ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

THE EVALUATION AND COMPARISON OF PRIMALITY TESTING ALGORITHMS

M.Sc. THESIS

Gözde SARIKAYA

Department of Applied Informatics

Cybersecurity Engineering and Cryptography Programme

JUNE 2019

ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

THE EVALUATION AND COMPARISON OF PRIMALITY TESTING ALGORITHMS

M.Sc. THESIS

Gözde SARIKAYA
(707151013)

Department of Applied Informatics

Cybersecurity Engineering and Cryptography Programme

Thesis Advisor: Assoc. Prof. Dr. Enver ÖZDEMİR

JUNE 2019

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

ASALLIK TESTİ ALGORİTMALARININ İNCELENMESİ VE KARŞILAŞTIRILMASI

YÜKSEK LİSANS TEZİ

Gözde SARIKAYA
(707151013)

Bilişim Uygulamaları Anabilim Dalı

Bilgi Güvenliği Mühendisliği ve Kriptografi Programı

Tez Danışmanı: Doç. Dr. Enver ÖZDEMİR

HAZİRAN 2019

Gözde SARIKAYA, a M.Sc. student of ITU Informatics Institute with student ID 707151013, successfully defended the thesis entitled “THE EVALUATION AND COMPARISON OF PRIMALITY TESTING ALGORITHMS”, which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Dr. Enver ÖZDEMİR** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Ergün YARANERİ** Istanbul Technical University	.....
	Assist. Prof. Dr. Elif Segah ÖZTAŞ Karamanoglu Mehmetbey University	.....

Date of Submission : 03 May 2019
Date of Defense : 14 June 2019

To my beloved family and nephew Umut,

FOREWORD

First and foremost, I would first like to thank my thesis advisor Assoc. Prof. Enver Özdemir. He supported me throughout the entire process with his patience, motivation, enthusiasm and immense knowledge on the topic. It was a great pleasure to join his research group at the National HPC Center of Turkey. I really appreciate all the efforts he has put into this thesis.

In addition to my education, the most important reason for me to feel belonging to a company is to feel friendliness, honesty, and trustworthiness that your colleagues offer. Thus, I would like to acknowledge to my dear colleagues whom I had worked together at UHeM, TUBITAK and CRYPTTECH. I appreciate all for making me feel like the part of this family.

Finally, I would like to express my profound gratitude to my lovely family. The words are not enough for me to express how grateful I am to all members for their sacrifices. My father always trusts all my decisions and gives encouragement to whatever I dream. My mother focuses on my achievements all the time to make me stronger whenever my self-confidence is weakened. My brother makes me feel very lucky by giving invaluable support and guidance as a polestar since my childhood. My sister deserves many lovely thanks for being patient in listening to me until I come to the decisions for my life. And, my sister-in-law always supports me spiritually through this process. This success could not be achieved without excellent encouragement and sacrifices. I will be grateful forever for their love and this achievement would not have been possible without them.

Finally, I dedicate this thesis to my nephew, Umut. He teaches all the people around him by laughing how to be strong even when everything goes wrong. Hope you never lose the light and joy in your eyes Umut!

Thank you for all.

June 2019 Gözde SARIKAYA

TABLE OF CONTENTS

Page

ABBREVIATIONS

CIA	**:** Confidentiality-Integrity-Availability
SPSP	**:** Strong Pseudoprime
DLP	**:** Discrete Logarithm Problem
IFP	**:** Integer Factorization Problem
ECC	**:** Elliptic Curve Cryptography
ECPP	**:** Elliptic Curve Primality Proving
ECDLP	**:** Elliptic Curve Discrete Logarithm Problem
GCD	**:** Greatest Common Divisor
GF	**:** Galois Field
RSA	**:** Rivest-Shamir-Adleman
AKS	**:** Agrawal-Kayal-Saxena
CRT	**:** Chinese Reminder Theorem
NAF	**:** Non-adjent Form

SYMBOLS

{a,b,c}	**:** A set with 3 elements
p	**:** Prime number
P	**:** A message to encrypt, i.e., plaintext
C	**:** Encrypted plaintext, i.e., ciphertext
$\mathbf{K_E}$	**:** Encryption key
$\mathbf{K_D}$	**:** Decryption key
$\mathbf{E_k(P)}$	**:** Encryption of a plaintext *P* with the key K_E
$\mathbf{D_k(C)}$	**:** Decryption of a ciphertext *C* with the key K_D
a | b	**:** *a* divides *b*
a ∤ b	**:** *a* does not divide *b*
π(n)	**:** Number of primes less than or equal to *n*
φ(n)	**:** Euler's totient function for *n*
$\left(\frac{a}{p}\right)$	**:** Legendre symbol
$\left(\frac{a}{n}\right)$	**:** Jacobi symbol
R	**:** Ring
G	**:** Group
Z/*n*Z	**:** Residue class modulo n
(Z/*n*Z)*	**:** Multiplicative group whose element are relatively prime to n
$\boldsymbol{F_q}$	**:** Finite field with $q = p^k$ a prime number
$\boldsymbol{F_p}$	**:** Finite field with *p* elements, where *p* is prime
E	**:** Elliptic curve with simplified equation $y^2 = x^3 + ax + b$
$\mathbf{E/F_p}$	**:** Elliptic curve over F_p
$\boldsymbol{P_\infty}$	**:** Point at infinity on elliptic curve *E*

LIST OF TABLES

Page

LIST OF FIGURES

Page

THE EVALUATION AND COMPARISON OF PRIMALITY TESTING ALGORITHMS

SUMMARY

A prime number is an integer without a non-trivial divisor. In modern cryptography, several secure digital communication methods need to use large prime numbers. For example, one of the most popular public key cryptosystems, RSA [22], uses prime integers with more than 150 digits. Thus, it has been an interest of researchers to give a generic formula for defining all prime numbers. Although, there was no initial concern to detect prime integers theoretically, from the beginning of late 60's, several researches have presented a practical method for the primality test. Currently, there are methods [7–10, 16, 17] to decide the primality of the big numbers. These methods can be divided into two categories: probabilistic and deterministic primality tests.

Probabilistic tests are very fast and consist of some set of mathematical equations and procedures. The common property of these tests is having error probability with some composite numbers. For a given integer n to determine whether it is prime, the test is 100% accurate if it labels that number as a composite. However, the other case where the test output is prime is not always true, that is to say, there is always a probability that a composite number passes the test and is labeled as prime. To decrease this probability and increase the accuracy of the test, we always need to repeat the test for several times. As examples of probabilistic tests, we provide explanations on Euler's Test, Fermat's Test, Solovay-Strassen Primality Test and Miller-Rabin Test.

To overcome the drawback of probabilistic tests, deterministic tests are invented. Besides probabilistic tests, deterministic tests guarantee that if the test labels a given number n as composite, the number is, in fact, composite. Additionally, deterministic tests also guarantee the primality of numbers. However, the running time of deterministic tests is not satisfactory for frequent use in commercial applications. As an example of deterministic test, we give details of Agrawal-Kayal-Saxena (AKS) primality testing algorithm with a pseudocode, examples, running time analysis, its accuracy and drawbacks.

Moreover, elliptic curve primality testing methods and theorems are widely used. Although some algorithms require very long execution time for several-million digit integers, the results are deterministic. Thus, we include Shafi-Killian and Atkin's elliptic curve primality proving algorithms into our analyses.

In this thesis, general mathematical theorems which are very fundamental in number theory are explained to the reader. Then, their applications on primality testing are given by commonly used primality test algorithms. The current algorithms analyzed in terms of computational complexity, illustrated with examples to make the algorithms clearer and finally evaluated with its advantages and disadvantages. This literature review is given to increase the knowledge required for the singular cubic curve test [35]. Then, known primality tests and their analyses are given to provide a base for comparison.

Finally, the theoretical and experimental comparison results are provided in the last chapter. Known primality tests are compared to the singular cubic primality test by using the same dataset which includes both primes and composites with different number of digits. Implementation and testing phrases show that the singular cubic curve algorithm catches all composite number up to 10^{21} which were strong pseudoprimes to base 2 according to commercially used Miller-Rabin test. Additionally, it successfully detects the compositeness of large integers that have several hundred digits. Thus, singular cubic curve algorithm is a candidate to be a primality testing algorithm with running time of $O(\log^{2+\varepsilon} n)$.

ASALLIK TESTLERİ ALGORİTMALARININ İNCELENMESİ VE KARŞILAŞTIRILMASI

ÖZET

Asal sayılar, kendisi ve 1 dışında herhangi bir böleni olmayan sayılardır. Günümüzde yaygın olarak kullanılan kriptolojik tekniklerde ve güvenilir haberleşme protokol tasarımlarında yüzlerce ve hatta binlerce rakamdan oluşan asal sayılar kullanılmaktadır. Örnek vermek gerekirse, popüler ve çok bilinen algoritmalardan biri olan RSA [22] algoritması, 150'den fazla rakam içeren iki asal sayının çarpımından oluşmaktadır. Bu sebeple, tüm asal sayılar için geçerli olabilecek genel formüller ve testler geliştirilmesi, araştırmacılar tarafından halen çalışmaları devam eden bir alan olmuştur. 1960'lı yıllardan öncesinde teorik olarak kanıtlanan bir asallık testi olmamasına rağmen, o yıllardan itibaren bu konuda birden fazla pratik metotlar bulunmuştur [7–10, 16, 17]. Çok büyük sayılar için kullanılması uygun olan algoritmalar vardır ve bunlar genel olarak olasılıksal testler ve deterministik testler olarak iki gruba ayrılabilir.

Olasılıksal testler birkaç gruptan oluşan matematiksel denklemleri içerir ve diğer testlerle karşılaştırıldığında daha hızlı olduğu kanıtlanmıştır. Bu testlerin ortak özelliği; asal sayıları tanımlamakta (çok küçük ve önemsiz düzeyde de olsa) bir hata payı içermesidir. Test edilmek üzere verilen bir n sayısı için, sayı eğer asal değil ise, olasılıksal testler bunu tespit etmekte ٪100 gerçek sonuç verir. Fakat diğer yönden, verilen sayıya algoritmanın döndüğü tanımlama, sayının asal olduğu ise, bu sonuç tam olarak güvenilir değildir. Çünkü bilinen bazı asal olmayan sayılar vardır ki; bu testi asal bir sayıymış gibi geçebilirler. Bu yüzden, olasılıksal testlerin doğruluğunu ve geçerliliğini artırmak için, test aynı sayı için farklı rastgele tabanlar seçilerek t defa tekrar edilir. Bu tekrar, aynı zamanda test sonucunda oluşan hata payının düşmesini sağlar. Aynı sayının defalarca tekrar edilmesi sebebiyle, olasılıksal bir sonuç elde edilir. Olasılıksal testlere örnek olarak; Euler ve Fermat testleriyle birlikte, Solovay-Strassen [10] ve Miller-Rabin [16, 17] tarafından geliştirilen testleri de verilmiştir.

Olasılıksal testler bölümünde görüldüğü üzere; bazı yalancı asallar vardır ki, rastgele seçilen baz değerine göre asal sayılar gibi testin tüm şartlarını sağlarlar. Tarihsel olarak sıralandığında, günümüz gelişmelerine doğru testlerin yarı asal oluşturma olasılığı gitgide azalmıştır. Örneğin; 10^4 limitinden az Fermat testi için (baz 2 alındığında) yalancı asalların sayısı 22 iken; Euler testiyle birlikte bu sayı 12'ye düşmüştür. Miller-Rabin olasılıksal testi aynı baz değeri için değerlendiğinde, bu toplam sadece 5'tir. Ancak; en az hata veren Miller-Rabin testinin hata oranı incelendiğinde; testin t defa tekrar etmesi sonucunda hata oranı $(\frac{1}{4})^t$ değerinden azdır. Bu değer küçük ve önemsiz olsa da, asal sayıların yaygınca kullanıldığı günümüz algoritmaları düşünüldüğünde, net sonuçlar veren asallık testlerine olan ihtiyaç görülmektedir.

Olasılıksal testlerin defalarca tekrarlanarak olasılıksal bir sonuç dönmesinin eksikliğini kapatmak üzere, deterministik testler geliştirilmiştir. Deterministik testlerin temel

özelliği, çok yüksek bir doğruluk payı içermeleridir. Diğer bir deyimle, deterministik testler verilen sayı için sonuç olarak asal tanımlaması yaptığında, verilen sayının gerçekte asal olması doğrulanmış olur. Aynı şekilde; verilen sayının tanımlaması asal olmadığı şeklinde ise, sayı gerçekte de asal değildir. Ancak, bu testlerin doğruluk özelliğinin net olmasının yanında, pratik uygulamalarda sıklıkla kullanmak için olasılıklar testler kadar hızlı sonuç vermezler. Deterministik testlere örnek vermek gerekirse, Agrawal-Kayal-Saxena (AKS) algoritması ve analizi örneklerle beraber incelenmiştir. Bu test, polinom zamanlı deterministik bir test olmasına rağmen, çalışma zamanı bakımından değerlendirildiğinde olasılıklsal ve elliptik eğri testlerine nazaran çok yavaş kalmaktadır. Bu sebeple, tez içinde karşılaştırılmaya dahil edilmemiştir.

Bu testlerin dışında, eliptik eğriler ve türevleri kullanılarak geliştirilen asallık testi algoritmaları da günümüzde yaygınca kullanılmaktadır. Bazı eliptik eğri asallık testi algoritmalarının, çok büyük sayıda rakamdan oluşan sayıların asal olup olmadığını tespit etmeleri çok uzun zaman alsa da, dönülen sonuç deterministiktir. Bu testlere örnek olarak Shafi-Killian ve Atkin-Morain olmak üzere, iki eliptik eğri asallık ispatlama algoritması incelenmiştir.

Öncelikle, Shafi-Killian [9] ve Atkin-Morain [8] tarafından geliştirilen algoritmaların baz aldığı temel işleyiş metodolojisi verilmiştir. Ayrıca; her iki algoritmanın da ana temelini oluşturan Pocklington Teoremi'nden bahsedilmiştir. Her iki algoritma da benzer metodolojiye farklı bir bakış açısı sunduğundan ve birebir bir karşılaştırma sunulması açısından, bu algoritmalar 5 temel adımda incelenmiştir.

Shafi-Killian tarafından geliştirilen algoritmada, ilk aşamada Miller-Rabin testi gibi olasılıksal bir test kullanılır. Bu aşama sayesinde, sadece olasılıksal testin asal olmadığını ispatlayamadığı sayılar için denenmiş olur. Daha sonra, rastgele bir a ve b değeri seçilerek eliptik eğri denklemi oluşturulur. Bu eğrinin üzerindeki tüm noktaların sayısını bulmak için, nokta sayma algoritmalarından olan Schoof metodu [11] kullanılır. Bu metot ile bulunan değerin çarpanlara ayrılması gerekir ve bu aşamana Lenstra'nın çarpanlara ayırma metoduna yer verilmiştir. Eğer değer çarpanlara ayrılmazsa, en başa dönülerek yeni bir eğri seçilir ve diğer adımlar tekrarlanır. Bu ilk 3 aşamanın zaman maliyetli olması sebebiyle, Atkin-Morain eliptik eğri algoritması geliştirilmiştir.

Atkin-Morain algoritmasında kullanılacak eliptik eğri rastgele değil, belli bir ön aşamadan sonra oluşturulur. Bu metodun temel farkı olarak karışık çarpım (CM) yöntemine yer verilmiştir. CM sayesinde, eğri üzerindeki nokta sayısı için olası değerler elde edilir ve bu sayıların çarpanlara ayrılması denenir. Çarpanlara ayrılan değer bulunduğunda, eliptik eğri bu değere göre yine rastgele a ve b değerleri seçilerek oluşturulur.

Eliptik eğri oluşturma aşamasında farklılaşan Shafi-Killian ve Atkin-Morain algoritmaları, eliptik eğri seçildikten sonra aynı operasyonları uygular. Her iki algoritma da, belirlenen eğri üzerinde, yani eğri denklemini sağlayan, bir nokta seçer. Daha sonra bu nokta üzerinde grup operasyonu uygular. Bu testler, ilk aşamada verilen sayının asal olduğunu kabul edip, grup operasyonu sırasında asal olmadığını bir hata ile bulma mantığına dayalı olduğu için, grup operasyonunda hata olana kadar testler devam eder. Bu sebeple, bu testlerin çalışma zamanını net olarak ölçmek mümkün değildir. Bu tezde, eliptik eğriler ve grup operasyonları hakkında temel bilgilere

değinilmiştir. Aynı zamanda, çalışma zamanı olarak verimli grup operasyonu ve skaler çarpım algoritmalarına yer verilmiştir.

Genel olarak, bu tezde, öncelikle sayılar teorisinde temel olarak bilinen ve kullanılan genel matematiksel altyapılar için tanımlamalar ve teoremler verilmiştir. Daha sonra devam eden bölümlerde eliptik eğrilere dair detaylar anlatılmıştır. Literatür taraması olarak, bilinen asallık testi algoritmaları altyapılarıyla birlikte verilerek, bu testler okuyucuya daha açık bir anlatım sunabilmek için örneklerle beraber pekiştirilmiştir. Bilinen algoritmaların kendilerine dair özellikleri dışında, diğer algoritmalarla karşılaştırıldığında oluşan avantajlar ve dezavantajları değerlendirilmiştir.

Bilinen algoritmaların yanına ek olarak, önceden teorik olarak geliştirilen bir asallık testi önermesine [35] yer verilmiştir. Bu önermenin algoritma olarak detayları verilerek, neden asallık testi olabileceğine dair detaylardan bahsedilmiştir. Bu algoritmanın bilgisayar ortamında C++ dili ile yazılım tabanlı gerçeklenmesi yapılıp pratik ortamda kanıtlanması sağlanmıştır. Testler Miller-Rabin algoritmasının doğru olarak yakalayamadığı sayılardan olan "baz 2" sayıları seçilmiştir. Baz 2'ye göre yalancı olan sayılar gerçekte asal olmamasına rağmen, Miller-Rabin algoritmasından asal olarak geçebilen sayılardır. Bu testin gerçeklenmesinden elde edilen sonuçlar, önerilen asallık testinin 2^{64}'e kadar olan tüm baz 2'ye göre yalancı asal olan sayıların gerçekte asal olmadığını tanımladığını göstermektedir. Aynı zamanda, çok büyük rakam içeren ve yüksek hassaslıktaki rastgele sayılar denenmiş ve bu testin doğru sonuçlara kısa zamanda ulaştığı görülmüştür.

Bu testin gerçekliğinin pratikte denenmesinin yanında; testin diğer algoritmalar ile çalışma zamanı karşılaştırılmıştır. Teorik karşılaştırma ve çalışma zamanına dair sonuçlar göstermektedir ki; bu test olasılıklar testler kadar hızlı ve pratikte kullanılabilecek kadar az algoritma karmaşıklığına sahiptir.

