \left(\frac{a}{2^k} + \frac{b}{2^k} i \right)}{(1+i)^\ell}, \quad r = 2k + \ell.$$

If $r = 2k,\ k \in \mathbf{N}$ we shift the real and the imaginary parts of α by k to the right and multiply them by -1 and/or interchange them depending on the value of $3k$. If r is odd, there is an additional subtraction and addition to perform.

The primarisation function we used consists of a few congruency tests and it also determines the number of times we have to multiply α by i to get the primary associate of α.

The computation of $\mathrm{Red}_\beta(\alpha)$ is done according to [9] using an Euclidean division and rounding appropriately.

To find the representation of α we proceed as follows. First calculate the norm of $\alpha,\ N(\alpha)$. Then find j maximal such that $2^j \mid N(\alpha)$. Divide α by $(1+i)^j$ and transform the result into its primary associate.

In the implementation of the algorithm we need to ensure $(1+i) \nmid \beta$ and $\gcd(\alpha, \beta) \sim 1$. The first requirement is taken care of by applying the primarisation function on β. If we cannot find a primary associate, β is divisible by $(1+i)$ and we terminate. For the second condition we check in every iteration whether $\mathrm{Red}_\beta(\alpha) \to 0$. This would imply $\gcd(\alpha, \beta) \not\sim 1$ and we terminate.

4.3 Algorithm of Damgård and Frandsen

Description The most expensive operation used in the algorithm described above is $\mathrm{Red}_\beta(\alpha)$. Damgård and Frandsen present in [2] an efficient algorithm for computing the cubic residue symbol in the ring of Eisenstein integers $\mathbf{Z}[\zeta]$. Their algorithm can be transformed into an algorithm for the quartic residue symbol in the ring of Gaussian integers.

There are three main differences to the basic algorithm. Instead of using $\mathrm{Red}_\beta(\alpha)$ to reduce α, they suggest using $\alpha - \beta$. This takes much less time but increases the number of iterations needed. Furthermore they only interchange α and β, if $N(\alpha) < N(\beta)$. It is not necessary to calculate $N(\cdot)$ exactly for this purpose, an approximation $\tilde{N}(\cdot)$ suffices. They demonstrate how one can compute an approximate norm $\tilde{N}(\alpha)$ in linear time. Instead of adding up the squares of the real and the imaginary part of α, one replaces all but the 8 most significant bits of the real and the imaginary part of α with zeroes and computes the norm of the resulting Gaussian number.

Their algorithm takes $O(\log^2 N(\alpha\beta))$ time to compute $\chi_\beta(\alpha)$.

4.4 Other Algorithms

In addition to the above, we studied papers concerning algorithms for the quartic residue symbol by Weilert. In [17] he presents a fast gcd algorithm for Gaussian integers. Based on this gcd algorithm and using some properties of the Hilbert symbol he demonstrates in [18] how to construct an algorithm for the quartic residue symbol. This algorithm involves calculating an Euclidean descent and storing some intermediate results for later use. This algorithm presents a very fast asymptotic complexity which is even better than that of Damgård and Frandsen.

Algorithm 2 Damgård and Frandsen's Algorithm Quartic Residuosity in $\mathbf{Z}[i]$

Require: $\alpha, \beta \in \mathbf{Z}[i] \setminus \{0\}$, $\gcd(\alpha, \beta) \sim 1$ and $(1+i) \nmid \beta$
Ensure: $c = \chi_\beta(\alpha)$ $(c = 0 \Leftrightarrow \chi_\beta(\alpha)$ is not defined)
1: let primary $\alpha_1, \beta_1 \in \mathbf{Z}[i]$ be defined by
   $\alpha = (i)^{i_1} \cdot (1+i)^{j_1} \cdot \alpha_1$ and
   $\beta = (i)^{i_2} \cdot \beta_1$
2: let $m, n \in \mathbf{Z}$ be defined by $\beta_1 = m + ni$
3: $t \leftarrow \frac{m-n-n^2-1}{4} j_1 + \frac{m^2+n^2-1}{4} i_1 \bmod 4$
4: replace α with α_1, β with β_1
5: **if** $\tilde{N}(\alpha) < \tilde{N}(\beta)$ **then**
6: interchange α and β and adjust t
   $t \leftarrow t + \frac{(\tilde{N}(\alpha)-1)(N(\beta)-1)}{8} \bmod 4$
7: **end if**
8: **while** $\alpha \neq \beta$ **do**
9: (LOOP INVARIANT: α, β are primary)
10: let primary α_1 be defined by $\alpha - \beta = (i)^{i_1} \cdot (1+i)^{j_1} \cdot \alpha_1$
11: let $m, n \in \mathbf{Z}$ be defined by $\beta = m + ni$
12: $t \leftarrow t + \frac{m-n-n^2-1}{4} j_1 + \frac{m^2+n^2-1}{4} i_1 \bmod 4$
13: replace α with α_1
14: **if** $\tilde{N}(\alpha) < \tilde{N}(\beta)$ **then**
15: interchange α and β and adjust t
   $t \leftarrow t + \frac{(\tilde{N}(\alpha)-1)(N(\beta)-1)}{8} \bmod 4$
16: **end if**
17: **end while**
18: **if** $\alpha \neq 1$ **then** $c \leftarrow 0$ **else** $c \leftarrow i^t$ **end if**

