Factorization Attacks against RSA – Individual Project Contribution

Yuqing Liu (786354):

For the written report, I wrote sections for Continued Fraction Factorization method (CFFM) and the comparison among CFFM, Quadratic Sieve Algorithm, and Shor's Algorithm. For the presentation, I was responsible for the introduction and the part for CFFM. As a team member, I reviewed others' section and gave feedback.

Shuai Wang (830166):

During our project, my teammates and I worked together and helped each other. For presentation, I was responsible for preparing the slides 5-6 about Quadratic Sieve algorithm. For the report, I wrote the abstract, introduction and Quadratic Sieve algorithm part. For report review, I read the whole report and provided feedbacks to my teammates.

Wenjun Zhou (771229):

I was responsible for preparing the slide 8-10 (from Shor's algorithm part to the end). I also put all our parts together to make sure the slides look consistent. In the project report part, I was responsible for writing the Shor's algorithm, the recommendation and the conclusion sections. I added a table in the comparison section and I also proofread the whole report and made changes when it is necessary to make sure it looks like written by one person.

Factorization Attacks against RSA

Yuqing Liu; Shuai Wang; Wenjun Zhou

{yuqingl8@student.unimelb.edu.au},
{shuaiw6@student.unimelb.edu.au},
{wenjunz1@student.unimelb.edu.au}

Abstract. This report introduces three representative factorization algorithms in different time against RSA encryption, specifically Continued Fraction Factorization Method (CFFM), Quadratic Sieve Algorithm (QS), and Shor's Algorithm. Both CFFM and QS are developed based on Fermat's method and takes exponential time to factor an *n*-bit prime using traditional computers, while Shor's Algorithm takes polynomial time by using quantum computers. By comparing these algorithms, we recommend RSA cryptosystem use at least 2048-bit encryption modulus. This report also encourages the creation of a new encryption scheme to prevent quantum computers attacks in the future.

Keywords: RSA; factorization; Continued Fraction Factorization Method; Quadratic Sieve Algorithm; Shor's Algorithm

1 Introduction

The RSA public-key cryptosystem, proposed by Ron Rivest, Adi Shamir and Len Adleman in 1977 [1], is widely used for secure data transmission. It masks the original information and makes it only be readable to the authorised parties. This mechanism relies on the hard problem of factoring public key *n*. In practice, *n* is typically a large prime which makes the adversaries take infeasible long time to get the prime factors of *n*. Nonetheless, smart researchers have been coming up with efficient algorithms to get the answer in a shorter time. This way of cracking RSA cryptosystems is called factorization attack. In this report, we introduce three representative integer factorization algorithms in different stages of RSA encryption, namely Continued Fraction Factorization Method (CFFM), Quadratic Sieve Algorithm (QS) and Shor's Algorithm.

2 Factorization Methods

2.1. Continued Fraction Factorization Method

Continued Fraction Factorization Method (CFFM) proposed by Morrison and Brillhart in 1975 is based on one of the earliest factoring algorithms, Fermat's algorithm [2]. Fermat is regarded as the father of number theory by many people and Fermat's algorithm has excellent performance when a factor of a number is close to the square root of the number [3]. For a better understanding of CFFM, the basis of Fermat's algorithm is given in Formula 2.1.1.

$$n = a^2 - b^2 = (a + b)(a - b) \tag{2.1.1}$$

It shows that for a given number *n*, if we can find two square numbers, e.g. a^2 and b^2 that make the difference between them is equal to *n*, then both the sum of *a* and *b* and the difference between *a* and *b* are factors of the number *n*. The procedure of Fermat's algorithm based on this formula is shown below.

Steps:

1. Let *x* be $\lceil\sqrt{n}\rceil$, where $\lceil\sqrt{n}\rceil$ is the smallest integer larger than $\sqrt{n}$.
2. Test whether the difference between x^2 and *n* is a square number denoted as b^2.
3. If it is, two factors of *n* are found, i.e. $(x - b)$ and $(x + b)$.
4. If it is not, let *x* be $(x + 1)$ and go to step 2.

