# NP-completeness Reduction for Semiprimes Factorization Problem

Yen-Lung Lai

Monash University Malaysia,
Jalan Lagoon Selatan, Bandar Sunway, 47500 Subang Jaya, Selangor
yenlung.lai@monash.edu

**Abstract.** We show a reduction of integer (semiprimes) factorization problem to a $NP$-complete problem related to coding. Our results rigorously imply the existence of a quantum computer could possibly devastate existing security system relies on NP-hard problem.

**Keywords:** · Integer Factorization · NP-complete Problems· Complexity Theory

## 1 Introduction

Integer factorization is a commonly studied problem and has been well applied to cryptographic primitive like asymmetric cryptographic system (i.e., RSA). There are some notable works on reducing the integer factorization problem to other computational problems [1] [2] [3], named as low exponent RSA and full RSA. These computational problems rely on the existence of the straight-line programs, which are sub-class of generic ring algorithm that can perform addition, subtraction, and multiplication, and multiplicative inverse of modulo $N$. However, the later works by Boneh and Vekatesan[4] have shown that there are no such straight-line programs reducing integer factorization problem to low exponent RSA or full RSA problem, unless the integer factorization problem itself is easy. Apart from this, the work by Jager and Schwenk [5] have analysed large class of subset membership problem related to integer factorization, again, shown that there is no algorithm solving these kinds of problem efficiently unless integer factorization is easy. Their analysis has included the well-known quadratic residuosity problem and subgroup decision problem.

### 1.1 The NP-complete problems

The problem of class $P$ corresponds to polynomial-time (deterministic) algorithm. It defines a set of problems which can be solved in computational time polynomial to the input size. The problems over class $P$ are generally regarded as intractable, i.e., solving linear equations, finding the minimum cut in flowgraph, etc.

Alternately, the class $NP$ refer to the problems solvable by a non-deterministic algorithm whose running time is bounded by a polynomial in the input size. A

non-deterministic algorithm is one which, may produce different results in different courses run (simultaneously). The consequence of different courses run would be the exponential growth of the number of possible solutions, and the algorithm is said to solve the problem given one of the solutions produces the correct answer. Above description explained that $NP$ corresponds to *non-deterministic polynomial-time* algorithm.

Showing that the class of problem $P \subseteq NP$ is trivial since both solutions for $NP$ and $P$ are verifiable in polynomial time. However, the proof of the converse whether $NP = P$ remains a central open question of computer science. The *Cook-Levin theorem* (in 1971) has proved that *any* problem in $NP$ can be reduced to the satisfiability problem. Given a polynomial-time algorithm can be used to solve the satisfiability problem, so would any other $NP$ problem like travelling salesman problem, integer programming problem, etc. However, a lot of researchers have work on these $NP$ problems many years without producing an efficient (running in polynomial-time in the input size) algorithm for any one of them. In later Krap [9] showed that the satisfiability problem itself can be reduced to other $NP$-problem. These class of problem is now called the $NP$-complete class, corresponds to the hardness problem in $NP$. If anyone of the NP-complete problem possesses polynomial-time algorithm, so does every $NP$ problem hence $P = NP$.

Up till now, there are literally more than hundreds NP-complete problem can be found in book [10]. No one has ever found a polynomial-time algorithm for any one of these NP-complete problem. It is natural to believe that $P \neq NP$.

It is an interesting exploitation over the integer factorization problems, which is commonly believed that this problem is in the class of $NP$. Researchers have been trying to come out with a more efficient factorization algorithm. Despite a long time of researching, to our best knowledge, there have no existing any reduction prove for integer factorization problem to be in problem of class either $NP$-hard or $NP$ complete. Even so without efficient algorithm to solve the integer factorization problem, there have no reason to believe that factoring is hard other than the fact that many people have failed to solve it. Therefore, the exact complexity of factoring integers (specifically its decision version) remains as a major open question. Recent result obtained by Shor [6], showing that integer factorization problem can be done in polynomial time by reducing it to the order finding of the multiplicative group (mod $N$) using a quantum computer.

