The Attack of the RSA Subgroup Assumption

JIANG WENG1,2, YUN-QI DOU1,2 AND CHUAN-GUI MA1,2
1State Key Laboratory of Mathematical Engineering and Advanced Computing
2Zhengzhou Information Science and Technology Institute
Zhengzhou, 450001 P.R. China
E-mail: {wengjiang858}@163.com

In TCC 2005, Groth proposed the cryptographic usefulness of a small subgroup \( G \) of \( \mathbb{Z}_n^* \) of hidden order. So far, the best attack of previous method for a subgroup of \( \mathbb{Z}_n^* \) had a complexity about \( \mathcal{O}(\sqrt{p'}) \). In this paper, we propose the interval and the double walks method to speed up the computation of the semi-smooth RSA subgroup problem. Our new algorithm reduces the complexity to \( \mathcal{O}(\sqrt{q'}) \) rather than \( \mathcal{O}(\sqrt{p'}) \). Besides the theoretical analysis, we also compare the performances of our new algorithm with the previous algorithm in experiments, and the efficiency of our new algorithm is approach to 50% faster than the previous.

Keywords: RSA moduli, hidden order, subgroup, cryptanalysis, semi-smooth RSA

1. INTRODUCTION

In order to speed up cryptographic computations it is of interest to find as small groups of hidden order as possible. In TCC 2005, Groth [1] proposed a collection of cryptographic primitives based on small RSA subgroups of \( \mathbb{Z}_n^* \) of hidden orders. Cryptographic schemes for integer commitment and digital signatures have been suggested over large subgroups of \( \mathbb{Z}_n^* \), by reducing the order of the groups we obtain quite similar but more efficient schemes. The underlying cryptographic assumption resembles the strong RSA assumption.

The RSA moduli \( N \) used in TCC 2005 are of the form:

\[
N = pq = (2p' + 1)(2q' + 1),
\]

where \( p, q, p', \) and \( q' \) are prime integers and \( r, s \) are random integers. Then there exists a unique subgroup \( G \) of \( \mathbb{Z}_n^* \) of order \( p'q' \).

Groth [1] showed that the best attack has complexity \( \mathcal{O}(p') \). Therefore, Groth [1] suggested to use \( l_p = l_q = 100 \) as security parameters, where \( l_p \) and \( l_q \) are the bit length of the primes \( p' \) and \( q' \). Furthermore, Coron et al. [2] proposed a faster attack that recover the secret factors of \( N \) in \( \mathcal{O}(\sqrt{p'}) \). Under Groth’s [1] security parameter assumption, Coron et al. [2] can recover the secret factor in 250 attack. Obviously, it is potentially insecure.

We introduce the suggested parameters of Groth [1] in section 2, where \( r \) and \( s \) are smooth integers. In the case of a semi-smooth RSA subgroup modulus, the factorization would also tell us the factors \( p', q' \) and we can break the decisional semi-smooth RSA sub-
group assumption.

The security of many cryptographic schemes is based on the strong RSA subgroup assumption. The well-known attack method such as Pollard's rho method, the baby-step giant-step algorithm [3], Pollard's $\lambda$-method [4] and Pollard's $p - 1$ method [5], the complexity of all these methods is about $2^\kappa$ modular multiplications. Furthermore, the elliptic curve method (ECM) [6] and the general number field sieve (GNFS) [7] also be used to attack many cryptographic schemes.

In particular, the scheme proposed by Damgard et al. [8] used a subgroup of prime order $r$ of $\mathbb{Z}_N^*$, where $r$ is a factor of both $p - 1$ and $q - 1$. On the other hand, it applies to the subgroup variant of the Paillier cryptosystem [9]. The parameter choice from the original paper was more conservative than that of Groth [1].

In the recent attack the semi-smooth RSA subgroup assumption, Coron et al. [2] proposed square-root attack. Multipoint evaluation of univariate polynomials has been used in Coron et al.'s [2] method. One is actually interested in the following problem: given two lists \{a_i\} and \{b_j\} of numbers modulo $N$, find a pair $(a_i, b_j)$ such that $\gcd(a_i - b_j, N)$ is non-trivial. Instead, we use multipoint evaluation differently, as a way to compute certain products of $m$ elements modulo $N$ in $O(m)$ polynomial-time operations.

In this paper, we give a new algorithm to improve the efficiency of the computation semi-smooth RSA subgroup problem. The experiment results show that we can achieve the speedup by a factor extremely close to 50%, which is the best performance one can achieve in theory, using the new algorithm with the interval method and the double walks on the equivalence class.

The remainder of this paper is organized as follows. Firstly we recall a brief overview of the semi-smooth subgroup notation, and then discuss the previous attack methods in Section 2. Section 3 presents our new attack against the semi-smooth RSA subgroup problem with the negation map and the equivalence. We briefly describe fast multiplication algorithm in $\mathbb{Z}_N$ in Section 4. Section 5 we give the complexity analysis and the pseudocode of our new algorithm. We conclude the paper in section 6.

2. PRELIMINARIES

In this section, we recall a brief overview of the semi-smooth subgroup notation, and then introduce the previous attack methods.

