iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 11 **Issue:** VIII **Month of publication:** Aug 2023

DOI: https://doi.org/10.22214/ijraset.2023.55341

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VIII Aug 2023- Available at www.ijraset.com

Enhancing Post-Quantum Cryptography: Exploring Mathematical Foundations and Comparative Analysis of Different Cryptographic Algorithm

Aayush Shah[1], Prabhat Mahato[2], Aadarsh Bhagat[3]
St. Xavier's College, Maitighar, Nepal

Abstract: ***This research paper surveys the landscape of cryptography, encompassing historical origins and modern applications. Beginning with foundational concepts, it explores encryption, decryption, ciphers, and keys, spanning symmetric and asymmetric cryptography. Historical context unfolds, tracing cryptography from ancient Egyptian hieroglyphs to Julius Caesar's cipher. The study then transitions to contemporary subjects. Elliptic Curves and Cryptography are investigated, showcasing their significance in secure communication, demonstrating ECC key exchange and AES-GCM encryption using python and Comparative analysis of ECC, RSA, and Diffie-Hellman sheds light on their performance. Zero-Knowledge Proofs are introduced as tools for privacy-preserving verification followed by an exploration of various Zero-Knowledge Proof (ZKP) protocols. By presenting practical implementation examples using Python, the paper illustrates how these proofs can be applied in real-world scenarios. Random Number Generation is examined and distinction between pseudorandom number generators (PRNGs) and cryptographically secure PRNGs (CSPRNGs) is emphasized conducting a thorough comparative analysis of PRNGs and CSPRNGs, considering factors like correlation, independence, periodicity, and entropy. Furthermore, the section evaluates the performance of different random number generation techniques. Fully Homomorphic Encryption emerges as a groundbreaking concept, discussing its mathematical properties, practical implementation, parameter selection, and optimization techniques enabling computation on encrypted data. Cryptographic Secret Sharing Schemes are explored for secure information distribution. The paper concludes by delving into the Chinese Remainder Theorem's applications within modern cryptographic protocols, particularly in RSA decryption and the integration factorization process of the RSA public key cryptosystem. It also provides a comprehensive overview of the theoretical foundations of primality testing, a pivotal aspect of the RSA algorithm. Overall, this research paper provides a comprehensive exploration of cryptography's historical context, core concepts, advanced techniques, and practical implementations, offering valuable insights into the realm of secure communication.***

Keywords: Cryptography, encryption, decryption, RSA, PRNG, CSPRNG, Entropy, Periodicity, Homomorphic, CRT, Elliptical Curves, Bootstrapping.

I. INTRODUCTION

Cryptography [15] is a method of developing techniques and protocols to prevent a third party from accessing and gaining knowledge of the data from the private messages during a communication process. There are several terms related to cryptography, including encryption, decryption, cipher, and key.

The components of cryptography are plaintext, encryptions, ciphers, secret keys, ciphertext and decryption. In today's age of computers [16], cryptography is often associated with the process where ordinary plain text is converted to ciphertext, which is the text made such that the intended receiver of the text can only decode it; hence, this process is known as encryption. The process of converting ciphertext to plain text is known as decryption.

There are basically three different ways in which cryptographic algorithms can be performed:

1) *Symmetric-key Cryptography:* In symmetric-key cryptography, also known as secret-key cryptography, there is only one key. The key is used for both encryption and decryption. Using a common single key creates a security problem of transferring the key between the sender and the receiver.

©IJRASET: All Rights are Reserved

2) *Asymmetric-Key Cryptography:* Asymmetric-key cryptography, also known as public-key cryptography, uses a pair of keys, an encryption key and a decryption key, named public key and private key respectively. The key pair generated by this algorithm consists of a private key and a unique public key that is generated using the same algorithm.
3) *Hash Functions:* Hash functions do not make use of keys. Instead, it uses a cipher to generate a hash value of a fixed length from the plaintext. It is nearly impossible for the contents of plain text to be recovered from the ciphertext.

A. History

Cryptography is not a modern-day invention. In fact, there have been many different types of cryptography throughout history, with the first known evidence of cryptography being traced to the use of hieroglyphs." Some 4000 years ago, the Egyptians used to communicate by writing messages in hieroglyphs. This code was a secret known only to the scribes who used to transmit messages on behalf of the kings. Julius Caesar also used a form of encryption to convey secret messages to his army generals on the war front in 100 BC. This substitution cipher, known as the Caesar cipher, is perhaps the most frequently mentioned historic cipher in academic literature. During the 16th century, Vigenere designed a cipher that was supposedly the first cipher that used an encryption key.

II. ELLIPTIC CURVES AND CRYPTOGRAPHY:

A. Introduction

Elliptic curve cryptography (ECC) is a type of public key cryptography that relies on the mathematical properties of elliptic curves. An elliptic curve is a smooth, continuous curve defined by an equation of the form:

$$y^2 = x^3 + ax + b$$

Where a and b are constants that determine the shape and position of the curve. The curve also contains a special point at infinity, which serves as the identity element. The key idea in ECC is to perform cryptographic operations using points on these curves. Elliptic curves have a mathematical structure that forms an abelian group under an operation called point addition. Given two points P and Q on the curve, you can find a third point R such that P + Q = R. This operation is commutative, and each point has an inverse.

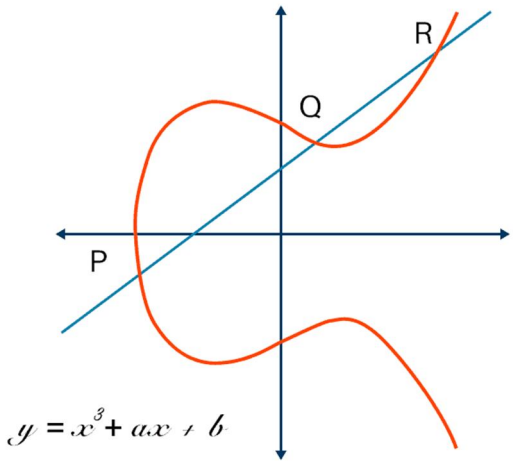


Figure 1: Elliptic Curve

ECC doesn't rely on factorization but instead solves equations (elliptic curves) of the form.[1] The security of ECC is based on the difficulty of a problem called the "discrete logarithm problem." In simple terms, if you have a point P on the curve and another point Q that is a result of multiplying P by a secret number (private key) such that Q = kP, the challenge is to figure out what that secret number is, given only P and Q.

This problem is believed to be difficult to solve, even with modern computing power, and that's what makes ECC secure. It's different from RSA, where security relies on the difficulty of factoring large numbers.

B. ECC in Cryptography

ECC is primarily used for two purposes in cryptography: key exchange and digital signatures.