In this correspondence, we provide a concrete factorization algorithm depicted as algorithm PF. We show a reduction of the integer (semiprimes) factorization problem to an $NP$-complete problem related to coding. As factoring a semiprimes is refer to the hardest instance of integer factorization problem, our result implies that integer factorization problem is in problem class of $NP$-complete. Accompanies with the assertion of quantum algorithm can be used to solve integer factorization problem in polynomial-time, our results would denote the emergence requirement of post-quantum security system.

## 2 Integer(semiprimes) Factorization Problem

The semiprimes (integer) factorization problem can be described as follow.

$$N = pq \tag{1}$$

*Given $N \in \{0,1\}^n$, for all $p$ and $q$ to be some variables (primes number) s.t. $p, q \in \{0,1\}^k$, where $k < n$, find $p$ or $q$.*

Since $p, q \in \{0,1\}^k$, we shall have the worst-case value for $N$ to be $2^n = (2^k)(2^k)$, therefore:

$$k \leq \lceil \log(N)/2 \rceil , \tag{2}$$

must hold for all $p, q \in \{0,1\}^k$.

By letting $p = a - b$ and $q = a + b$, for all random variables $a, b \in \{0,1\}^k$ (not necessary to be primes), we can express the product of $p$ and $q$ from Eq. 1 as:

$$\begin{aligned} N &= (a-b)(a+b), \\ &= a^2 - b^2. \end{aligned} \tag{3}$$

In such a case, our factorization problem now turns to look for $a$ **and** $b$ instead of $p$ **or** $q$. Intuitively, doing so means we are looking for two integers $a$ and $b$ where their distance give rise to the desired prime number, $p$ or $q$.

To look for $a$ and $b$, there are only two cases we have to consider, which are when $a = b$, and $a \neq b$. Clearly, given $a = b$, we shall have the trivial solution:

$$N = a^2 - b^2 = 0.$$

On the other hand, to look for non-trivial solutions when $a \neq b$, we need to first define some error vector $e \in \mathcal{E}$ in some random distribution $\mathcal{E}$ over $\{0,1\}^k$. In particular, we refer all error vectors $e \in \mathcal{E}$ are of weight $\|e\| = \lfloor k\epsilon \rfloor$ parametrized some error rate $\epsilon \in [1/k, 1/2]$, s.t.:

$$\begin{aligned} a &= e \oplus 0^k, \\ b &= e \oplus 1^k, \end{aligned} \tag{4}$$

By doing so, $\|a \oplus b\| \geq 1$ always hold for $\epsilon \in [1/k, 1/2]$. Therefore $a \neq b$ by their uniqueness.

Viewing $a = (e \oplus 0^k)_{10}$ and $b = (e \oplus 1^k)_{10}$ both in based 10, with both side mod $(N)$ in Eq. 2, it follows that:

$$\begin{aligned} N \mod (N) &= a^2 - b^2 \mod (N), \\ 0 &= (e \oplus 0^k)_{10}^2 - (e \oplus 1^k)_{10}^2 \mod (N). \end{aligned} \tag{5}$$

Since the value of $a$ and $b$ can be determined by knowing $e$, therefore, our main goal is to look for such $e$ in $\mathcal{E}$ over $\{0,1\}^k$ with the help of strings $0^k$, $1^k$, and $N$. Under the event where the distance between $a$ and $b$ is known, determine the value of $\epsilon$ is straightforward since $\|a \oplus b\| = \|e\| = \lfloor k\epsilon \rfloor$. However, since $a$ and $b$ are unknowns without the knowledge of $p$ and $q$, one has no cruel about the weight of $e$. Nevertheless, a straightforward way is to brute-force such $e$ by using different possible values of $\epsilon$. Remark that since $\|e\| \geq 1$, the values of $a$ and $b$ must be different, thus, their corresponding solutions must be unique. Therefore two different solutions (a unique pair of $(a, b)$) can be obtained with a single trial for $e \in \mathcal{E}$. In such a case, it is enough to choose $\epsilon$ up to $1/2$ s.t. $\epsilon = \frac{1}{k}, \frac{2}{k}, \frac{3}{k}, \ldots, \frac{1}{2}$.

