# *La Garde Ultime*: Analysis of NP-Hard Cryptography

Ellijah Darrellshane Suryanegara - 13522097
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*13522097@std.stei.itb.ac.id*

***Abstract*—This paper explores the strategic application of NP-Hard cryptography as a robust method for enhancing the security of cryptosystems, capitalizing on the inherent intractability associated with problems characterized by a P-hardness complexity within the realm of NP-hard. A comprehensive analysis of two noteworthy NP-Hard cryptosystems, namely Merkle—Hellman and Naccache—Stern, brings to light vulnerabilities susceptible to specific algorithms. These vulnerabilities, with the potential to navigate around the intrinsic intractability of the problems and the looming prospect of quantum computers, prompt a reassessment of the reliance on NP-hardness alone. The findings underscore the importance of exercising prudence and considering alternative cryptographic paradigms in the quest for constructing an impervious cryptosystem.***

***Keywords—cryptography, P vs NP, NP-hardness***

## I. Introduction

Cryptography is all about securing and delivering data in such a way that only specific individuals can understand or access it. We want it to be simple so that people who are granted access can acquire the transmitted data with ease while simultaneously being very complex to prevent unsanctioned actors from attaining equivalent access. Achieving security in this degree however is getting harder and harder—even seemingly impossible—as computational capabilities of modern computers are advancing with remarkable speed and efficiency. Problems that are deemed impossible, to the point of taking an early generation computer 10.000 years to solve, would now presently pose an inconsequential challenge to the likes of a quantum computer which would effectively resolve the identical issue within a mere second. Hence, it is only natural to wonder whether if designing the perfect cryptosystem—*la garde ultime* ("the ultimate guard" in French)—akin to chasing the end of a rainbow?

Well, a straightforward solution would be to adopt an algorithm whose efficient solution eludes contemporary computational capabilities as the bedrock of our cipher. In computational complexity theory, the problem in hand has much similarity as the *P versus NP problem*. The *P versus NP problem* is basically about asking, "If the solution to a problem can be verified (NP) in polynomial time, can it be found (P) in polynomial time?". In the context of cryptography, we want to find out if a key to our cryptosystem which inherently demands verifiability in a reasonable amount of time—lest the purpose of the cipher, ensuring accessible data, be nullified—can also feasibly, meaning in a reasonable amount of time as well, be deciphered illegitimately. As we aspire to craft an ideal cypher, one that ensures verifiability while remaining virtually impervious to decryption in the absence of the designated key, an implication is made that we are operating in the assumption that $P \neq NP$. With this assumption in place, we can now enter the realm of NP-hard problems and ask, "Is NP-Hard Cryptography the answer for our journey in finding *la garde ultime*?"

## II. Theoretical Basis

### A. Intractability

The first difficulty in developing a theory of average-case intractability is to come up with a formal definition of what it means for a problem to be "intractable on average" or, equivalently, what it means to be "average-case tractable." A natural definition would be to consider an algorithm efficient-on-average if it runs in expected polynomial time. Such a definition has various shortcomings (related to the fact that it is too restrictive). For example, if an algorithm A runs in time t(x) on input x, and its simulation B (on a different model of computation) runs in time $t^2(x)$ on input x, it is natural that we would like our definition to be such that A is efficient-on-average if and only if B is.

Suppose, however, that our inputs come from the uniform distribution, and that A runs in time $n^2$ on all inputs of length n, except on one input on which A takes time 2n. Then the expected running time of A is polynomial but the expected running time of B is exponential. Looking at the median running time of an algorithm gives us a more robust measure of complexity, but still a very unsatisfactory one: if an algorithm runs in polynomial time on 70% of the inputs, and in exponential time on 30% of the inputs, it seems absurd to consider it an efficient-on-average algorithm. The right way to capture the notion of "efficient on typical instances" should be that it is fine for an algorithm to take a large amount of time on certain inputs, provided that such inputs do not occur with high probability: that is, inputs requiring larger and larger running times should have proportionally smaller and smaller probability. This is the idea of Levin's definition of average-case complexity.