However, as mentioned by Damgård et al. in [2], the fastest algorithms for practical inputs in the case of the Jacobi symbol are based on binary gcd algorithms [11]. Weilert proposed a binary gcd algorithm for Gaussian integers in [16] as well, but did not adapt it to the computation of the quartic residue symbol. Algorithms for cubic and quartic residue symbols taking this approach was proposed by Damgård et al. in [2] arguing that this is likely to provide a more efficient algorithm than the asymptotically fast variant of Weilert [18] in practice. Therefore, we have chosen to implement Algorithm 2 which takes this binary approach since we need a fast algorithm for practical inputs rather than the best asymptotic complexity.

5 Discrete Logarithm in a Hidden Subgroup

One suitable homomorphism for the generalized MOVA scheme is the one mentioned in Subsection 3.3. This homomorphism ϕ satisfies $\phi(x) = \log_g(x^r \bmod p)$. It consists of a modular exponentiation followed by a discrete logarithm computation. The modular exponentiation can be implemented by the classical methods such as the square-and-multiply method. For the discrete logarithm computation we consider three variants which are the use of a precomputed table of all discrete logarithms, the Baby Step Giant Step (BSGS) algorithm and Pollard's

Rho method. The choice of the algorithm will strongly depend on the amount of memory the signer has at disposal, namely the Pollard's Rho method requires almost no memory while the BSGS method is a time-memory tradeoff. Below we discuss the method of precomputed table and we refer to [10] for a description of the two other methods.

Given p prime, g a generator of a cyclic group G, subgroup of $\mathbf{Z}_p^*$, and $d = |G|$, we construct a table with entries (g^j, j) for $0 \leq j \leq d$. Building this table is a very time and memory consuming task, but once the table exists, finding the discrete logarithm consists of a simple look up operation.

There are several ways of constructing such a table. One can use a two dimensional array and sorting it by the first component. Finding the discrete logarithm is then reduced to a binary search. Alternatively, one can use conventional hashing on the first component to store the entries in a hash table, in which case placing an entry and searching for an entry in the table takes constant time. Another advantage is the fact, that we do not need space for g^i. Especially when $p \gg d$, this can save an enormous amount of memory. The only difficulties are finding a suitable hash function and dealing with collisions without losing too much time.

Time complexity of the construction of the table is $O(d)$ multiplications (plus $O(d \log d)$ comparisons to sort). Space complexity is $O(d(\log d + \log p))$ for the sorted table, resp. $O(d \log d)$ for the hash table. The running time for the sorted table is $O(\log d)$, for the hash table O(1).

6 Implementation

The implementation of all algorithms has been written in C using the GNU Multiple Precision Arithmetic Library (GMP) [6]. This library provides highly optimized arithmetic functions on large numbers. Most of the basic computations in $\mathbf{Z}$ have been performed using GMP such as integer multiplication or the modular exponentiation. For all implemented homomorphisms, we focused on the case where the modulus n is of size of 1024 bits.

6.1 Quartic Residue Symbol

Our principal optimization effort focused on the two algorithms computing the quartic residue symbol. In particular, we minimized the number of function calls, used some of the more sophisticated GMP functions, reduced the number of `mpz_t` (C data type for a multiple precision integer) variables whenever possible and applied general C optimization techniques such as described in [4, 8]. In addition, we used profiling and tried out different compiler optimization levels.

The basic algorithm has been implemented using the above remarks as well as the methods for computing the subfunctions which are explained in Subsection 4.2. We proceed in the same way for the algorithm of Damgård and Frandsen. Additionally, we tested whether the use of an approximative norm allows to obtain effective improvements. We implemented both the standard norm and the

norm Damgård and Frandsen suggest. The standard norm consists of only two GMP functions: one multiplication and one combined addition/multiplication whereas the approximate norm involves one bit scan to determine the size of the real part, one shift operation to extract the 8 most significant bits, one multiplication for the squaring of these 8 bits and another shift operation to put the result back to its correct position. We apply the same procedure on the imaginary part and we add the two approximate squarings up. In short, we need four additional operations to reduce the size of the numbers we have to multiply.