To ensure the correctness in getting $e$ so for $a$ and $b$, all possible weight of solutions for $\|e\|$ must be considered s.t. $\|e\| = 1, 2, \ldots, \lfloor k/2 \rfloor$, including the worst-case scenario where $a$ and $b$ come with hamming distance arbitrary close to $k/2$. More generally, the error vector $e \in \{\mathcal{E}_1, \ldots, \mathcal{E}_{\lfloor k/2 \rfloor}\}$ can be found over a family of error distributions $\{\mathcal{E}_1, \ldots, \mathcal{E}_{\lfloor k/2 \rfloor}\}$ parametrized by different value of $\epsilon$.

To minimise the number of brute-force trials, a smaller value of $\epsilon$ is always desired, which corresponds to smaller weight solution for $\|e\|$. Noting that the brute-force search can be carried out in parallel with different value of $\epsilon$, thus, the overall brute-force complexity can be expressed as:

$$\binom{k}{\lfloor k\epsilon \rfloor} \leq 2^{\lfloor kh_2(\epsilon) \rfloor} \leq 2^{kh_2(\epsilon)} = O(2^{kh_2(\epsilon)}), \tag{6}$$

where $h_2(\epsilon) = -\epsilon \log(\epsilon) - (1-\epsilon)\log(1-\epsilon)$ refers to the binary entropy function of rate $\epsilon$.

Recall that for every single trial of $e \in \{\mathcal{E}_1, \ldots, \mathcal{E}_{\lfloor k/2 \rfloor}\}$, we shall have a unique pair of $(a, b)$. If follows that we can look for their corresponding solutions $x$ and $y$ follow:

$$\begin{aligned} x &= a^2 \mod N, \\ y &= b^2 \mod N. \end{aligned}$$

Suppose $x$ or $y$ is a square, by computing the gcd of $N$ with the different (or sum) of $(a, x)$ and $(b, y)$, it follows that $\gcd(x+a)\gcd(x-a)|N$ or $\gcd(y+b)\gcd(y-b)|N$, hence the factor of $N$ can be obtained. Despite the output may be the trivial factor (1 or $N$), given that all possible weight of $\|e\| = 1, 2, \ldots, \lfloor k/2 \rfloor$ are considered, all possible solution of $(a, b)$ and $(x, y)$ should be covered. The collection of all the possible solutions is known as the congruence of square. In such a case, at least one of the solution would produce the non-trivial factor of $N$, which is $p$ or $q$. Note that computing the congruence of square is indeed a common technique used for many integer factorization method such as the Fermet's factorization, quadratic sieve and Dixon's factorization.

### 2.1 The Reduction

Our reduction utilised one of the coding problem which has proven to be $NP$-complete by Berlekamp et al., [7] (through reduction to the *three dimensional matching problem*). More precisely, the problem can be formalised into a decision problem as follow:

**Input**: *A binary matrix* $H$ *and non-negative integer* $z > 0$
**Problem**: *Is there exits a vector* $w$ *of hamming weight* $\|w\| \leq z$ *such that* $Hw = 0$

Based on above problem, we can always express the error vector in Eq. 4 with some $k \times n$ binary matrix $H$ and random binary string $w \in \mathbb{F}_2^n$, s.t.:

$$a = e \oplus 0^k = Hw \oplus H0^n = H(w \oplus 0^n),$$
$$b = e \oplus 1^k = Hw \oplus (1^k \oplus H0^n) = 1^k \oplus H(w \oplus 0^n).$$

With such expression, one can easily verify that the process of looking for $e$ is equalvalent to look for $w$ given $H$ and $a$ (or $b$). In reverse, if one could factor $N$ and yield $p$ (or $q$), he/she should be able to determine $a$ and $b$ easily, so for $e$ as well. Once $e$ is known, determine its weight $\|e\|$ is straightforward, then the answer for the above problem can be obtained immediately. This showcase a reduction of our problem in looking for $e$ to an $NP$-complete problem [8] related to coding.