Factorization with this method is only suitable for small integers and requires $O(n)$ steps, where *n* is the number to be factored [4]. Maurice Kraitchik introduces "an enhancement of Fermet's difference of squares technique" [3], that is to search for pairs of a and b that satisfy $a^2 \equiv b^2 \ mod \ n$, where *n* is the integer to be factored and *a* and *b* both relatively prime to *n*. Thus, either $gcd(a - b, n)$ or $gcd(a + b, n)$ is a non-trivial factor of *n* [5]. This is "the basis of most modern factoring algorithms" and allows sub-exponential time complexity [3]. Based on this formula, CFFM speeds up the process of finding suitable pairs of square numbers and was "the fastest factorization method before Quadratic Sieve" [6].

The key for CFFM is to find suitable square numbers that satisfy $a^2 \equiv b^2 \ mod \ n$. Convergents of a simple continued fraction expansion of $\sqrt{n}$ are used for finding such numbers. We will introduce related mathematical knowledges before we give the description of this method.

A Continued Fraction is a number x expressed in Formula 2.1.2.

$$x = a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \ddots}}} \tag{2.1.2}$$

A simple continued fraction is when b_i is equal to 1 for all *i*. As we only focus on rational parts of a number, i.e. a_i for all *i* in the above expression, the number *x* can be written

as another notation $[a_0, a_1, a_2, a_3, \dots]$. These rational terms are called convergents. For a positive irrational number, the number of rational terms is infinite. We can use the first *k* successive convergent to approximately express this number, i.e. $[a_0, a_1, \dots, a_k]$. A corresponding theorem is described as in Theorem 1.

Theorem 1. Assume that $p_{-2} = 0, p_{-2} = 1, \dots, p_k = a_k p_{k-1} + p_{k-2}$, and $q_{-2} = 1, q_{-2} = 0, \dots, q_k = a_k q_{k-1} + q_{k-2}$. Then $[a_0, a_1, \dots, a_k] = \frac{p_k}{q_k}$. $\frac{p_k}{q_k}$ is called the kth convergent of $[a_0, a_1, a_2, a_3, \dots]$ [4].

Using CFFM to factor a number *n*, we need first express $\sqrt{n}$ in the form of a simple continued fraction to generate a number Q_k whose definition is shown in Definition 1.

Definition 1. Assume $\frac{p_k}{q_k}$ is the *k*th convergent of *n*. Then $(-1)^k Q_k = {p_{k-1}}^2 - n{q_{k-1}}^2$.

It implies that $(-1)^k Q_k \equiv {p_{k-1}}^2 \ mod\ n$. Using Theorem 1, a set *Q* i.e. $\{Q_1, Q_2, \dots, Q_k\}$ can be constructed. If we can find a certain number of elements from *Q* and the product of them is a square number, i.e. $\prod_i (-1)^i Q_i$ is a square number, $\prod_i (-1)^i Q_i$ and corresponding $\prod_i {p_{i-1}}^2$ satisfy the formula $a^2 \equiv b^2 \ mod\ n$ introduced by Kraitchik. Thus, $\gcd(\sqrt{\prod_i (-1)^i Q_i} - \prod_i p_{i-1}, n)$ is highly likely a non-trivial factor of *n*. If we cannot find items from *Q* to yield a square number when all these items multiply together or $\gcd(\sqrt{\prod_i (-1)^i Q_i} - \prod_i p_{i-1}, n)$ is not a non-trivial factor of *n*, we can increase *k* for larger convergent.

To select an appropriate subset of *Q* for a perfect square number, we factor each element Q_i in *Q* using trivial division and express it in the form of a prime power factorization. That is to test every prime less than $\sqrt{Q_i}$ to see if it is a factor of Q_i. According to primes factored from Q_i, we determine a subset of *Q* to construct a square number.

An example used by Morrison and Brillhart when they first introduced CFFM is to factor 13290059 denoted as *n* [2]. Table 2.1.1 contains Q_i and corresponding p_{i-1}.