As GMP is a highly optimized library, computing the standard norm takes little time and the additional operations of the approximate norm only amortise if the real and the imaginary part are larger than 2048 bits. This and the fact that the norm of α and β decreases with each iteration convinced us to use the standard norm instead.

6.2 Discrete Logarithm

Here, we would like to present how we manage the computation of the discrete logarithm in the case of the precomputed table.

In this suggested variant of the generic homomorphic signature scheme, p is typically a 512 bit and d a 20 bit prime. Creating a table with d entries of size 532 bits is impossible on a usual desktop computer. Therefore we decided to use a hash table (key 512 bits, data 20 bits, 2^{20} entries). We found some existing hash table data structures written in C, but they do not fulfill our requirements. They are either too slow, support C types only, do not allow tables that large and/or they store the key as well.

To avoid problems, we did not adapt any of the existing data structures, but implemented a hash table ourselves providing enough storage and a collision handling mechanism suitable for our needs. Our solution is a hash table consisting of an array of unsigned integers. This array is of maximal length (2^{24}) to reduce collisions.

An unsigned integer is 32 bits long, so it was possible to store the data for the logarithm as well as using one of the higher order bits as a flag for collisions. Because the key is large and we wanted to avoid any unnecessary computation, we chose to use the 24 least significant bits of the key as the index into the hash table, in case of collision the next 24 bits, etc. By selecting 24 bits instead of the possible 20 bits, we minimize the occurrence of collisions. Tests have shown that most collisions are resolved by choosing the next 24 bits. We tried out other hash functions, but we did not achieve a gain of speed. This way, the size of the table is 64 MB.

To find the correct discrete logarithm for $y \in G$, one has to check if the collision flag at the corresponding array field is set, to decide if one can return the logarithm stored in the field or if one has to continue with the next field.

The implementation of the BSGS was done in a similar way. As the table contains much less entries, collisions hardly ever occur. The implementation of the Pollard's Rho method did not require any special treatment.

7 Results

In this section we present the results of the timing measurements we conducted to determine how well the different algorithms perform. In order to measure the running time precisely, we used functionalities offered by `frequence_cpu.h` by Victor Stinner [15]. The tests have been done on an Intel(R)4 1.4 GHz Desktop Computer with 256 MB RAM. Our results are average values produced by test series of 1000 tests.

7.1 Quartic Residue Symbol

We have considered the quartic residue symbol $\chi_\beta(\alpha)$ where α is a Gaussian integer with real and imaginary part of 1024 bits and $\beta = \pi\sigma$ a product of two primes and of size of 512 bits in each component. In such a situation, we have to consider a variant of the Damgård and Frandsen algorithm, we call the mixed algorithm. Namely, since α is much bigger than β it is more efficient in this case to compute first $\hat{\alpha} \leftarrow \mathrm{Red}_\beta(\alpha)$ and apply the Damgård and Frandsen algorithm on $\chi_\beta(\hat{\alpha})$. Timed results and number of iterations are given in Table 1.

	time in ms	iterations
Basic algorithm	32.12	248.81
Damgård's algorithm	50.63	766.12
Mixed algorithm	24.65	511.92

Table 1. Quartic Residue Symbol with $\beta = \pi\sigma$

The mixed algorithm is then the most judicious choice for fast implementations. The same phenomenon occurs for the case $\beta = \pi$ as well.

7.2 Signature Generation

Here, we finally compare the time required for generating a MOVA signature with the different homomorphisms. We consider a signature size of 20 bits. We omit the time required by the generation of the values Xsig_i's. Hence, we just have to compare the time required for computing 20 Jacobi symbols $(\cdot/p)_2$ (or $(\cdot/q)_2$), 20 quartic residue symbols with $\beta = \pi\sigma$, 10 quartic residue symbols with $\beta = \pi$, 1 homomorphism based on the discrete logarithm in a hidden subgroup and 1 RSA homomorphism. We recall that for all these homomorphisms, we take a modulus n of size of 1024 bits. Results are given in Table 2.

We have implemented the Jacobi symbol using a similar algorithm as Algorithm 1 and the basic GMP subroutines in order to have a fair comparison with our implementation of the quartic residue symbol. We note that the highly optimized GMP implementation of the Jacobi symbol `mpz_jacobi` provides the fastest signature generation and that the quartic residue symbol χ_π is about 4

Homomorphism	time in ms
Quartic Residue Symbol ($\beta = \pi\sigma$)	493.01
Quartic Residue Symbol ($\beta = \pi$)	90.32
Jacobi Symbol (ordinary algorithm)	25.22
Jacobi Symbol (`mpz_jacobi`)	2.32
Discrete Logarithm (Precomputed Table)	9.66
Discrete Logarithm (BSGS)	19.47
Discrete Logarithm (Pollard's rho)	74.93
RSA	33.87

Table 2. Results Comparison Signature Schemes

times slower than our implementation of the Jacobi symbol. This is mainly due to the fact that all operations are performed in $\mathbf{Z}[i]$ instead of $\mathbf{Z}$. We remark that the variant $\chi_{\pi\sigma}$ is much slower than for χ_π since we have to perform two times more quartic residue computations and that β is two times greater. The variants of the discrete logarithm offer a very competitive homomorphism. In particular, except for the variant using the Pollard's rho method this homomorphism is even more efficient than RSA. Finally, we can see that a 20-bit MOVA signature can be three times faster than a regular RSA signature.