*Levin's definition*

*an algorithm is polynomial-time-on-average if there is a constant $c > 0$ such that the probability, over inputs of*

*length n, that the algorithm takes more than time T is at most poly(n)/T^c.*

## B. *P = NP?*

In the 1995 episode of The Simpsons, Treehouse of Horror VI (Season 7 Episode 6), Homer Simpson stumbles upon a portal to the enigmatic Third Dimension while attempting to evade his in-laws. Amid a dark expanse marked by green gridlines and adorned with geometric shapes, he encounters equations, one of which posits the seemingly straightforward assertion that P = NP. It was the same equation that became the main topic of a 2002 survey, where 61 mathematicians and computer scientists expressed skepticism about P equating to NP, while only nine believed in the equality [1]. Notably, some of those endorsing the equality admitted doing so merely for contrarian reasons. However, the question remains unanswered, making the equivalency of P and NP a pivotal inquiry in theoretical computer science—one of seven problems for which the Clay Mathematics Institute offers a million-dollar reward for proof or disproof.

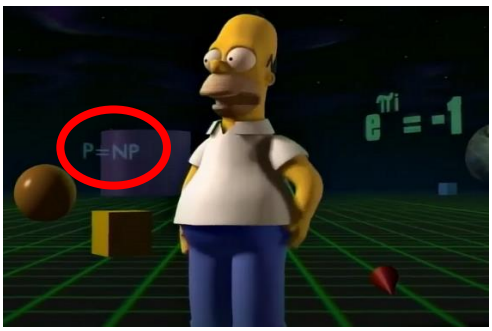


Fig. 1 Appearance of P = NP in The Simpsons: Season 7, Episode 6

Fundamentally, computer science revolves around a core question: how long it takes to execute a given algorithm. This duration is not measured in minutes but rather in relation to the quantity of elements the algorithm manipulates.

Consider an algorithm seeking the largest number in an unsorted list. While it must inspect all numbers, its execution time, if it maintains a record of the largest number seen so far, is directly proportional to the number of elements, denoted as N. Although most algorithms are more intricate, many share execution times proportional to $N^2$ or N Log N.

The "P" in "P = NP" corresponds to polynomial expressions involving N's. While differences between algorithms with execution times proportional to N or $N^3$ are substantial, they pale in comparison to distinctions involving exponential expressions, such as $2^N$. For instance, if an algorithm takes a second for a computation with 100 elements, an algorithm with a time complexity of $2^N$ would require an astronomical 300 quintillion years. This discrepancy escalates dramatically with larger values of N.

NP (nondeterministic polynomial time) comprises problems with solutions verifiable in polynomial time. Despite this verifiability, solving many NP problems appears to demand exponential time. Notably, finding prime factors of a large number, a quintessential exponential-time problem, exemplifies this challenge, requiring systematic exploration of numerous candidates.

## C. *NP-Hard*

An NP-hard (Non-deterministic Polynomial-time hard) problem is a type of computational problem that is at least as hard as the hardest problems in NP (Non-deterministic Polynomial-time). The class NP consists of problems for which a given solution can be checked quickly, but finding the solution efficiently is not necessarily guaranteed. An NP-hard problem doesn't need to have its solutions efficiently verifiable, but if you had a polynomial-time solution to an NP-hard problem, you could use it to solve any problem in NP in polynomial time.

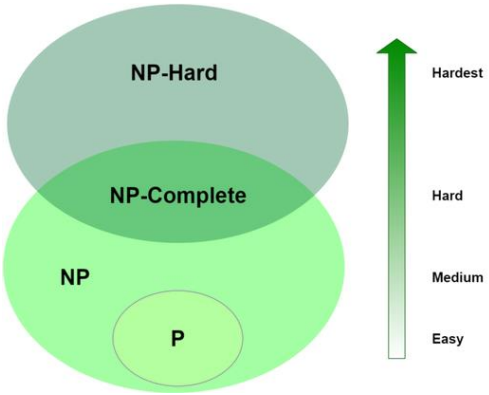


Fig. 2 Complexity of P-hardness in the realm of P ≠ NP

Key characteristics of NP-hard problems:

**a) Difficulty of Solution**

NP-hard problems are considered computationally difficult, as finding an algorithm that provides an optimal solution in polynomial time is not known (or may not even exist).