1. INTRODUCTION

1.1 Motivation and Objectives

The primality testing algorithms are the keystone of cryptography and computational number theory. Almost all security of cryptographic algorithms depend on their underlying mathematical procedures and theorems. For example, the most commonly used public-key cryptosystem, RSA, is solely using exponentiation and congruences modulo some integer. In addition, the security of RSA is based on the difficulty of integer factorization problem. In other words, the product of two integers can easily be calculated with any computers; nevertheless, the factorization of that calculated number into its products is challenging. Therefore, if an efficient algorithm that factorizes very large integers could be found, then RSA will not be safe to use anymore. In Chapter 2, we provide the RSA algorithm whose key generation step involves the multiplication of two very large primes. As seen from the algorithm, we need to deterministically ensure the primality of those large numbers. This provides our motivation to study on primality testing.

The primality testing algorithms can be categorized into two main categories: probabilistic and deterministic. The probabilistic tests are fast but if the test results in composite for a given number n, it is definitely composite. However, if the test does not say an integer is composite, it is very hard to conclude that the number is prime. The forebear of probabilistic tests is Fermat's Little Theorem and continues with Euler's extensions, Solovay-Strassen, and the Miller-Rabin test (the most recent). As seen from the tables of the number of pseudoprimes in Chapter 4, Fermat's test has less accuracy than the Miller-Rabin Test. In other words, considering just one trial, the probability of having an error is at most fifty percent in Fermat's Test, whereas the Miller-Rabin Test only has at most twenty-five percent. Although every probabilistic primality test contains some extra conditions to reduce the likelihood of

error, deterministic tests are needed as probabilistic tests would not be sufficient for every application.

Deterministic tests have been developed to mitigate this disadvantage in probabilistic tests. These tests can determine the primality of an integer more precisely, i.e., if the test results as a prime then the number is definitely prime. However, deterministic tests are not efficient in practice. Thus, there is still a need for practical deterministic algorithms.

Because of this motivation, we prepared this thesis to serve a brief introduction to the history and evolution of primality testing algorithms through time from Fermat's Little Theorem to modern tests that use elliptic curves. In addition, we included a newly presented algorithm which uses singular cubics to satisfy the drawback in Miller-Rabin test. After the introductory chapters including the fundamental definitions and theorems of number theory, primality testing and proving algorithms are examined by their algorithms, examples, and analyses. Finally, evaluation and comparison are given to examine the accuracy and execution rate of known algorithms.

1.2 Thesis Structure

The goal of this thesis is to cover the necessary background from number theory and to explain significant primality testing algorithms. It consists of six chapters with each chapter covering the following themes:

In Chapter 2, we will give an overview of the encryption schemas together with their definitions and security requirements. Then, we will give explanations, advantages, and drawbacks of two types of cryptosystems: secret-key cryptosystems and public-key cryptosystems. Since prime numbers and primality testing algorithms are a major building block of public-key cryptography, we will take a comprehensive look at public-key cryptosystems along with their underlying mathematical problems that are not yet computationally feasible. After describing the discrete logarithm problem with its two applications, the integer factorization problem will be examined.

In Chapter 3, we will present definitions and preliminaries of elliptic curves. After explaining its group structure as the next issue, we continue with curve arithmetic which forms a base for modern primality testing algorithms. Furthermore, coordinate

systems for elliptic curves and its comparison will be explained to give the idea of speeding up the group operation.

In Chapter 4, we will present the leading primality testing algorithms along with their principle definitions and theorems coming from number theory. There are two categories of primality testing algorithms: probabilistic and deterministic tests. Both methods will be intensively described in this chapter together with their applications and examples. The oldest techniques illustrated with some examples and tables herein to explain the basic facts on pseudo-primality. The AKS primality testing algorithm will be examined as a reference for a deterministic test. Next, we will introduce the well-known Pocklington's theorem and Cornacchia's algorithms, which are the principle strategies used in elliptic curve primality proving. The given ECPP algorithms will be divided into five main steps to evaluate gradually the difference in perspective of two groups of researchers, Shafi-Killian and Atkin-Morain respectively. Finally, we also give the details of newly presented algorithm [35] that uses singular cubics.

In Chapter 5, we will first provide our theoretical comparison result which includes algorithmic complexity of some selected probabilistic and deterministic algorithms along with singular cubic primality testing algorithm. Afterward, the details of our implementation will be provided with hardware and software specifications. The running time comparison results of our implementation will be given at the end of this chapter.

Finally, in Chapter 6, we will conclude the singular cubic curve primality test and its contribution to the research area of algebraic and computational number theory. The final discussion will include suggestions for future research.

2. ENCRYPTION SCHEMAS

2.1 Introduction

In modern technology, *information* is valuable. Our e-mail contents, passwords, pictures or text messages are all private information that should be kept secret. Cryptography provides rule-based techniques, which consist of a set of mathematical calculations. It is referred almost solely to *encryption* operation, which basically converts the information to an unreadable form. *Decryption* operation is the reverse of encryption, which reveals the encrypted message to the original form. Cryptography provides the security of communication between two parties by using encryption and decryption techniques. Cryptosystems are intended to prevent other people from reading and changing the confidential message, except for the person who encrypts and sends the confidential message. Using encryption techniques in our daily lives, our encrypted messages can be transmitted securely over a secure channel and decrypted by only the message recipient.

Information security and cryptography built upon three main concepts known as *CIA triad*, which refers to confidentiality, integrity, and availability of the information. All encryption algorithms or any protocols aim to keep secure at least one of these components. Now, we provide elementary descriptions of each security measurement.

- **Confidentiality:** This measurement ensures to keep secret as a secret, i.e., nobody could be able to read the data while transmission between different parties. Encryption algorithms are used to provide this property. As we will discuss at the following chapter, symmetric cryptosystems or asymmetric cryptosystems are two ways of keeping confidentiality of messages.

- **Integrity:** This measurement states that the message (or even some parts) cannot be modified by any unauthorized users. Checksums or hash values can be used for comparison between each version whether there is an unknown change.

- **Availability:** This measurement refers to being always available for authorized users and protecting system-level attacks, which interrupt communication between systems.

In addition to CIA triad, there are also extra measurements as following:

- **Non-repudiation:** This measurement refers to non-deniability and proving of doing something, for example, sending an e-mail or perform an action on systems. Legal policies are for providing this assurance.
- **Authentication:** This measurement is a trustiness to initial message sender is who he/she claims to be.

2.2 Cryptography Basics

Before prividing further details, we will use the following definition to explain the key components that each cryptosystem has.

Definition 2.2.1. An encryption scheme is a tuple (P,C,K_E,K_D,E,D), where P,C,K_E and K_D are arbitrary sets (not necessarily distinct), and E and D are sets of functions, such that for each $k \in K_E$, there is a function $E_k : P \to C$, and for every $k \in K_D$ there is a function $D_k : C \to P$. This tuple must satisfy the condition that for every $t \in K_E$, there is a unique $s \in K_D$, such that $D_s(E_t(p)) = p$ for all $p \in P$.

From the above definition; the symbols P,C,K_E and K_D are known as *plaintext*, *ciphertext*, *encryption key* and *decryption key* respectively. Then, the symbols E and D be the *encryption* and *decryption functions* respectively. Now, we provide some non-technical explanations of each term:

- **Plaintext (P):** The original sensitive information which is transmitted between two parties.
- **Encryption Key (K_E):** The secret parameter for which changes the structure and content of plaintext, and generates its ciphertext.
- **Encryption (E):** The set of procedures and permutations to hide the original data from unauthorized users by using an encryption key and plaintext as an input.

- **Ciphertext (C):** The final unintelligible message after an encryption process of a plaintext.
- **Decryption Key (K_D):** The secret parameter for which enables to obtain the original message. It can be derived from the public key as in asymmetric cryptosystems or is the same as the encryption key as in symmetric cryptosystems.
- **Decryption (D):** The reverse operation of an encryption schema to retrieve original text from plaintext by using decryption key and ciphertext as an input.

In every encryption schema, if one inputs the plaintext P with an encryption key K_E provided before, encryption algorithm E outputs an unpredictable and unintelligible ciphertext message C. Conversely, the original message can be obtained by applying a decryption algorithm D, which has similar design with the encryption algorithm. Thus, decryption algorithm D inputs ciphertext C and the decryption key K_D, then outputs the original message P. Both encryption and decryption algorithms rely on *Correctness Property* such as $D_{K_D}(E_{K_E}(P)) = P$.

As we have seen from the definition that, encryption is a set of mathematical procedures which concern with the design and analysis of secure communication between two parties by protecting their sensitive information from unauthorized access. The purpose of the encryption process is to provide confidentiality of data by scrambling it in a way of which just the only people who have *the key* can reveal the original data. Thus, it is obvious that the main demand from an encryption schema is not to allow any unauthorized users to decrypt the encrypted message without having the key used in encryption.

In cryptography, the encryption and decryption algorithms should be designed not to compromise the security of systems. The algorithms should provide substitution and permutation to make the prediction of any secret message hard and complex. Now, we give one of the main principles used in any cryptosystem.

Kerckhoffs' Principle. *When designing or evaluating the security of cryptosystems, it should be secure even if everything about the system, except the key, is public knowledge.*

In ancient cryptography, the design of cryptosystems kept its secret to form a barrier when an attacker intended to hack it. However, the security of an algorithm should not be depended on the design of the algorithm according to Kerckhoffs' Principle. It should not cause any vulnerability even if the algorithm is known broadly so that the security of the algorithm should solely depend on the security of the encryption key K_E. By adopting this design principle, the users of the algorithm can be aware of possible attacks and it is also open for any cryptanalysis by experts to have a more secure system. Thus, Kerckhoffs' Principle became a fundamental design criterion in modern cryptography.

There are also two entities who want to communicate with each other: (1) Alice as a sender and (2) Bob as a receiver. During secret message transmission, If Alice has K_E and Bob has matched K_D, it is known as *one-way secure communication channel*. However, if Alice and Bob have K_E, and Alice has matched K_D, there is a *two-way secure communication channel*. We later present more broad explanations of them as symmetric and asymmetric encryption.

2.3 Types of Cryptosystems

In modern cryptography, there are two types of cryptosystems when encryption and decryption system considered. We provide explanations and features of each in the following sections:

2.3.1 Secret-key cryptosystems

The secret key cryptosystems, also known as symmetric cryptosystems, are known as the oldest techniques used in history since the ancient cryptography era. In secret key cryptosystems, both parties agree on a secret *shared key* at the initial stage of the encryption process and the same key is used in the decryption process. The secret keys should be a randomly chosen text or integer and initially transmitted to each party in a secure channel. Thus, the security of the symmetric key cryptosystems is mostly based on keeping the key transmission channel secure. Even the message integrity can be provided by one of the modes of operation, if an attacker corrupts the key during transmission, he can decrypt all plaintext and so reveal all original messages.

The key property of this type of cryptosystems is the need for secure key transmission between parties. Also, the algorithms provide the same security with shorter key size comparing to public-key cryptosystems. Because of the efficiency of this feature, many cryptosystems use secret-key encryption schemas. Besides, one of the drawbacks of a secret key cryptosystem is generating different random keys for each pair of entities. If there is a group communication including n people, $n(n-1)/2$ different key pairs are needed to be generated to enable communication between any two-party. Generating many key pairs also leads to the need for secure storage and changing the key regularly to prevent possible attacks.

Thus, secret key cryptography (such as DES, AES, 3DES, etc.) is widely used in many applications even key establishment between two-party and trustiness are still open questions.

2.3.2 Public-key cryptosystems

Public-key cryptosystems (PKC), also known as asymmetric cryptosystems, consist of a pair of a public and private key for encryption and decryption process. Public keys can be sharable with anyone; however, the mathematically related private key is owned and known by just its owner. These cryptosystems have been invented to eliminate the need to share the secret key with each party in a secure channel. The secret message encrypted with the receiver's public key, and only the receiver that has matched private key can decrypt the encrypted information. Thus, any adversaries who do not have the receiver's secret key cannot decrypt the secret message.

The security of this type of cryptosystems is based on the generation of keys and securing the private keys. One of the keys in pair is the public key, which can be known and distributed to everyone without adjusting any security agreement and it does not easily reveal the generation of the private key. Thus, key pair generation algorithms are mostly based on some unsolved mathematical problems which provide computational hardness to not to derive the private key from its paired public key. Additionally, public-key cryptosystems require a relatively longer key size which leads to slower encryption schemas comparing to secret key cryptosystems.

Moreover, one of the challenges in PKC is the trustiness and proof of public keys used in encryption. The sender needs to ensure that the public key is not corrupted

by any malicious third party. This problem leads to a solution by having a Public Key Infrastructure (PKI) to manage and certificates public keys. However, public-key cryptosystems such as RSA, ElGamal Signature Schema and Diffie-Hellman Key Exchange schema are also widely used if one needs to replace the limitations of secret key cryptosystems.

The security of this type of cryptosystem is based on the protection of private key, which is mathematically derived from the public key. Thus, we present some mathematical problems such as discrete logarithm problem and integer factorization problem, which are computationally unsolved yet. We will give problem definitions of both in the following sections 2.4 and 2.5 respectively.

2.4 Discrete Logarithm Problem

2.4.1 Problem definition

Definition 2.4.1. Let g be the generator of the group G with the order of n. Given group and n, for any integer

$$h = g^x \tag{2.1}$$

computation of x is the discrete logarithm problem. Such an integer x is called a *discrete logarithm* of h to base g.

To solve the DLP in general, the first algorithm which comes to mind is a trivial exhaustive search. However, its complexity is quite high and not efficient in terms of running time. Although there is no efficient method to solve DLP efficiently in general, there are some algorithms that can solve very efficiently for some cases. The methods are called Index-Calculus, Pohling-Hellman, Shank's Baby-Step-Giant-Step Method, Pollard Rho Method, and Lenstra's Elliptic Curve Method. (See Section 11.6 in the book [19] for more information)

2.4.2 DLP based PKC

In previous subsections, we gave the problem definition of the discrete logarithm. In this subsection, we present examples of some public key encryption systems whose

security is based on the difficulty of the computationally intractable discrete logarithm problem.

2.4.2.1 Diffie-Hellman key exchange

As seen in Section 2.3, secret key sharing between two entities is a critical issue in symmetric type cryptosystems. Thus, Whitfield Diffie and Martin Hellman invented the algorithm in 1976, which allowed sharing a secret without the need for a secure communication channel.

Alice	**Bob**
Agreement Stage p: prime g: generator s.t. $g < p$ and $g \in G$.	
$a \leftarrow \mathbb{Z}_n$ and $a < p$ $h_A = g^a \bmod p$ $k = (h_B)^a = g^{ab} \bmod p$	$b \leftarrow \mathbb{Z}_n$ and $b < p$ $h_B = g^b \bmod p$ $k = (h_A)^b = g^{ab} \bmod p$

Figure 2.1 : Key Exchange Schema.

As we have seen from the figure above, Alice and Bob initially agree on a finite cyclic group G with the order of n. Then, they choose a prime integer p and a primitive root of p such that $g < p$ and $g \in G$. Then, Alice chooses a random integer $a \in \mathbb{Z}/n\mathbb{Z}$ such that $a < p$ as her secret key and calculates $h_A = g^a \ (\bmod p)$ as a public key. She sends her public key h_A to Bob. (Remember from the Section 2.3.2 that the secret key should be unknown except Alice; however, the public key h_A is sharable to anyone.) At the same time, Bob chooses his secret key $b \in \mathbb{Z}/n\mathbb{Z}$ such that $b < p$ and calculates his public key $h_B = g^B \ (\bmod p)$. Bob sends his public key h_B to Alice. Now, each party has other's public key. Bob calculates $(h_A)^b$ and Alice computes $(h_B)^a$. At the final stage, they obtain the same calculation since $(h_A)^b = g^{ab} = (h_B)^a$.

Example. Choose $p = 281$ and a primitive element $a = 3$ of p. Then, Alice and Bob choose $a = 59$ and $b = 181$ respectively. Before key exchange stage; Alice computes $3^{59} \bmod 281 = 74$ and Bob computes $3^{113} \bmod 281 = 270$. After that, they exchange public keys. Then; Alice computes $74^{113} = 82$ and Bob computes $270^{59} = 82$.

Security. Assume that an attacker knows everything except the secret keys a and b given in Figure 2.1. Even the values of p, g and the result of the computation g^a, g^b are known to the attacker, it is not computationally feasible to compute the secret keys in a reasonable amount of time. The problem with the use of very large primes is still very expensive in terms of calculation, even for modern supercomputers. Thus, the security of the Diffie-Hellman key exchange method is based on the difficulty of solving the discrete logarithm problem.

2.4.2.2 ElGamal encryption schema

We have seen from Diffie-Hellman Protocol that it is a great idea to exchange the secret key between two parties, which was a major problem in secret key cryptography. However, the secret message is not encrypted in the Diffie-Hellman method. Thus, in 1984, Taher ElGamal invented a new public-key cryptosystem, which is similar to Diffie-Hellman protocol. However, it also encrypts messages and generates digital signatures of entities. Moreover, it also provides probabilistic encryption which means that a message can be encrypted with many possible ways depending on the choice of the key. The method consists of 3 main stages: key generation, encryption, and decryption. Now, we describe each part by given algorithms and an example.

Alice	**Bob**
Key Generation Stage	
p_A: prime g_A: generator modulo p_A a: random st. $2 \leq a \leq p_A - 2$ $b_A = g_A^a \mod p_A$	p_B: prime g_B: generator modulo p_B b: random st. $2 \leq b \leq p_B - 2$ $b_B = g_B^b \mod p_B$
Public: $\{p_A, g_A, b_A\}$ Private: a	Public: $\{p_B, g_B, b_B\}$ Private: b

Figure 2.2 : ElGamal Key Generation.

Since ElGamal Methods is a public-key cryptosystem, we generate two different keys: a public key is used for encryption and a private key is used for decryption. Now, we give details on encryption and decryption stages by the following figures:

Let us illustrate the encryption schema with an example:

Example.

1. Key Generation:

Encryption
Public key received st. $\{p, g, b\}$ Select random k st. $1 \leq k \leq p-1$ Calculate $x = g^k \mod p$ Calculate $y = m \cdot (g^a)^k \mod p$ where $0 \leq m \leq p-1$
Ciphertext: (x, y)

Figure 2.3 : ElGamal Encryption.

Decryption
Ciphertext received st. $\{x, y\}$ Calculate $m = x^{-a} \cdot y \mod p$
Plaintext: $(g^k)^{-a} \cdot m \cdot (g^a)^k \mod p = m$

Figure 2.4 : ElGamal Decryption.