2.1 RSA with Semi-smooth Subgroup

Groth [1] introduced the definition of semi-smooth subgroup as follows. Let $(N, g)$ be a semi-smooth RSA subgroup pair. Consider randomly choosing $p, q$ satisfying $p = 2p_1p_2\cdots p_m + 1$, $q = 2q_1q_2\cdots q_m + 1$, where $p_1, \ldots, p_m, q_1, \ldots, q_m$ are distinct odd primes smaller than some small bound $B$. The integer $N=pq$ is called semi-smooth integer.

**Definition 1:** Let $N = (2p_1p_2\cdots p_m + 1)(2q_1q_2\cdots q_m + 1)$ be a random integer, the unique subgroup $G$ of order $p^aq$ is called the semi-smooth subgroup of $\mathbb{Z}_N^*$.

Define $P = \prod_{p \in \mathbb{P}, \pi(p) \leq \kappa} p$. Choosing $h$ at random and setting $g = h^P$, we have overwhelming probability of $g$ generating $G$. In other words, it is easy to find a generator
of $G$ if $N$ is given. Therefore, $g$ is saved and $N$ is public.

There is no probabilistic polynomial-time algorithm such that only given $N$, and it is easy to factor $N$ with non-negligible probability. Currently, the best known general purpose factoring algorithm is the general number field sieve (GNFS). Heuristically, this algorithm runs in time $\exp(O((\ln(n))^{1/3} \ln(\ln(n))^{1/3}))$ on average to factor a number $N$ of length $O(n)$.

**Lemma 1:** Let $m$ be a positive integer, and let $a, x$ and $y$ be integers with $ax = ay (\text{mod } m)$. Suppose that $m$ and $a$ are coprime. Then $x = y (\text{mod } m)$.

**Lemma 2:** (Fermat’s Little Theorem) Let $p$ be a prime number. Then $x^p \equiv x (\text{mod } p)$ for all integers $x$. Moreover if $x$ is coprime to $p$ then $x^{p-1} \equiv 1 (\text{mod } p)$.

### 2.2 Previous Attack

To introduce notation and the central ideas, we first give an overview of the method proposed by Coron et al. [2], which will be used in our later calculations. Recall that the RSA modulus $N=pq$ is such that:

$$N=pq=(2p'r+1) (2q's+1),$$

where $p'$ and $q'$ are primes. Besides, $g$ is a generator of the subgroup $G$ of order $p'q'$.

Groth [1] showed that factor $N$ in complexity $O(p')$. In PKC 2011, Coron et al. [2] proposed an attack with factor $N$ in time complexity $O(\sqrt{p'})$ and space complexity $O(\sqrt{p'})$. The core idea is as follows.

Because of $g^{p'q'} \equiv 1 \mod N$, and $g^{p's} \equiv 1 \mod p$. It is easy to see that $g^{p'} \equiv 1 \mod p$. Let $l$ denote the bit-length of $p'$. Without loss of generality, assume it is even, and writes $\Delta = 2^{l/2}$.

Denotes $p' \equiv a + \Delta b$, where $0 \leq a, b < 2^{l/2}$. Then

$$g^{a} = (g^{a})^{-b} \mod p. \quad (1)$$

From the left and right parts of the Eq. (1) generates the following two lists:

$$\{L_i = \{g^i \mod p : 0 < i < 2^{l/2}\}\}$$

$$\{L'_i = \{(g^{a})^{-j} \mod p : 0 < j < 2^{l/2}\}\}. \quad (2)$$

Obviously the prime factor $p$ is unknown, equivalently, using the following two lists modulo $N$ instead of Eq. (2):

$$\{L_N = \{x_i = g^i \mod N : 0 < i < 2^{l/2}\}\}$$

$$\{L'_N = \{y_j = (g^{a})^{-j} \mod N : 0 < j < 2^{l/2}\}\}. \quad (3)$$

The above equations mean that there exists $i=a$ and $j=b$ for all $x_i \in L$ and all $y_j \in L'$ such that $x_a - y_b = 0 \mod N$. It is equivalent to $x_a - y_b \equiv 0 \mod p$, which in turn implies
that \( \gcd(x_a - y_b, N) \) computation yields a non-trivial factor of \( N \). The problem of factor \( N \) has been reduced to find two values \( x_a \) and \( y_b \) with the stated properties.

Multipoint evaluation of univariate polynomials has been used to find such two values \( x_a \) and \( y_b \). First it generates the polynomial: \( f(x) = \prod_{x \in L'} (x - x_j) \mod N \). Second, for all \( y_j \in L' \), evaluate \( f \) at \( y_j \) and compute \( \gcd(f(y_j), N) \). There exists \( y_b \in L' \) such that \( f(y_b) = \prod_{x \in L'} (y_b - x_j) = (y_b - x_0) \cdot R = 0 \mod p \), computing \( \gcd(f(y_j), N) \) reveals the factor of \( N \) for \( j = b \). This would reveal the factors of \( N \). This attack exhibits the complexity in \( O(\sqrt{p^3}) \) both in time and space.

3. FAST MULTIPLICATION

In this section, we briefly introduce fast multiplication algorithm. Fast multiplication algorithm for Newton interpolation and evaluation rely on the base change between the monomial basis and the Newton basis. We focus on the special case of interpolation and evaluation points in geometric progression. In this case, the complexities of the Newton evaluation and interpolation on a geometric progression of size \( n \) can be computed within \( M(n) + O(n) \) base field operations. As well as, the conversions between the monomial and Newton bases can be done with the same asymptotic complexity of \( M(n) + O(n) \) base field operations.