1) *Key Exchange:* ECC can be used to establish a shared secret between two parties through the Diffie-Hellman key exchange protocol. This shared secret can then be used to derive symmetric keys for secure communication.
2) *Digital Signatures:* ECC can also be used to create digital signatures. A private key holder can sign a message with their private key, and anyone with the corresponding public key can verify the authenticity of the signature.

C. Security Properties of ECC

1) *Short Key Length:* ECC offers the same level of security as traditional systems like RSA and Diffie-Hellman but with significantly shorter key lengths. For example, a 256-bit ECC key is considered as secure as a 3072-bit RSA key.
2) *Efficiency:* ECC operations are faster than many other cryptographic systems due to their underlying mathematical structure. This makes ECC well-suited for resource-constrained devices like mobile phones and Internet of Things (IoT) devices.
3) *Resistance to Quantum Attacks:* ECC is believed to be resistant to quantum attacks based on Shor's algorithm, which can efficiently factor large integers and break RSA-like systems. ECC's resistance to such attacks makes it a promising choice for post-quantum cryptography.

D. Python Code Example: Secure Communication with ECC Key Exchange and AES-GCM Encryption

This code demonstrates the complete process of generating keys using ECC, deriving encryption keys, encrypting, and decrypting data using AES in GCM mode, and including the authentication tag for verification during decryption.

Code Snippet: https://colab.research.google.com/drive/1AiCXLKEs9peJaLGYA8RNwvynS-1AtR8U#scrollTo=d8x0X6F_DCPE&line=6&uniqifier=1

E. Comparative Performance Analysis of Cryptographic Algorithms: ECC, RSA, and Diffie-Hellman

Code Snippet: https://colab.research.google.com/drive/1AiCXLKEs9peJaLGYA8RNwvynS-1AtR8U#scrollTo=qvSWG5RrDeW2&line=46&uniqifier=1

This code measures the time taken for key generation, encryption, and decryption using various cryptographic algorithms (ECC, RSA, Diffie-Hellman) and different data sizes. It then displays the results in a table format using the tabulate library. The purpose is to compare the performance of these algorithms in terms of time.

Table 1: Output of the Comparative Performance Analysis of Cryptographic Algorithms: ECC, RSA, and Diffie-Hellman

Input Size (bits)	Algorithm	Operation	Average Time (seconds)
128	ECC	keygen	0.000183535
128	ECC	encrypt	0.00798869
128	ECC	decrypt	0.00812683
128	RSA	keygen	9.99212e-05
128	RSA	encrypt	0.0162943
128	RSA	decrypt	0.00853505
128	Diffie-Hellman	keygen	0.0001122
128	Diffie-Hellman	encrypt	0.00999649
128	Diffie-Hellman	decrypt	0.00802293
256	ECC	keygen	0.000106907
256	ECC	encrypt	0.0080472
256	ECC	decrypt	0.00904293
256	RSA	keygen	0.000103092
256	RSA	encrypt	0.0080174
256	RSA	decrypt	0.00787609
256	Diffie-Hellman	keygen	0.000133419
256	Diffie-Hellman	encrypt	0.00797482
256	Diffie-Hellman	decrypt	0.00811784

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

512	ECC	keygen	0.000100136
512	ECC	encrypt	0.00946639
512	ECC	decrypt	0.0102986
512	RSA	keygen	9.88245e-05
512	RSA	encrypt	0.0078618
512	RSA	decrypt	0.00813406
512	Diffie-Hellman	keygen	0.000107694
512	Diffie-Hellman	encrypt	0.00801425
512	Diffie-Hellman	decrypt	0.00786257
1024	ECC	keygen	0.000101542
1024	ECC	encrypt	0.0090966
1024	ECC	decrypt	0.00989945
1024	RSA	keygen	0.000138521
1024	RSA	encrypt	0.00799379
1024	RSA	decrypt	0.00914361
1024	Diffie-Hellman	keygen	0.000127673
1024	Diffie-Hellman	encrypt	0.00811684
1024	Diffie-Hellman	decrypt	0.00802405
2048	ECC	keygen	0.00010078
2048	ECC	encrypt	0.00801871
2048	ECC	decrypt	0.00793025
2048	RSA	keygen	0.000101805
2048	RSA	encrypt	0.0167916
2048	RSA	decrypt	0.0172631
2048	Diffie-Hellman	keygen	0.000187469
2048	Diffie-Hellman	encrypt	0.0173658
2048	Diffie-Hellman	decrypt	0.0171651

Here's a comparison of the algorithms' performance for each operation and input size:

1) Key Generation

- For small input sizes (128 bits), ECC is the fastest.
- As input sizes increase (256, 512, 1024, 2048 bits), ECC remains consistently fast, followed by Diffie-Hellman and then RSA.

2) Encryption

- For small input sizes (128 bits), ECC is the fastest, followed by Diffie-Hellman and then RSA.
- As input sizes increase (256, 512, 1024, 2048 bits), ECC consistently maintains its speed advantage for encryption, followed by Diffie-Hellman and RSA.

3) Decryption

- For small input sizes (128 bits), ECC and Diffie-Hellman show similar decryption times, while RSA is slightly slower.
- As input sizes increase (256, 512, 1024, 2048 bits), RSA's decryption time remains slower compared to ECC and Diffie-Hellman, which still exhibit similar performance.

In summary, ECC generally performs faster for key generation, encryption, and decryption across various input sizes. Diffie-Hellman also performs well, particularly for encryption and decryption. RSA tends to be slower in comparison, especially for encryption and decryption tasks.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

III. ZERO-KNOWLEDGE PROOFS

A. Introduction

Zero-Knowledge Proofs (ZKPs) are fascinating cryptographic concepts that allow one party (the prover) to prove to another party (the verifier) that a certain statement is true without revealing any information about the statement itself. The main goal of a ZK-proof is to persuade the verifier that a claim is true without revealing any information other than the claim's veracity. [2] This concept was introduced by Shafi Goldwasser, Charles Rackoff, and Silvio Micali in the 1980s.

B. Basic Concepts

1) *Completeness:* If the statement is true, an honest verifier will be convinced by an honest prover.
2) *Soundness:* If the statement is false, no cheating prover can convince an honest verifier.
3) *Zero-Knowledge:* The verifier learns nothing beyond the truth of the statement.

C. ZKP Protocols

1) Schnorr Protocol

The Schnorr protocol is a basic ZKP that allows a prover to prove knowledge of a discrete logarithm without revealing the logarithm itself. It forms the basis for many advanced protocols.