**b) No Efficient Verification**

Unlike problems in NP, solutions to NP-hard problems are not necessarily quickly verifiable. If you propose a solution, it may take a long time to verify its correctness, but once a solution is given, it can be checked for correctness in polynomial time.

**c) Relation to NP**

If you could find a polynomial-time solution to any NP-hard problem, you could use that solution to efficiently solve any problem in NP. This is known

as the Cook-Levin theorem, which establishes the concept of NP-completeness.

**d) Theoretical Foundation**

NP-hard problems are foundational to theoretical computer science and computational complexity theory. They help establish the limits of what can be efficiently computed.

Examples of NP-hard problems include:

- **Traveling Salesman Problem (TSP)**

  Given a list of cities and the distances between each pair of cities, the objective is to find the shortest possible route that visits each city exactly once and returns to the original city.

- **Knapsack Problem**

  In the 0/1 Knapsack Problem, you are given a set of items, each with a weight and a value, and you need to determine the maximum value that can be obtained by selecting a subset of the items such that the total weight doesn't exceed a given limit.

- **Boolean Satisfiability Problem (SAT)**

  Given a Boolean formula, the goal is to determine whether there exists an assignment of truth values to the variables that makes the entire formula true.

- **Graph Coloring Problem**

  Given an undirected graph, the objective is to assign colors to the vertices in such a way that no two adjacent vertices have the same color, using as few colors as possible.

.

## III. Analysis of NP-Hard in Cryptography

### *A. Merkle-Hellman*

One of the earliest form in implementing a subset of an NP-Hard problem is the Merkle-Hellman Knapsack Cryptosystem, which as stated in the name implemented a knapsack sequence which theoretically would only be cracked in exponential time if it is non-superincreasing.

**Key Generation**

1. Choose a block size *n*. Integers up to *n* bits in length can be encrypted with this key.
2. Choose a random superincreasing sequence of *n* positive integers

$$W = (w_1, w_2, \ldots, w_n)$$

Superincreasing refers to a sequence in which each element ($w_k$) is larger than the sum of all the previous elements in the sequence. In mathematical terms,

$$w_k > \sum_{i=1}^{k-1} w_i, \text{ for } 1 < k \leq n.$$

3. Choose a random integer *q* such that

$$q > \sum_{i=1}^{n} w_i$$

4. Choose a random integer *r* such that gcd(r,q) = 1. In other words, *r* and *q* must be coprime
5. Calculate the sequence by multiplying each element in the sequence with *r* (mod *q*)

$$B = (b_1, b_2, \ldots, b_n)$$
$$\text{where } b_i = rw_i \bmod q.$$

6. The resulting sequence *B* will be set as the public key while *W*, *q*, *r* will be set as the private key.

**Encryption**

Let *m* be an *n*-bit message consisting of bits $m_1m_2m_3...m_n$, with $m_1$ being the highest order bit. Select each $b_i$ for which $m_i$ is nonzero, then sum all of the product for each $b_i$ and $m_i$. Therefore, cipher text *c* can be generated as following

$$c = \sum_{i=1}^{n} m_i b_i$$

**Decryption**

1. Calculate the modular inverse of r (mod q) using the Extended Euclidean Algorithm.

$$r' := r^{-1} \pmod q$$

2. Calculate c'
3. Solve the subset sum problem for c' using the superincreasing sequence W' with the greedy algorithm below

```
def
subset_sum_greedy(superincreasing_sequence
, target_sum):

# Initialize the list X to store indices of
selected elements
X = []

# Find the largest element in W which is
less than or equal to the target sum
# Find the largest element less than or
equal to the target sum
while target_sum > 0:
     max_element =
     max(filter(lambda x: x <= target_sum,
     superincreasing_sequence))

# Subtract the selected element from the
target sum
target_sum -= max_element

# Append the index of the selected element
to the list X
index =
superincreasing_sequence.index(max_element
)

X.append(index)
```

```
# Return the list of selected indices
return X

# Example usage:
W = [2, 7, 11, 21, 42, 89]
target_sum = 57
result = subset_sum_greedy(W, target_sum)
print("Selected indices:", result)
```