First we introduce the monomial and Newton bases. The monomial basis is defined as \( 1, x, \ldots, x^{n-1} \), and the Newton basis is defined as \( 1, (x - x_0), (x - x_0)(x - x_1), \ldots, (x - x_{n-1}), \ldots, (x - x_0) \).

3.1 Polynomial Interpolation

In what follows, suppose that \( n \) is a power of 2, and the subproduct tree \( T \) associated to \( x = x_0, \ldots, x_{n-1} \) is then a complete binary tree. In order to compute a polynomial given as a product of \( n \) terms. The following result was first pointed out by Horowitz [10], Gathen and Gerhard [11].

**Proposition 1:** The subproduct tree associated to \( x_0, \ldots, x_{n-1} \) can be computed within \( 1/2 M(n) \log(n) + O(n \log(n)) \) base field operations.

If the points \( x \) form a geometric sequence, it is indeed an acceleration (by a logarithmic factor) can be obtained. We now treat the question of fast interpolation in the Newton basis.

The problem of interpolation in the Newton basis is described as follows. Given the values \( x_0, \ldots, x_{n-1} \) and \( v_0, \ldots, v_{n-1} \), Newton interpolation consists in determining the unique coefficients \( a_0, \ldots, a_{n-1} \) such that the polynomial

\[
    f = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_{n-1}(x - x_0) \cdots (x - x_{n-2})
\]

satisfies \( f(x_i) = v_i \), for \( i = 0, \ldots, n - 1 \).
3.2 Polynomial Evaluation

In the following, we evaluate \( f \mod N \) at each of the points \( x_i \) with \( 0 \leq i \leq n-1 \). Since \( x_i \) is geometric progression, fast evaluation on geometric progression is introduced as follows.

Well known problem of evaluation in the Newton basis is described as: given \( x_0, \ldots, x_{n-1} \) and \( a_0, \ldots, a_{n-1} \), Newton evaluation consists in computing the values \( v_0 = f(x_0), \ldots, v_{n-1} = f(x_{n-1}) \), where \( f \) is the polynomial given by Eq. (4).

Fast algorithm for evaluation in the monomial basis have complexity in \( O(M(n) \log(n)) \), where \( M(n) \) denotes the cost of multiplying univariate polynomials of degree less than \( n \). Using FFT-based multiplication algorithms, \( M(n) \) can be taken in \( O(n \log(n) \log \log(n)) \).

**Proposition 2:** [12] Let \( k \) be a field and let \( q \in k \) such that the elements \( x_i = q^i \) are pairwise distinct, for \( i = 0, \ldots, n-1 \). Then Newton interpolation and evaluation on the geometric sequence \( 1, q, \ldots, q^{n-1} \) can be done using \( M(n)+O(n) \) base field operations.

3.3 Conversions Between Monomial and Newton Bases

In our new algorithm, we mainly contain two steps, polynomial interpolation and evaluation in geometric sequence. It involves the conversions between the monomial and Newton bases in two steps. The following proposition gives the complexity of conversions between the monomial and Newton bases.

**Proposition 3:** [12] Let \( k \) be a field and let \( q \in k \) such that the elements \( x_i = q^i \) are pairwise distinct, for \( i = 0, \ldots, n-1 \). Then the conversion between the monomial basis and Newton basis associated to \( 1, q, \ldots, q^{n-1} \) can be done using \( M(n)+O(n) \) base field operations.

4. NEW ALTERNATIVE APPROACHES

In this section we discuss that how to construct a useful univariate polynomial which can be more efficient to compute and evaluate. Then we propose double walks method on the equivalence class for the multipoint evaluation of univariate polynomial, which is to obtain a speedup, hopefully by a factor of 2.

4.1 Motivation

The order of the congruence class of \( g \) modulo \( p \) is by definition the smallest positive integer \( d \) such that \( g^d = 1 \mod p \). Since \( g \) is an integer coprime to \( p \) then it follows from Fermat’s theorem that \( g^{\phi(p)} = 1 \mod p \). It follows from Lemma 2 that the order of \( g \) modulo \( p \) divides \( \phi(p) \), thus \( \phi(p) = 2p' \) is divided by \( d \).

Since \( p' \) is a prime integer, it is easy to obtain the congruence \( g^{p'} = 1 \mod p \). Now it follows from Lagrange theorem that the order of any element of a finite group divides the order of the group. Then we apply this result to the group of congruence classes
modulo $p$. It implies that the sequence of powers of $g$ is periodic with $d = p'$. The congruence $g^{d} = 1 \pmod{p}$ has at most $p'$ solutions modulo $p$. Since $p$ is prime, the congruence classes of $1, g, g^2, \cdots, g^{p-2}$ modulo $p$ are distinct. To better understand the double walks method, we have the following fact.

**Proposition 4:** Let $p$ be a positive integer, let $g$ be an integer coprime to $p$, and let $j$ and $k$ be positive integers. Then $g^{j} = g^{k} \pmod{p}$ if and only if $j = k \pmod{p'}$, where $p'$ is the order of the congruence class of $g$ modulo $p$.