- Alice chooses $p = 41$ and finds the primitive element $g = 6$,
- Chooses her private key $a = 29$, and
- Calculates her public key $b = g^a = 6^{29} \mod 41 = 22$.

2. Encryption:
   - Bob receives Alice's public key $\{p = 41, g = 6, b = 22\}$
   - Bob chooses $k = 17$ and the message $\mathbf{m} = \mathbf{35}$,
   - Calculates $x = g^k = 6^{18} = 26 \mod 41$,
   - Calculates $y = m \cdot (g^a)^k = 35 \cdot (6^{29})^{17} \mod 41 = 20$, and
   - Sends the ciphertext pair $(x, y) = (26, 20)$.
3. Decryption:
   - Alice finds the random $k = x^a = 26^{29} = 24 \mod 41$, and
   - Reveals the secret message $m = k^{-1} \cdot y = 24^{-1} \cdot 20 \mod 41 = \mathbf{35}$.

Security. We know from the algorithm that secret values are the private key a, random integer k and the message m. If the attacker knows the private key a of Alice, then he can decrypt the messages. However, finding a by knowing public values $\{p, q, b, x, y\}$ is discrete logarithm itself. Thus, the security of ElGamal is as hard as the discrete logarithm problem as we mentioned in Section 2.4.1.

2.5 Integer Factorization Problem

As we have already seen from the previous sections, the general idea behind the public-key cryptography is to have different keys for encryption and decryption. Since the private key is derived from the public key, key derivation algorithms were based on mathematical unsolved problems, i.e., the solutions are not feasible even with modern supercomputers. One of the methods is to choose the Discrete Logarithm Problem (DLP) to be a base for the security of the algorithm. Diffie-Hellman Key Exchange and ElGamal Encryption are examples where the security only depends on DLP.

The usage of prime numbers in the RSA algorithm increased the challenge in prime factoring for many years. Thinking of the last factored RSA number is RSA-220 which consist of 220 digits, integer factorization problem is not easy to solve in a timely manner. In this section, we give another unsolved mathematical problem, which constructs the security of RSA.

2.5.1 Problem definition

According to Fundamental Theorem of Arithmetic, every integer has unique prime factors. It is trivial if the number is prime since prime numbers have no divisors other than 1 and itself. Otherwise, any factorization algorithm gives the multiplies of composite numbers. However, the difficulty of problem increases for very large composites.

The next definition is the main idea behind the factorization:

Definition 2.5.1. Let n denote a composite number which can be written as the form of

$$n = \prod_{i=1}^{k} p_i^{e_i} \tag{2.2}$$

Then, integer Factorization Problem (IFP) can be defined as a decomposition of n into p_i with repeating count of e_i.

Even there is no efficient prime factorization algorithm yet, there are some algorithms with their drawbacks. The first and the simplest one is to try all possible numbers to

check whether it divides n or not. Besides the elementary methods, we will mention elliptic curve factorization method in Chapter 3, which is most recent.

2.5.2 IFT based PKC: RSA

RSA algorithm is invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT and became one of the most widely used public-key cryptosystems for years. RSA algorithm has a key pair such that a known public key and a private key derived from the public key, as well as the other previous cryptosystems. Furthermore, the security of the RSA algorithm depends on the difficulty of factorization of large number (see Section), which is the product of two hundred-digit prime numbers. Now, we provide details of the method afterward giving examples and security discussion.

First, we come up with the key generation stage by the following figure:

Key Generation
Choose: $\{p,q\}$ st. p,q are large primes Calculate: $n = p \cdot q$ Choose: e st. $gcd(e,\phi(n)) = 1$ and $1 < e < \phi(n)$ Calculate: $d = e^{-1} (\text{mod } \phi(n))$
Public key: (e,n) Private key: (d,n)

Figure 2.5 : RSA Key Generation.

As we have seen from the figure that $\phi(n)$ is the Euler's Totient Function as we will discuss the details in Chapter 4. Also, one of the public key e and the private key d are mathematically related to each other, i.e., one of each is multiplicative inverse of another modulus $\phi(n)$.

After the key preparation step is completed, Bob gets Alice's public key pair $\{e_a, n_a\}$ and computes the ciphertext $c = m^{e_a} \mod n_a$ where m is the secret message. When Bob sends the ciphertext c to Alice, she decrypts using her private key pair $\{d_a, n_a\}$ by doing the calculation $m = c^{d_a} = m^{e_a \cdot d_a} \mod n_a$.

Example.

1. Key Generation:

   - Alice chooses $p = 151$, $q = 607$

- Calculates $n = p * q = 91657$ and $\phi(91657) = 90900$.
- Chooses $e = 7$ st. $gcd(90900, 7) = 1$
- Calculates $d = e^{-1} \pmod{90900} = 51943$

2. Encryption:

   - Bob receives Alice's public key $\{e = 7, n = 91657\}$
   - Bob encrypts the message $\mathbf{m} = \mathbf{96}$ by calculating $96^7 \bmod 91657 = 76779$,

3. Decryption:

   - Alice decrypts the message by calculating $m = 76779^{51943} \bmod 91657 = \mathbf{96}$.

Correctness.

The correctness of the algorithm can be demonstrated by using the following theorem and its proof.

Theorem 2.5.1 (RSA Correctness)**.** *$m^{ed} \equiv m \pmod{n}$ holds for all integers $m \in \mathbb{Z}_n$.*

Proof. It suffices to prove both in modulus p and q because of the Chinese Reminder Theorem (CRT). Since p and q are distinct primes such that $gcd(p, q) = 1$, CRT states that if

$$m^{ed} \equiv m \pmod{p} \quad \text{and} \quad m^{ed} \equiv m \pmod{q} \tag{2.3}$$

then it implies that

$$m^{ed} \equiv m \pmod{n} \tag{2.4}$$

So, we first prove for modulus p, then the further will be the same for modulus q.

Firstly, we know from the key generation stage of RSA that for $e > 0$ and any $k > 0$

$$e \cdot d = 1 \pmod{\phi(n)}$$
$$e \cdot d = k \cdot \phi(n) + 1 = k \cdot (p-1) \cdot (q-1) \text{ for some integer } k > 0$$

holds. Additionally, Euler's extension to Fermat's Little theorem asserts that

$$m^{p-1} \equiv 1 \pmod{p} \tag{2.5}$$

Therefore;

$$\begin{aligned} m^{e\cdot d} &= m^{1+k\cdot(p-1)\cdot(q-1)} \cdot \ (\mathrm{mod}\ p) \\ &= m \cdot (m^{p-1})^{(q-1)}\ (\mathrm{mod}\ p) \\ &= m \cdot (1)^{(q-1)}\ (\mathrm{mod}\ p) \\ &= m\ (\mathrm{mod}\ p) \end{aligned}$$

Now, we have seen that $m^{e\cdot d} \equiv m\ (\mathrm{mod}\ p)$ for all integers m. The proof also satisfies if one changes p with q. Thus, we have proved by using CRT and Fermat's Little Theorem that

$$m^{e\cdot d} \equiv m\ (\mathrm{mod}\ n) \tag{2.6}$$

holds for all integer m.

□

Security. The security of RSA solely depends on the difficulty of solving the integer factorization problem. Since RSA is the first easily used public-key cryptosystem, it is implemented in many systems for years and its security analyzed in many ways. The usual attack is prime factoring very large integers. From 1991, RSA Laboratories arrange factoring challenges to encourage research on cracking RSA by the area of computational number theory. Even they did not succeed yet, they applied many attacks on RSA such as timing attack, partial key exposure attack, implementation attacks, etc. [30]. Finally, we conclude that RSA can be broken at all if one discovers a fast integer factorization algorithm, which also works well with very large RSA modulus.

3. PRELIMINARIES OF ELLIPTIC CURVES

An elliptic curve is an algebraic structure which used in many applications in cryptography because it provides the same security level with lower key size. In terms of number theory, these curves are used in modern primality testing and integer factorization algorithms. Elliptic curves first introduced by Miller and Koblitz in 1980, then Lenstra used them for integer factorization.

Since this thesis includes primality testing algorithms with elliptic curves, we give some brief notations and definitions in this chapter.

3.1 Definitions

Definition 3.1.1 (Generalized Weierstrass Equation)**.** An elliptic curve over a field K is given by

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x_2 + a_4x + a_6 \tag{3.1}$$

where the constants $a_i \in K$ for $i = 1,2,3,4,6$. Then, the set of points on the curve E can be expressed by the equation

$$K(K) = (x,y) \in K : y^2 + a_1xy + a_3y = x^3 + a_2x_2 + a_4x + a_6 \cup \infty \tag{3.2}$$

Definition 3.1.2 (Simplified Weierstrass Equation)**.** Let

$$E : y^2 = x^3 + ax + b \tag{3.3}$$

where the constants $a,b \in \mathbb{Z}_n$ is denotes the simplified Weierstrass form of an elliptic curve.

Definition 3.1.3 (Discriminant of the curve)**.** The discriminant of an elliptic curve given in the Weierstrass form is

$$\Delta = -{d_2}^2 d_8 - 8{d_4}^3 - 27{d_6}^2 + 9d_2d_4d_6 \tag{3.4}$$

where :

$$d_2 = a_1 + 4a_2$$

$$d_4 = 2a_4 + a_1a_3$$

$$d_6 = {a_3}^2 + 4a_6$$

$$d_8 = {a_1}^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$$

and $\Delta \neq 0$.

Definition 3.1.4 (Smoothness)**.** If the given elliptic curve has discriminant $\triangle \neq 0$, the curve is smooth which means that no points on the curve have two distinct tangent lines.

3.2 Elliptic Curve Arithmetic

Now, we have introduced the concepts of elliptic curve arithmetic including algebraic group operation.

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are two points on elliptic curve, we denote $P_\infty = (0, 1)$ as *point at infinity* or *identity point* of elliptic curve. Since elliptic curves are symmetric geometrically, we denote as the negative $-P = (x, -y)$. If we add two points P and Q algebraically, we denote this group operation as $\oplus$, we get an 3^{rd} point $P \oplus Q = R = (x_3, y_3)$ on the curve as shown in the figure below.

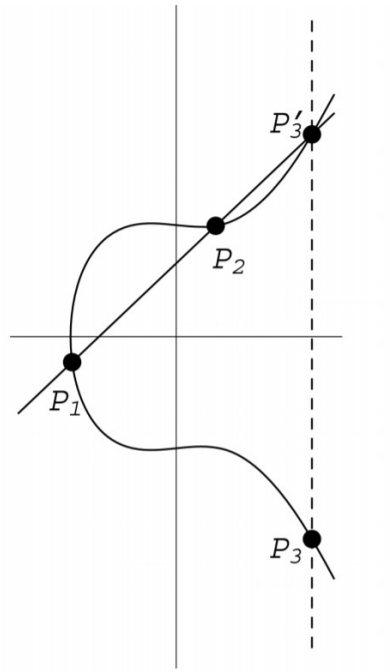

Figure 3.1 : Point addition on elliptic curves.

3.2.1 Group law

The following theorem gives a brief explanation of group operation on E.

Theorem 3.2.1. *The addition of points on an elliptic curve E satisfies the following properties:*

1. Addition: If $P, Q \in E$, *then* $P \oplus Q \in E$

2. Identity: $O \oplus P = P \oplus O = P$

3. Inverse: $P \oplus (-P) = O = P_\infty$

4. Associativity: $P \oplus (Q \oplus R) = (P \oplus Q) \oplus R$

5. Commutativity: $P \oplus Q = Q \oplus P$

If the last property, i.e., commutativity rule, satisfies, then we call this group as **abelian group**.

We give details about algebraic point addition $\oplus$ operation on proceeding sections.

3.2.2 Group order

The number of points in the elliptic curve E over the finite field F_q is called *the order of the curve* and represented by $\#E(F_q)$. Hasse's theorem on elliptic curve 3.2.2 gives the interval of the order of the curve E over finite fields [29]. It estimates the order by bounding the value both above and below which are only depend on the finite field and not the curve.

Theorem 3.2.2 (Hasse' Theorem)**.** *Let p be a prime number and* $q = p^n$ *for* $n \in \mathbb{N}$. *The order of an elliptic curve E over* F_q *is bounded by*

$$q + 1 - 2\sqrt{q} \leq \#E(F_q) \leq q + 1 + 2\sqrt{q} \tag{3.5}$$

Proof. See [29] for the complete proof. □

Hasse's Theorem is used in many elliptic curve cryptosystems over years. For instance, R. Schoof used this theorem [11] to generate an algorithm for computing the number of the points on elliptic curves. Additionally, elliptic curve primality proving algorithms included this theorem in their proofs.

After describing the main theorem of group order of elliptic curves, we now give details on their group structures.

3.2.3 Group structure

Let E denote an elliptic curve over the field K. There is a *chord-and-tangent* rule which generates the 3rd point on $E(K)$ by drawing a line passing over two distinct points or a tangent line of a single point on the curve E.

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points on elliptic curve E over $\mathbb{F}_p$. Then, we can uniquely describe a third point by adding $P \oplus Q = (x_3, y_3)$.

The point addition and doubling operation can be done by following way:

$$x_3 = \lambda^2 + 2a - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1. \tag{3.6}$$

where

$$\lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{if } P \neq Q \\ \dfrac{3x_1^2 - 4ax_1 + a^2}{2y_1} & \text{if } P = Q \end{cases}$$

Note that $-P = (x, -y)$ and $P \oplus (-P) = P_\infty$ where P_∞ is called point O or point at infinity. We should also mention here that there are many other ways to perform group operation in $E(\mathbb{F}_q)$. (For other methods, see [2, 24] related chapters.)

In this way, the computation of the slope (λ) requires to take the multiplicative inverse of either "$x_1 - x_2''$ or "$2y_1''$ in modulus n. If the greatest common divisor of any of the denominator and n is not equal to 1, the operation fails to compute point addition and returns a factor of n. This is the key point of Lenstra's Elliptic Curve Factorization Method [20].

3.2.4 Scalar multiplication

The scalar multiplication operation consists of a positive integer $k > 0$ and a point P on elliptic curve E. These three ingredients form the basis of the security level of elliptic curve cryptosystems which are based on the difficulty of finding the scalar multiplier k in a given multiplication operation $Q = [k]P$ for given point P, Q on E and the group

operation denoted by $\oplus$. The scalar multiplication is mainly applying group addition operation to the point P repeatedly to itself and finding another point Q on curve E. It means that

$$Q = [k]P = \underbrace{P \oplus P \oplus P \oplus ... \oplus P}_{k-times}$$

This definition can be extended for all integer k by $[0]P = P_\infty$ and $[-k]P = [k](-P)$ for $k < 0$.

This basic recursive point addition method is very expensive because it requires the longest addition operation chain for large values of k. Here, the problem arises to retrieve scalar multiplier k with the least number of steps. Since finding the shortest addition chain is actually a NP-complete problem, various heuristics methods have been invented to reduce the number of additions and doublings. For simplicity, we will not give a detailed analysis of these methods; but, general structures of algorithms will be given at this point.

3.2.4.1 Binary method

The 'school book' method to retrieve n-bit integer k starts with binary representation of $k = k_0 + 2k_1 + 2^2k_2 + ... + 2^{n-1}k_{n-1}$ where $k_0, k_1, ...k_{n-1} \in \{0,1\}$. By this way, the multiplication cost depends on the length of k and the number of 1s in representation. As a result, binary method takes $n-1$ doublings and $w(k)$ additions such that w is the Hamming weight which represents the number of non-zero bits in k. Thus, this method speeds-up retrieving the multiplier k by having $n/2$ additions on average without any precomputation step.

The method works as follows:

Algorithm 1 Binary Method Point Multiplication

Require: Point P, n-bit integer k

1: **procedure** CALCULATE($[k]P$)
2: **if** $k_{n-1} = 1$ **then** $Q \leftarrow P$
3: **else** $Q \leftarrow 0$
4: **for** $i = n-2 \rightarrow 0$ **do**
5: $Q \leftarrow Q \oplus Q$
6: **if** $k_i = 1$ **then** $Q \leftarrow Q \oplus P$
7: **return** Q

Signed Digits

The elliptic curve representation has negative points $P = (x, -y)$ and the point subtraction costs the same as the point addition. However, this representation of any given positive integer speeds up the computation by reducing the number of non-zero digits comparing to signed digit binary representation of the same number. This unique representation is called *width-w non-adjent form* and denoted by $NAF_w(k) = \sum_{i=0}^{n-1} k_i 2^i$ for a positive integer k where each $k_i \in \{-1, 0, 1\}$ and width size w. We can easily see that NAF_w conversion of an integer has fewer non-zero digits. It is proved in [MorOl, 1990] that the density is approximately $1/3$ for $w = 2$ and $1/(w+1)$ in general. [Sol, 2000].

Now, we will give how to find NAF_w representation of given $n-$bit scalar k.

Algorithm 2 Converting to NAF_w Representation

Require: n-bit integer k and parameter w

procedure $NAF_w(k)$
  $n \leftarrow 0$
  while $k > 0$ **do**
    if k is odd **then**
      $k_n \leftarrow k$ ***mods*** 2^w
      $k \leftarrow k - k_n$
    else $k_n \leftarrow 0$
    $k \leftarrow k/2$
    $n \leftarrow n + 1$
  return $(k_{n-1}, k_{n-2}, ..., k_1, k_0)_{NAF_w}$

where ***mods*** function can be computed as:

procedure MODS
  if $k \mod 2^w \geq 2^{w-1}$ **then**
    return $((k \mod 2^w) - 2^w)$
  else
    return $(k \mod 2^w)$

Each point multiplication method can be modified by adding the computation of NAF representation of k as a pre-step. Thus, we first give the modified version of the binary multiplication method as follows:

Algorithm 3 Binary Method Point Multiplication by using NAF Representation

Require: Point P, n-bit integer k

1: **procedure** CALCULATE($[k]P$)
2: **if** $k_{n-1} = 1$ **then** $Q \leftarrow P$
3: **else**$Q \leftarrow 0$
4: **for** $i = n-2 \rightarrow 0$ **do**
5: $Q \leftarrow Q \oplus Q$
6: **if** $k_i = 1$ **then** $Q \leftarrow Q \oplus P$
7: **if** $k_i = -1$ **then** $Q \leftarrow Q \ominus P$
8: **return** Q

3.2.4.2 Windowing method

This method is similar to the binary method; however, it consists of precomputation step and processes blocks of w (window size) bits in one time. Besides it has almost the same complexity with the binary method, it provides fewer additions with extra memory to store precomputed values. There are two ways of applying windowing method: fixed-size windowing method and dynamic (sliding) windowing method. We give the algorithms and their analysis respectively.