4. Construct the message $m$ with a 1 in each $x_i$ bit position and a 0 in all other bit positions

$$m = \sum_{i=1}^{k} 2^{n-x_i}$$

**Runtime Complexity**

Merkle and Hellman originally suggested using knapsacks of approximate size $n = 100$. However, Schroeppel and Shamir developed an algorithm to solve knapsacks of this size. By trading time and space their method can solve the knapsack problem in time $T = O(2^{n/2})$ and space $O(2^{n/4})$ [5], which is recently improved by Jesper Nederlof and Karol Węgrzycki to $O(2^{n/2})$ time and $O(2^{0.249999n})$ space randomized algorithm for solving worst-case Subset Sum instances with $n$ integers [3]. For $n = 100$, $T = 2^{50} \approx 10^{15}$. Thus a single processor can find a solution in 11,574 days (which is roughly 32 years) [5]. But for $n = 200$, assuming 8.64 x $10^{10}$ instructions per day, the challenge is intractable.

However, the system was finally shown to be insecure after Brickell invented a polynomial time algorithm which recreated simple knapsack vectors from hard and complex knapsack vectors. Using Brickell's algorithm, it has been roughly calculated that the algorithm would require a runtime as explained by Ernest F. Brickell himself. [4]

"Breaking Iterated Knapsacks"
Ernest F. Brickell
page 355—357

*The worst case running time of the $L^3$ algorithm is $O(m^6D^3)$. In practice however, the running time appears to be $(mD^3)$. Also, we might have to make $O(\frac{n}{n-m})^3$ choices of an m-set of weights before we get a good one. To find the order of the weights, we must take $O(n)$ determinants. Each determinant takes $O(Y^3)$ multiplications of integers with length D. So the running time for finding the order is $O(nY^3D^2)$. The total running time on the first part of the algorithm is*

$$O(m\,(\frac{n}{n-m})^3\,D^3) + O(nY^3D^2)$$

*The second part of the algorithm is solving for the $a_i$'s after a cypher is received. The running time for this part is*

$$O(n^2) + O(2^{\varepsilon/2}) + O(Y^2)$$

Table 1. Tests of the $L^3$ Algorithm

| Type | N | Y | R | D | m | Time | Con |
|---|---|---|---|---|---|---|---|
| GS | 40 | 4 | 14 | 82 | 18 | 27.6 | 2.78E-6 |
| MH | 40 | 5 | 10 | 93 | 28 | 108.2 | 4.80E-6 |
| MH | 40 | 5 | 20 | 113 | 30 | 210.6 | 4.86E-6 |
| MH | 40 | 7 | 10 | 105 | 30 | 138.1 | 3.98E-6 |
| MH | 40 | 10 | 10 | 123 | 32 | 250.1 | 4.20E-6 |
| MH | 50 | 5 | 7 | 97 | 28 | 108.7 | 4.25E-6 |
| GS | 50 | 5 | 16 | 96 | 24 | 92.3 | 4.34E-6 |
| MH | 50 | 10 | 7 | 127 | 32 | 263.2 | 4.02E-6 |
| GS | 64 | 5 | 64 | 158 | 37 | 653.7 | 4.48E-6 |
| GS | 64 | 10 | 16 | 140 | 32 | 451.8 | 5.15E-6 |
| GS | 100 | 5 | 16 | 151 | 30 | 295.5 | 2.86E-6 |
| GS | 100 | 10 | 100 | 270 | 53 | 3542.0 | 3.40E-6 |
| GS | 100 | 20 | 20 | 260 | 52 | 3355.3 | 3.67E-6 |

*table note* [4]
Type – Merkle-Hellman (MH) or Graham-Shamir (GS)
N – the number of weights
R – the random bits used in constructing the superincreasing sequence. For MH knapsacks, all random bits are the low order bits. For GS knapsacks, half of the random bits are the low order bits, and the other half are the high order bits
Y – the number of iterations
D – the number of bits in the final (public) weights
m – the number of weights used in the lattice reduction
Time – the running time in seconds of the $L^3$ algorithm
Con – Time/$(mD^3)$

**Implication**

As shown, using Brickell's $L^3$ algorithm has effectively solved the special case of the Subset Sum problem (in itself a subset of the Knapsack Problem) which the Merkle-Hellman cryptosystem is based on—although it should be noted that the general Subset Sum and Knapsack Problem is still intractable up to this day. This demonstrates that tackling a cryptosystem grounded in NP-hard problems can be achieved by simplifying the algorithm to a problem with a lower level of P-hardness in complexity. Such strategic approach allows one to circumnavigate around the inherent intractability associated with the NP-hard foundation of the cryptosystem.