To generate this table, $\sqrt{13290059}$ is first expressed in the form of a simple continued fraction. Using Theorem 1, in the process of computing the 52nd convergent of $\sqrt{13290059}$, we compute $\{p_0, p_1, \dots, p_{52}\}$ first and thus construct a set *Q* containing $\{Q_1, Q_2, \dots, Q_{52}\}$. Using trivial division, each element in set *Q* is factored shown in column Q_i *factored*. Through this column, we observe that $Q_5 = 2 \times 5^2 \times 41$, $Q_{22} = 41 \times 113$, and $Q_{23} = 2 \times 113$. The product of them is $2^2 \times 5^2 \times 41^2 \times 113^2$. It is obvious that their product is a square number. The result of $Q_5 \times Q_{22} \times Q_{23} \ mod\ n$ is 46330^2. Multiply their corresponding p_{i-1}, i.e. p_4, p_{21} and p_{22}. The result of $p_4 \times p_{21} \times p_{22} \ mod\ n$ is 1469504. Then we can find that *gcd*(*h*, 13290059) where $h \in$(46330, 1469504), i.e. 4261 is a factor of 13290059.

Table 2.1.1. Q_i and Corresponding p_{i-1} of 13290059

i	Q_i	$p_{i-1} (mod\ n)$	Q_i *factored*
-1	13290059	0	…
0	1	1	…
1	4034	3645	2×2017
2	3257	3646	3257
3	1555	7291	5×311
4	1321	32810	1321
5	2050	171341	$2\times5^2\times41$
10	1333	6700527	31×43
22	4633	5235158	41×113
23	226	1914221	2×113
26	3286	11455708	2×31×53
31	5650	1895246	$2\times5^2\times113$
40	4558	3213960	2×43×53
52	25	2467124	5^2

The time complexity for CFFM is $O(n^{\sqrt{(2+o(1))\,log\,log\,n/\,log\,n}})$ [7]. However, when Brillhart and Morrison came up with this method, they did not give the analysis of time complexity for this method at the same time. In Wunderlich's paper, he gives an estimation of the running time of this method [8]. According to his estimation, for a number *n*, the time consumed to factor this number is not over $n^{\varepsilon(n)}$ where $\varepsilon(n)$ tends to zero as *n* approaches to positive infinity. Wunderlich also uses more than one thousand numbers between fifteen and forty-two digits to test the actual time consumed for factorization using CFFM. The result shows that the estimation of the running time for a number *n* in this range is cn^{152} where *c* is a constant. In 1981, Dixon gave a rigorous analysis of the time complexity for a slightly different version of CFFM, i.e. Dixon's algorithm. It shows the steps for his algorithm is about $exp = (\sqrt{2\,log\,n\,log\,log\,n})$ [9]. CFFM is the first factorization method that allows sub-exponential running time, but its limitation is that it can only process a number with around fifty digits at most [10].

2.2. Quadratic Sieve Algorithm

In this section, we will introduce another integer factorization algorithm based on Fermat's Method, called Quadratic Sieve Algorithm (QS). Carl Pomerance proposed it in 1981 [11]. QS is still the fastest known algorithm for integers in the range of 40 to 100 decimal digits [12]. We give some definitions below for a better understanding of QS algorithm.

Definitions:

i. ***B-Smooth*:** a number is a *B-Smooth* number if all its prime factors are less than or equal to *B*. The number *B*, called Smoothness Bound, is used to limit the length of exponent vector and the number of vectors required to construct *a* and *b*. E.g. $300 = 2^2 \times 3 \times 5^2$. 300 is a *5-Smooth* number or *7-Smooth* number.

ii. **$\pi(B)$:** indicate how many primes are less than *B*. E.g. $\pi(5)=2$.

iii. **Exponent vector:** factor a number, and then take all exponents of prime factors *mod 2* to generate a vector. E.g. $\overrightarrow{v_{(300)}} = (2,1,2)\ mod\ 2 = (0,1,0)$.

iv. **Gaussian Elimination:** is an algorithm for solving systems of linear equation. In our case, it can be used to find some exponent vectors whose sum is zero vector.