**Proof:** We may suppose without loss of generality that $j < k$. If $j = k \pmod{p'}$ then $k - j$ is divisible by $p'$, and hence $g^{k-j} = 1 \pmod{p}$. But then $g^{j} = g^{k-j} = g^{k} \pmod{p}$. Conversely suppose that $g^{j} = g^{k} \pmod{p}$ and $j < k$. Then $g^{j} g^{k-j} = g^{k} \pmod{p}$. But $g^{j}$ is coprime to $p$, it follows from Lemma 1 that $g^{k-j} = 1 \pmod{p}$. Thus if $k - j = q p' + r$, where $q$ and $r$ are integers and $0 \leq r < p'$, then $g^{r} = 1 \pmod{p}$. But then $r = 0$, since $p'$ is the smallest positive integer for which $g^{p'} = 1 \pmod{p}$. Therefore $k - j$ is divisible by $p'$, and thus $j = k \pmod{p'}$. This completes the proof.

### 4.2 The New Algorithm

We now present how to use interval method and construct double walks on the equivalence classes. Then we implement these techniques to give a more efficient algorithm for the attack semi-smooth RSA subgroup.

The basic idea is to run the Coron et al. [2] algorithm on the set of equivalence classes. A further speedup is given by defining double walks on the equivalence classes and analyzing the polynomial fast multiplication. The congruence $g^{x} = (g^{x})^{a} \pmod{p}$ is known.

**The interval:** We shift the problem (multiplying both sides by $g^ {- \frac{a}{2}}$ which requires an inversion in the group) so that it is of the form $g^{x} \cdot g^ {- \frac{a}{2}} = (g^{x})^{a} \cdot g^ {- \frac{a}{2}} \pmod{p}$, which implies that

\[
g^{x} = (g^{-\frac{a}{2}})^{a} \pmod{p}.
\]  

(5)

Notice that $0 \leq a, b < \Delta$, then denote $a' = a - \frac{a}{2}$, $b' = \frac{1}{2} + b$, we see that $-\frac{a}{2} \leq a' < \frac{a}{2}$, $1 \leq b' < \Delta$, and $p'$ is the smallest positive integer for which $g^{p'} = 1 \pmod{p}$. It implies that the sequence of powers of $g$ is periodic with period $p'$. Therefore, we applied Proposition 4 to the equation $g^{x} = (g^{x})^{a'} \pmod{p}$, and we obtain the desired equation $a' = -\Delta \cdot b' \pmod{p'}$.

**Double walks:** Let $p'$ be an odd prime. We know that $\Delta = 2^{i+2}$, thus $-\Delta \cdot b'$ is an even integer. For instance $|\Delta \cdot b'| < p'$, then $-\Delta \cdot b' \pmod{p'}$ is an odd integer. Therefore $a' \pmod{p'}$ must be an odd integer. Since $-\frac{a}{2} \leq a' < \frac{a}{2}$, thus $|a'| < p'$. Obviously the left of the equation $a'$ is an odd integer when $a' > 0$, and $a'$ is an even integer when $a' < 0$.

Now we only need to deal with the case where $a' > 0$ is odd. From the left and right parts of the Eq. (5), we generate the following two lists:
\[
\begin{cases}
L_p = \{x_i = g^{2\Delta i} \mod p : 0 \leq i < \frac{\Delta}{2}\} \\
L_p' = \{z_i = (g^{-\Delta})^i \mod p : 1 \leq i \leq \Delta\}.
\end{cases}
\]  

(6)

Since the prime factor \( p \) was unknown, equivalently, we generate the following lists modulo \( N \) instead of Eq. (6):

\[
\begin{cases}
L_N = \{x_i = g^{2\Delta i} \mod N : 0 \leq i < \frac{\Delta}{4}\} \\
L_N' = \{z_i = (g^{-\Delta})^i \mod N : 1 \leq i \leq \Delta\}.
\end{cases}
\]  

(7)

\( \hat{L}_N = \{ \hat{x}_i = x_i^{-1} \mod N : 0 \leq i < \frac{\Delta}{4}\} \). Next we describe how to find such two values \( x_a \) and \( y_b \). Then we generate the following two polynomials:

\[
f(x) = \prod_{i=0}^{\frac{\Delta}{4} - 1} (x - x_i) \mod N \quad \text{and} \quad g(x) = \prod_{i=0}^{\frac{\Delta}{4} - 1} (x - \hat{x}_i) \mod N.
\]

Because we have got the set \( L_N \), then infer the set \( \hat{L}_N \) from \( L_N \), where \( x_i \hat{x}_i = 1 \mod N \) for \( i \) from 0 to \( \frac{\Delta}{4} - 1 \). The modular inverse of \( x_i^{-1} \mod N \) can be computed by means of the Extended Euclidean Algorithm.

We evaluate \( f \) and \( g \) at \( z_i \) for all \( z_i \in \hat{L}_N \), and calculate \( \gcd(f(z_i), N) \) and \( \gcd(g(z_i), N) \) respectively. More precisely, we have the following algorithm.