*B. Naccache–Stern*

The Naccache-Stern Cryptosystem is a public-key cryptosystem designed by David Naccache and Jacques Stern in 1997. It belongs to the class of asymmetric key algorithms, meaning it uses different keys for encryption and decryption. The Naccache-Stern Cryptosystem is known for its resistance against known attacks, particularly against certain types of quantum attacks.

**Encryption—Decryption**

[6] Let p be a large public prime and denoted by n the largest integer such that

$$p > \prod_{i=0}^{n} p_i, \text{where } p_i \text{is the i} - \text{th prime, p0} = 2$$

The secret key s < p – 1 is a random integer such that gcd(p-1, s) = 1 and the public keys are the n+1 roots generated with the Pohlig-Hellman

$$v_i = \sqrt[s]{p_i} \bmod p$$

$$m = \sum_{i=0}^{n} 2^i m_i \in M \text{ is encrypted as } c = \prod_{i=0}^{n} {v_i}^{m_i} \bmod p$$

which is recovered by

$$m = \sum_{i=0}^{n} \frac{2^i}{p_i - 1} * (\gcd(p_i, c^s \bmod p) - 1)$$

The security foundation of the Naccache-Stern Cryptosystem lies in the challenge posed by the multiplicative knapsack problem associated with its trapdoor function, specifically given the product

$$c = \prod_{i=0}^{n} {v_i}^{m_i} \bmod p$$

The task is to recover the individual $m_i$'s. Notably, this multiplicative knapsack problem differs from its additive counterparts, like the one employed in Merkle-Hellman cryptosystems, rendering conventional techniques such as Euclidean lattice reduction ineffective.

The primary generic attack involves tackling the discrete logarithm problem to deduce *s* from *p*, $p_i$, $v_i$, a challenge deemed formidable for classical computers. However, the advent of quantum algorithms, notably Shor's algorithm, efficiently solves the discrete logarithm problem, raising concerns in the face of quantum advancements. It's noteworthy that, as of 2023, there exists no conclusive proof establishing the reduction of the Naccache-Stern knapsack problem to the discrete logarithm problem. A specific attack identified in 2018 exploits the birthday theorem, aiming to partially invert the function without prior knowledge of the trapdoor. This attack assumes a message with an exceptionally low Hamming weight, showcasing the ongoing efforts to scrutinize and fortify the cryptographic resilience of the Naccache-Stern Cryptosystem.

**Example**

n = 7 with prime p = 9700247 > 2 * 3 * 5 * 7 * 11 * 13 * 17 * 19 and the secret s = 5642069 yield the v-list:

$v_0 = \sqrt[s]{2} \bmod p = 8567078$ $v_4 = \sqrt[s]{11} \bmod p = 8643477$

$v_1 = \sqrt[s]{3} \bmod p = 5509479$ $v_5 = \sqrt[s]{13} \bmod p = 6404090$

$v_2 = \sqrt[s]{5} \bmod p = 2006538$ $v_6 = \sqrt[s]{17} \bmod p = 1424105$

$v_3 = \sqrt[s]{7} \bmod p = 4340987$ $v_7 = \sqrt[s]{19} \bmod p = 7671241$

*encryption*

$m = 202 = 11001010_2$

$c = v_7^1 * v_6^1 * v_5^0 * v_4^0 * v_3^1 * v_2^0 * v_1^1 * v_0^0 \bmod p = 7202882$

*decryption*

by exponentiation, we retrieve:

$c^s \bmod p = 7202882^{5642069} \bmod 9700247 = 6783$

whereby:

$6783 = 19^1 * 17^1 * 13^0 * 11^0 * 7^1 * 5^0 * 3^1 * 2^0 \rightarrow m = 11001010_2$

**Shor's Algorithm**

Shor's algorithm, named after mathematician Peter Shor, is a quantum algorithm that efficiently factors large composite numbers into their prime factors and solves the discrete logarithm problem. Both of these problems are classically hard, meaning that no known polynomial-time algorithm exists for solving them on a classical computer. Shor's algorithm, however, demonstrates quantum superiority in addressing these problems exponentially faster than the best-known classical algorithms.