As mentioned earlier, Fermat's method described that an odd integer *n* can be factored if there exist two positive integers satisfying $a^2 \equiv b^2$ (*mod n*) and $a \not\equiv \pm b$ (*mod n*). Either *gcd*(*n*, *a* + *b*) or *gcd*(*n*, *a* – *b*) is the prime factor of *n* [13]. The main challenge is to find *a* and *b*. The idea of QS is to input a small quantity of possible numbers of *a* to calculate $a_i{}^2\ mod\ n$, and to check whether the result is a *B-Smooth* number and then to choose a sequence of such results to construct *a* and *b*.

To find two prime factors *a* and *b* of *n*, the main steps are described as follow.

Steps:

1. Choose a smoothness bound B.
2. Set the range of $a_i \in [\lceil\sqrt{n}\rceil - K, \lceil\sqrt{n}\rceil + K]$ where *K* is a constant.
3. Compute $b_i = (a_i)^2\ mod\ n$. Select those b_i that is a *B-Smooth* number.
4. Factor b_i and obtain its exponent vectors.
5. Find $\pi(B) + 1$ exponent vectors which add to the zero vector with Gaussian Elimination.
6. Multiply b_i corresponding to such set of vectors together which yields a square b^2, and take the square root of the square to yield *b*.
7. Multiply a_i corresponding to those b_i to yield *a*.
8. Compute $p = gcd(n, a + b)$ and $q = gcd(n, a – b)$. If *p* or *q* is trivial (1 or *n*), go back to step 5, choose another subset of vectors.
9. $n = p \times q$. The number *n* has been factored.

To have a deep understanding of QS, we take a concrete example for $n = 87463$. We choose the smoothness bound $B = 37$. Table 2.2.1 shows all prime number basic factor bases which are less than *B*.

Table 2.2.1. Basic factor base for $B = 37$

prime	2	3	5	7	11	13	17	19	23	29	31	37

However, not all of primes above can be in the final factor base. Using the concept of quadratic residues [14] to determine which prime can be in factor base. If $\left(\frac{n}{p}\right) = 1$, it means *p* does not divide *n*, and can be a prime factor in final factor base. If $\left(\frac{n}{p}\right) = 0$, it

means *p* can divide *n*, and cannot be a prime factor in final factor base. If $\left(\frac{n}{p}\right) = -1$, it means *p* cannot divide *n* and is non-residuosity, cannot be a prime factor in final factor base. Thus, choose those factors with $\left(\frac{n}{p}\right) = 1$ to obtain the final factor base (Table 2.2.2).

Table 2.2.2. Factor base with quadratic residues

prime	2	3	5	7	11	13	17	19	23	29	31	37
n/prime	1	1	-1	-1	-1	1	1	1	-1	1	-1	-1

Thus, the final factor base is 2, 3, 13, 17, 19 and 29.

We now set a range of $a_i \in [\lceil\sqrt{n}\rceil - 30, \lceil\sqrt{n}\rceil + 30]$ where $a_i \in [265, 325]$. Compute $b_i = (a_i)^2 \ mod \ n$ on this range, and check if b_i can be factored over the final factor base (if it is a *37-Smooth* number). We found 265, 278, 269, 299, 307 and 316 are 37-Smooth number, and represent them with exponent vector in Table 2.2.3.

Table 2.2.3. Exponent vectors

a_i	b_i	-1	2	3	13	17	19	29
265	**70225**	1	1	1	0	1	0	0
269	**72361**	0	0	0	0	1	0	0
278	**77284**	1	0	1	1	0	0	1
299	**1938**	0	1	1	0	1	1	0
307	**6786**	0	1	0	1	0	0	1
316	**12393**	0	0	0	0	1	0	0

By using Gaussian Elimination, we found exponent vectors corresponding to 265, 278, 296 and 307 adding to zero vector, which is $\overrightarrow{v_{(265)}} + \overrightarrow{v_{(278)}} + \overrightarrow{v_{(296)}} + \overrightarrow{v_{(307)}} = \vec{0}$. Hence, we can compute *a* and *b* with Formula 2.2.1 and 2.2.2.

$$a = 265 \cdot 278 \cdot 296 \cdot 307 \ (mod \ 87463) = 34757 \qquad (2.2.1)$$

$$b = \sqrt{(265^2 - 87463) \cdot (278^2 - 87463) \cdot (296^2 - 87463) \cdot (307^2 - 87463)} \ (mod \ 87463) = 28052 \qquad (2.2.2)$$

So, $p = gcd(87463, 34757 + 28052) = 587$ and $q = gcd(87463, 34757 - 28052) = 149$. Thus, $n = p \times q = 87463 = 587 \times 149$.