Fixed-size Windowing

Algorithm 4 Fixed-size Windowing Method

Require: Window size w, Point P, n-bit integer with k

1: $k = k_0 + 2^2 k_1 + 2^{2w} k_2 + ... + 2^{(n-1)w} k_{n-1}$
2: **procedure** CALCULATE($[k]P$)
3: **if** $k_{n-1} = 1$ **then** $Q \leftarrow P$
4: **else**$Q \leftarrow 0$
5: **for** $i = n-2 \rightarrow 0$ **do**
6: $Q \leftarrow Q \oplus Q$
7: **if** $k_i = 1$ **then** $Q \leftarrow Q \oplus P$
8: **return** Q

Sliding Window

Instead of fixed size windowing method, this method has a dynamic windows approach:

Algorithm 5 Sliding Window Method for Point Multiplication

Require: Window size w, Point P, n-bit integer $k = (k_{n-1}...k_0)_2$ and precomputed points $[3]P, [5]P, ..., [2^w - 1]P$

1: **procedure** CALCULATE($[k]P$)
2: $Q \leftarrow P_\infty$
3: $i \leftarrow n-1$
4: **while** $i \geq 0$ **do**
5: **if** $k_i = 0$ **then**
6: $Q \leftarrow [2]Q$ and $i \leftarrow i-1$
7: **else**
8: $s \leftarrow max(i-k+1, 0)$
9: **while** $n_s = 0$ **do** $s \leftarrow s+1$
10: **for** $h = 1 \rightarrow i-s+1$ **do** $Q \leftarrow [2]Q$
11: $u \leftarrow (n_i...n_s)_2$
12: $Q \leftarrow Q \oplus [u]P$
13: $i \leftarrow s-1$
14: **return** Q

3.2.4.3 Montgomery ladder

The mainly used scalar multiplication algorithm in the context of elliptic curves is proposed by [33] and redesigned for many years. From the computational the point of view, this method has the advantage of time and power consumption. However, it exposes to *side-channel attack* shown in [15] that the full private key can be extracted after performing timing against 200 signatures.

Algorithm 6 Montgomery Ladder

Require: Point P, n-bit integer k

1: **procedure** CALCULATE($[k]P$)
2: $Q \leftarrow 0$
3: **for** $i = 1 \rightarrow n$ **do**
4: **if** $k_{n-i} = 0$ **then**
5: $P \leftarrow Q \oplus P$ and $Q \leftarrow 2Q$
6: **else**
7: $Q \leftarrow Q \oplus P$ and $P \leftarrow 2P$
8: **return** Q

4. PRIMALITY TESTING AND PROVING ALGORITHMS

4.1 Mathematical Underpinnings

The primality testing is one of the most important topics in computational number theory and cryptography since ancient times. For a given integer n, it is trivial to prove primality if n is even or a very small odd number. However, if we think very large integers that have several hundred digits, the problem was not so easy because of the lack of efficient primality testing algorithms that run in polynomial time. In this section, we present some theorems and definitions regarding the fundamentals of number theory and then describe some primality testing algorithms in details in the next sections.

Firstly, we introduce the important theorem which was proposed by Carl Friedrich Gauss in 1801, which is a remarkable principle in theory.

Theorem 4.1.1 (Fundamental Theorem of Arithmetic)**.** *Every integer $n > 1$ and in $\mathbb{Z}$ is either a prime itself or the product of prime numbers.*

Example. Some examples of non-negative integers as given below:

$$255 = 3 \cdot 5 \cdot 17 \quad , \quad 2520 = 2^3 \cdot 3^2 \cdot 5 \cdot 7 \quad , \quad 60291 = 3^3 \cdot 7 \cdot 11 \cdot 29$$

This theorem asserts that the numbers except for primes are the form of the product of two (or more) prime numbers, which are known as *composite* numbers. Since this theorem states a unique representation and factorization if the order is ignored. The integers can be written as the form

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k} = \prod_{i=1}^{k} p_i^{e_i} \tag{4.1}$$

where p_i's are distinct primes with the order of $p_1 < p_2 < ... < p_k$ and e_i's are positive integers corresponding occurrences of each prime p_i. Since 1's could be added finitely to this product, so 1 is not accepted as prime (or even composite) [27] to prevent the

uniqueness of factorization. If 1 would be a prime, the factorization would not be unique since for example $13 = 13 \cdot 1 \cdot 1 \cdot 1 \cdots$.

Proof. For a complete proof, see section 2.3 in the book [6].

□

Hence, prime numbers are building blocks of all integers in $\mathbb{Z}$. The subsequent theorem gives a point of view on the density of prime numbers until a limit. Now, we introduce a very remarkable theorem which conjectured by Carl Fredrich Gauss in 1792 and proved elementarily by Paul Erdös (1949) and Atle Selberg (1950).

Theorem 4.1.2 (Prime Number Theorem)**.** *For any $x \in \mathbb{R}$, let $\pi(x)$ denote prime-counting function which returns the number of primes that not exceed the bound x and $ln(x)$ is natural logarithm. So; $\pi(x)$ can be denoted as*

$$\pi(x) \sim \frac{x}{ln(x)}$$

Here we also give the first 10 calculations of $\pi(x)$ and $\frac{x}{ln(x)}$ in the table below:

Table 4.1 : Density of primes.

x	$\pi(x)$	$\frac{x}{\pi(x)}$
10	4	2.5
10^2	25	4
10^3	168	5.952
10^4	1229	8.137
10^5	9592	10.425
10^6	78498	12.740
10^7	664579	15.047
10^8	5761455	17.357
10^9	50847534	19.667
10^{10}	455052511	21.975

As we have seen from the graph and table that $\pi(x)$ is monotonically increasing and the density of prime numbers up to a certain large limit is less frequent. So, it is difficult to find very large prime numbers in a reasonable amount of time. The largest prime number founded in the universe, known under the code name M77232917, has more than 23 million digits [39]. However, the following theorem proposes that there is no limit to find a prime number.

Theorem 4.1.3 (Euclid's First Theorem)**.** *Infinitely many prime numbers exist.*

Proof. Before giving the existence and uniqueness proofs of the theorem above, we first introduce the lemma used in the proof.

Lemma 4.1.1 (Euclid's Lemma)**.** *Let $p \in \mathbb{Z}$ be a prime number and $a, b \in \mathbb{Z}$. Then, p is divisible by the product of ab if and only if p is divisible by a or b.*

We follow the proof by contradiction method by assuming that the number of primes is finite. Let call this count as n and the bound as N. Now, denote that $N = p_1 \cdot p_2 \cdot p_3 \cdots p_n + 1$. It is apparently seen that p_n is the largest prime, $N > p_n$ and N is not prime. By using Theorem 4.1.1, we deduce that N must have a prime factor in our list, call p_i. The number N will have a remainder 1 when divided by one of the p_i. This contradiction makes N a prime aside from $N \neq 1$.

□

Primality tests deal with determining and proving the primality of given a number n. In high school mathematics, we all know that one of the basic methods for determining primality is founded by Sieve of Eratosthenes. The idea behind this test is checking if n is divisible by all numbers less than or equal to $\sqrt{n}$. If any number in the list divides n, the test outputs the compositeness of n; otherwise, n will be a prime. Here, there may be a modification in the list since we do not need to check for all even numbers. For example, if n is not divisible by 2, then it will also not divisible by multiples of 2 up to $\sqrt{n}$, let say bound B. Moreover, the idea is the same for other small prime numbers 3,5,7 and so on. If the given number n is not divisible by any small prime, it also will not divisible by its multiples, so we mark its multiples to not to check again. Thus, we continue to check if unmarked numbers divide n or not. However, this method is very inefficient considering very large numbers that have several hundred digits. Hence, we cover mostly the theorems and algorithms used for primality test in an efficient way.

The primality tests are more broader in two categories: deterministic tests and probabilistic tests. Deterministic tests can exactly determine the primality of a number but it is slower than probabilistic tests. Although the probabilistic tests are quite fast, these tests can erroneously determine (with a very small probability) a composite

number as prime or a prime number as composite. The next sections we describe and provide some examples on both probabilistic and deterministic tests.

4.2 Probabilistic Primality Tests

4.2.1 Fermat's test and Euler's extension

Fermat's test

Fermat's Little Theorem and primality testing algorithm is an ancestor of later developed algorithms. There are some algorithms which are just an extension of Fermat's theorem. Thus, we state primality testing algorithms by beginning from Fermat's test.

Theorem 4.2.1 (Fermat's Little Theorem)**.** *If p is prime and $a > 0$ with $p \nmid a$, then for all $a \in \{1, 2, ..., p-1\}$*

$$a^{p-1} - 1 \equiv 0 \pmod{p} \tag{4.2}$$

Proof. See [6] for more detailed proof. □

Moreover, there is a useful corollary which can be obtained by multiplying both sides of the congruence stated in 4.2.1 by a. The restated form does *not* have a restriction for a to be relatively prime to p. This variant can be applied to any a (regardless of it is relatively prime to p). The form can be written as:

$$a^p \equiv a \pmod{p} \tag{4.3}$$

Fermat's Little Theorem states that if p is prime, then $a^p - a$ is always divisible by p for any base a. Now, we illustrate the use of this theorem with an example.

Example.

Let us choose a prime number $p = 101$ and an integer $a = 13$ such that $p \nmid a$.
By using Theorem 4.2.1, we calculate that $13^{100} - 1 \equiv 0 \pmod{101}$.

The Fermat's little theorem is a proper fit to reduce residue if one calculates some large powers of a number. For example; $7^{214529} \equiv (7^{652})^{329} \equiv 7^{21} \equiv 186 \pmod{653}$ because $7^{652} \equiv 1 \pmod{653}$. Moreover, we can obtain information on the compositeness of a

number such that if the reminder for the division of a^{p-1} by p is not 1 for an integer $a > 1$, p is definitely composite.

However, there is a drawback such that some numbers satisfies the primality test algorithm and acts like a prime number even though they are actually not prime. Now, we introduce this snag and special composites which are also the main focus of this thesis.

Definition 4.2.1 (Fermat pseudoprimes)**.** The composite number $n \geq 2$ which succeeds the condition such that $a^{n-1} - 1 \equiv 0 \pmod{n}$ for an integer $a > 0$ are called Fermat pseudoprimes to base a, or a-pseudoprime, $psp(a)$ for short.

If the number n does not pass the case, we can say that it is a composite; however, the converse of this proposition is not always true as the following example shows.

Example.
Let us choose the number $n = 341$ to test for base $a = 2$ and calculate the condition provided by Theorem 4.2.1. Although the calculation is resulted as $2^{340} - 1 \equiv 0 \pmod{341}$, n is actually not prime since $341 = 11 \cdot 31$. We define the *base 2* as the following definition states:

Definition 4.2.2 (Witness/Liar)**.** For a given odd number n, the congruence

$$a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n} \tag{4.4}$$

holds or not for some values of base a such that $\gcd(a,n) = 1$.

If n does not satisfy the congruence 4.2.2 for base a, we prove that n is definitely composite. Therefore, we define the base a as *witness base* because it helps to prove the compositeness of n. In contrary, if n passes the congruence even though it is actually composite, we call a as *liar base* because it causes an incorrect result.

From the previous example which we examine whether compositeness of $n = 341$, base 2 is a Fermat's liar base for 341.

The results in the research [36] show that Fermat pseudoprimes are relatively rare than prime numbers, however, there are still infinitely many [34]. The following table [37] gives the number of 2-pseudoprimes less than 10^n where $n > 2$. We call such pseudoprimes to base 2 as *Poulet numbers*.

Table 4.2 : Number of 2-pseudoprimes.

limit	#Fermat-psp(2)
10^3	3
10^4	22
10^5	78
10^6	245
10^7	750
10^8	2057
10^9	5597
10^{10}	14884
10^{11}	38975
10^{12}	101629
10^{13}	264239
10^{14}	687007
10^{15}	1801533
10^{16}	4744920
10^{17}	12604009
10^{18}	33763684
10^{19}	91210364
2^{64}	118968378

The following table contains more examples of different values of base $a \leq 10$:

Table 4.3 : Examples of a-pseudoprimes.

a	n
2	341, **561**, 645, 1105, 1387, ...
3	91, 121, 286, 671, 703, ...
4	15, 85, 91, 341, 435, ...
5	4, 124, 217, **561**, 781, ...
6	35, 185, 217, 301, 481, ...
7	6, 25, 325, **561**, 703, ...
8	9, 21, 45, 63, 65, 117, ...
9	4, 8, 28, 52, 91, 121, ...
10	9, 33, 91, 99, 259, 451, ...

We see from the table 3.3 that some numbers are pseudoprimes to more than one bases. For example; we know that 9 is a composite number but holds the congruences for base 8 and 10 such that $8^{9-1} \equiv 1 \pmod 9$ and $10^{9-1} \equiv 1 \pmod 9$ respectively. This increases the probability of having an error if n passes the test. Thus, we need to repeat this test for different bases to find the compositeness of a number. If a number is a probable prime for a base a, we give a try to another base because of the existence of Fermat's liar bases. If a number n fails at any base a, we say that it is non-prime.

We provide the pseudocode of Fermat's algorithm that uses Fermat's Little Theorem as primality testing:

Algorithm 7 Fermat's Pseudoprime Test

Require: $a \geq 1, n > 2, \gcd(a,n) = 1$

```
t ← repeat time.
procedure TEST(n)
    for i = 1 → t do
        a ← {2, n − 1}
        if gcd(a, n) > 1 then
            return composite
        else if a^{n−1} ≢ 1 (mod n) then
            return composite
        else
            continue
    return probable prime
```

As we have seen from the algorithm that the test chooses a random integer a and applies the condition containing a quadratic equation. If the test stops any of the steps before completing t-times, it correctly declares that n is composite and has non-trivial factors (not including factorization step). We hope that if we try for all bases a such that $1 < a < n-1$, we will prove also the primality of a number. However, there are some special numbers which are not convenient for this test even we test with Fermat's Little Theorem. We are unable to prove and predict their compositeness even after t-trials of Fermat's test. Now, we give the definition of these troublesome composites which are pseudoprimes for many bases and proposed by Carmichael.

Definition 4.2.3 (Carmichael Numbers)**.** The composite integer n is called Carmichael number if it passes the Fermat's primality test for every base a such that $n \nmid a$.

Carmichael numbers are very rare rather than a-pseudoprimes and 561 is the smallest for base 2. Now, we illustrate the use of this definition with an example.

Examples.

- $2^{560} \equiv 1 \pmod{561}$
- $5^{560} \equiv 1 \pmod{561}$
- $7^{560} \equiv 1 \pmod{561}$

As this example shows, 561 always passes the Fermat's test to base a such that $\gcd(a,n) > 1$ despite its compositeness. Thus, Carmichael numbers cause trouble in determining compositeness of an integer with Fermat's Test. Because of existence such numbers, if the test does not stop at any condition for t-trials, the test outputs that n is prime or Carmichael number with probability greater than $1 - \frac{1}{2^t}$. This lead to following generalization theorem proposed by Euler.

Euler's extension

We see from the previous section that Fermat's theorem could only be applied when a modulus is a prime number. Otherwise, there would be false generalization since $2^{10} \equiv 4 \pmod{10}$, not equal to 1. Euler generalized the Fermat's Little Theorem for all numbers n. Before explaining the Euler's theorem, we first give a definition of Euler's Totient function that provides a base for this theorem, whereupon illustrate the definition with examples.

Definition 4.2.4 (Euler's Totient Function)**.** For an integer n, the function $\phi(n)$ is expressed as the count of integers between 1 and $n-1$ which are relatively prime to n.

Examples.

- Choose $n = 18$. To find $\phi(n)$, search for all integers not exceeding n. The set $S = \{1,5,7,11,13,17\}$ contains all numbers which are co-prime to 18. Thus, $\phi(18) = 6$.
- Choose $n = 11$. The set $S = \{1,2,3,4,5,6,7,8,9,10\}$ contains all numbers which are co-prime to 11. Thus, $\phi(11) = 10$.

The following table gives some preliminary values of $\phi(n)$. We assume $\phi(1) = 1$ by convention.

Table 4.4 : Euler Phi Values.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\phi(n)$	1	1	2	2	4	2	6	4	6	4	10	4	12	6	8	8	16	6	18	8

We easily see from two examples above that 1 is co-prime to each number. Moreover; we can find that the count $\phi(n)$ is equal to $n-1$ if n is prime. Now, we give some calculations for composite numbers by the subsequent theorem [28].

Theorem 4.2.2. *Let p be a prime and e be positive number which denotes the power of p. Then*

$$\phi(p^e) = p^e - p^{e-1} = p^e \cdot \left(1 - \frac{1}{p}\right) \tag{4.5}$$

We illustrate this theorem by the following examples.

Examples.

- $\phi(7^2) = 7^2 - 7 = 42$
- $\phi(13^3) = 13^3 - 13^2 = 2018$

Before giving the theorem for calculation of $\phi(n)$ for composite integers, we first introduce the multiplication property of the ϕ function, proposed by Pettofrezzo and Byrkit in 1970 [5].

Theorem 4.2.3. *Let a and b be positive integers and co-prime. Then, $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$*

Proof. See [6] for more detailed proof. □

Now, by using the unique factorization theorem 4.1.1 and multiplication property given in theorem 4.2.3, we find $\phi(n)$ for all composite integers n.

Theorem 4.2.4. *Let n be a composite and $e > 0$ be a number which denotes the power of n. Then, n can be written as the form of*

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k} \tag{4.6}$$

and Euler's totient function can be found by

$$\phi(n) = n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_k}\right) \tag{4.7}$$

Proof. We first apply the multiplication property stated in Theorem 4.2.3 such that

$$\phi(n) = \phi(p_1^{e_1}) \cdot \phi(p_2^{e_2}) \cdots \phi(p_k^{e_k})$$

where $n = \prod_{i=1}^{k} p_i^{e_i}$. Then, we know from the Theorem 4.2.2 that

$$\phi(p_i^{e_i}) = p_i^{e_i} - p_i^{e_i - 1} = p_i^{e_i} \cdot \left(1 - \frac{1}{p_i}\right)$$

If we multiply each $\phi(p_i^{e_i})$ where $1 \leq i \leq k$, we found that

$$\phi(n) = p_1^{e_1} \cdot \left(1 - \frac{1}{p_1}\right) \cdot p_2^{e_2} \cdot \left(1 - \frac{1}{p_2}\right) \cdots\cdots p_k^{e_k} \cdot \left(1 - \frac{1}{p_k}\right)$$

$$= n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdots\cdots \left(1 - \frac{1}{p_k}\right)$$

□

The following example illustrates the Theorem 4.2.4.
$\phi(120) = \phi(2^3 \cdot 3 \cdot 5) = \phi(2^3) \cdot \phi(3) \cdot \phi(5) = 120 \cdot \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{3}\right) \cdot \left(1 - \frac{1}{5}\right) = 32$

This theorem is a generalization of Fermat's Little Theorem 4.2.1 with Euler's phi function.