\begin{algorithm}
\caption{New Algorithm for Pollard’s \( p - 1 \) with the interval}
\textbf{Input:} A semi-smooth RSA subgroup modulus \( N = pq = (2p'q' + 1)(2q's + 1) \), \( g \) is a generator of the subgroup \( G \) of order \( p'q' \);
\textbf{Output:} A non-trivial factor of \( N \);
\begin{algorithmic}[1]
\State 1 : \( \Delta \leftarrow 2^2 \);
\State 1 : \( \Delta \leftarrow 2^2 \);
\State 1 : \( \Delta \leftarrow 2^2 \);
\State 1 : \( \Delta \leftarrow 2^2 \);
\State 2 : \For {i = 0 to \( \frac{\Delta}{4} - 1 \) do}
\State 3 : \( x_i \leftarrow g^{2\Delta i} \mod N \);
\State 4 : \( y_i \leftarrow x_i^{-1} \mod N \);
\State 5 : \EndFor
\State 6 : \( z_i \leftarrow (g^{-\Delta})^i \mod N \);
\State 7 : \End
\State 8 : Generate the polynomial
\State \( f(x) \leftarrow \prod_{i=0}^{\frac{\Delta}{4} - 1} (x - x_i) \mod N \), \( g(x) \leftarrow \prod_{i=0}^{\frac{\Delta}{4} - 1} (x - y_i) \mod N \);
\State 9 : \For {j = 0 to \( \Delta - 1 \) do}
\State 10 : Evaluate \( f(z_j) \in Z_N \) and \( g(z_j) \in Z_N \);
\State 11 : Attempt to factor \( N \) by computing \( \gcd(f(z_j), N) \) and \( \gcd(g(z_j), N) \).
\State 12 : \EndFor
\end{algorithmic}
\end{algorithm}

5. COMPLEXITY ANALYSIS OF ALGORITHM 1

We now give complexity estimates for the problems of computation and evaluation as the following form:
\[ f(x) = \prod_{i=0}^{\frac{d}{2}-1} (x-x_i) \mod N, \]

where \( \frac{d}{2} = 2^{\frac{\theta}{2}} \).

For all \( z_i \in L_N \), evaluate \( f \) and \( g \) at \( z_i \), then compute \( \gcd(f(z_i), N) \) and \( \gcd(g(z_i), N) \). Since we have \( f(z_i) = \prod_{j=0}^{i} (x_j-x_i) \cdot R = 0 \mod p \), or \( g(z_i) = \prod_{j=0}^{i} (x_0-x_i) \cdot R = 0 \mod p \). So the \( \gcd(f(z_i), N) \) or \( \gcd(g(z_i), N) \) can reveal the factor of \( N \) for \( i = h \).

In fact, when we run the Algorithm 1, it contains two steps: first compute the coefficients of the polynomial \( f(x) = \prod_{i=0}^{\frac{d}{2}-1} (x-x_i) \mod N \), and then evaluate \( f(x) \mod N \) at each of the points \( z_i \).

It is a classical fact [13] that the coefficients of \( f(x) = \prod_{i=0}^{\frac{d}{2}-1} (x-x_i) \mod N \) can be computed using a product tree, and have complexity in \( O(M(\frac{\theta}{2}) \log \frac{\theta}{2}) \), where \( M(\frac{\theta}{2}) \) denotes the cost of multiply univariate polynomials of degree less than \( \frac{\theta}{2} \). Similarly, with a remainder tree, \( f(x) = \prod_{i=0}^{\frac{d}{2}-1} (x-x_i) \mod N \) can be evaluated at all points \( z_j, 0 \leq j < \frac{\theta}{2} \), also have complexity in \( O(M(\frac{\theta}{2}) \log \frac{\theta}{2}) \). \( M(\frac{\theta}{2}) \) can be taken in \( O(\frac{\theta}{2} \log \frac{\theta}{2} \log(\log \frac{\theta}{2})) \) using FFT-based multiplication algorithms. So the above complexity is nearly optimal, up to logarithmic factors.

In our case, because \( (x_i) \) and \( (z_i) \) are geometric progressions, these steps of polynomial interpolation and polynomial evaluation reduce to a variant of the discrete Fourier transform [14, 15]. Simplifications in Newton evaluation and interpolation formulas also arise by Schoenberg [16] when the sample points form a geometric progression. Hence the Newton basis conversion algorithms of Bostan and Schost [12] are more efficient than others.

To better understand the Algorithm 1, we give the pseudocode of Algorithm 1, as described in Algorithm 2. The core of the Algorithm 2 mentioned up three operations: Newton interpolation, Newton evaluation and the conversions between the monomial basis and the Newton basis. By propositions 2 and 3 we know that the complexity of Newton interpolation, Newton evaluation and the conversions between the monomial basis and the Newton basis can be taken in \( M(n) + O(n) \).

In order to simplify this task, each step of Algorithm 2 will be studied separately and then the results will be added up to obtain the average cost. We can break down the attack in two steps: first compute the coefficients of the polynomial \( f(x) = \prod_{i=0}^{\frac{d}{2}-1} (x-x_i) \mod N \) by Newton interpolation, and then evaluate \( f(x) \mod N \) at each of the points \( z_j \) by Newton evaluation. In this two steps, it includes that the conversions between the monomial basis and the Newton basis.

In our case, the result is an overall time complexity of \( 3M(\frac{\theta}{2}) + O(\frac{\theta}{2}) \) for the complete attack. Space requirements are also \( O(\frac{\theta}{2}) \), to store a few polynomials of degree \( \frac{\theta}{2} \). Thus, the complexity of our attack is about \( O(\sqrt{p/2}) \) both in time and space.

In our implementation, we apply the Newton interpolation and evaluation algorithms of Bostan and Schost [12] to compute and evaluate \( f(x) = \prod_{i=0}^{\frac{d}{2}-1} (x-x_i) \mod N \). More precisely, we provide the pseudocodes of Newton interpolation, Newton evaluation, conversion from monomial to Newton basis and from Newton to monomial basis on the geometric sequence as Algorithms 3, 4, 5 and 6 respectively in Appendix A.
5.1 Experiments

The main obstruction of attacking semi-smooth RSA subgroup is time and space memory. However, it is not obvious how the algorithm can be efficiently parallelized to distribute the computation.