Zero-Knowledge Proof Simulation (Schnorr Protocol – Python Code Snippet: https://colab.research.google.com/drive/1rJwGBJ_Zyd6o5bIWBEK46Msqs9VSI2qq#scrollTo=Q0ilWbbxJjpo&line=8&uniqifier=1

2) Fiat-Shamir Transform

This technique is used to turn certain interactive proofs into non-interactive ones, making them suitable for practical implementation. It takes a protocol with interaction between prover and verifier and transforms it into a protocol based solely on the prover's message.

3) ZK-SNARKs (Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge)

ZK-SNARKs are highly efficient ZKP protocols that allow for compact proofs and verification. They are commonly used in blockchain systems like Zcash to provide private transactions without revealing transaction details.

4) Bulletproofs

Bulletproofs are a range proof protocol that allows proving that a secret value lies within a certain range, without revealing the actual value. They are used to enhance privacy in confidential transactions and are more efficient than some previous approaches.

5) zk-STARKs (Zero-Knowledge Scalable Transparent ARguments of Knowledge):

zk-STARKs are ZKP protocols that provide scalability by eliminating the need for a trusted setup. They find applications in scenarios where trust in the setup process is a concern.

D. Types of Zero-Knowledge Proofs

There are several types of Zero-Knowledge Proofs. Some of them are:

1) Interactive Zero-Knowledge Proofs (iZKPs)

Interactive ZKPs involve a back-and-forth interaction between the prover and verifier. The protocols usually go through multiple rounds of communication. A well-known example of an interactive ZKP is the "Three-coloring Proof," where the prover demonstrates that a given graph can be colored with three colors without revealing the actual coloring. The Schnorr protocol is another example.

Interactive Zero-Knowledge Proof for Graph Coloring - Python Code Snippet:
https://colab.research.google.com/drive/1A7L8RMrOcCYYPr3VyXkstN1ZlIwL7r11#scrollTo=CbUUgsEljU0G&line=7&uniqifier=1

2) Non-Interactive Zero-Knowledge Proofs (NIZKPs)

NIZKPs eliminate the need for repeated interactions between the prover and verifier. These proofs are highly efficient and have found significant applications in blockchain and privacy-enhancing technologies. One of the most famous NIZKP constructions is the "zk-SNARK" (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) used in the Zcash cryptocurrency.

Age Verification System Using zk-SNARKs and RSA Encryption - Python Code Snippet: https://colab.research.google.com/drive/1A7L8RMrOcCYYPr3VyXkstN1ZlIwL7r11#scrollTo=p7SnAEc7m1RX&line=32&uniqifier=1

3) Statistical Zero-Knowledge Proofs

Statistical ZKPs allow a proof to be convincing only with high probability, rather than with absolute certainty. These proofs are useful when achieving perfect zero-knowledge is too resource-intensive.

Statistical Zero-Knowledge Proof Example: Demonstrating Secrecy Without Revealing - Python Code Snippet: https://colab.research.google.com/drive/1A7L8RMrOcCYYPr3VyXkstN1ZlIwL7r11#scrollTo=xwlUpq4lrSVs&line=27&uniqifier=1

4) Proofs of Knowledge

Proofs of knowledge are a subset of zero-knowledge proofs where the prover not only convinces the verifier of the truth of a statement but also demonstrates that they possess certain knowledge without revealing it. These are commonly used in authentication protocols.

Demonstrating Proof of Knowledge Using ECDSA on SECP256k1 Curve - Python Code Snippet: https://colab.research.google.com/drive/1A7L8RMrOcCYYPr3VyXkstN1ZlIwL7r11#scrollTo=4ZFtjTSf_X3x&line=17&uniqifier=1

E. Applications of Zero-Knowledge Proofs

1) *Privacy-Preserving Transactions:* ZKPs are widely used in cryptocurrencies and blockchain systems to enable private transactions. For instance, in Zcash, ZK-SNARKs are used to ensure transaction validity and balance without revealing sender, receiver, or transaction amount.
2) *Authentication:* ZKPs can be employed for password less authentication. A user can prove their identity to a server without transmitting a password or any personal information, enhancing security and privacy.
3) *Data Sharing and Proofs:* ZKPs can enable sharing specific information from sensitive datasets without revealing the entire dataset. This is useful in scenarios where parties want to prove certain properties about their data without exposing the data itself.
4) *Digital Signatures:* ZKPs can improve digital signatures by allowing users to prove the ownership of a private key without actually revealing the private key. This enhances security against attacks involving key exposure.
5) *Anonymous Credentials:* Zero-knowledge proofs can be used to issue digital credentials that provide proof of certain attributes (like age or qualifications) without revealing the actual attributes themselves.

F. Challenges and Considerations

1) *Efficiency:* While ZKPs offer enhanced privacy, their computational complexity can be a challenge, making them resource-intensive in certain scenarios.
2) *Trusted Setup:* Some ZKP protocols require a trusted setup phase, raising concerns about the security of this setup process. Compromises during setup can undermine the entire system's security.
3) *Standardization and Adoption:* Widespread adoption of ZKPs requires standardization efforts to ensure interoperability and security across different implementations.

In summary, zero-knowledge proofs are powerful tools that can significantly enhance privacy and authentication in cryptographic systems.

They enable efficient and secure ways to prove statements without revealing sensitive information, making them a crucial building block for privacy-preserving technologies. However, their deployment requires careful consideration of their mathematical underpinnings, efficiency, and potential security challenges.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VIII Aug 2023- Available at www.ijraset.com

IV. RANDOM NUMBER GENERATION

A. Introduction

CSPRNG (Cryptographically Secure Pseudo-Random Number Generator) and PRNG (Pseudo-Random Number Generator) are two types of algorithms used to generate random numbers in computer systems. However, they serve different purposes and have different levels of security.

1) PRNG (Pseudo-Random Number Generator)

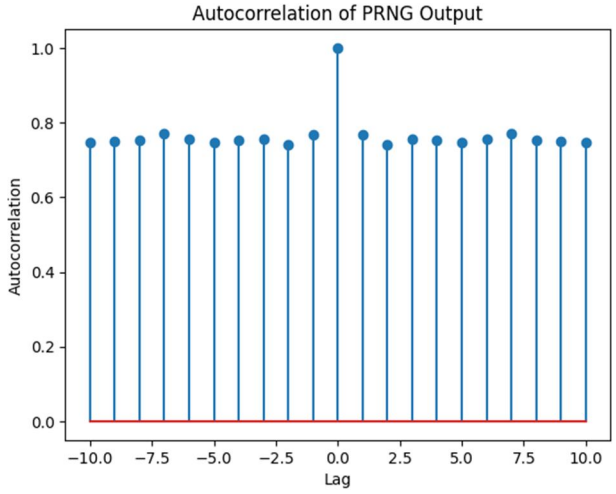
A pseudorandom number generator (PRNG), also known as a deterministic random bit generator (DRBG),[3] is an algorithm for generating a sequence of numbers whose properties approximate the properties of sequences of random numbers. They are actually determined by an initial value called a seed. Once seeded, a PRNG generates a sequence of numbers that may seem random, but they are generated in a deterministic way. Although sequences that are closer to truly random can be generated using hardware random number generators, pseudorandom number generators are important in practice for their speed in number generation and their reproducibility.[4] PRNGs are used in various applications like simulations, games, and some non-cryptographic tasks. They are not suitable for cryptographic purposes because their predictability can be exploited by attackers if they can determine or guess the seed.