Theorem 4.2.5 (Euler's Theorem)**.** *For a positive integer n and a such that $n \nmid a$, then*

$$a^{\phi(n)} - 1 \equiv 0 \pmod{n} \tag{4.8}$$

Proof. See [6] for detailed proof. □

We illustrate the use of this theorem with an example.

Example. Let us pick a number $p = 100$ and an integer $a = 7$ such that $p \nmid a$.
By using Theorem 4.2.5, we calculate that $7^{\phi(100)} \equiv 7^{40} - 1 \equiv 0 \pmod{100}$ where $\phi(100) = \phi(2^2) \cdot \phi(5^2) = 2 \cdot 20 = 40$

Accuracy. In the last part of the previous section, we see that Fermat's test is not successful for detecting some composites especially Carmichael numbers which pass the test for all bases less than $n - 1$. However, thinking the frequency of Carmichael numbers among pseudoprimes are very rare, the probability of a Fermat prime is a Carmichael can be negligible. Thus, for a given a non-Carmichael number, the probability of correctly detecting composite number as composite is at least $\frac{1}{2}$. Then, if we try Fermat's Little Theorem for t times with different base values a, the probability of labeling a given composite incorrectly is at most $\frac{1}{2^t}$. Hence, having large enough repeating count increases the accuracy of the test.

4.2.2 Euler's test

In order to introduce Euler's test, we first give definitions (without their proofs) which provides a base for Euler's test.

Definition 4.2.5 (Quadratic Residue)**.** For a prime p and a positive integer a such that $(a,n) = 1$, we say a is a quadratic residue if the congruence

$$x^2 \equiv a \pmod{p} \tag{4.9}$$

has a solution. If not, we denote a as quadratic non-residue.

Example. To find quadratic residues of $p = 7$, we try all numbers less than 7.

$$1^2 \equiv 1 \pmod{7} \qquad 4^2 \equiv 2 \pmod{7}$$

$$2^2 \equiv 4 \pmod{7} \qquad 5^2 \equiv 4 \pmod{7}$$

$$3^2 \equiv 2 \pmod{7} \qquad 6^2 \equiv 1 \pmod{7}$$

Hence; $1,2,4$ are the entire set of quadratic residues of 7 whereas quadratic nonresidues of 7 are $3,5,6$. A list for quadratic residues where $p \leq 20$ can be found at the sequence number A046071 in OEIS database [37].

Now, we give the definition for Legendre symbol which is based on the law of quadratic reciprocity.

Definition 4.2.6 (Legendre Symbol)**.** For any prime p and $b > 0$, we define Legendre symbol as

$$\left(\frac{a}{p}\right) = \begin{cases} 1, & \text{if } a \text{ is quadratic residue mod } p. \\ 0, & \text{if } a \text{ and } p \text{ are not co-prime.} \\ -1, & \text{if } a \text{ is quadratic nonresidue mod } p. \end{cases}$$

Example.

Let use the previous example above.

$$\left(\frac{1}{7}\right) = \left(\frac{2}{7}\right) = \left(\frac{4}{7}\right) = 1$$

$$\left(\frac{3}{7}\right) = \left(\frac{5}{7}\right) = \left(\frac{6}{7}\right) = -1$$

Now, we give some properties of the Legendre symbol to be used in calculations:

Theorem 4.2.6. *Let p,q be distinct primes and a,b be any positive integers which are co-prime to p. Then the followings hold:*

- *If* $a \equiv b \pmod{p}$, *then* $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$
- $\left(\frac{a \cdot b}{p}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right)$
- $\left(\frac{-1}{p}\right) = \left(-1\right)^{\frac{p-1}{2}}$

  Equivalently, we can write that

$$\left(\frac{-1}{p}\right) = \begin{cases} 1, & p \equiv 1 \pmod{4} \\ -1, & p \equiv 3 \pmod{4} \end{cases}$$

- $\left(\frac{2}{p}\right) = \left(-1\right)^{\frac{p^2-1}{8}}$

  Equivalently, we can write that

$$\left(\frac{2}{p}\right) = \begin{cases} 1, & p \equiv \pm 1 \pmod{8} \\ -1, & p \equiv \pm 3 \pmod{8} \end{cases}$$

- *(Law of quadratic reciprocity)*
  $\left(\frac{p}{q}\right) \cdot \left(\frac{q}{p}\right) = \left(-1\right)^{(p-1)/2 \times (q-1)/2}$

  Equivalently, we can write that

$$\left(\frac{p}{q}\right) = \begin{cases} \left(\frac{q}{p}\right), & p \equiv 1 \pmod{4} \text{ or } q \equiv 1 \pmod{4} \\ -\left(\frac{q}{p}\right), & p \equiv q \equiv 3 \pmod{4} \end{cases}$$

As we see from the definition 4.2.6 that the bottom element p is always a prime integer and the top can be any integer. Since Legendre symbol has a restriction and can be just applied on prime modulus, we give a slightly different but somehow equivalent definition which is also generalization for modulus to be chosen from any odd number n.

Definition 4.2.7 (Jacobi Symbol)**.** Let n be a positive odd integer which has the form of

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k} = \prod_{i=1}^{k} p_i^{e_i} \tag{4.10}$$

where each of p_i are primes and e_i are positive integers. Then, Jacobi symbol can be expressed as

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \cdot \left(\frac{a}{p_2}\right)^{e_2} \cdots \left(\frac{a}{p_k}\right)^{e_k} = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}. \tag{4.11}$$

Jacobi symbol is an extension on Legendre symbol; even the same if n is prime. Now, we give a small note on Jacobi symbol: If a and p are co-prime, $\left(\frac{a}{n}\right) = -1$ means that the number a is a quadratic non-residue $\bmod n$, i.e., there is no solution of the equation $x^2 \equiv a \pmod{n}$. Moreover, if there is a solution to that equation, then $\left(\frac{a}{n}\right) = 1$. However, the converse is *not* always true. If Jacobi symbol $\left(\frac{a}{n}\right) = 1$, we can not say that a is a quadratic residue in p. So, there may be some number whose Jacobi is 1 but still not square in *mod n*. For example;

$$\left(\frac{54}{77}\right) = \left(\frac{54}{7}\right) \cdot \left(\frac{54}{11}\right) = \left(\frac{-2}{7}\right) \cdot \left(\frac{-1}{11}\right) = \left(-1\right) \cdot \left(-1\right) = 1$$

However, the result of Jacobi is inconclusive since there is no solution to both equations $x^2 \equiv 54 \pmod{11}$ and $x^2 \equiv 54 \pmod{7}$. This means that the result of Jacobi symbol 1 is the multiplication of 2 Legendre symbol, i.e., $(-1) \cdot (-1) = 1$.

Theorem 4.2.7 (Euler's Criterion)**.** *Let p be an odd prime and a be a positive integer which is not divisible by p, then*

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p} \tag{4.12}$$

Proof. See [6] for more detailed proof. □

If the congruence in 4.2.7 does not hold, we say that n is not prime. However, if the congruence holds, we are not able to say that the number is prime regarding the *note* on Jacobi symbol. This makes the test a compositeness test rather than a primality test. So, Euler's criteria does not guarantee to prove the primality of that integer. All numbers which are satisfying the criteria are called *probable primes* until a counterexample found to prove its compositeness. The following definition generalizes the counterexamples.

Definition 4.2.8 (Euler pseudoprimes)**.** The composite numbers $n \geq 2$ which succeeds Euler Criterion 4.2.7 for an integer $a > 0$ are called Euler pseudoprimes to base a, or a-pseudoprime, $Epsp(a)$ for short.

Table 4.5 : Number of Euler 2-pseudoprimes.

limit	**#Euler-psp(2)**
10^3	1
10^4	12
10^5	36
10^6	114
10^7	375
10^8	1071
10^9	2939
10^{10}	7706
10^{11}	20417
10^{12}	53332
10^{13}	124882

As shown from the table above, the number of Euler pseudoprimes are relatively rare than pseudoprimes and Euler pseudoprimes within the same base and range. For example; there are just 124882 strong pseudoprimes base 2 less than 10^{13}, while the number of Fermat pseudoprimes was 264239 in the same conditions. So, we can say that we decrease the pseudoprimes by half with Euler's criteria.

Now, we provide some initial examples of the Euler pseudoprimes for several bases less than or equal to 10.

Table 4.6 : Examples of Euler a-pseudoprimes.

a	**n**
2	341, 561, 1105, 1729, 1905, 2047, ...
3	121, 703, 1541, 1729, 1891, 2465, ...
4	341, 561, 645, 1105, 1387, 1729, ...
5	217, 781, 1541, 1729, 5461, 5611, ...
6	185, 217, 301, 481, 1111, 1261, 1333, ...
7	25, 325, 703, 817, 1825, 2101, 2353, ...
8	9, 21, 65, 105, 133, 273, 341,481, ...
9	91, 121, 671, 703, 949, 1105, 1541, ...
10	9, 33, 91, 481, 657, 1233, 1729, ...

Algorithm. Now, we give basic algorithm which repeats the Euler' criteria to check primality of a number n.

Algorithm 8 Euler's Primality Test

Require: $a \geq 1, n > 2, \gcd(a,n) = 1$
1: $a \leftarrow \{2, n-1\}$
2: $t \leftarrow$ repeat time.
3: **procedure** TEST(n)
4: **for** $a = 1 \rightarrow t$ **do**
5: **if** $a^{(n-1)/2} \not\equiv \pm 1 \pmod{n}$ **then**
6: **return** composite
7: **else**
8: **continue**
9: **return** probable prime

Accuracy. We see from the Euler's test that it is actually compositeness test for a given integer n. If the criteria does not hold, the test guarantees that n is composite. However, the converse is not always true. In other words, the test could not prove the primality of that number. If the test holds, we need to repeat the test for a different choice of bases a, because there are liar bases for composite numbers that provide them to not to fail the test.

4.2.3 Solovay-Strassen primality test

In this section, we introduce a new type of probabilistic primality test developed by Robert M. Solovay and Volker Strassen [10] and used public key cryptography. This test is based on Euler's criterion explained in the section 4.2.7. We first give the main theorem which is the key idea behind the test.

Theorem 4.2.8 (Solovay-Strassen)**.** *Let n be a composite odd integer. There exists a number* $a < n$ *and* $\gcd(a,n) = 1$ *hold the condition such that*

$$a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n} \tag{4.13}$$

where $\left(\frac{a}{n}\right)$ *is the Jacobi Symbol.*

Algorithm. If given input n is not definitely composite (i.e., probable prime), it repeats itself with different bases a which is co-prime to n and $1 < a < p$. This is because of the existence of counterexamples for the theorem 4.2.7, i.e., Euler pseudoprimes.

Algorithm 9 Solovay-Strassen Algorithm

```
1: procedure TEST(n)
2:     a ← {2, n − 1}
3:     if gcd(a, n) > 1 then
4:         return composite
5:     j = a^((n−1)/2) (mod n)
6:     if j ≠ J(a, n) then
7:         return composite
8:     else if j = J(a, n) then
9:         return probable prime
```

As we see from the theorem above, the Solovay-Strassen test has two main calculations. Let n denotes the number that we want to check the primality. First, the test selects random integer a for a base and computes very large power of it modulus n. Secondly, it computes the Jacobi Symbol mentioned in 4.2.7 for these numbers. Checking whether these two calculations are equal or not is the key idea behind the Solovay-Strassen Test.

Example.

Let us try to check primality of an integer $n = 217$. Then, $(n-1)/2 = 108$.

As stated in the algorithm, we first randomly select an integer $1 < a < n$ and check if $\gcd(a,n) = 1$. Now, we compute the followings:

- $\gcd(6, 217) = 1$
- $a^{(n-1)/2} \pmod{n} = 6^{108} \pmod{217} = 1$
- $\left(\frac{6}{217}\right) = 1.$

Since we found that n is probable prime, we repeat the test with different base: $n = 11$.

- $\gcd(11, 217) = 1$
- $a^{(n-1)/2} \pmod{n} = 11^{108} \pmod{217} = 64$
- $\left(\frac{11}{217}\right) = -1$

As we have discussed from the example, 217 is *not* prime after 2 trials. While base 11 is a witness for the compositeness of n, base 6 is a liar because of an incorrect result.

Repeating the algorithm t times increases the accuracy of the test and decreases the probability of having an incorrect result.

Running Time and Accuracy. Finding GCD and Jacobi Symbol can be computed within the time complexity of $O((log\ n)^2)$. Then, computing the power of the base a run in complexity $O((log\ n)^3)$. Thus, the algorithm has $O(k \cdot log^3 n)$ running time for k-times pass the test where k is the repeat count of the test. Since this algorithm is a probabilistic algorithm, every result is *not* definitely true. If the given number n is prime, the test correctly tags it as probable prime. However, if the number is composite, the result will not be accurate. As we have seen in the previous chapter, some composite integers exist by holding the cases mistakenly for these algorithms. Hence, the probability of the test outputs incorrect result is at most 2^{-t} after t trials, which is the same as Euler's test.

Drawbacks. Firstly, since the results in Euler test are impractical for Carmichael numbers which are pseudoprimes to every bases and Solovay-Strassen test is basically based on repeating the Euler's criteria 4.2.7, the Solovay-Strassen test is *not* successful to catch Carmichael numbers. Secondly, this test is *not* a deterministic test, so that the results are probabilistic when checking the primality of a random number. This means that the accuracy of the test will depend on the number of repetitions. Thus, the algorithm requires choosing a sufficiently large repetition count of t to use this algorithm in practical applications.

4.2.4 Miller-Rabin strong pseudoprime test

In this section, we give details of the most useful and common probabilistic primality test algorithm which is also a generalization of Solovay-Strassen test. As the name of the algorithm implies, the test is discovered by Gary L. Miller [16] and modified by Michael O. Rabin [17] in 1980. This test also depends on some set of equalities proposed by Fermat and Euclid like the tests given in previous sections. The idea behind all tests considered until now was to have a failure in some cases for composite numbers even there exists counterexamples known as pseudoprimes. The Miller-Rabin test shares the same idea, however, for an integer n, it provides more efficient and accurate results than the tests mentioned in previous sections. Even though we know

the test is a probabilistic test, it has also a deterministic version runs in polynomial time by assuming the Generalized Riemann Hypothesis (GRH) is true.

Firstly, we give some definitions and theorems which will be used as key ingredients of the singular cubic primality test.

Lemma 4.2.1. *Let p be a prime and $a, b > 0$ be any integers such that*

$$a \cdot b \equiv 0 \pmod{p} \tag{4.14}$$

Then,

$$a \equiv 0 \pmod{p} \quad \text{or} \quad b \equiv 0 \pmod{p} \tag{4.15}$$

The lemma states that if prime p divides the multiplication $a \cdot b$, then it also must divide at least one of the multiplicands. Since we mentioned some set of definitions in 4.2.5, we now give the theorem of the square-root of the quadratic equation $x^2 \equiv a \pmod{p}$ related to the Lemma 4.2.1.

Definition 4.2.9. We call x is a *square-root* if it satisfies the quadratic equation $x^2 \equiv a \pmod{p}$ for a prime p and integer a such that $(a, n) = 1$.

Example.

$$1^2 \equiv 1 \pmod{7} \qquad 4^2 \equiv 2 \pmod{7}$$

$$2^2 \equiv 4 \pmod{7} \qquad 5^2 \equiv 4 \pmod{7}$$

$$3^2 \equiv 2 \pmod{7} \qquad 6^2 \equiv 1 \pmod{7}$$

Theorem 4.2.9 (Square-root)**.** *Let p be a prime and $x > 0$ be any integers. The congruence*

$$x^2 \equiv 1 \pmod{p} \tag{4.16}$$

holds if and only if

$$x \equiv \pm 1 \pmod{p} \tag{4.17}$$

Proof. See [28] for more detailed proof. □

If we write the congruence $x^2 - 1 \equiv 0 \pmod{p}$, we obtain that $(x-1)\cdot(x+1) \equiv 0 \pmod{p}$ by factorization of two squares. By using the Lemma 4.2.1, we see that if n is prime, x is congruent to at least either $+1$ or -1, i.e., $x \equiv \pm 1 \pmod{p}$ holds and we call x as a *trivial square-root of 1*. Besides that, if one can find square-roots other than ± 1, say *non-trivial square-roots of 1*, we found n as a composite. Checking whether there exists a square-root in mod n form a basis to the Strong Pseudoprime Test used in Miller-Rabin Algorithm.

Theorem 4.2.10 (Strong Pseudoprime Test)**.** *Let $n > 2$ is an odd number which can also be expressed by $n - 1 = 2^e \cdot m$ where m is an odd number and $e > 0$. If n is prime, either*

$$a^m \equiv 1 \pmod{n} \quad or \quad a^{m2^i} \equiv -1 \pmod{n} \tag{4.18}$$

holds for some $0 \leq i < e$, and then n passes the test for base a which is between 1 and n-1.

Proof. The Fermat's Little Theorem given in 4.2.1 states that the congruence $a^{n-1} \equiv 1 \pmod{n}$ holds if n is prime. Since we have the expression $n - 1 = 2^e \cdot m$, by taking the power of both sides, we get $a^{n-1} = a^{2^e \cdot m} = 1$. Now, we compute the following sequence by taking the square of each previous item:

$$a^m, a^{2m}, a^{2^2 m}, a^{2^3 m}, ..., a^{2^{e-1} m}, a^{2^e m} \mod n$$

We know that the last item in the sequence is $a^{2^e m} = 1$. Considering square-root theorem 4.2.9, the sequence will have two forms:

- The first element a^m is equal to 1. Since the next item a^{2m} is a square of the first item, this will be 1 too. Thus, we will get the whole sequence containing all 1's.
- The first element a^m is **not** equal to 1. This is also possible if there exist a k-th element ($1 < k \leq e$) which is $a^{2^{k-1} m} = -1$ and whose square makes its next element $a^{2^k m}$ equal to 1.

□

We also can prove the Theorem 4.2.10 follows:

By taking power of both sides in the equation $n-1=2^e\cdot m$ with base a result as $a^{n-1}=a^{2^e\cdot m}$. From the Fermat's Little Theorem given in 4.2.1, left hand side of the equation is $a^{n-1}-1\equiv 0\ (\bmod\ n)$ if n is prime. Thus, the right-hand side $a^{2^e\cdot m}$ will be equal to $0\ \bmod n$. Now, we do some factorizations as follows:

$$\begin{aligned}
a^{2^e m}-1 &= (a^{2^{e-1}m})^2-1\\
&= (a^{2^{e-1}m}-1)\cdot(a^{2^{e-1}m}+1)\\
&= (a^{2^{e-2}m}-1)\cdot(a^{2^{e-2}m}+1)\cdot(a^{2^{e-1}m}+1)\\
&\ \ \vdots\\
&= (a^m-1)\cdot(a^m+1)\cdot(a^{2m}+1)\cdot(a^{2^2m}+1)\cdots(a^{2^{e-1}m}+1)\equiv 0\ (\bmod\ n)
\end{aligned}$$

One or some of these elements must be equal to 0 if n is prime. This means that either the first element is $a^m-1\equiv 0\ (\bmod\ n)$ or every subsequent is $(a^{2^i m}+1)\equiv 0\ (\bmod\ n)$ where $0\leq i<e$.