The time complexity of QS is represented with Formula 2.2.3 as follows:

$$e^{(1+o(1))\sqrt{ln\, n\, ln\, ln\, n}} \qquad (2.2.3)$$

QS is a general-purpose factorization algorithm, which refers to its running time only depends on the length of integer to be factored not on special property of the integer. It is still the fastest known algorithm for integers in the range of 40 to 100 decimal digits.

However, when QS has been used to factor integers above 100 decimal digits, the performance is poor. This is because we cannot easily determine whether a polynomial b_i is *B-smooth* by factoring it when the number *n* to be factored is larger than 100 decimal digits.

2.3. Shor's Algorithm

Because all the current integer factorization algorithms in a classical computer can only run in an exponential time complexity, when the sender use a long key (usually longer than 2048 bits) to encrypt the message, the attacker will not be able to decrypt the message in a feasible time with current computational power. In this section, we are going to introduce an innovative algorithm which is called Shor's algorithm proposed by Peter Shor in 1994 [15]. This algorithm requires the use of a quantum computer to solve the integer factorization problem in polynomial time.

The integer factorization problem has been explained in the introduction section. RSA encryption is based on the challenge of integer factorization problem which takes exponential computational time to decompose the *n* into two primes. It means that the longer bits the *n* has, the exponential longer time it will take to get the decomposition of the *n*. Shor's algorithm solves the exponential time complexity problem by using Quantum Fourier Transformation (QFT). QFT is one kind of discrete Fourier transformation that decomposes the original equation into the product of a number of simple unitary matrixes. One may still ask why this has relationship with cryptography. Indeed, the QFT has no direct relationship with cryptography; however, the QFT can effectively work out the periodicity of modulo arithmetic functions, which is the key to crack RSA [16]. Since the focus of our paper is not on quantum mechanics, we will mask the QFT process and proof, and only provide a general idea of Shor's algorithm.

There is only one solution (one unique tuple of *p* and *q*) to the modulus *n* in RSA, and this property is a clue for cryptanalysts to find the answer of the semiprimes. We only need to find either a non-trivial *p* or a non-trivial *q* which can divide *n* evenly to solve the $n = p \times q$ problem. Pick a random value *a* which does not share any factors with *n*, then for some integer *x*, $a^x \equiv 1 \ (mod\ n)$ holds. Shor's algorithm consists of two parts: one is in classical computing, the other falls into quantum computing, and the whole processes are simplified as described below.

Steps:

1. Randomly pick an *a*;
2. Use a quantum computer to get the period (*d*) of a^x mod n;
3. If the *d* is odd, go back to step 1; if the *d* is even, go to step 4;
4. If $\left(a^{\frac{d}{2}} + 1\right)$ is divisible by *n*, go back to step 1; if $\left(a^{\frac{d}{2}} + 1\right)$ is not divisible by *n*, go to step 5;

5. The factor p and q is the product of $gcd(n, \left(a^{\frac{d}{2}} - 1\right))$ and $gcd(n, \left(a^{\frac{d}{2}} + 1\right))$, and this can simply be achieved by using the Euclidean algorithm.

According to N. David Mermin, the probability of going through the whole algorithm steps without going back to step 1 is 50% [16]. The time complexity of Shor's algorithm is represented as in Formula 2.3.1 [17].

$$O((\log n)^2(\log\log n)(\log\log\log n)) \tag{2.3.1}$$

IBM has successfully implemented the Shor's algorithm on a 7-qubit quantum computer in 2001 [18]. However, the implementation only demonstrated the factorization of 15 (= 3 × 5). After IBM's demonstration, there were several different implementations which successfully factored some small semiprimes. Nevertheless, the largest integer that was factored by a quantum computer by using Shor's algorithm is only 21 [19]. Interestingly, some researchers theoretically show that getting the factorization of 291,311 on a 6-qubit quantum computer is possible through another optimized algorithm [20].