To evaluate the efficiency of the new algorithm, we implemented and compared the attack running times of PKC 2011 with our new algorithm on an Intel(R) Core(TM)2 Duo CPU E8500 3.12GHz as follows:

Table 1. Experimental attack running times for 1024-bit moduli.

<table>
<thead>
<tr>
<th>( l = \lceil \log_2 p' \rceil )</th>
<th>PKC 2011’s Algorithm</th>
<th>Algorithm 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>26 bits</td>
<td>1.9 seconds</td>
<td>0.9 seconds</td>
</tr>
<tr>
<td>28 bits</td>
<td>4.0 seconds</td>
<td>1.9 seconds</td>
</tr>
<tr>
<td>30 bits</td>
<td>8.1 seconds</td>
<td>4.0 seconds</td>
</tr>
<tr>
<td>32 bits</td>
<td>16.5 seconds</td>
<td>8.2 seconds</td>
</tr>
<tr>
<td>34 bits</td>
<td>33.5 seconds</td>
<td>16.6 seconds</td>
</tr>
<tr>
<td>36 bits</td>
<td>68.9 seconds</td>
<td>33.4 seconds</td>
</tr>
</tbody>
</table>

Obviously, our new algorithm takes two bits advantage than PKC 2011’s algorithm. However, the running time of the attack is exponential growth when the bit length \( l \) of the prime \( p' \) is linear growth. Therefore, it is difficult to deal with larger parameters because the running time is exponential growth, which is out of our finite computational ability. So we direct extrapolation yields the following estimates from Table 1, which also implemented in C using the FLINT library [17] and the same computing environment.

Table 2. Compare the running time for 1024-bit moduli.

<table>
<thead>
<tr>
<th>( l = \lceil \log_2 p' \rceil )</th>
<th>PKC 2011’s Algorithm</th>
<th>Algorithm 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>60 bits</td>
<td>3 days</td>
<td>≈1.6 days</td>
</tr>
<tr>
<td>80 bits</td>
<td>9 years</td>
<td>≈4.4 years</td>
</tr>
<tr>
<td>100 bits</td>
<td>9000 years</td>
<td>≈4548.8 years</td>
</tr>
</tbody>
</table>

In Table 2, we compare the running time of our attack with the PKC 2011’s in the same computing environment. The running time of PKC 2011’s algorithm is \( O(\sqrt{p'}) \) elementary operations. Obviously, our attack running time is only about half of PKC 2011’s. The memory of our attack was also compared to the PKC 2011’s in Table 3.

From Table 3, we know that the space requirements of our new algorithm is about half of PKC 2011’s algorithm.

Table 3. Compare the memory for 1024-bit moduli.

<table>
<thead>
<tr>
<th>( l = \lceil \log_2 p' \rceil )</th>
<th>PKC 2011’s Algorithm</th>
<th>Algorithm 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>60 bits</td>
<td>≈1 GB</td>
<td>≈0.5 GB</td>
</tr>
<tr>
<td>80 bits</td>
<td>≈1 TB</td>
<td>≈0.5 TB</td>
</tr>
<tr>
<td>100 bits</td>
<td>≈1 PB</td>
<td>≈0.5 PB</td>
</tr>
</tbody>
</table>
In conclusion, consider the original parameters proposed by Groth [1]. The bit length \( l_p' \) and \( l_q' \) of the primes \( p' \) and \( q' \) as \( l_p' = l_q' = 100 \), so the space requirements is about \( O(\sqrt[p']{p'}) \) bits, which is about 1 PB. Obviously, it is a serious hurdle. This means that we can’t settle these data in practice. In this case, we only need 0.5 PB storage and \( O(\sqrt[p']{p'}/2) \) elementary operations.

In general, the efficient of other algorithms is a tradeoff between time complexity and space requirements. However, our algorithm reduces the complexity of both time and space. So our algorithm is better than the PKC 2011’s algorithm.

**Algorithm 2**: Pseudocode

**Input**: A semi-smooth RSA subgroup modulus \( N = p q = (2p'r + 1) (2q's + 1) \), \( g \) is a generator of the subgroup \( G \) of order \( p'q' \), the polynomial \( F \) of degree less than \( n = \Delta; \)

**Output**: A non-trivial factor of \( N; \)

1: \( p \leftarrow 1; q \leftarrow 1; s \leftarrow 1; u \leftarrow 1; z \leftarrow 1; x \leftarrow 1; \)

2: Initialize \( U, U', Z, Z', W, W' \) as zero polynomials of degree \( \frac{n}{2} - 1 \) and \( M, K, K', S \)

as zero polynomials of degree \( n - 1 \)

3: \( U_0 \leftarrow u, Z_0 \leftarrow z, W_0 \leftarrow w \)

4: \( \text{for } i = 1 \text{ to } \frac{2}{3} \text{ do} \)

5: \( p \leftarrow p \cdot g \mod N \)

6: \( x \leftarrow p^2 \mod N \)

7: \( p \leftarrow x \cdot p \mod N \)

8: \( q \leftarrow q \cdot p \mod N \)

9: \( s \leftarrow s \cdot (p - 1) \mod N \)