The strong pseudoprime test returns two results:

- the number fails the test and found as definitely composite, or
- the number passes the test and found as probable prime.

There are some composite numbers that pass the test.

Definition 4.2.10 (Strong Pseudoprime)**.** For any given composite integer n, if n passes the test for base a, then we define n as strong pseudoprime to base a, $spsp(a)$ for short and a as *liar base*.

Also, we can say that each strong pseudoprime to base a is also Euler pseudoprime to a even the converse is not *always* true.

Example.
Let us evaluate $n=2047$. We write $n-1=2\cdot 1023$. Then, the congruences $2^{1023}\equiv 1\ (\bmod\ 2047)$ and $2^{2046}\equiv 1\ (\bmod\ 2047)$ hold for base $a=2$. However, 2047 is not prime because it can be factorized as $23\cdot 89$.

Theorem 4.2.11 (Generalized Riemann Hypothesis (GRH))**.** *For an odd composite integer $n>0$, let $n-1=2^e\cdot m$ for some power $e>0$ and odd integer m. For all a*

which is $1 < a < 2(log\ n)^2$*, either*

$$a^m \equiv 1 \pmod{n} \quad or \quad 2^i \cdot m \equiv -1 \pmod{n} \text{ for some } 0 \leq i < e \tag{4.19}$$

holds, then n is prime.

The following algorithm is the deterministic version of Miller-Rabin Test. As seen from the algorithm, it outputs definitely prime or composite if Generalized Riemann Hypothesis is true.

Algorithm 10 Miller Rabin Deterministic Algorithm

Require: $a > 0, n > 2$

```
1: procedure TEST(n)
2:     n − 1 = 2^e m
3:     for each a = 2 → min(n − 1, 2(logn)^2) do
4:         for each i = 0 → m − 1 do
5:             if (a^m ≢ 1 (mod n)) & (a^{2^i m} ≢ −1 (mod n)) then
6:                 return composite
7:     return prime
```

Since GRH has not been proven yet, we give the probabilistic version of the Miller-Rabin Algorithm:

Algorithm 11 Miller Rabin Probabilistic Test

Require: $a > 0, n > 2$

```
1: n − 1 = 2^e m
2: t ← repeat time.
3: procedure TEST(n)
4:     for i = 1 → t do
5:         a ← {2, n − 1}
6:         x ← a^m (mod n)
7:         if x ≡ ±1 (mod n) then
8:             continue
9:         for i = 1 → e − 1 do
10:            x ← x^2 (mod n)
11:            if x = 1 then
12:                return composite
13:            else if x = −1 then
14:                continue
15:        return composite
16: return probable prime
```

Theorem 4.2.12. *Let $n > 9$ be an odd composite integer such that $n-1 = m2^e$ for some integer $e > 0$ and odd integer m. Let*

$$S = \{a \in (\mathbb{Z}/n\mathbb{Z})^{\times} \quad | \quad a^m \equiv 1 \mod n \text{ or } a^{m2^i} \equiv -1 \mod n \text{ for some } 0 \leq i < e\}.$$

Then

$$\frac{\#S}{\phi(n)} \leq \frac{1}{4}$$

where $\phi()$ is the Euler's phi function.

The theorem below states that the algorithm has error-probability bound less than $\left(\frac{1}{4}\right)^t$ for t-trials.

Table 4.7 : Number of Spsp(2).

limit	**#spsp(2)**
10^3	0
10^4	5
10^5	16
10^6	46
10^7	162
10^8	488
10^9	1282
10^{10}	3291
10^{11}	8607
10^{12}	22407
10^{13}	58892
10^{14}	156251
10^{15}	419489
10^{16}	1135860
10^{17}	3115246
10^{18}	8646507
10^{19}	24220195

As shown from the table above, the number of strong pseudoprimes are relatively very rare than pseudoprimes and Euler pseudoprimes within the same base and range. For example; there are just 3291 strong pseudoprimes base 2 less than 10^{10}, while the count for Fermat pseudoprimes were 14884 and for Euler pseudoprimes were 7706 in the same conditions.

The following table contains the initial examples of strong pseudoprimes for several bases less than 10.

Table 4.8 : Examples of Spsp(a).

a	n
2	2047, 3277, 4033, 4681, 8321, ...
3	121, 703, 1891, 3281, 8401, 8911, ...
4	341, 1387, 2047, 3277, 4033, 4371, ...
5	781, 1541, 5461, 5611, 7813, ...
6	217, 481, 1111, 1261, 2701, ...
7	25, 325, 703, 2101, 2353, 4525, ...
8	9, 65, 481, 511, 1417, 2047, ...
9	91, 121, 671, 703, 1541, 1729, ...

Running Time and Accuracy. The running time of this algorithm is $O(t \cdot log^3(n))$ where t is the repeating count. In terms of accuracy, the Miller-Rabin test has lower error probability for a single round when compared with other probabilistic tests. For any random composite n, we have 50% chance to determine correctly by using other probabilistic tests. However, the probability of success is at least 75% in Miller-Rabin test for all bases which are relatively prime to n. Moreover, this probability will decrease if the repeating count of k is chosen large enough. For example; if we choose $t = 20$, the probability of a composite number passes the test is less than $\frac{1}{4^{20}} = \frac{1}{2^{40}}$ Moreover, there are too many liar bases for Carmichael numbers when one uses other tests, but the Miller-Rabin test gets rid of this snug.

Drawbacks. Although the test is very quick and useful for very large integers which contain several hundred digits, this test is also a probabilistic test like other tests such as Fermat, Euler, and Solovay-Strassen tests. There is always error probability in the probabilistic version. In case of deterministic version, if one assuming GRH is true, the test will have a reliable limit for repeating count t. If a number does not fail until that limit, we could prove that number is prime. Thus, the deterministic version of the Miller-Rabin algorithm is conditional to Generalized Riemann Hypothesis which is needed to be proven.

4.3 Deterministic Primality Tests

4.3.1 AKS primality test

As we have seen probabilistic primality test until this section, some primality tests such as Fermat and Miller-Rabin correctly label prime numbers despite that some

conclusive results may occur while detecting composites. Even there were some classifying errors for composites that pass the test, probabilistic tests could guarantee the compositeness of a given integer n if they fail some sort of number theoretical theorems and conditions. There were two outputs in the probabilistic test; the result was either composite or probable prime since the tests do not guarantee the result of primality. Moreover, there are always some counterexamples for some numbers which pass the test as prime as if it were not really a prime, for instance, Carmichael numbers. Thus, finding a deterministic primality test algorithm which decides the primality or compositeness of a given number in polynomial time without dependency on some unproven assumptions was an unsolved problem for many years.

Then, in 2002, Manindra Agrawal, Neeraj Kayal, and Nitin Saxena invented the first deterministic polynomial time primality test in their paper [7]. This algorithm was the first example of deterministic tests which prove primality or compositeness of a given number in polynomial time. In other words, the algorithm guarantees both primality and compositeness by not depending on the form of a given number, i.e., it also runs on any form of number such as Mersenne numbers, Carmichael numbers or Fermat pseudoprimes, etc. Since the deterministic version of Miller-Rabin Pseudoprime test depends on unproved Generalized Riemann Hypothesis, AKS test is unconditional so that it does not depend on any of unproven hypothesis.

Before giving the AKS primality test algorithm, we give underlying definitions which are the used in the algorithm.

Definition 4.3.1 (Multiplicative order)**.** Let $ord_n(a)$ denote the multiplicative order of a given number a modulus n such that $\gcd(a,n) = 1$. Then, the order of $a \pmod{n}$ is defined as the smallest integer $k > 0$ that satisfies the equation $a^k \equiv 1 \pmod{n}$.

Example.

$$4^3 \equiv 1 \pmod{7} \Rightarrow ord_7(4) = 3$$

$$18^{16} \equiv 1 \pmod{97} \Rightarrow ord_{97}(18) = 16$$

$$653^{6161} \equiv 1 \pmod{12323} \Rightarrow ord_{12323}(653) = 6161$$

Theorem 4.3.1. *Let p be a prime. If $0 < i < p$, then $\binom{p}{i} \equiv 0 \pmod{p}$.*

Proof. We can expand the binomial as follows:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{4.20}$$

As seen from the formula that the numerator of the division will always be greater than the denominator since $0 < i < p$. Since the denominator cannot eliminate p, we get 0 in modulus p. □

Now, we give the key theorem which leads to the AKS primality test.

Theorem 4.3.2. *For any integer $n \geq 2$ and a such that a and p are co-prime, n is prime if the congruence*

$$(x+a)^n \equiv x^n + a \pmod{n} \tag{4.21}$$

holds (by assuming x is just a formal symbol).

Proof. See Section 2 in the paper [7] for complete proof. □

Example.

The following example illustrates and verifies the theorem:

1. Let us choose $n = 5$ and $a = 3$.

$$\begin{aligned}(x+3)^5 &= \binom{5}{0}x^5 3^0 + \binom{5}{1}x^4 3^1 + \binom{5}{2}x^3 3^2 + \binom{5}{3}x^2 3^3 + \binom{5}{4}x^1 3^4 + \binom{5}{5}x^0 3^5 \\ &= x^5 + 15x^4 + 90x^3 + 270x^2 + 405x + 243 = x^5 + 3 \pmod{5}\end{aligned}$$

   Hence; we see that $n = 5$ is prime and so the congruence $(x+3)^5 = x^5 + 3 \pmod{5}$ holds.

2. Let us choose $n = 6$ and $a = 5$.

$$\begin{aligned}(x+5)^6 &= \binom{6}{0}x^6 5^0 + \binom{6}{1}x^5 5^1 + \binom{6}{2}x^4 5^2 + \binom{6}{3}x^3 5^3 + \binom{6}{4}x^2 5^4 + \binom{6}{5}x^1 5^5 + \binom{6}{6}x^0 5^6 \\ &= x^6 + 30x^5 + 375x^4 + 2500x^3 + 9375x^2 + 18750x + 15625 \\ &= x^6 + 3x^4 + 4x^3 + 3x^2 + 1 \neq x^6 + 3 \pmod{6}\end{aligned}$$

   Hence; we see that $n = 6$ is not prime and so the congruence $(x+5)^6 \neq x^6 + 3 \pmod{5}$ does not hold.

Moreover, Euler's totient function which is mentioned in the section 4.2.4 will be used in the algorithm. Now, we give the pseudocode of the AKS algorithm.

Algorithm 12 AKS Algorithm

Require: $a \geq 1, n > 2, \gcd(a,n) = 1$

procedure TEST(n)
  if n is the form of a^b **then**
    return composite
  Find the smallest r s.t. $ord_r(n) > (log_2 n)^2$ and $(r,n) = 1$
  if $\exists a \geq r$ s. t. $1 < \gcd(a,r) < 1$ **then**
    return composite
  if $r \geq n$ **then**
    return prime
  for $a = 1 \rightarrow \lfloor \sqrt{\varphi(r)} \cdot log_2(n) \rfloor$ **do**
    if $(X+a)^n \not\equiv x^n + a \pmod{X^r - 1, n}$ **then**
      return composite
  return prime

We have mentioned from the beginning that AKS primality test is unconditional, deterministic and polynomial time primality test algorithm. Also, we know from the previous algorithm that if the given number n fails at one of the steps, then it is definitely composite. The converse is normally *not* true, however, AKS test holds in both ways. This means that, if the AKS algorithm given above results as prime, then the number will be labeled as definitely prime. We do not give the full proof of this theorem, but the proof of this claim can be found in [7]. Now, let us illustrate the algorithm by an example.

Example.

We first check by AKS algorithm whether $n = 97$ is prime or not.

(i) To check if n is the form of a^b, we take the rational power of 97 such that 97^x where are prime numbers less than $log(97) \approx 6$.

$$97^{(\frac{1}{2})} = 9.85 \quad 97^{(\frac{1}{3})} = 4.6 \quad 97^{(\frac{1}{5})} = 2.5$$

Since none of the results are a natural number, we say that 97 is not perfect square. (Passed)

(ii) We found $r = 59$ which is the smallest number satisfying $ord_r(n) > log_2^2(97) = 43.56$. (Passed)

(iii) For integers $1 \leq a \leq 59$, we found that all of them have $\gcd(a, 97) = 1$. (Passed)

(iv) $n = 97 > r = 29$. (Passed)

(v) $\sqrt{\phi(59)} \cdot log_2(97) = 44$. Since the condition $(x+a)^n \equiv x^n + a \pmod{x^r - 1, n}$ satisfies for all a such that $1 \leq a \leq 44$.

Running Time and Accuracy. In terms of accuracy, we say that the results are 100% accurate since the test is deterministic.

Drawbacks. Besides it is algorithmic complexity was in P, it is not a practical algorithm in terms of running time. From the theoretical computer science point of view, the AKS algorithm is best in algorithmic complexity. Due to its inefficiency, "almost" polynomial time algorithms such as Miller-Rabin algorithm are better in choice of real-time applications.

4.4 Elliptic Curve Primality Proving

In this section, we give a detailed explanation of one of the modern and widely used primality test which use elliptic curves over a finite field $\mathbb{F}_p$. Elliptic curve primality test is a deterministic test which certificates the primality and it seems to be polynomial time under some plausible assumptions. Besides its algorithmic complexity, the running time of algorithms is hard to estimate. Because we first assume the given number n is prime, i.e., it has passed a probabilistic primality test such as Miller-Rabin test. By this assumption, we also assume that all elements in modulo n is invertible so that group operation can be done successfully as if n is prime. However, if any error occurs in group operation when performing point addition or doubling in $\mathbb{Z}/n\mathbb{Z}$, i.e., a non-invertible element appears, the algorithm then outputs a non-trivial factor of n by taking greatest common divisor (GCD) of that non-invertible element and n. In this manner, the algorithm successfully finds the compositeness of that given number. Since the algorithms immediately stop when an unlikely error occurs, it makes running time estimation quite hard.

To give more details on elliptic curve primality testing theorems and algorithms, we denote $E(\mathbb{Z}/n\mathbb{Z})$ as elliptic curve E over $\mathbb{Z}/n\mathbb{Z}$ in the form of Weierstrass equation

$$y^2 = x^3 + ax + b \tag{4.22}$$

where $a, b \in \mathbb{Z}/n\mathbb{Z}$ and the determinant $\triangle = 4a^3 + 27b^2 \in (\mathbb{Z}/n\mathbb{Z})^*$.

The main idea of all elliptic curve primality testing algorithms is the following theorem:

Theorem 4.4.1 (Pocklington's Theorem)**.** *Let n be an integer and s be divisor of* $n-1$. *Suppose that there is an element a in* $\mathbb{Z}/n\mathbb{Z}$ *such that*

$$\left.\begin{array}{l} a^{n-1} \equiv 1 \pmod{n} \\ gcd(a^{n-1/q}, n) = 1 \end{array}\right\} \tag{4.23}$$

for each prime divisor q of s. Then, any prime p dividing n satisfies $p \equiv 1 \pmod{s}$. *In particular, if* $s > \sqrt{n}$, *then n is prime.*

The following elliptic curve primality testing theorem is based on the analog of Theorem 4.4.1.

Theorem 4.4.2 (Elliptic Curve Primality Testing)**.** *Let* $n > 2$ *be an odd positive integer such that* $gcd(n,6) = 1$ *and E be an elliptic curve over* $\mathbb{Z}/n\mathbb{Z}$ *given in the form of the equation 4.22. Let m and s be positive integer such that s is a divisor of m. Suppose that there is a point P on the curve E satisfying the conditions:*

- $[m]P$ *is identity;*
- $[m/q]P$ *is defined and not identity for any prime divisor q of s.*

Then, every prime p dividing n satisfies $\#E(\mathbb{Z}/p\mathbb{Z}) \equiv 0 \pmod{s}$. *In particular, if* $s > (\sqrt[4]{n}+1)^2$, *then n is prime.*

Proof. See the Theorem 5.2 in [23]. □

Next, we give the general methodology of elliptic curve primality testing algorithms:

Algorithm 13 General ECPP Method

Require: An odd integer n
Ensure: Primality certificate or compositeness with a factor of n

1: **procedure** $ECPP(n)$
2: Generate a non-singular elliptic curve in $E(\mathbb{Z}/n\mathbb{Z})$
3: $m \leftarrow$ the order of the curve E
4: Choose a prime factor q of m such that $q > (\sqrt[4]{n}+1)^2$
5: Pick a random point P on E
6: **if** $[m]P \neq 0$ **then goto** Step 5.
7: **if** $[m/q]P = 0$ **then goto** Step 5.
8: **if** an error occurs **then return** a factor of n.
9: **return** prime

As seen from the Algorithm 13, the first step requires to choose an elliptic curve E over $\mathbb{Z}/p\mathbb{Z}$ and the order of that curve as m. If we assume every calculation upon the assumption of primality of n, the order can be written of the form $m = kq$ such that q is probable prime by a probabilistic primality test and $q > (\sqrt[4]{n}+1)^2$. These two steps differ in Shafi-Killian algorithm [9] and Atkin-Morain test [8] which we will give more details in proceeding sections. The following steps are all common in both algorithms. The idea is to find a random point P on the elliptic curve which holds the conditions given in the Theorem 4.4.2 and to perform group operation in $\mathbb{Z}/p\mathbb{Z}$. Finally, it continues recursively by setting the number $n' = q$ and applies the same process again. Hence, the random point P proves the primality of n if q is prime. In other words, finding primality of n will be reduced to the proving that a smaller number is prime.

This key idea leads first to Shafi-Killian primality testing algorithm. Due to some drawbacks of Shafi-Killian algorithm, Atkin and Morain developed a more elegant solution in 1993. We will give details, examples, and analyses of both algorithms respectively in the next two sections.

4.4.1 Shafi-Killian algorithm

The main methodology of elliptic curve primality test [Theorem 4.4.2] prepared a base with some uncertainties on how to choose the curve E, find the order in the form of $m = kq$ and pick a point P which holds conditions. Now, we will give details of Shafi-Killian [9] algorithm in five steps.

Algorithm.

Step 1: Choose a random curve over $\mathbb{Z}_n$

In this step, the algorithm picks a and b values at random and constructs the curve E in the form given in 4.22

Step 2: Curve Order

This step is one of the drawbacks of the algorithm that consumes time in terms of running time complexity. The order of the curve E, i.e., the number of points on the curve can be calculated by using Schoof's Method [11]. It is possible to determine the compositeness of the given n if it gets an error and stops immediately at this step. In other words, if this algorithm does not succeed to find the cardinality of the curve in a timely manner, this will allow determining a non-trivial factor of n.