As indicated before, the algorithm itself seems no limitations in terms of solving the integer factorization problem. Based on the fact that there are no presence of powerful quantum computers exist to run Shor's algorithm and factorize large integers, the limitation of this algorithm is only the absence of real world applications.

3 Comparison

Both CFFM and QS are based on the advanced technique introduced by Kraitchik to find square numbers that satisfy $a^2 \equiv b^2 \ mod \ n$. This allows that CFFM and QS both have sub-exponential running time. CFFM as the first factorization method using this technique requires $\exp(\sqrt{2\log n \log\log n})$ steps at best. However, QS optimizes the search of *B-smooth* integers and thus halves the steps to $\exp(\sqrt{\log n \log\log n})$. As modern factorization methods, CFFM and QS can factor relatively large numbers with up to 100 digits and have no requirement of the form of the number to be factored [2]. Nonetheless, algorithm used is not the only contributing factor to the speed of factorization. This is also related to the quantity and performance of computers used. Compared to traditional computers, the advent of quantum computers indicates a new age for factorization. Shor's algorithm successfully reduces the time for factorization to polynomial time using a quantum computer.

We summarize all the features in Table 3.1 as shown below.

Table 3.1 Comparison between algorithms complexity and fields

Algorithms	Time complexity	Fields
CFFM	$\exp(\sqrt{2\log n \log\log n})$	Classical Computing
QS	$\exp(\sqrt{\log n \log\log n})$	Classical Computing
Shor's algorithm	$O((\log n)^2(\log\log n)(\log\log\log n))$	Quantum Computing

4 Recommendation

Despite the fact that QS is one of the fastest integer factorization algorithms, it still has exponential time complexity. Simply extending the key length can prevent integer factorization attacks from happening. Here, we go through the whole factorization attack history and the changes of key length requirement to demonstrate the possible future changes.

The most recent factorization attack cracked 768-bit RSA in 2009 by using the general number field sieve algorithm [21]. For the other earlier attacks, we refer to the summary from Stefania Cavallar et al. work in Table 4.1 [22].

Table 4.1. Factoring records since 1970

Decimals Number	Date or Year	Algorithm Used	Effort (MIPS years)
39	Sep 13, 1970	CF	N\A
50	1983	CF	N\A
55-71	1983-1984	QS	N\A
45-81	1986	QS	N\A
78-90	1987-1988	QS	N\A
87-92	1988	QS	N\A
93-102	1989	QS	N\A
107-116	1990	QS	275 for C116
RSA-100	Apr 1991	QS	7
RSA-110	Apr 1992	QS	75
RSA-120	Jun 1993	QS	835
RSA-129	Apr 1994	QS	5000
RSA-130	Apr 1996	NFS	1000
RSA-140	Feb 1999	NFS	2000
RSA-155	Aug 1999	NFS	8400

Note. From Factorization of a 512-Bit RSA Modulus, by Cavallar S. et al., 2000, Springer, Berlin, Heidelberg. Copyright 2013 by Springer, Berlin, Heidelberg. Reprinted with permission.

As stated in the table, although the effort that we need to spend on cracking RSA increases exponentially and the time gap between cracking different level of semiprimes increases, we still observe researchers working on this field. This may be a warning that longer RSA encryption key may be broken in the near future. According to Barker and Roginsky, in 2022, the bit length of *n* should be at least 2000, and after that, an at least 3000-bit key is required for high sensitive data encryption. In 2011, U.S. Department of Commerce published a transition standard for suggesting the length of RSA encryption key, and they suggested that the proper security strength is based on the sensitivity of the data that requires to be protected [23]. The RSA key agreement schemes that they proposed was: the use of 1024-bit long *n* should be deprecated from 2011 to 2013. After 2013, using a 2048-bit *n* is considered as acceptable.