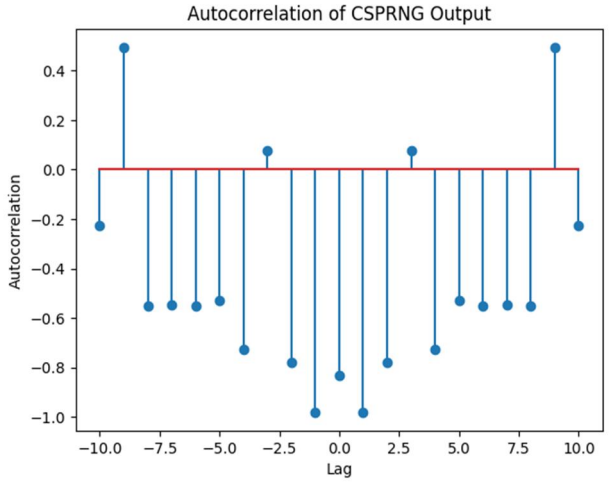
2) CSPRNG (Cryptographically Secure Pseudo-Random Number Generator)

A cryptographically secure pseudorandom number generator (CSPRNG) or cryptographic pseudorandom number generator (CPRNG)[5] is a pseudorandom number generator (PRNG) with properties that make it suitable for use in cryptography**.** They are constructed in a way that makes it computationally infeasible to predict their output or to distinguish their output from truly random data, even if an attacker knows some of the internal state of the generator. It is also loosely known as a cryptographic random number generator (CRNG),[6][7] which can be compared to "true" vs. pseudo-random numbers. CSPRNGs are crucial for various security-sensitive tasks, such as generating encryption keys, nonces, initialization vectors, and other cryptographic values. They are designed to resist various types of attacks, including statistical analysis and deterministic prediction.

B. Comparative Analysis between PRNGs and CSPRNGs

1) Based on Correlation and Independence.





Python Code Snippet: https://colab.research.google.com/drive/1-fHjy5LrArY0wfhbCe1mesdTXwngOo16#scrollTo=JyDjflRJi6yN&line=31&9uniqifier=1

(Note: The appearance of the graphs may vary due to the randomness of the data each time the code is executed, but the underlying properties remain consistent.)

From above Graphs, the autocorrelation values for the PRNG generator are mostly positive and relatively high, especially when the lag is 0 (where autocorrelation is always 1 by definition). This suggests that the generated random numbers are somewhat correlated with each other, especially when close in time. The values gradually decrease as the lag increases, but they still indicate some degree of correlation.

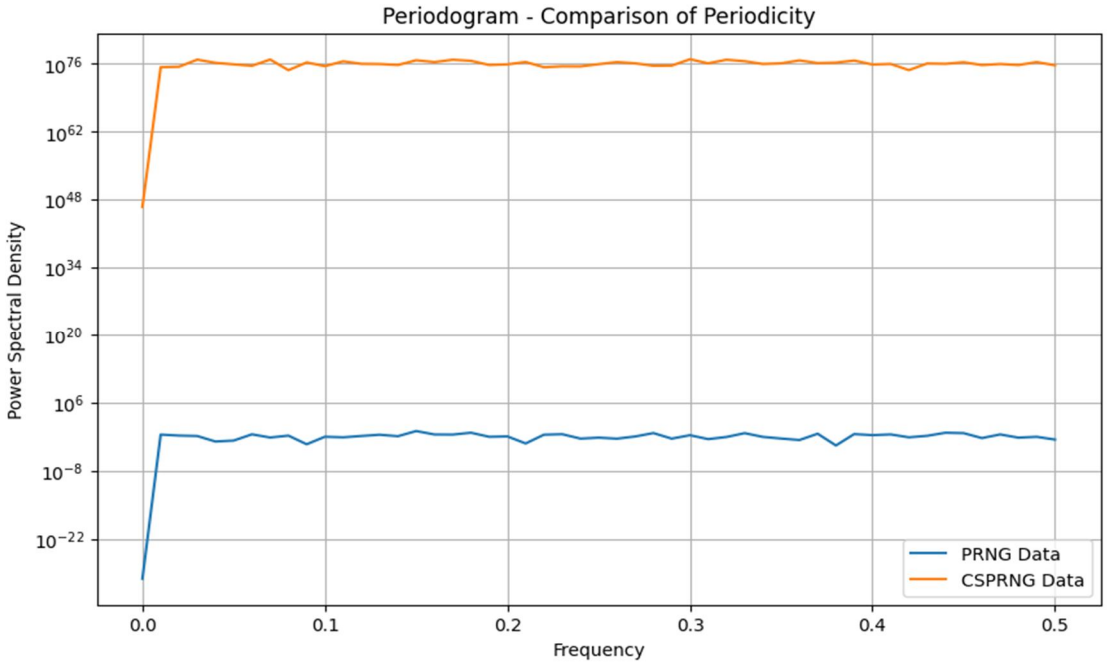
©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VIII Aug 2023- Available at www.ijraset.com

For the CSPRNG generator, the autocorrelation values are both positive and negative, indicating a more complex pattern. The values vary across the lag range, and some are relatively high, while others are quite low. The presence of negative values suggests that there is some level of anti-correlation, meaning that the sequence alternates between higher and lower values over certain lags.
In summary, while both generators exhibit some degree of autocorrelation, the autocorrelation pattern for the CSPRNG generator appears to be more complex and less uniform compared to the PRNG random generator. This suggests that PRNG generator's sequence might be more correlated and less random than the sequence generated by the CSPRNG.

2) Based on Periodicity

The provided periodogram consists of two periodogram graphs: one labelled as "PRNG Data Periodogram" and the other labelled as "CSPRNG Data Periodogram." A periodogram is a tool used in signal processing and statistics to analyse the frequency components of a signal.



Python Code Snippet: https://colab.research.google.com/drive/1-fHjy5LrArY0wfhbCe1mesdTXwngOo16#scrollTo=itRzj56ihcc9&line=31&uniqifier=1

(Note: The appearance of the graphs may vary due to the randomness of the data each time the code is executed, but the underlying properties remain consistent.)