10: \( u \leftarrow u \cdot p / (1 - p) \mod N \)

11: \( z \leftarrow (-1)^i u / q \mod N \)

12: \( w \leftarrow q / (s \cdot u) \mod N \)

13: \( U_i \leftarrow u, Z_i \leftarrow z, W_i \leftarrow w \)

14: \( \text{end for} \)

15: \( Z \leftarrow (-1)^{\frac{n}{2}} s_{\frac{n}{2} - 1} Z \mod N \)

16: \( W \leftarrow \text{mul}'(\xi - 1, U, W) \)

17: \( \text{for } i = 0 \text{ to } \frac{2}{3} \text{ do} \)

18: \( W_i \leftarrow W_i \cdot Z_i \mod N \)

19: \( \text{end for} \)

20: \( p' \leftarrow 1; q' \leftarrow 1; s' \leftarrow 1; u' \leftarrow 1; z' \leftarrow 1; w' \leftarrow 1; x' \leftarrow 1 \)

21: \( U'_0 \leftarrow u, Z'_0 \leftarrow z', W'_0 \leftarrow w' \)

22: \( g \leftarrow \frac{1}{s'x'} \mod N \)

23: \( \text{for } i = 1 \text{ to } \frac{2}{3} \text{ do} \)

24: \( p' \leftarrow p' \cdot g \mod N \)

25: \( x' \leftarrow p'^2 \mod N \)

26: \( p' \leftarrow x' \cdot p' \mod N \)

27: \( q' \leftarrow q' \cdot p' \mod N \)

28: \( s' \leftarrow s' \cdot (p' - 1) \mod N \)

29: \( u' \leftarrow u' \cdot p' / (1 - p') \mod N \)
30: \( z' \leftarrow (-1)^i u' / q' \mod N \)
31: \( w' \leftarrow q' / (s' \cdot u') \mod N \)
32: \( U', Z', S' \leftarrow z', w' \)
33: \( \textbf{end for} \)
34: \( Z' \leftarrow (-1)^{i+1} s' \cdot Z' \mod N \)
35: \( W' \leftarrow \text{mul}'(\xi - 1, U', W') \)
36: \( \textbf{for} \ i = 0 \ \textbf{to} \ \xi - 1 \ \textbf{do} \)
37: \( W'_i \leftarrow W'_i \mod N_i \)
38: \( \textbf{end for} \)
39: \( g \leftarrow \frac{1}{n} \mod N \Rightarrow g \leftarrow g^{-\xi} \)
40: \( p \leftarrow 1; q \leftarrow 1; s \leftarrow 1; u \leftarrow 1; m \leftarrow 1 \)
41: \( U \leftarrow 0; M \leftarrow 0 \)
42: \( U_i \leftarrow u, M_i \leftarrow m, S_i \leftarrow s \)
43: \( \textbf{for} \ i = 1 \ \textbf{to} \ n - 1 \ \textbf{do} \)
44: \( p \leftarrow p \cdot g \mod N \)
45: \( q \leftarrow q \cdot p \mod N \)
46: \( s \leftarrow s \cdot (p - 1) \mod N \)
47: \( u \leftarrow u \cdot (1 - p) \mod N \)
48: \( m \leftarrow (-1)^i u / q \mod N \)
49: \( S_i \leftarrow s, U_i \leftarrow u, Z_i \leftarrow z \)
50: \( \textbf{end for} \)
51: \( \textbf{for} \ i = 1 \ \textbf{to} \ \xi - 1 \ \textbf{do} \)
52: \( W'_i \leftarrow W'_i / m \mod N \)
53: \( W'_i \leftarrow W'_i / m \mod N \)
54: \( \textbf{end for} \)
55: \( K \leftarrow \text{mul}'(n - 1, Z, W) \)
56: \( K' \leftarrow \text{mul}'(n - 1, Z, W') \)
57: \( \textbf{for} \ i = 0 \ \textbf{to} \ n - 1 \ \textbf{do} \)
58: \( K_i \leftarrow (-1)^i K \cdot U_i \mod N \)
59: \( K'_i \leftarrow (-1)^i K' \cdot U'_i \mod N \)
60: \( \textbf{end for} \)
61: \( K \leftarrow K \cdot S \)
62: \( K' \leftarrow K' \cdot S \)
63: \( \textbf{for} \ i = 0 \ \textbf{to} \ n - 1 \ \textbf{do} \)
64: \( \text{if} \ \gcd(K_i, N) \neq 1 \ \textbf{then return} \ \gcd(K_i, N) \)
65: \( \text{else if} \ \gcd(K'_i, N) \neq 1 \ \textbf{then return} \ \gcd(K'_i, N) \Rightarrow \text{Factor found!} \)
66: \( \textbf{end if} \)
67: \( \textbf{end for} \)

6. CONCLUSIONS

In this paper, we proposed a new attack against the semi-smooth RSA subgroup of hidden order described in TCC 2005 that works in time and space complexity \( O(\sqrt{p}/2) \) respectively. While the best of the previous attack had complexity in \( O(\sqrt{p}) \).
The main part of our technique is to use the interval and the double walks on the equivalence class method. The experiment result shows that we can achieve the speedup by a factor extremely close to 50% using the new approach, which is the best performance one can achieve in practice.