Step 3: Order Factorization

Once the order is found by using Schoof's method (see Algorithm 7.5.6 in [12]), the algorithm proceeds to determine if that order m is of the form of $m = kq$ where q is prime and $q > (\sqrt[4]{n}+1)^2$. If we cannot factor the order by using known factorization algorithms, we discard that curve E and step back to choosing random values of a, b and generation another random curve.

Step 4: Random Point on $E_{a,b}(\mathbb{Z}_n)$

Once the curve is successfully generated, we pick a random point on the constructed curve and perform group operations. Here is how we pick a random point on a given curve $E_{a,b}(\mathbb{Z}_n)$:

Algorithm 14 Picking a random point P on a given elliptic curve E

Require: Elliptic curve $E : y^2 = x^3 + ax + b$ and integer n
Ensure: (x, y) on E

procedure $RandomPoint(a, b, n)$
  $found \leftarrow false$
  do
    do
      $x \in \{0, n-1\}$ ▷ Random integer
      $y = (x(x^2 + a) + b) \mod n$
    while $Jacobi(y/n) = 1$
    $t \leftarrow Sqrt(y, n)$ ▷ See Algorithm 15
    if t is found **then**
      $P = (x, y)$
      $found \leftarrow true$
  while $!found$
  if $(P_y)^2 \mod n \neq y$ **then** ▷ n is composite
    $P = \infty$

The Algorithm 14 requires to compute square roots mod p. Thus, we give the following algorithm developed by Tonelli in 1891, which outputs a solution to the equation

$$x^2 \equiv a \pmod{p} \tag{4.24}$$

for a prime p and given integer a such that $Jacobi\left(\frac{a}{p}\right) = 1$.

Algorithm 15 Tonelli's Square Root mod p Algorithm

Require: Integer a and prime number p
Ensure: Square root x

procedure $Sqrt(a, p)$
  $a \leftarrow a \mod p$
  if $p \equiv 3, 7 \pmod{8}$ **then** ▷ Case for 3,7
    return $a^{(p+1)/4} \mod p$
  else if $p \equiv 5 \pmod{8}$ **then** ▷ Case for 5
    $x \leftarrow a^{(p+3)/8} \mod p$
    $c \leftarrow x^2 \mod p$
    if $c \neq a \mod p$ **then**
      $x = x2^{(p-1)/4} \mod p$
    return x

Algorithm 15 Tonelli's Square Root mod p Algorithm - Continued

else
  while $\left(\frac{d}{p}\right) \neq -1$ **do** ▷ Case for 1
    $d \leftarrow d \in [2, p-1]$
  $p-1 = 2^s t$
  $A \leftarrow a^t \mod p$
  $D \leftarrow d^t \mod p$
  $m \leftarrow 0$
  for $i = 0 \rightarrow s$ **do**
    if $(AD^m)^{2^{s-1-i}} \equiv -1 \pmod p$ **then**
      $m \leftarrow m + 2^i$
  $x \leftarrow a^{(t+1)/2} D^{m/2} \mod p$
  return x

Step 5: Point Operation

The details of group operation methods and its analyses are given in Chapter 3 and subsection 3.2.4

Running Time and Accuracy. The results show that the algorithm is expected to be in polynomial time.

Drawbacks. The construction of the curve is costly because the test requires Schoof's method [11] (which is very slow and cumbersome) for counting the number of points on the elliptic curve. Secondly, the test finds a curve whose number of points m is of the form $m = kq$ where q is probable prime. This construction increases the computational complexity of the algorithm.

4.4.2 Atkin-Morain ECPP

As we discussed in the previous subsection, Shafi-Killian Elliptic Curve Primality Proving (ECPP) algorithm selects and constructs an elliptic curve randomly. Additionally, the point counting step increases the time and computational complexity of the algorithm. Due to the lack of this pre-step in this algorithm, Atkin and Morain developed a new algorithm in the same year (1986) which finds curves using complex multiplication (CM) method. The key point of the new algorithm was to choose a suitable discriminant D to make the computation of the order of the curve m, i.e., the

number of points on E, easier. After this step, the curve will be generated based on this order. Now, we first give details of the algorithm in five steps for a given off number n.

Algorithm.

Step 0: Miller Rabin and Sieve test

The initial step of this algorithm requires to eliminate by checking whether the given number is composite by Miller-Rabin probabilistic test Algorithm 11 with a small round value (for example, only 10 or 20 rounds). If not, Sieve of Eratosthenes method can be used to check the given number is less than a bound.

Step 1: Choose discriminant

This step is the keystone of Atkin-Morain algorithm comparing to Shafi-Killian algorithm. It will be very easy to find a discriminant because the $Jacobi(D,n)$ should be equal to 1. If not, the algorithm tries to find another discriminant from the list of increasing values of the class number $h(D)$. The following algorithm gives the details:

Algorithm 16 Choosing Discriminant

Require: Number n
Require: Initialized discriminant array for class number 1 and 2
Ensure: The order $m = kq$ and the discriminant d

```
1:  procedure ChooseDiscriminant(n)
2:      index ← 0
3:      size ← array size
4:      while index + 1 < size do
5:          d ← array[i]
6:          (x, y) ← ModifiedCornacchia(d, n)          ▷ See Algorithm 17
7:          if Jacobi(d, n)! = 1 then continue
8:          else if (x, y)! = null then continue
9:          else if OrderFactorization(x, y, d, n) is not found then continue
10:         elsebreak
11:     return d
```

Note: The discriminant list for class number 1 and 2 is taken from the results of the work [Cox 1989]. (See Algorithm 7.5.9 in [12] for complex multiplication method). We used a closed-form solution by using the explicit parameter sets for all D for class number 1 and 2 computed in Table 7.1 in [12].

We pick a discriminant value in order and check if Jacobi test results in 1 or not. If the discriminant value does not pass the Jacobi test, the next discriminant will be chosen

from the initial array. In the case of $Jacobi(D,n) = 1$, the algorithm proceeds to find a solution (x,y) to the Diophantine equation:

$$x^2 + |D|y^2 = 4n \tag{4.25}$$

If there exists no such solution to the above equation, then it returns *null* and skips to the next D value in the array.

Finding a solution to the equation 4.25 can be easily computed by using *Modified Cornacchia Algorithm* for a given prime p and discriminant D such that $|D| < 4p$ and $d \equiv 0,1 \pmod 4$. The details of algorithm are given below:

Algorithm 17 Modified Cornacchia Algorithm

Require: Discriminant D and prime number p
Ensure: The solution (x,y)

procedure *Cornacchia*(D,p)
  if $p = 2$ **then** ▷ Initial case
    $k \leftarrow D+8$
    if k is square **then**
      return $(\sqrt{k},1)$
    return *null*
  if $\left(\frac{D}{p}\right) < 1$ **then** ▷ Solvability test
    return *null*
  $x_0 \leftarrow \sqrt{D} \mod p$ ▷ See the Algorithm 15
  if $x_0 \not\equiv D \pmod 2$ **then**
    $x_0 = p - x_0$
  $a \leftarrow 2p$ ▷ Initialize Euclid chain
  $b \leftarrow x_0$
  $c \leftarrow \lfloor 2\sqrt{p} \rfloor$
  while $b > c$ **do** ▷ Euclid chain
    $l \leftarrow a \mod b$
    $a \leftarrow b$
    $b \leftarrow l$
  $t \leftarrow 4p - b^2$ ▷ Results
  if $t \not\equiv 0 \pmod{|D|}$ **then**
    return *null*
  if $t/|D|$ is not square **then**
    return *null*
  return $(\pm b, \pm\sqrt{t/|D|})$

Step 2: Order Factorization

If a suitable discriminant D and the solutions x, y are found (via Algorithm 17), it yields the possible curve order m as the following:

$$m \in \begin{cases} \{n+1 \pm x \,,\, n+1 \pm 2y\} & \text{for } D = -4 \\ \{n+1 \pm x \,,\, n+1 \pm (x \pm 3y)/2\} & \text{for } D = -3 \\ \{n+1 \pm x\} & \text{for } D < -4 \end{cases}$$

If the order m has any factor q such that $q > (\sqrt[4]{n}+1)^2$ and passes the Miller-Rabin test (Algorithm 11). Once an acceptable factor q of m is not found in a timely manner, then the algorithm returns back to choose new discriminant (Step 1).

There are three widely used integer factorization algorithm such as Pollard's p-1, Pollard's p and Lenstra's Elliptic Curve Factorization Method. The most efficient of all is Lenstra's ECM method which consists of group operation on the elliptic curve.

Step 3: Curve Parameters

Since the standard elliptic curve Weierstrass equation $E : y^2 = x^3 + ax + b$ where $a, b \in \mathbb{F}$, we need to find a and b values to construct the curve E.

Before we provide a detailed description of the curve generation algorithm, we need to find a suitable quadratic non-residue g which has no solution to the equation $x^2 \equiv g \pmod{p}$. The following algorithm helps to find g value at random.

Algorithm 18 Finding a non-residue value

Require: Discriminant D and number n
Ensure: Quadratic non-residue g

```
1:  procedure NonResidue(D, n)
2:      p ← (n − 1)/3
3:      do
4:          g ∈ {0, n − 1}
5:          if g = 0 or (g/n) ≠ −1 then
6:              continue
7:              if d = −3 then
8:                  t ← g^p mod n
9:                  if t = 1 then
10:                     continue
11:                 break
12:     while true
```

Now, we use the complex multiplication (CM) method to construct the curve E with parameters a and b. If the chosen discriminant is -3, then 6 isomorphism classes of elliptic curves can be generated with points $(0,-g^k)$ and the curve $y^2 = x^3 - g^k$ where $k \in \{0,5\}$. Similarly, If $D = -4$, then 6 points can be generated with points $(-g^k,0)$ and the curve $y^2 = x^3 - g^k x$ where $k \in \{0,5\}$. Otherwise, either Hilbert class or Weber class polynomial calculation is needed. Since all coefficients of these calculations take time even for small discriminant, we prefer a closed-form solution by using in Table 7.1 [12] which includes precomputed (x,y) values for each discriminant D.

The following algorithm generated curves for a given non-residue value g, discriminant D and number n.

Algorithm 19 Elliptic Curve Generation

Require: Discriminant D and number n
Ensure: Curve constants a and b

```
1: procedure GenerateCurve(D, n)
2:     g ← NonResidue(D, n)                          ▷ See Alg. 18
3:     if d = −3 then
4:         return (0, −g^k) where k ∈ {0,5}          ▷ 6 curves: y^2 = x^3 − g^k
5:     else if d = −4 then
6:         return (−g^k, 0) where k ∈ {0,3}          ▷ 4 curves: y^2 = x^3 − g^k x
7:     else
8:         Select precomputed values at Table 7.1 [12]
9:         return (−3rs^3 g^{2k}, 2rs^5 g^{3k}) where k ∈ {0,1}
```

Step 4: Random Point

Once the elliptic curve is generated, the rest of the algorithm works as in the Shafi-Killian method 14.

Step 5: Point Operation

Once a random point is chosen by using Algorithm 14, we perform group operation on this point. The group operation consists of two different parts: addition an doubling. The details of group operation methods and its analyses are given in Chapter 3 and subsection 3.2.4

The next page includes full pseudocode of the Atkin-Morain ECPP Test.

Algorithm 20 Atkin-Morain ECPP

Require: Number n
Ensure: Prime or composite

```
1: procedure AtkinMorain(n)
2:     if MillerRabin(n) is composite then return composite
3:     else if SieveTest(n) then return composite
4:     else
5:         found ← false                                   ▷ For calculation
6:         result ← false                                  ▷ Compositeness or primality
7:         while !found do
8:             d ← ChooseDiscriminant(n)                   ▷ See Algorithm 16
9:             do
10:                k ← 0
11:                while !found and GenerateCurve(d,n,k) is found do
12:                    P ∈ E(a,b)                          ▷ See Algorithm 14
13:                    if P is not found then break        ▷ New random point
14:                    Q ← [m/q]P mod n
15:                    if Q is not found then              ▷ Composite
16:                        found ← true
17:                        result ← false
18:                        break
19:                    else if Q ≠ ∞ then
20:                        R ← [q]Q mod n
21:                        if R is not found then
22:                            result ← true
23:                            n ← q                       ▷ Try for factor q
24:                            points = 100                ▷ Go to choose discriminant step
25:                            break
26:                    k++
27:                points+ = k
28:            while !found and points < 100
```

Now, we give the following example to illustrate the algorithm for the prime number $2^{89}-1$.

Example.

- **number** = $2^{89}-1$ = 618970019642690137449562111

  D = -3

  (x,y) = (48215832688019 , 7097266064519)

  m = 618970019642738353282250131

  q = 57306024217633

  a = 0, b = 576847241968978529162657802

  P = (257330503798390012319382377 , 611426850505584859173079386)

Q = (505491236768072315331388501 , 40526714008984340075565116 6)

R = (0 , 1)

- **number** = 57306024217633

  D = -3

  (x,y) = (13318945 , 4156513)

  m = 57306037536579

  q = 9014635447

  a = 0, b = 31449508861799

  P = (1732983213944 , 4894216390798)

  Q = (2855554411665 , 14026450171402)

  R = (0 , 1)

- **number** = 9014635447

  D = -3

  (x,y) = (168821 , 50193)

  m = 9014804269

  q = 1908703

  a = 0, b = 7340921285

  P = (5819216896 , 6330263360)

  Q = (2558437617 , 8734715535)

  R = (0 , 1)

- **number** = 1908703

  The given number is **proven prime** because *1908703* is prime.

Running Time and Accuracy. Since Atkin-Morain test avoids very expensive point-counting step, it runs faster than the Shafi-Killian case practically. Moreover, the heuristic results show that the whole primality proof can be done in $O(log^{5+\varepsilon}n)$ where $\varepsilon > 0$ using fast scalar multiplication techniques.

Drawbacks. Since this algorithm uses the complex multiplication method, the construction of the curve with order m is not costly as in Shafi-Killian algorithm. However, finding a good discriminant D and factoring the order m is still a costly

operation. This factorization operation increases the algorithmic complexity of the algorithm.

4.5 Singular Cubic Curve Primality Test

This algorithm [35] is designed for strong pseudoprimes which pass the Miller-Rabin test for base 2. The algorithm first eliminates all composites for base 2 since the Miller-Rabin test guarantees the compositeness of numbers which do not pass the test. Then, the singular cubic primality test generates a singular cubic curve E by choosing a particular prime number, a point on the curve E and tries to compute certain power of that point as we have also given theorems on elliptic curve arithmetic.

The main tool used for this algorithm is singular cubics which are defined below.

Definition 4.5.1 (Singular Cubic Curves)**.** Let E denote the singular curve and a is a non-zero integer. E is defined by an algebraic equation of the form of

$$y^2 = x(x-a)^2 \tag{4.26}$$

where a is a singular point in $\mathbb{Z}/n\mathbb{Z}$.

The generalized Jacobian group, $Jac(E)$, is introduced by Maxwell Rosenlicht in 1954 and associated to curve with a divisor. It is similar to the elliptic curve group [1, 24]. Thus, we can perform the same group law, which is mentioned in Chapter 3, for generalized Jacobian groups.

Let p be a prime, $a > 0$ be a integer such that $gcd(a,n) = 1$ and E be a singular cubic curve defined by the equation 4.26. We have the following theorem from [24] which gives the structure of the group $E(\mathbb{Z}_p)$.

Theorem 4.5.1. *The order of cyclic group $E(\mathbb{Z}_p)$ is either $p-1$ if $Jacobi(a/p) = 1$ or $p+1$ if $Jacobi(a/p) = -1$*

Proof. See the proof of Theorem 2.31 at [24]. □

The key idea in the following theorem is used as a backbone for singular cubic curve algorithm:

Theorem 4.5.2 (Lenstra's Elliptic Curve Factorization Method)**.** *Assume that n be a composite integer with prime factors (p and q), E be an elliptic curve, $k > 0$ and D be an element in generalized Jacobian group, Jac(E). If kD(mod p) gives D_p and kD(mod q) gives D_q such that the only one of D_p or D_q is the identity. Then, the computation fails when computing kD(mod n) and returns a non-trivial factor (q or p) of n.*

Proof. See the proof in [20]. □

By using theorems above, this algorithm conjectures to a primality test which has additional steps to strong pseudoprime test to base 2 and computes some powers of certain points on the curve modulo n. Now, we describe the algorithm by using these mathematical preliminaries on elliptic curves as well as singular cubic curves.

The Algorithm.

Step 0: Miller Rabin Test

The first step of the algorithm is the strong pseudoprime test to base 2 which we have already presented at this chapter. The steps in the singular cubic primality test are for numbers which are already a probable prime in the Miller-Rabin test, so we eliminate composites with the Miller-Rabin test.

Step 1: Curve Generation

Then, the algorithm chooses a convenient small prime number a such that $0 < a < 100$ and the Jacobi $\left(\frac{a}{n}\right) = -1$. The selection of random a is a try-and-error method starting from the smallest prime 2. After finding a suitable prime a (several points may be tried), the algorithm constructs the singular curve E which is of the form $y^2 = x(x-a)^2 \bmod n$.

Step 2: Random Point

Once the singular cubic is generated, the algorithm proceeds to select a random point on the curve which is the form of $P = (1, 1-a)$.

Step 3: Point Operation

At this step, the algorithm computes a certain power of a randomly chosen point P on the generated curve E. The group operation is based on the idea of finding either *illegal*

operations or the infinity point while computing the power of the point P. If there is an illegal group operation, the algorithm proves the compositeness of the number.

In this algorithm, the group operation will be computed by using singular curves. The nice thing about singular curves, we already know the order of group. It's $p-1$ if a is square mod p, and $p+1$ otherwise .

From this observation [35], the order of the curve E mod n must be "$n+1$" if n is "prime". According to the proof of this conjecture, the algorithm first tries with small orders i where $i < 200$ to check $[i]P \pmod{n}$ is identity. If there is no illegal group operation and $[i]P \pmod{n}$ is not identity, then it computes $[n-1]P \pmod{n}$. The final step is to look at the $n+1$-th power of P, i.e., $[n-1]P \oplus [2]P = [n+1]P$, and it must be identity. If $[n+1]P$ is *not* identity, then n is "definitely composite".

The results [3, 21, 25, 26] show that the strong pseudoprimes to base 2 tend to have only two prime factors. For example, the experimental results in [3, 26] show that the majority of spsp(2) are of the form $n = pq$ where p and q are primes and $p-1 = d(q-1)$ such that $d < 20$ and $\gcd(p-1, d_1(n-1)) = p-1$ with $d_1 < 20$. Thus, the algorithm shows that the above algorithm can easily detect compositeness of such integers.