For both graphs, the frequency values are distributed in x-axis from 0 to 0.5. These frequencies represent different cycles per unit time (or normalized frequency) in the analysed signal. The "Power Spectral Density" in y-axis associated with each frequency indicate the amount of power or energy that each frequency component contributes to the overall signal. Higher power spectral density values indicate stronger presence of that frequency component in the signal.

3) Frequency Data (PRNG and CSPRNG)

a) Both PRNG and CSPRNG generate data at different frequency points ranging from 0 to 0.5.

b) The frequencies are evenly distributed and cover a wide range.

4) Power Spectral Density Data (PRNG and CSPRNG)

a) PRNG

- The power spectral density of the PRNG data shows a pattern with peaks and valleys across different frequencies.
- The presence of recognizable patterns in the power spectral density might indicate some level of predictability or regularity in the generated data.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

b) CSPRNG

- The power spectral density of the CSPRNG data appears to be much more chaotic and lacks the distinct peaks and valleys observed in the PRNG data.
- The lack of prominent patterns suggests that the CSPRNG data is less predictable and potentially more random compared to the PRNG data.

C. Interpretation and Comparison for Cryptography

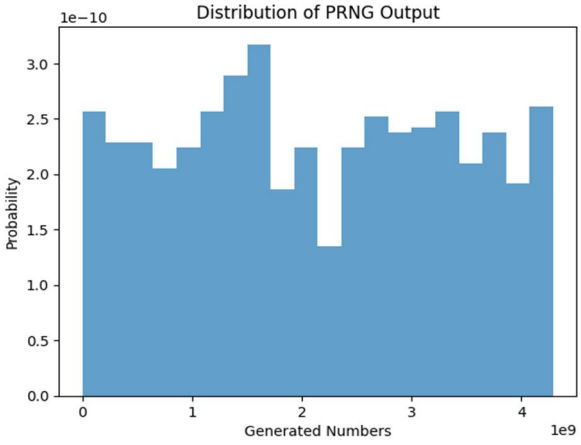
Cryptography relies heavily on the unpredictability and randomness of generated numbers. From the data and its interpretation:

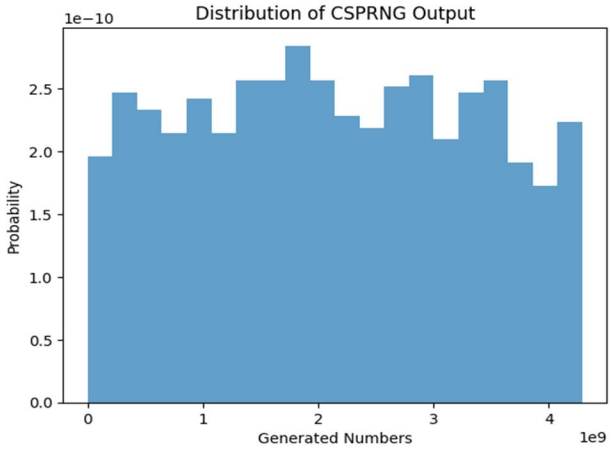
- The CSPRNG data appears to have a more uniform and chaotic distribution in the power spectral density, indicating higher randomness compared to the PRNG data.
- PRNG data shows patterns that could potentially be exploited by an attacker who understands these patterns.

Based on this analysis, the cryptographically secure pseudo-random number generator (CSPRNG) seems more suitable for cryptography. Its output appears to be more random and less susceptible to patterns or predictability, which is crucial for cryptographic applications where the security of the generated random data is paramount.

1) Based on Entropy

The provided two histogram shows the distribution of PRNG Output and distribution of CSPRNG Output.





Python Code Snippet:
https://colab.research.google.com/drive/1fHjy5LrArY0wfhbCe1mesdTXwngOo16#scrollTo=csymE5ftgI9k&line=30&uniqifier=1
(Note: The appearance of the graphs may vary due to the randomness of the data each time the code is executed, but the underlying properties remain consistent.)

a) PRNG Data Histogram: The PRNG data is generated using a linear congruential generator (LCG), which can have certain patterns and repetitions in its output. In the histogram:

- We see clumps or groups of bars indicating that certain values are more likely to occur than others.
- There are visible gaps or areas with no bars, suggesting that some numbers are not being generated at all.
- The histogram shows a relatively simple distribution pattern, lacking the complex and uniform randomness expected in secure random data.

b) CSPRNG Data Histogram: The CSPRNG data is generated using os.urandom(), which is designed to provide high-quality random bytes suitable for cryptographic purposes. In the histogram:

- The bars are approximately evenly distributed across the range, indicating that each value is equally likely to occur.
- There aren't any noticeable patterns, clumps, or gaps in the histogram.
- The distribution looks relatively flat, with no significant peaks or valleys.

2) Interpretation

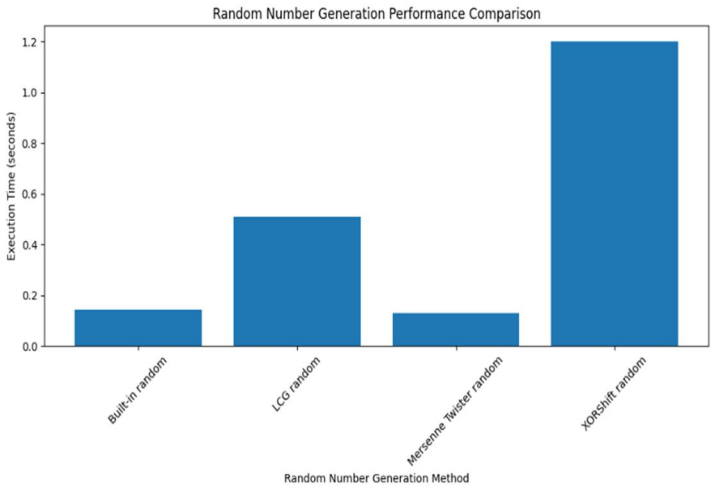
When comparing the histograms of the two datasets:

- The PRNG data histogram might show signs of non-randomness, such as clumping or patterns.
- The CSPRNG data histogram should appear more uniformly distributed and without any discernible patterns.

Histograms can provide visual insights into the distribution and randomness of data. In cryptography, the goal is to have data that appears indistinguishable from true random data, and a histogram is one way to analyse this property. In this case, the CSPRNG data histogram should be more suitable for cryptographic applications due to its more random appearance and better distribution characteristics.

D. Comparison of Random Number Generation Performance Analysis

Here's comparison of the performance of four different random number generation methods by measuring how long it takes each method to generate a specified number of random numbers. The purpose of this comparison is to evaluate the efficiency of these methods in terms of execution time for generating random numbers.