However, simply extending the key size should not be an ultimate solution, especially when researchers are working hard on building quantum computers. With the increasing popularity of research in quantum field, powerful quantum computers will be built to crack RSA eventually. By that time, no matter how long the key size is, RSA encryption will no longer be safe. Thus, this research would encourage a new scheme of encryption to be developed soon, for example, post-quantum cryptography is a new emergence. Since the focus of this paper is not on quantum mechanics, we would not develop further suggestions on that.

5 Conclusion

Three representative integer factorization algorithms were introduced in this paper and it is still a challenge both in classical and quantum computing fields to successfully apply a serious RSA attack. With the growing computational power, the key length of RSA also keeps growing. RSA can be considered as a secure encryption function until quantum computer builds. Nevertheless, it is inevitable that quantum computers will be developed in the future, and when that happens, there needs to be a new encryption scheme ready to guard against the information security.

Reference

[1] W. Stallings, Cryptography and network security. Boston: Pearson, 2017.

[2] M. A. Morrison and J. Brillhart, "A Method of Factoring and the Factorization of F_7," Mathematics of Computation, vol. 29, pp. 183-205, 1975.

[3] F. J. Smith, "A Brief History of Factorization Techniques," University of Washington, 2006.

[4] Barnes Connelly, "Integer factorization algorithms," Oregon State University, 2004.

[5] C. Pomerance, "A tale of two sieves," Notices Amer.math.soc, vol. 43, pp. 1473--1485, 1996.

[6] "Fermat's Factorization Method -- from Wolfram MathWorld", Mathworld.wolfram.com, 2017. [Online]. Available: http://mathworld.wolfram.com/FermatsFactorizationMethod.html.

[7] C. Pomerance, "Analysis and comparison of some integer factoring algorithms," Computational Methods in Number Theory Part, 1982.

[8] M. C. Wunderlich, A running time analysis of Brillhart's continued fraction factoring method: Springer Berlin Heidelberg, 1979.

[9] J. D. Dixon, "Asymptotically fast factorization of integers," Mathematics of Computation, vol. 36, pp. 255-260, 1981.

[10] J. R. Steuding and R. Slezeviciene, "Factoring with continued fractions, the Pell equation, and weighted mediants," Fiz.mat.fak.moksl.semin.darb, vol. 6, pp. 120-130, 2003.

[11] C. Pomerance, "The quadratic sieve factoring algorithm", Springer Berlin Heidelberg, vol. 209, pp. 169–182, 1985.

[12] C. Seibert, "Integer Factorization using the Quadratic Sieve", University of Minesota, 2011.

[13] R. S. Lehman, "Factoring large integers", Math. Comp., vol. 28, pp. 637-646, 1974.

[14] D. M. Bressoud, S. Wagon, "A course in computational number theory", Key College Pub., 2000.

[15] P. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", SIAM Journal on Computing, vol. 26, no. 5, pp. 1484-1509, 1997.

[16] N. Mermin, Quantum computer science. Cambridge: Cambridge University Press, 2007.

[17] D. Beckman, A. Chari, S. Devabhaktuni and J. Preskill, "Efficient networks for quantum factoring", Physical Review A, vol. 54, no. 2, pp. 1034-1063, 1996.

[18] L. Vandersypen, M. Steffen, G. Breyta, C. Yannoni, M. Sherwood and I. Chuang, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance", Nature, vol. 414, no. 6866, pp. 883-887, 2001.

[19] E. Martín-López, A. Laing, T. Lawson, R. Alvarez, X. Zhou and J. O'Brien, "Experimental realization of Shor's quantum factoring algorithm using qubit recycling", Nature Photonics, vol. 6, no. 11, pp. 773-776, 2012.

[20] N. S. Dattani, N. Bryans, "Quantum factorization of 56153 with only 4 qubits", arXiv:1411.6758v3 [quant-ph], 2014.

[21] "768-bit RSA encryption cracked", Infosecurity, vol. 7, no. 1, p. 7, 2010.

[22] S. Cavallar, B. Dodson, A. Lenstra, W. Lioen, P. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam and P. Zimmermann, "Factorization of a 512-Bit RSA Modulus", Advances in Cryptology — EUROCRYPT 2000, pp. 1-18, 2000.

[23] E. Barker and A. Roginsky, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths", 2011.