This version of the algorithm given below is for detecting the compositeness of strong pseudoprimes base 2. However, our experimental results also encourage us to claim if $[n+1]P \pmod{n}$ is identity, then given n is "definitely prime". Thus, it is actually a primality test depending on our results.

Now, we give details of the algorithm below:

Algorithm 21 Singular Curve Primality Test Algorithm

Require: An odd integer n

1: **procedure** $PrimalityTest(n)$
2: **if** $MillerRabin(n)$ is composite **then**
3: **return** composite
4: **else**
5: $a \leftarrow a \in \{0,100\}$ where a is prime **and** $\left(\frac{a}{n}\right) = -1$
6: $E(\mathbb{Z}_n) : y^2 = x(x-a)^2$
7: $P \leftarrow (1, 1-a)$
8: **for** $i = 0 \rightarrow 200$ **do**
9: $Q_i \leftarrow [i]P \mod n$
10: **if** $Q_i = \infty$ **then**
11: $P \leftarrow (r^2, r(r-a))$ for some $r \neq a$
12: $Q \leftarrow [n-1]P + [2]P \mod n$
13: **if** $Q \neq \infty$ **then**
14: **return** composite

Running Time and Accuracy. In this algorithm, we need to find a prime $a < 100$ such that $Jacobi(a/n) = -1$. the Jacobi symbol (a/n) can be computed with very low complexity. As seen in the 12^{th} step, we need to compute $(n+1)^{th}$ power of P in the Jacobian of the singular cubic curve. In order to compute $(n-1)P \bmod n$, we need to perform at most $2\log n + 1$ addition or doubling in $E(\mathbb{Z}_n)$. Thus, the running time of the algorithm is $O(\log^{2+\varepsilon} n)$. In terms of accuracy, we tested all strong pseudoprimes less than 2^{64} and algorithm successfully finds the compositeness of all. Additionally, the algorithm finds the primality or compositeness of very large numbers with several hundred digits in just seconds. (see Table 5.2 for execution time analyses).

Drawbacks. Since this algorithm is not tested with numbers larger than 10^{21}, the test is just an observation.

5. COMPARISON RESULTS

In this chapter, the theoretical and computational comparison details of selected algorithms will be given.

The selected algorithms are:

- Sieve of Eratosthenes
- Miller-Rabin Probabilistic Primality Test
- Atkin-Morain ECPP Implementation
- Atkin-Morain GMP-ECPP Executable [40]
- Singular Curve Primality Test

These algorithms will be evaluated based on their algorithmic complexity and experimental tests. Since each algorithm has distinct properties, running time analysis may not give very strict comparable results. For example, some tests include elliptic curve primality testing analog (see Section 4.4 in Chapter 4 for more details) which have not an estimated running time because of randomized values inside. However, the average execution time value will be used in comparison results.

5.1 Theoretical Results

The following table evaluates the theoretical comparison results:

Table 5.1 : Theoretical comparison results.

Primality Test	Running Time
Miller-Rabin	$O(\log^4 n)$ assuming GRH.
ECPP	Hard to make running time analysis. [23]
AKS Algorithm	$O(\log^{12+\varepsilon} n)$
Singular Cubic Primality Test	$O(\log^{2+\varepsilon} n)$

5.2 Computational Tests

In this section, we will examine the running time comparison of the selected algorithms according to execution time until it finds primality or compositeness of a given number. For a fair comparison, we prepared a list of prime numbers which have different numbers of bits and used the same list for each algorithm. There will be two metrics to compare:

- Execution time of each algorithm when determining composites with an increased number of bits
- Execution time of each algorithm when determining primes with an increased number of bits
- Execution time of all algorithms with the same odd composite integer
- Execution time of all algorithms with the same odd prime integer

5.2.1 Implementation details

The algorithms could be implemented in some higher-level programming language like Java or MATLAB; however, they may lack performance during testing. So, the algorithm is implemented and tested by using C++ programming language, because it makes the algorithm very portable and fast with GCC compilers.

Since primality testing is a challenge for very large integers, we needed a library to provides us fast arithmetic for rational numbers and large integers which has several hundred digits. The next thing was to choose of a portable multiple precision library, so we chose GNU Multiple Precision Arithmetic Library (GMP [41]) which is also implemented in C programming language.

Finally, the calculation results may vary due to model or properties of hardware as well as the version of software, we provide the following specifications which include both hardware and software specifications:

Hardware Specifications:

- Processor: Intel Xeon E-2176M (6 cores, 12 threads)
- Main Memory: 32 GB DDR4-2666 RAM (non-ECC)
- Cache Memory: 12 MB
- Disk Capacity: 1 TB PCIe NVMe M.2 SSD
- GPU: NVIDIA Quadro P2000 (Memory: 4GB)

Software Specifications:

- Programming Language: C++ (c++11, c++14 and c++17)
- Compiler: g++
- IDE: CodeLite, Gedit
- Libraries:
  - The GNU Multiple Precision Arithmetic Library (GMP) (Version 6.1.2) [41]
  - Pari/GP (Version 2.11.1) [42]
  - Open Multi-Processing Library (OpenMP) [43]
  - A High-Performance Message Passing Library (MPI) [44]
  - Gmp-Ecpp (version 2.49) [40]

5.2.2 Experiments

- **Sieve of Eratosthenes:** The experimental results show that this method is suitable for only small integers which have less than 60 digits. We tested with random 600 primes which has 1 to 600 digits, the method fails after at 39^{th} prime. Moreover, it fails at 59^{th} composite over random composites which has increased number of digits. Finally, for the case which includes the first 10^5 primes take 236.816 seconds, other cases ignored because of very long execution time.
- **Miller-Rabin probabilistic test:** As we mentioned in the previous chapter, this test is a probabilistic test which can output some composites as prime with some error probability. Since our test includes random primes and composites not including strong pseudoprimes to some special base, the outputs are all correctly labeled.

Additionally, the execution time of this test increases linearly when determining the primality of an integer with an increased round count. However, the round count is not important when determining the compositeness because the algorithm terminates without waiting for the round loop ends. All in all, the Miller-Rabin test results are the best in terms of lowest execution time among other tests.

- **ECPP:** We compared our Atkin's elliptic curve primality testing implementation with GMP-ECPP implementation. The randomization techniques may differ in these two implementations, so we get the time difference in approximate values when the calculation of the first 10^5 primes case. However, our implementation gets better results when $10^{200} + 153$ is calculated. Since both ECPP implementations include the Miller-Rabin test for composites, the case for composites gives actually the running time Miller-Rabin test. Because of this, both ECPP implementation gets much lower running time for composites compared to primes. Finally, all these execution time averages are still slower compared to the Miller-Rabin algorithm.

- **Singular curve primality test:** When we compared the proposed algorithm [35] with well-known primality tests, the experimental results show that the algorithm runs nearly like a probabilistic test Miller-Rabin, additionally, it is not probabilistic. For instance, we try 1500 digits prime numbers, the singular curve primality test takes only 1.5 seconds to prove its primality. Other elliptic curve primality testing algorithms never find the result as fast as the singular curve primality test.

  Moreover, we give another experimental result that we conduct to observe if there is any counterexample in the singular curve primality testing algorithm. Thus, we tested all strong pseudoprimes base 2 less than $2^{64} \approx 1.89 \cdot 10^{19}$ [36] and the algorithm catches all composite numbers. In addition, we tested to extend the limit of the number of pseudoprimes [3], but we were able to find up to 10^{10} without using parallel programming on supercomputers. Finally, we tested several million high precision strong pseudoprimes base 2 and again the algorithm detects compositeness of such large numbers with very short execution time.

The following table contains the execution time of each algorithm for different cases which includes random primes and composites with a different number of digits:

Table 5.2 : The execution time of each algorithm in seconds.

Cases	Primality?	# Integers	# Digits	Miller-Rabin (10)	Miller-Rabin (100)	Our ECPP	GMP-ECPP	Singular Curve Test
First 10^5 primes	Yes	10^5	1-7	2.1686	22.211	$\approx$ 274.496	$\approx$ 98.550	551.08
$2^{81}-1$	Yes	1	27	0.000572	0.000881	$\approx$ 0.069679	$\approx$ 0.063651	0.005677
Random	Yes	1	53	0.00056	0.001728	$\approx$ 1.86175	$\approx$ 1.738	0.008462
$10^{200}+153$	Yes	1	200	0.001893	0.014376	$\approx$ 134.467	$\approx$ 247.712	0.029982
Primes	Yes	600	1-200	0.320846	3.13526	-	-	8.59567
Composites	No	600	1-200	0.036731	0.035407	$\approx$ 0.049723	$\approx$ 0.361965	1.15666
Random	Yes	1	300	0.005432	0.045727	-	-	0.066517
Random	Yes	1	500	0.018514	0.173619	-	-	0.15979
Random	Yes	1	700	0.047668	0.470509	-	-	0.300855
Random	Yes	1	1000	0.119327	1.179	-	-	0.647198
Random	Yes	1	1200	0.197571	2.02547	-	-	0.928898
Random	Yes	1	1500	0.384182	3.7633	-	-	1.52689

6. CONCLUSION

Prime numbers are crucial in the field of number theory and especially in cryptography. Different types of primality tests have their own advantages and disadvantages. Probabilistic tests are faster; on the other hand, deterministic tests can determine results without employing a doubt.

In this thesis, we analyze the trade-offs between known primality tests. We first introduce general number theoretical methods, such as Fermat's and Euler's test. Because determining and proving the primality of a potential prime number cannot be done with certainty with these methods, we proceed with deterministic tests. A disadvantage of the deterministic Agrawal–Kayal–Saxena (AKS) test is that it is hard to implement, so we then move on to elliptic curve primality testing and proving algorithms that are developed by Shafi-Killian and Atkin-Morain.

In general, the evaluation of primality testing algorithms is illustrated with examples. The theoretical results show that deterministic polynomial-time algorithms are the best. However, our practical results demonstrate that probabilistic methods are faster for commercial applications.

Moreover, we also included singular cubic primality testing algorithm into our comparison, because the widely used Miller-Rabin algorithm has strong pseudoprimes and the conjectured test [35] catches them. The experimental results show that the conjectured algorithm catches all strong pseudoprimes (base 2) up to 2^{64}. Moreover, even the compositeness of high-precision integers is successfully detected by this algorithm with a very short running time (see Table 5.2).

All in all, the execution time of the singular cubic curve primality test is close to that of the probabilistic Miller-Rabin test; additionally, the algorithm does not employ a probability when determining compositeness or primality of a given integer. When this hypothesis is proved to be a successful primary test, we have been demonstrated that

the singular cubic curve primality test had good theoretical and experimental results, as evaluated in this thesis.

REFERENCES

[1] **Cohen, H.** (2000). A Course in Computational Algebraic Number Theory, *Springer-Verlag*.

[2] **Cohen, H. and Frey, G.** (2005). Handbook of Elliptic and Hyperelliptic Curve Cryptography, *Chapman & Hall/CRC*.

[3] **Jaeschke, G.** (1993). On Strong Pseudoprimes to Several Bases, *Math. Comp. (204)*, 915–926.

[4] **Adleman, L. M., Pomerance, C. and Rumely, R. S.** (1983). On distinguishing prime numbers from composite numbers, *Ann. of Math. 2, 117:1*, 173–206.

[5] **Pettofrezzo, A. J. and Byrkit, D. R.** (1994). Elements of Number Theory (2nd Edition), *Orange Pub*.

[6] **Rosen, K.H.** (1984). Elementary Number Theory and Its Applications, *Pearson/Addison Wesley*.

[7] **Agrawal, M., Kayal, N. and Saxena, N.** (2004). Primes is in P, *Annals of Math.* 160, 781–793.

[8] **Atkin, A. O. L. and Morain, F.** (1993). Elliptic curves and primality proving. *Math. Comp.*, 61(203): 29-68.

[9] **Goldwasser, S. and Killian, J.** (1999). Primality testing using elliptic curves, *J. ACM 46*, 4, 450-472.

[10] **Solovay, R. M. and Strassen, V.** (1977). A fast Monte-Carlo test for primality, *SIAM Journal on Computing 6*, 84-85.

[11] **Schoof, R.** (1995). Counting points on elliptic curves over finite fields, *Journal de Théorie des Nombres de Bordeaux*, tome 7, no 1, p. 219-254.

[12] **Crandall, R. and Pomerance, C.** (2001). Prime numbers: a computational perspective, *Springer*, New York.

[13] **Jiang, Y. and Deng, Y.** (2014). Strong pseudoprimes to the first eight prime bases, *Math. Comp. 83*, No 290, 2915-2924.

[14] **Koblitz, N.** (1987). Elliptic curve cryptosystems, *Math. Comp. 48*.

[15] **Benger, N., Pol, J.V.D., Smart, N.P. and Yarom, Y.** (1977). "Ooh Aah... Just a Little Bit": A small amount of side channel can go a long way, *CHES 2014*, 75-92.

[16] **Miller, G.** (1976). Riemann's hypothesis and tests for primality, *J. Comput. and System Sci. 13*, 300-317.

[17] **Rabin, M. O.** (1980). Probabilistic algorithms for testing primality, *J. Number Theory 12*, 128-138.

[18] **Muller, S.** (2003). A Probable Prime Test With Very High Confidence for n≡ 3 mod 4, *J. Cryptology*, Vol 16, No. 2, pp. 117-139.

[19] **Mullen, G.L. and Panario., D.** (2013). Handbook of Finite Fields, *Chapman and Hall/CRC.*

[20] **Lenstra, Jr. H. W.** (1987). Factoring integers with elliptic curves, *Ann. of Math. (2)*, 126(3), 649-673.

[21] **Pomerance, C., Selfridge, J. L. and Wagstaff, Jr. Samuel S.** (1980). The pseudoprimes to $25 * 10^9$, *Math. Comp. 35*, 1003-1026.

[22] **Rivest, R. L.**, **Shamir, A. and Adleman, L.** (1978). A method for obtaining digital signatures and public key cryptosystems, *Commun. of the ACM*, 21:120-126.

[23] **Schoof, R.** (2008). Four primality testing algorithms, Algorithmic Number Theory, *MSRI Publications*, Volume 44, 101-126.

[24] **Washington, L. C.** (2008). Elliptic Curves: Number Theory and Cryptography, 2nd edition. *Chapman & Hall/CRC.*

[25] **Zhang, Z.** (2001). Finding strong pseudoprimes to several bases, *Math. Comp. 70*, 863-872.

[26] **Zhang, Z. and Tang, M.** (2003). Finding strong pseudoprimes to several bases II, *Math. Comp. 72*, 2085-2097.

[27] **Wells, D.** (1986) The Penguin Dictionary of Curious and Interesting Numbers, *Penguin Books*, Penguin Press Science.

[28] **Jones, G. A. and Jones, J. M.** (1998). Elementary Number Theory, *London: Springer*

[29] **Hasse, H.** (1936). Zur Theorie der abstrakten elliptischen Funktionenkorper, I, II, III, *Crelle's Journal(175)*,55-63,69-88,193-208.

[30] **Boneh, D.** (2002). Twenty Years of Attacks on the RSA Cryptosystem, *Notices of the American Mathematical Society 46(3).*

[31] **Gaudry, P.** (2014). Integer factorization and discrete logarithm problems, *Notes d'un cours donné aux Journées Nationales de Calcul Formel.*

[32] **Buchman, J. A.** (2004). Introduction to cryptography (2nd Edition), *Springer-Verlag*, New York.

[33] **Montgomery, P.L.** (1987). Speeding the Pollard and elliptic curve methods of factorization, *Mathematics of Computation*, 48(177):243–264.

[34] **Solomon, R.** (2003). Abstract Algebra, *American Math. Soc.*, 48(177):243–264.

[35] **Ozdemir, E.** (t.y.). Primality Test with Singular Cubic Curves.

[36] **Url-1** <http://www.cecm.sfu.ca/Pseudoprimes/index-2-to-64.html>, date retrieved 04.05.2019.

[37] **Url-2** <http://oeis.org>, date retrieved 04.05.2019.

[38] **Url-3** <http://oeis.org/A006880>, date retrieved 04.05.2019.

[39] **Url-4** <http://primes.utm.edu/primes/lists/all.txt>, date retrieved 04.05.2019.

[40] **Url-5** <https://sourceforge.net/projects/gmp-ecpp/>, date retrieved 04.05.2019.

[41] **Url-6** <https://gmplib.org/>, date retrieved 04.05.2019.

[42] **Url-7** <ftp://pari.math.u-bordeaux.fr/pub/pari/manuals/2.3.5/users.pdf>, date retrieved 04.05.2019.

[43] **Url-8** <https://www.openmp.org/>, date retrieved 04.05.2019.

[44] **Url-9** <https://www.mpich.org/>, date retrieved 04.05.2019.

CURRICULUM VITAE

Name Surname **:** Gözde SARIKAYA

Place and Date of Birth **:** Eminönü, 07 June 1992

E-Mail **:** sarikayagozde@outlook.com

EDUCATION

- **B.Sc. :** 2015, Fatih University, Faculty of Arts and Sciences, Mathematics.
- **B.Sc. :** 2015, Fatih University, Faculty of Engineering, Computer Engineering.
- **M.Sc. :** 2019, Istanbul Technical University, Informatics Institute, Information Security Engineering and Cryptography.

PROFESSIONAL EXPERIENCE:

- 2014, Undergraduate Research Intern, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, UK.
- 2015, Undergraduate Research Intern, Department of Computer Engineering, Fatih University, Turkey.
- 2015, Researcher, National High-Performance Center of Turkey (UHeM), Istanbul Technical University, Turkey.
- 2016, Graduate Student, European Union PRACE Project - Summer of HPC Programme, IT4 National HPC Center, Czech Republic.
- 2016, Researcher, National Research Institute of Electronics and Cryptology (UEKAE), TUBITAK, Turkey.
- 2017, R&D Software Engineer, CRYPTTECH, Yildiz Technical University, Turkey.

ACHIEVEMENTS & HONORS:

- 2012, Ranked 1st in the Department of Mathematics.
- 2012, Ranked first in the Competition for Game Development with Java.
- 2010-2015, High Honor List for eight semesters in Department of Mathematics.
- 2015, Ranked third in Graduation from Department of Computer Engineering.

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- Ozdemir E., **Sarikaya G.,** 2016. A Primality Testing Algorithm. *ACMES - Computational Discovery in Mathematics*, May 12-15 2016, Western University, Ontario, Canada.

- Ozdemir E., **Sarikaya G.,** 2019. A Note on Primality Testing: Algorithms and Analyses. *ICOMAA - 2nd International Conference on Mathematical Advances and Applications*, May 3-5 2019, Yildiz Technical University, Istanbul, Turkey.