### 2.2 The Complexity

Observe that the brute-force complexity is bounded by order $O(2^{kh_2(\epsilon)})$. To make sure our brute-force trials can be done in an efficient manner, we hope to show that $2^{kh_2(\epsilon)} \leq 2^{\log(\mathsf{poly}(n))} = \mathsf{poly}(n)$. Clearly, this can be done if:

$$kh_2(\epsilon) \leq \log(\mathsf{poly}(n)). \tag{7}$$

In viewed of this, the efficiency of $\mathsf{PF}$ can be evaluated based on any polynomial function of $n$ to determine the maximum value of $\epsilon$. For instance, let $\mathsf{poly}(n) = n^3+n+3$, the efficiency claim for $\mathsf{PF}$ only holds when $2^k \leq n^3 + n + 3$. This means one could choose a higher value of $\epsilon$ (given $k$) compare to $\mathsf{poly}(n) = n+1$. This result also implies any polynomial function can only be evaluated (or verified) efficiently given their entropy is larger or equal to the Shannon entropy of some error rate $\epsilon$.

Since $2^{kh_2(\epsilon)} \leq 2^{kh_2(1/2)} = 2^k$ which is maximum when $\epsilon = 1/2$. This implies there should have some worst-case polynomial function which require $2^k$ number of brute-force trials. Viewed this way, the error rate $\epsilon$ indeed determined the number of polynomial function can be evaluated efficiently. To be specific, to make sure **all** polynomial functions $\mathsf{poly}(n) \in \{0,1\}^n$ can be evaluated or verified

efficiently, we must have $2^k$ brute-force trials with some $k < n$. Relaxing such requirement with $\epsilon > 0$ would allow us to evaluate more polynomial function (with smaller order) but indeed reducing the number of function can be evaluated or verified in an efficient manner.

As a summary, given $\epsilon > 0$ and $k < n$, we can evaluate and verify any polynomial function efficiently only if:

$$2^{kh_2(\epsilon)} \leq \mathsf{poly}(n). \tag{8}$$

For instance, given $n = 16$, for $n \geq 2k$ (see Eq. 2), we shall have maximum $k = 8$. Suppose we trial all $\epsilon = 1/k, \ldots, 1/2$ (up to $1/2$), hence $2^k = 256 \leq \mathsf{poly}(n)$ is needed for efficient evaluation over $\mathsf{poly}(n)$. Let $\mathsf{poly}(n) = n^x \geq (2k)^x$, then the minimum value of $x \geq 2$, thus factoring $p$ and $q$ can be done efficiently in $O(n^2)$. Noting that the order of complexity will increase with higher value of $n$. Given $n = 256$ bits, the order of complexity increased up to $O(n^{17})$. Noticing such complexity is constrained by the strict bound of $n \geq 2k$. The remaining question is whether one can derive a better lower bound representation for $k < n$. The solution for this question could further reduce the factoring time significantly.

## 3 Concluding Remark

We showcase a reduction of semiprimes factorization problem to an $NP$-complete coding problem. Our result reveals the semiprimes factorization can in reverse, use to solve the $NP$-complete problem related to coding. Follows the work in [9] stated, if any $NP$-complete problem possesses a polynomial time algorithm to solve, then so does every $NP$-problem. The Shor's algorithm is proven to be efficient for factorization of $N$ in polynomial time ($O(n^3)$). This strongly argued that every $NP$-problem could be solve in polynomial time by the existence of quantum computer. In computational complexity theory [10], the class of problem solvable by an quantum computer in polynomial time is refer to the bounded-error quantum polynomial time (BQP). It is clearly that factoring $N$ is in BQP class. In such a case, our results would imply all $NP$ problem can be solved by a quantum computer hence $NP \subseteq BQP$.

Despite of such discovery, we still have no known existing prove that $P = BQP$, the answer to such question would lead to $P = NP$ since $NP = BQP = P$.

As a site note, the works in [11] (yet to be verified) also pointed out that another $NP$-complete problem related to coding could be solved in polynomial time with proper encoding and decoding function.

