Fast Algorithm for Finding a Small Root of a Quadratic Modular Equation

Hidenori Kuwakado and Hatsukazu Tanaka
Kobe University, 1-1 Rokkodai Nada Kobe 657, Japan
E-mail: {kuwakado, tanaka}@eedept.kobe-u.ac.jp

Abstract. The security of some cryptosystems is based on the difficulty of solving a quadratic modular equation. This paper shows a new algorithm for finding the small root of the quadratic modular equation. While previous algorithms for finding the small root of the modular equation are based on the LLL algorithm, the new algorithm is based on the continued fraction. Using the new algorithm, we can find the root less than $n^{1/4}$, where $n$ is the modulus. The new algorithm is more efficient than previous algorithms even if the modulus is large.

1 Introduction

If the factorization of the modulus is unknown, then it seems computationally difficult to solve the modular equation. The security of some public-key cryptosystems is based on its difficulty. Recently, Coppersmith proposed an algorithm for solving the modular equation (the Coppersmith algorithm) [2]. The size of the root that the Coppersmith algorithm can find is much smaller than that of the modulus.

In this paper, we focus on quadratic modular equations because the security of the Rabin scheme [5] and the reciprocal scheme [3] is based on the difficulty of solving the quadratic modular equation. We propose a new algorithm for computing the root of the quadratic modular equation. Although the size of the root found with our algorithm is smaller than that of the root found with the Coppersmith algorithm, our algorithm is more efficient than the Coppersmith algorithm. Differing from the Coppersmith algorithm, our algorithm is based on the continued fraction. Even if the modulus is large, our algorithm can be efficiently computed the small root.

It is known that solving the quadratic modular equation is one of the random self-reducible problems; if a non-negligible fraction of the instances can be solved efficiently, the entire instances can be done as well. Hence, the Coppersmith algorithm and our algorithm are effective for restricted instances. However, we can make use of these algorithms in the cryptanalysis as discussed in Sect. 4.
2 Proposed Algorithms

2.1 Notation

For any positive rational number \( r \), the continued fraction expansion of \( r \) is defined as follows.

\[
    r = c_0 + \frac{1}{c_1 + \frac{1}{c_2 + \cdots + \frac{1}{c_{m-1} + \frac{1}{c_m}}}}
\]

where \( c_i \in \mathbb{Z} \) (\( 0 \leq i \leq m \)), \( c_0 \geq 0, c_i > 0 \) (\( 1 \leq i \leq m - 1 \)) and \( c_m \geq 2 \). Using the Euclid algorithm, we can compute \( c_i \) efficiently. The continued fraction expansion of \( r \) is denoted by \( (c_0, c_1, \cdots, c_m) \). We call \( m \) the length of the continued fraction expansion of \( r \).

2.2 Extended continued fraction algorithm

Let \( \alpha/\beta \) and \( \gamma/\delta \) be positive irreducible fractions satisfying

\[
    \frac{\alpha}{\beta} = \frac{\gamma}{\delta} \left(1 - \varepsilon\right),
\]

where \( \varepsilon \) is a non-zero rational number. Suppose that \( \alpha/\beta \) is known, and \( \gamma/\delta \) and \( \varepsilon \) are unknown. If \( \varepsilon \) is the small positive rational number, then \( \gamma/\delta \) can be found with the continued fraction [8].

We extend the algorithm due to [8]; the extended continued fraction algorithm (ECFA) can compute \( \gamma/\delta \) regardless of the sign of \( \varepsilon \). In this algorithm, \( m \) is the length of the continued fraction expansion of \( \alpha/\beta \).

**ECFA:**

**Input** \( \alpha, \beta \).

**Output** \( \gamma, \delta \).

**Step 1** Set \( i \leftarrow 0 \).

**Step 2** If \( i > m \), then the ECFA fails; otherwise compute \( c_i \) of the continued fraction expansion of \( \alpha/\beta \).

**Step 3** For \( j = 0, 1 \), execute the following steps.

**Step 3-1** Compute the rational number \( \gamma'/\delta' = (c_0, \cdots, c_{i-1}, c_i + j) \).

**Step 3-2** If \( \gamma'/\delta' \) is equal to \( \gamma/\delta \), then return \( \gamma' \) and \( \delta' \) and halt.

**Step 4** Set \( i \leftarrow i + 1 \) and go to Step 2. \( \square \)

In Step 3-2, there must exist the checking method that determines whether \( \gamma'/\delta' \) is correct without knowing \( \gamma/\delta \).

If \( |\varepsilon| \) is not small enough, i.e., \( \alpha/\beta \) is not a good approximation to \( \gamma/\delta \), then the ECFA can not find \( \gamma/\beta \). The sufficient condition such that the ECFA succeeds is

\[
    |\varepsilon| < \frac{2}{3\gamma\delta}. \quad (1)
\]

This inequality can be obtained from the property of the continued fraction [8].