REFERENCES

APPENDIX A

Algorithm 3: Newton interpolation: compute the polynomial \( F \) of degree less than \( n \) such that \( F(p_i) = v_i \), where \( p_i = q^i, 0 \leq i \leq n-1 \).

1: function NewtonInterpGeom \( (p_0, \ldots, p_{n-1}; v_0, \ldots, v_{n-1}) \)
2: \( q_0 \leftarrow 1; u_0 \leftarrow 1; w_0 \leftarrow v_0; \)
3: for \( i = 1 \) to \( n-1 \) do
4: \( q_i \leftarrow q_{i-1} \cdot p_{i-1}; \)
5: \( u_i \leftarrow u_{i-1} \cdot (q^i \cdot 1); \)
6: \( w_i \leftarrow v_i / u_i; \)
7: end for
8: \( G \leftarrow (\sum_{i=0}^{n-1} w_i x^i) \cdot (\sum_{i=0}^{n-1} (-x)^i q / u_i); \)
9: return Coeff\((G, 0)/ q_0, \ldots, Coeff\((G, n-1)/ q_{n-1}\));
10: end function

Algorithm 4: Newton evaluation: evaluate the polynomial \( F \) at all points \( p_i = q^i, 0 \leq i \leq n-1 \).

1: function NewtonEvalGeom \( (p_0, \ldots, p_{n-1}; a_0, \ldots, a_{n-1}) \)
2: \( q_0 \leftarrow 1; u_0 \leftarrow 1; g_0 \leftarrow a_0; \)
3: for \( i = 1 \) to \( n-1 \) do
4: \( q_i \leftarrow q_{i-1} \cdot p_{i-1}; \)
5: \( u_i \leftarrow u_{i-1} \cdot (q^i \cdot 1); \)
6: \( g_i \leftarrow q_i a_i; \)
7: end for
8: \( G \leftarrow (n-1) \sum_{i=0}^{n-1} g_i x^i \cdot (\sum_{i=0}^{n-1} u_i x^i); \)
9: return Coeff\((G, 0)/ q_0, \ldots, Coeff\((G, n-1)/ q_{n-1}\));
10: end function

Algorithm 5: Monomial to Newton basis, \( p_i = q^i, 0 \leq i \leq n-1 \).

1: function MtoN \( (p_0, \ldots, p_{n-1}; v_0, \ldots, v_{n-1}) \)
2: \( q_0 \leftarrow 1; u_0 \leftarrow 1; w_0 \leftarrow v_0; z_0 \leftarrow 1; a_0 \leftarrow v_0; \)
3: for \( i = 1 \) to \( n-1 \) do
4: \( p_i \leftarrow q_{i-1} \cdot p_{i-1}; \)
5: \( u_i \leftarrow u_{i-1} \cdot p_i / (1 - p_i); \)
6: \( w_i \leftarrow v_i / u_i; \)
7: \( z_i \leftarrow (-1)^i u_i / q_i; \)
8: \( a_i \leftarrow (-1)^i w_i q_i; \)
9: end for
10: \( G \leftarrow \text{mul}'(n-1, \sum_{i=0}^{n-1} z_i x^i, \sum_{i=0}^{n-1} a_i x^i); \)
11: return \( z_0 \cdot \text{Coeff}\((G, 0)/ \ldots, z_{n-1} \cdot \text{Coeff}\((G, n-1);\)
12: end function
Algorithm 6: Newton to Monomial basis, \( p_i = q^i, 0 \leq i \leq n - 1 \).

1: function NtoM \( (p_0, \ldots, p_{n-1}, a_0, \ldots, a_{n-1}) \) 
2: \( q_0 \leftarrow 1; v_0 \leftarrow a_0; u_0 \leftarrow 1; w_0 \leftarrow a_0; z_0 \leftarrow 1; \)
3: \( \text{for } i = 1 \text{ to } n - 1 \text{ do} \)
4: \( q_i \leftarrow q_{i-1} \cdot p_{i-1}; \)
5: \( v_i \leftarrow (-1)^{i} a_{q_i}; \)
6: \( u_i \leftarrow u_{i-1} \cdot p_i / (1 - p_i); \)
7: \( w_i \leftarrow v_i / u_i; \)
8: \( z_i \leftarrow (-1)^{i} u_i / q_i; \)
9: \( \text{end for} \)
10: \( G \leftarrow \text{mul}(n-1, \sum_{i=0}^{n-1} u_i x^i, \sum_{i=0}^{n-1} w_i x^i) \)
11: \( \text{return } z_0 \text{Coeff}(G, 0), \ldots, z_{n-1} \text{Coeff}(G, n-1) \)
12: end function

Jiang Weng (江文) is a Ph.D. student in State Key Laboratory of Mathematical Engineering and Advanced Computing, China. He received his Master degree in Zhengzhou Information Science and Technology Institute in 2012. His research focuses on elliptic curve cryptography and discrete logarithm problem.

Yunqi Dou (豆允旗) is a Ph.D. student in State Key Laboratory of Mathematical Engineering and Advanced Computing, China. He received his B.S. and M.S. degrees from the Zhengzhou Information Science and Technology Institute, China in 2010 and 2013. His research fields include Elliptic curve cryptography and side-channel attack.

Chuangui Ma (马传贵) received his Ph.D. degree in Applied Mathematics from Zhejiang University, China in 1999. He is currently a Professor in the State Key Laboratory of Mathematical Engineering and Advanced Computing. His main research interests include information security and public key cryptography.