Python Code Snippet: https://colab.research.google.com/drive/1-fHjy5LrArY0wfhbCe1mesdTXwngOo16#scrollTo=7_tMvWPXeE1N&line=42&uniqifier=1

(Note: The appearance of the graphs may vary due to the randomness of the data each time the code is executed, but the underlying properties remain consistent.)

The above Bar Graph shows the execution time of each method for generating 1,000,000 random numbers. Here's what it means:

1) *Built-in Random Generation Time:* 0.141330 Seconds: The built-in random number generator in Python took approximately 0.141 seconds to generate 1,000,000 random numbers.
2) *LCG Random Generation Time:* 0.511048 Seconds: The LCG random number generator implementation took approximately 0.511 seconds to generate 1,000,000 random numbers.
3) *Mersenne Twister Random Generation Time:* 0.130704 Seconds: The built-in Mersenne Twister random number generator in Python took approximately 0.130 seconds to generate 1,000,000 random numbers.
4) *XORShift Random Generation Time:* 1.202236 seconds: The XORShift random number generator implementation took approximately 1.202 seconds to generate 1,000,000 random numbers.

Interpretation of the Graph

- The built-in Mersenne Twister random number generator is the fastest among the tested methods.
- The LCG random number generator is relatively slower compared to the built-in Mersenne Twister generator.
- The XORShift random number generator is the slowest among the tested methods.

V. FULLY HOMOMORPHIC ENCRYPTION (FHE)

A. Introduction

Fully homomorphic encryption is a type of encryption scheme that allows you to perform arbitrary computations on encrypted data [8]. This technique enables computations to be performed on encrypted data without decryption. This technique solves the tough problem of doing important calculations with secret information while keeping that info private and secure.

Example is illustrated below:

Ram wants to calculate the sum of two numbers, 6 and 3, without revealing the individual numbers or the result.

1) *Encryption:* Alice uses FHE to encrypt the numbers: 6 becomes Enc (6) and 3 becomes Enc (3).
2) *Homomorphic Addition:* She performs a homomorphic addition on the encrypted numbers: Enc (6) + Enc (3) = Enc (9).
3) *Decryption:* Alice uses her secret key to decrypt Enc (9), revealing the plaintext sum: 9.

Through FHE, Ram was able to compute the sum of 6 and 3 on encrypted data without needing to decrypt the values, maintaining privacy throughout the process.

B. Mathematical properties of Fully Homomorphic Encryption

1) *Homomorphism Property:* The homomorphism property is a concept in mathematics, especially in algebra. It relates two algebraic structures by maintaining their operations. Given two structures (A, *) and (B, ·), a function φ is a homomorphism if it keeps the structure intact: φ (a * b) = φ(a) · φ(b) for elements a and b in A. This property helps us understand similarities and connections between different algebraic systems.
2) *Additive Homomorphism:* An additive homomorphism preserves addition in algebraic structures. For sets (A, +) and (B, ⊕), where + and ⊕ are additions in A and B, an additive homomorphism φ: A → B satisfies φ (a + b) = φ(a) ⊕ φ(b) for elements a, b. It maintains addition relations between structures. Common in linear and abstract algebra, it shows how one system's addition corresponds to another's.
3) *Multiplicative Homomorphism:* A multiplicative homomorphism is like a translator for multiplication in math. Imagine two groups, A and B, with their own multiplication rules (* and ·). A multiplicative homomorphism, φ, is like a bridge between them. It keeps the multiplication relationships intact: φ (a * b) = φ(a) · φ(b). In other words, it makes sure that if you multiply two things in group A and then use φ to translate, it's the same as multiplying their translations in group B. It helps us understand how multiplication in one group corresponds to another.
4) *Noise Accumulation and Bootstrapping:* Encrypting data accumulates noise due to encryption's randomness, risking computation errors. Fully Homomorphic Encryption (FHE) employs bootstrapping to address this. Bootstrapping refreshes ciphertext, reducing noise while maintaining security through complex math operations. This sustains data integrity, facilitating accurate operations despite noise.

While FHE's mathematical properties provide the foundation for its functionality, practical implementations and optimization techniques are crucial to making FHE feasible for practical use cases.

C. Practical Implementations

1) Bootstrapping

Bootstrapping is a "noise-cleaning" procedure that ensures your encrypted calculations stay accurate over many steps. It's a crucial technique to make Fully Homomorphic Encryption more practical and useful.

Imagine you're using a special type of encryption that lets you do calculations on encrypted data. However, with each calculation, a little bit of "noise" gets added to the encrypted data. After several calculations, this noise can make the results less accurate.

Bootstrapping is like giving your encrypted data a "refresh." It's as if you're cleaning up the noise that's built up during your calculations.

Here's how it works:

a) *Decrypt*: First, you use a secret key to decrypt the encrypted data.
b) *Clean Up:* Once it's decrypted, you apply some special techniques to remove the extra noise that has built up.
c) *Re-Encrypt:* After cleaning, you encrypt the data again, but this time it's cleaner – with less noise.

By doing this bootstrapping process, you're making your encrypted data accurate again. It's a bit like resetting the encryption so that you can keep doing accurate calculations on it without the noise getting in the way.

D. Parameter Selection

Selection of parameters is pivotal for realizing the practical applicability of Fully Homomorphic Encryption (FHE). It directly influences the delicate balance between security and performance. It's like finding the best balance between keeping things safe and making them work quickly. If the settings are too weak, the security might be at risk. If they're too strong, things can slow down a lot. Also, FHE needs to be fast enough for real-world tasks, but using big settings can make things complicated and slow. It's a bit like trying to balance speed and strength. If we don't choose the right settings, the encrypted calculations might not be secure or might take too long. But if we're smart about how we handle things like extra "noise" that comes up during calculations and customize settings based on what we're doing, we can make FHE work better. By finding that balance, we can make FHE a powerful tool that keeps things safe and still works well for many different things people do.

E. Optimization Techniques

Optimization techniques are essential for making Fully Homomorphic Encryption (FHE) more viable for real-world applications. They tackle the computational complexity and efficiency challenges, enhancing FHE's practicality. These techniques improve FHE operation speed, lower resource demands, and enhance its usability in real-world scenarios. By optimizing FHE, its performance is boosted, making it a more feasible and efficient solution for privacy-preserving computations.