An implementation of Shor's algorithm is as follows

1) Choose any random number let say *r*, such that $r < N$ so that they are co-primes of each other;
2) A quantum computer is used to determine the unknown period p of the function $f_{r, N}(x) = r^x \bmod N$;
3) If p is an odd integer, then go back to Step 1, else move to the next step;
4) Since p is an even integer so, $(r^{p/2} - 1)(r^{p/2} + 1) = r^p - 1 = 0 \bmod N$;
5) If the value of $r^{p/2} + 1 = 0 \bmod N$, go back to Step 1, else move to the next step;
6) Compute $d = \gcd(r^{p/2} - 1, N)$. [8]

```
from qiskit import IBMQ
from qiskit.aqua import QuantumInstance
from qiskit.aqua.algorithms import Shor

# Enter your API token here
IBMQ.enable_account('ENTER API TOKEN HERE')
provider = IBMQ.get_provider(hub='ibm-q')

# Specifies the quantum device
backend =
provider.get_backend('ibmq_qasm_simulator')

print('\n Shors Algorithm')
print('--------------------')
print('\nExecuting...\n')

# Function to run Shor's algorithm
# where 35 is the integer to be factored
factors = Shor(35)
```

```
result_dict                                    =
factors.run(QuantumInstance(backend,
shots=1, skip_qobj_validation=False))

# Get factors from results
result = result_dict['factors']

print(result)
print('\nPress any key to close')
input()
```

**Runtime Complexity**

Estimating the time complexity of Shor's algorithm involves considering several factors and referencing research findings. In the paper "Efficient Networks for Quantum Factoring" by David Beckman, Amalavoyal N. Chari, Srikrishna Devabhaktuni, and John Preskill, a notable estimation is provided, suggesting a time complexity of $\mathbf{72(\log N)^3}$ for quantum factoring[7], where N represents the number to be factored—resulting in O(log N) in Big-O terms.

However, it's crucial to acknowledge the complexities involved in comparing the number of steps on a quantum computer to those on a classical computer. The exact architecture of a quantum computer, which remains uncertain until its construction, significantly influences the number of steps required. Additionally, the comparison is complicated by the potential slower speed of individual steps on a quantum computer compared to classical counterparts. The challenge arises from maintaining quantum coherence, and until quantum computers are realized, the precise extent of this speed difference remains speculative. The time complexity of Shor's algorithm is intricately tied to the intricate interplay of quantum computation dynamics, emphasizing the need for empirical quantum computing developments to draw definitive conclusions.

**Implication**

Shor's algorithm stands as a groundbreaking milestone in the realm of quantum computing, showcasing the immense potential of quantum computers to address problems that are conventionally considered intractable for classical computers. Specifically, Shor's algorithm efficiently tackles challenges such as factoring large composite numbers and solving the discrete logarithm problem, both of which are integral to the foundation of various cryptographic systems. The significance of Shor's algorithm extends to the realm of public-key cryptography, where the security of widely used protocols, including RSA, relies on the presumed computational difficulty of factoring large numbers.

The quantum supremacy demonstrated by Shor's algorithm has profound implications for the security landscape of cryptographic systems based on NP-hard problems. NP-hard problems, which include various computational challenges fundamental to cryptographic protocols, are traditionally considered computationally infeasible to solve efficiently. However, Shor's algorithm challenges this assumption by illustrating the potential of quantum computers to significantly reduce the time complexity of certain problems that form the basis of cryptographic hardness.