8 Conclusion

We provided an overview of the implementation of the different candidates homomorphisms for the generalized MOVA scheme. Our principal case study concerned the quartic residue symbol, since the literature dedicated to its implementation is poor. We showed that the signature generation of the most efficient variant of the quartic residue symbol takes a little bit more than three times our implementation of the Jacobi symbol. The principal reason is that arithmetic operations are performed in $\mathbf{Z}[i]$ which are more costly than in $\mathbf{Z}$. We motivated the use of another variant of quartic residue symbol with two levels of secret by showing an application. This variant is the least efficient homomorphism of our comparison and requires about half a second to perform a signature generation on a classical workstation. When the signer has at least 64 MB memory, we demonstrated that using the homomorphism based on the discrete logarithm gives the most efficient signature generation after the Jacobi symbol implementation of GMP. However, if we take into account the cost of the confirmation protocol, this homomorphism is preferable to the characters. Finally, it is worthwhile to note that this implementation is about three times faster than a regular RSA signature scheme. We provided a clear overview of expected MOVA performances depending on the choice of the group homomorphism. Future work should further consider implementations with protection against side channels.

References

1. D. Chaum and H. van Antwerpen, *Undeniable Signatures*, Advances in Cryptology - Crypto '89, LNCS **435**, pp. 212-217, Springer, 1989.
2. I.B. Damgård and G.S. Frandsen, *Efficient Algorithms for GCD and Cubic Residuosity in the Ring of Eisenstein Integers*, FCT '03, LNCS **2751**, pp. 109-117, Springer, 2003.
3. H. Cohen, *A Course in Computational Algebraic Number Theory*, Graduate Texts in Mathematics **138**, Springer, 2000.
4. S. Garg, *How to optimize C/C++ Source - Performance Programming*, `http://bdn.borland.com/article/0,1410,28278,00.html`, 2002.
5. R. Gennaro, T. Rabin, and H. Krawczyk, *RSA-Based Undeniable Signatures*, Journal of Cryptology, **13**(4), pp. 397-416, Springer, 2000.
6. The GNU Multiple Precision Arithmetic Library, `http://www.swox.com/gmp/`.
7. K. Ireland and M. Rosen, *A Classical Introduction to Modern Number Theory: Second Edition*, Graduate Texts in Mathematics **84**, Springer, 1990.
8. M.E. Lee, *Optimization of Computer Programs in C*, `http://vision.eng.shu.ac.uk/bala/c/c/optimisation/l/optimization.html`, 2001.
9. V. Lefèvre, *Entiers de Gauss (sujet d'étude XM')*, `http://www.vinc17.org/math/index.fr.html`, 1993.
10. A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
11. S.M. Meyer and J.P. Sorendson, *Efficient Algorithms for Computing the Jacobi Symbol*, Journal of Symbolic Computation **26**(4), pp. 509-523, 1998.
12. J. Monnerat and S. Vaudenay, *Undeniable Signatures Based on Characters: How to Sign with One Bit*, PKC '04, LNCS **2947**, pp. 69-85, Springer, 2004.
13. J. Monnerat and S. Vaudenay, *Generic Homomorphic Undeniable Signatures*, Advances in Cryptology - Asiacrypt '04, LNCS **3329**, pp. 354-371, Springer, 2004.
14. R.L. Rivest, A. Shamir and L.M. Adleman, *A Method for Obtaining Digital Signatures and Public-key Cryptosystem*, Communications of the ACM, vol. **21**(2), pp. 120-126, 1978.
15. V. Stinner, `frequence_cpu.h`, `frequence_cpu.c`, `http://www.haypocalc.com/`, 2003.
16. A. Weilert, *(1+i)-ary GCD Computation in Z[i] is an analogue to the Binary GCD Algorithm*, Journal of Symbolic Computation **30**(5), pp. 605-617, 2000.
17. A. Weilert, *Asymptotically fast GCD Computation in Z[i]*, Algorithmic Number Theory, LNCS **1838**, pp. 595-613, Springer, 2000.
18. A. Weilert, *Fast Computation of the Biquadratic Residue Symbol*, Journal of Number Theory **96**, pp. 133-151, 2002.