Here's how optimization techniques make FHE more viable for real-world applications:

1) *Bootstrapping Efficiency:* Bootstrapping is essential for refreshing ciphertexts and reducing noise accumulation. Optimization techniques aim to make bootstrapping more efficient by reducing its computational cost. Faster bootstrapping enables more frequent noise reduction, making it possible to perform additional operations without compromising accuracy.
2) *Improved Computational Efficiency:* FHE operations can be computationally intensive due to the encryption and decryption processes involved. Optimization techniques streamline these operations, making them more efficient. Techniques like optimizing modular arithmetic, polynomial multiplication, and other fundamental operations reduce the time it takes to perform calculations, enabling FHE to handle real-world data in a reasonable timeframe.
3) *Noise Management:* Noise is introduced during FHE operations and can accumulate, potentially affecting computation accuracy. Optimization techniques manage noise, ensuring it doesn't grow too quickly. This is essential for maintaining the reliability of computations over multiple operations. By effectively managing noise, optimization techniques help prevent accuracy degradation and enable meaningful results.
4) *Hybrid Approaches:* Hybrid encryption combines FHE with other encryption techniques that might be more efficient for certain operations. Optimization techniques enable the smart integration of FHE with these techniques, allowing FHE to be used selectively where its benefits are most needed. This minimizes computational overhead while still maintaining strong security.
5) *Batch Processing:* Batching involves processing multiple pieces of data within a single ciphertext, reducing the overhead of encryption and decryption for individual data points. This technique is particularly useful for tasks like matrix operations and data analytics, where processing data in batches significantly improves efficiency by minimizing the number of FHE operations required

By addressing challenges and showcasing real-world examples, it demonstrates how FHE can be a valuable tool for ensuring privacy and security while also enabling innovative data usage across various industries.

VI. CRYPTOGRAPHIC SECRET SHARING SCHEMES

A. Introduction

Cryptographic secret sharing schemes are important cryptographic methods used to share secret information securely among many people. These methods make sure that a secret can only be put together again when a certain least number of participants work together and combine their parts.

Imagine you're a software developer who has created a revolutionary algorithm that enhances data encryption. To prevent any single organization from gaining full access to this powerful algorithm, you decide to employ a secret sharing scheme.

You divide the algorithm into six components and distribute them among three leading tech companies: Company X, Company Y, and Company Z.

C_____y X: _m_____e
C_____y Y: __p____t
C_____y Z: ___o____t

Each company holds only a portion of the algorithm, and its functionality remains incomplete without the collaboration of all three companies. This approach ensures that no single entity can exploit or manipulate the algorithm without the combined efforts of the other two, maintaining the security and integrity of your innovative encryption solution. The hierarchical distribution of the secret reflects the varying degrees of trust and expertise among the tech companies involved.

B. Types of Cryptographic Secret Sharing Schemes

1) Shamir's Secret Sharing

Shamir's Secret Sharing is an algorithm in cryptography created by Adi Shamir. The main aim of this algorithm is to divide secrets that needs to be encrypted into various unique parts [9].

Adi Shamir devised this scheme in 1979 as a way to distribute shares of a secret through polynomial interpolation. A distinctive feature is that the original secret can only be reconstructed when a certain minimum number of participants combine their shares. This minimum threshold acts as a safeguard against any single participant gaining access to the secret. The polynomial interpolation technique ensures that even if a subset of participants collaborates, they cannot extract the secret without the participation of enough authorized individuals.

Shamir's Secret Sharing Scheme Implementation in Python - Code Snippet:
https://colab.research.google.com/drive/1WFpGOW_dh7AFYru8mjqtHLD-AxgCNenT#scrollTo=PpOhBpvdEtXA&line=1&uniqifier=1

2) Threshold Cryptography:

This category builds upon the principles of secret sharing but extends its scope.

Imagine a valuable secret, like hidden treasure. You want it safe from one person's access, but usable when a certain number of friends unite. Threshold Cryptography manages this. Instead of one key, split it into parts for friends. Like puzzle pieces, a set number of friends (say, 3 out of 5) combine their pieces to unlock. This concept goes further. Friends team up for tasks like opening locked boxes (decryption) or signing papers (digital signing). Each adds a puzzle piece; with enough teamwork, the task completes. Even if one's unreliable or hacked, security stays. Threshold Cryptography is security teamwork. Key tasks need friend collaboration, preventing one friend from causing trouble. Distributed teamwork boosts safety and trust.

Implementation of Threshold Cryptography with Shamir's Secret Sharing for Digital Signing in python – code snippet:
https://colab.research.google.com/drive/1WFpGOW_dh7AFYru8mjqtHLD-AxgCNenT#scrollTo=-NjMlWm-XMl8&line=51&uniqifier=1

C. Mathematical Foundations Associated with Secret Sharing Schemes.

1) *Polynomial Interpolation:* Polynomial interpolation serves as a fundamental concept within secret sharing schemes, with Shamir's Secret Sharing being a prominent example. This method harnesses the power of polynomials within finite fields to securely distribute sensitive information among participants. Given a set of points (x, y), where x represents the participant's index and y is their assigned share value, a polynomial of degree k-1 can be constructed such that it passes through k of these points. The secret is then the constant term of the polynomial. This mathematical property ensures that any subset of at least k participants can reconstruct the polynomial and thereby the secret.
2) *Modular Arithmetic:* Secret sharing schemes use finite fields and modular arithmetic to work within a defined range of numbers. Finite fields offer a limited set of numbers with clear rules for adding and multiplying. This setup prevents numbers from becoming too large, maintaining accuracy and consistency in calculations. It's like having a mathematical boundary that keeps everything in check.
3) *Lagrange Interpolation:* Lagrange Interpolation is a type of polynomial interpolation in secret sharing. It forms the polynomial directly from given points. Each participant's share corresponds to a Lagrange basis polynomial. Summing these basis polynomials create secret polynomials. This guarantees each participant's input aligns with their share, preserving the right balance in the secret.

Secret sharing schemes play a vital role in ensuring the confidentiality and integrity of sensitive information in various domains, such as cryptography, data protection, and secure multi-party computation.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

D. Key Security Properties of Secret Sharing Schemes

1) *Adaptive Security:* Adaptive security ensures secret sharing schemes can handle changing participant groups without sacrificing confidentiality. Certain schemes can adjust to participants joining or leaving while maintaining security. This is vital for scenarios where participant dynamics evolve.
2) *Threshold & Access Structure:* Threshold and access structure choices impact scheme security and flexibility. A higher threshold enhances security by requiring more participants to reconstruct the secret yet may reduce availability. The access structure determines conditions for subsets to reconstruct. Balancing security and usability require careful access structure design.
3) *Robustness & Unconditional Security:* Robustness means resisting adversarial actions and sharing corruptions. A robust scheme prevents unauthorized reconstruction by colluding participants unless the threshold is met. Unconditional security, often tied to information-theoretic security, offers protection without relying on computational assumptions.
4) *Privacy & Information Theoretic Security:* Privacy refers to the security of individual participants' shares. An ideal secret sharing scheme should ensure that an adversary with partial information about some shares cannot infer any information about the secret. This property can be quantified using measures such as Shannon entropy and mutual information. Higher entropy and lower mutual information indicate stronger privacy.