## V. Conclusion

Leveraging NP-Hard cryptography represents a valid strategy for securing a cryptosystem, utilizing the inherent intractability of problems categorized with a P-hardness complexity of NP-hard. However, an insightful examination of two prominent NP-Hard cryptosystems, namely Merkle-Hellman and Naccache-Stern, reveals vulnerabilities susceptible to certain algorithms which could circumnavigate the intractability of the problem itself and potentially quantum computers. These vulnerabilities could unveil weaknesses in problems traditionally perceived as intractable. Consequently, prudence dictates against an exclusive reliance on NP-hardness as *la garde ultime* in the pursuit of crafting an impeccable cryptosystem.

## VI. Appendix

The applicative research presented in this paper owes a debt of gratitude to Mr. Ernest F. Brickle, the visionary mind behind the L3 algorithm, and Mr. Peter Shor, the brilliant architect of Shor's Algorithm. Their groundbreaking contributions and innovative algorithms have been pivotal in shaping the trajectory of this study. Without their seminal work, the insights and discoveries encapsulated in this paper would not have come to fruition.

Furthermore, it is imperative to extend special acknowledgment to all the distinguished researchers who have been trailblazers in the field of P vs NP. Their pioneering efforts, relentless dedication, and significant contributions have laid the foundation for the theoretical framework and computational paradigms explored in this research. The intellectual legacy they have established serves as a beacon, guiding subsequent generations of researchers, and their collective impact on the field is immeasurable. It is with deep appreciation and admiration that their names are invoked, recognizing the debt owed to these esteemed pioneers in the pursuit of unraveling the complexities of computational theory.

## VII. Acknowledgment

I am profoundly grateful to God Almighty, creator of the universe, for His guidance throughout the journey of crafting this paper. I would also like to express my sincere gratitude all the following individuals who have played pivotal roles in the completion of this paper:

1. Fariska Zakhralativa Ruskanda, S.T., M.T., my esteemed class professor, whose guidance and expertise provided invaluable insights throughout the course, elevating the quality of this work.
2. Dr. Ir. Rinaldi Munir, M.T., my dedicated course coordinator, whose research and syllabus have been instrumental in shaping the trajectory of this research endeavor.
3. My esteemed colleagues of IF'22, whose collaborative spirit and shared enthusiasm fostered an enriching

academic environment, stimulating meaningful discussions and enhancing the overall research experience.
4. Last but not least, my heartfelt appreciation goes to my parents for their enduring support, encouragement, and understanding. Their unwavering belief in my academic pursuits has been a constant source of inspiration, and I am truly grateful for their love and encouragement throughout this academic journey.

## REFERENCES

[1] https://news.mit.edu/2009/explainer-pnp, accessed on December 10th 2023.
[2] https://crypto.stackexchange.com/questions/58688/information-theoretic-security, accessed on December 10th 2023.
[3] Jesper Nederlof, Karol Węgrzycki, *Improving Schroeppel and Shamir's Algorithm for Subset Sum via Orthogonal Vectors*. Cornell University, 2021, arXiv:2010.08576 [cs.DS].
[4] Ernest F. Brickell, *Breaking Iterated Knapsacks*. Sandia National Laboratories, Albuquerque, New Mexico, 1984, pp. 342—358. Accessed through https://link.springer.com/content/pdf/10.1007/3-540-39568-7_27.pdf on December 10th, 2023.
[5] Jennifer Seberry, *Public Key Cryptography*. University of New South Wales, Canberra, 1987, 1—17. Accessed through https://ro.uow.edu.au/cgi/viewcontent.cgi?referer=&httpsredir=1&article=2045&context=infopapers on December 10th, 2023.
[6] David Naccache, Jacques Stern, *A New Public-Key Cryptosystem*. France, 1998. Accessed through https://www.di.ens.fr/~stern/data/St63.pdf on December 11th, 2023.
[7] David Beckman, Amalavoyal N. Chari, Srikrishna Devabhaktuni, and John Preskill, *Efficient Networks for Quantum Factoring*. American Physical Society, 1996, https://doi.org/10.1103/PhysRevA.54.1034.
[8] https://www.geeksforgeeks.org/shors-factorization-algorithm/, accessed on December 11th, 2023.

## STATEMENT OF ORIGINALITY

I hereby declare that this paper is an original composition of my own, not of any adaptation or translation from the authored works of others, and free from plagiarism.

Bandung, December 11th 2023

Ellijah Darrellshane Suryanegara
13522097