PF$(N, \epsilon)$

1 : $\mathcal{S} \leftarrow \emptyset$// Initiate an empty set
2 : $\mathcal{E} \leftarrow \{0,1\}^k$// Initiate a noise distribution with $\epsilon > 0$
3 : **Set** $k = \lceil \log(N)/2 \rceil$
4 :
5 : // Collect information to generate set of solutions for $\mathcal{S}$
6 : **for** $i = 1, 2, \ldots, 2^{\lfloor kh_2(\epsilon) \rfloor}$ **do**
7 : $e_i \xleftarrow{\$} \mathcal{E}$// Sample $e$ from $\mathcal{E}$ uniformly at random (without replacement), where $\|e\| = \lfloor k\epsilon \rfloor$
8 : **Set** $a = e_i \oplus 0^k$ and **Compute** $x = a^2 \mod N$
9 : **Set** $b = e_i \oplus 1^k$ and **Compute** $y = b^2 \mod N$
10 : **if** $\sqrt{x}$ **Divides** $x$
11 : **Set** $s = (\sqrt{x}, a)$
12 : $\mathcal{S} = \mathcal{S} \bigcup s$
13 : **elseif** $\sqrt{y}$ **Divides** $y$
14 : **Set** $s = (\sqrt{y}, b)$
15 : $\mathcal{S} = \mathcal{S} \bigcup s$
16 : **endif**
17 : **endfor**
18 : **return** $\mathcal{S}$
19 :
20 : // Verify the solutions with $\mathcal{S}$
21 : **for** $j = 1, \ldots, |\mathcal{S}|$
22 : $s_i = (s_i(1), s_i(2))$// Recall we have $s_i$ comes is pair (see Step 11 and 14)
23 : **Compute** $p = N/\gcd(s_i(1) + s_i(2))$ *or* $q = N/\gcd(s_i(1) - s_i(2))$
24 : **if** $(p \text{ or } q) \neq N$ **AND** $(p \text{ or } q) \neq 1$// only look for nontrivial $p$ and $q$
25 : **return** $(p \text{ or } q)$
26 : **BREAK**
27 : **endif**
28 : **endfor**

Step 1 to 18 of PF is viewed as data collection phase where it collects data (or information) with referring to Eq. 2, yielding a set of possible solutions over $\mathcal{S}$. The set of solution $\mathcal{S}$ is also known as congruence of square. The data collection phase can be done in parallel with different value of $\epsilon$, i.e. $\epsilon = \frac{1}{k}, \frac{2}{k}, \ldots, \frac{1}{2}$ correspond to different weight of solution for $\|e\| = 1, 2, \ldots, \lfloor k/2 \rfloor$.

On the other hand, Step 21 to 28 of PF is viewed as verification phase, where it verifies the all the possible solutions over $\mathcal{S}$. It the solution obtained is valid, it can factor $p$ and $q$ from $N$ successfully.

## References


1. G. Leander and A. Rupp, "On the equivalence of rsa and factoring regarding generic ring algorithms," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2006, pp. 241–251.
2. D. R. Brown, "Breaking rsa may be as difficult as factoring." *IACR Cryptology ePrint Archive*, vol. 2005, p. 380, 2005.
3. K. Altmann, T. Jager, and A. Rupp, "On black-box ring extraction and integer factorization," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2008, pp. 437–448.
4. D. Boneh and R. Venkatesan, "Breaking rsa may not be equivalent to factoring," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1998, pp. 59–71.
5. T. Jager and J. Schwenk, "The generic hardness of subset membership problems under the factoring assumption," Cryptology ePrint Archive, Report 2008/482, 2008, https://eprint.iacr.org/2008/482.
6. P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.
7. E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the inherent intractability of certain coding problems (corresp.)," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.
8. R. McEliece and H. Van Tilborg, "On the inherent intractability of certain coding problems(corresp.)," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.
9. R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.
10. M. R. Garey and D. S. Johnson, *Computers and intractability*. wh freeman New York, 2002, vol. 29.
11. Y.-L. Lai and Z. Jin, "Information-theoretic secure sketch for noisy sources of low entropy."