VII. CHINESE REMAINDER THEOREM

A. Introduction

It's often referred to as the Sun Zhu Theorem, named after the Chinese mathematician who developed it [10]. This theorem informs us that for a given set of congruences, there's a single, distinct solution within a specific modulus. This theorem is quite useful in solving systems of congruences, and it helps us determine this unique solution.

B. Theory and Formulation

Given a system of congruences:

$x \equiv a1 \pmod{m1}$

$x \equiv a2 \pmod{m2}$

⋮

$x \equiv an \pmod{mn}$

where a1,a2,…,ana1,a2,…,an are the remainders and m1,m2,…,mn are pairwise coprime moduli, there exists a unique solution for x modulo $M=m1 \cdot m2 \cdot \ldots \cdot mn$. This solution can be calculated using the Chinese Remainder Theorem formula:

$x \equiv (a1 \cdot M1 \cdot y1)+(a2 \cdot M2 \cdot y2)+\ldots+(an \cdot Mn \cdot yn) \pmod{M}$

where $Mi=M/mi$ and yi is the modular inverse of Mi modulo mi for each i.

Python code snippet for implementation of the Chinese Remainder Theorem (CRT) algorithm:
https://colab.research.google.com/drive/120VeJOLJj6p3gEbneCkKyw--N-ReT_nP#scrollTo=h-ysuah0ZLjK&line=27&uniqifier=1

C. Applications in Modern Cryptographic Protocols

1) *Speeding up Modular Arithmetic*

When dealing with large numbers or complex calculations involving modular arithmetic, the computations can become computationally demanding and time-consuming. This is where the Chinese Remainder Theorem (CRT) comes into play.

The Chinese Remainder Theorem provides a technique to break down a modular arithmetic problem into smaller components. It's especially useful when you need to perform calculations modulo different prime numbers or powers of prime numbers. The theorem states that if you have a system of simultaneous modular congruences (equations), you can solve for the common solution using the individual solutions for each congruence.

2) *RSA Encryption Acceleration*

RSA encryption is a popular method in cryptography. It uses big numbers that are hard to break into primes. It has a public key to encrypt and a private key to decrypt. It mainly uses modular math, raising a message to a power and getting the remainder when divided by a big number.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The Chinese Remainder Theorem (CRT) is mainly used to make RSA decryption faster. It's important to see how this improvement helps the entire RSA encryption method work better.

a) *RSA Decryption:* In RSA encryption, there are two main steps: encryption and decryption. Encryption involves raising the message to the recipient's public exponent and taking the result modulo the public modulus. Decryption is more demanding due to the larger key sizes. RSA decryption entails raising the ciphertext to the private exponent and then taking the result modulo the private modulus. This modular exponentiation can be slow, especially with big prime numbers and exponents.

b) *CRT-Based Enhancement:* The Chinese Remainder Theorem improves RSA decryption by capitalizing on the private modulus being the product of two primes (p and q). This enables two smaller modular exponentiations instead of a single large one, significantly boosting efficiency.

D. Applying CRT in RSA Decryption

Here's how the CRT is applied to RSA decryption:

Given the ciphertext C and the private key components (private exponent d, prime factors p and q), we compute two intermediate values:

M1 = C^d mod p: This is the result of raising the ciphertext to the power of the private exponent modulo the prime factor p.

M2 = C^d mod q: Similarly, this is the result modulo the prime factor q.

Using the CRT, we then reconstruct the final decrypted plaintext M as follows:

M = (M1 * q * q_inv) + (M2 * p * p_inv) mod N where q_inv is the modular multiplicative inverse of q modulo p, and p_inv is the modular multiplicative inverse of p modulo q.

Moreover, using the Chinese Remainder Theorem to improve RSA decryption makes the whole encryption process work better. It cuts down on computer work, makes decryption faster, and helps create a stronger and quicker system for keeping information safe and private.

VIII. INTEGRATION FACTORIZATION AND RSA

A. The RSA Public Key Cryptosystem

Prime numbers [13] play a fundamental role in encryption and cryptography, especially in asymmetric encryption algorithms like RSA (Rivest-Shamir-Adleman). The steps of the RSA algorithm involve the use of prime numbers to generate the public and private keys. Here's a overview of the RSA algorithm and its application of prime numbers:

1) RSA Algorithm

a) *Key Generation*

- Choose two large distinct prime numbers, p and q.
- Calculate their product, n = p * q. The number n is used as the modulus for both the public and private keys.
- Compute the Euler's totient function of n: φ(n) = (p – 1) * (q – 1).

b) *Public Key*

- Choose an integer e (1 < e < φ(n)) such that e and φ(n) are co-prime (i.e., gcd(e, φ(n)) = 1).
- The public key consists of the pair (e, n).

c) *Private Key*

- Calculate the modular multiplicative inverse of e modulo φ(n). Let's call this d.
- The private key consists of the pair (d, n).

d) *Encryption*

To encrypt a message M, convert it into an integer representation (usually using a reversible encoding scheme), and then apply the encryption formula:

$C \equiv M^e \pmod{n}$

C is the ciphertext.

e) Decryption

To decrypt the ciphertext C, apply the decryption formula:

M ≡ C^d (mod n)

M is the original message.

The security of RSA heavily relies on the difficulty of factoring large composite numbers into their prime factors. The keys' strength is directly related to the size and security of the chosen prime numbers (p and q). Larger prime numbers increase the difficulty of factorization and improve the security of RSA [14].

The public key (e, n) is designed for encryption and is distributed openly to anyone who wants to send encrypted messages to the owner of the private key. The private key (d, n) is kept secret and is used for decryption. The use of prime numbers in generating the keys ensures that the encryption and decryption processes are mathematically connected but computationally challenging to reverse without knowing the prime factors of n.

2) How this Actually Works?

Alice chooses two large prime numbers p, q, they will serve for her as the trapdoor. Now she computes the product N = pq, and chooses a further number e coprime to φ(N) [11]. Then she publishes N, e. Alice also computes the multiplicative inverse d of e modulo φ(N) [that is, de = φ(N)u + 1]. For her, this is easy to do: φ(N) = (p − 1)(q − 1).

If Bob wants to send the message $1 \leq m \leq N$ to Alice, then he computes c ≡ me mod N and sends it to Alice.

Then Alice simply raises c mod N to power d, obtaining

$$c^d \equiv (m^e)^d \equiv m^{ed} \equiv m^{\varphi(N)u+1} \equiv m.(m^{\varphi(N)})^u \equiv m \bmod N,$$

at least when gcd(m, N) = 1. For a randomly chosen message $1 \leq m \leq N$

$$Pr(gcd(m,N) = 1 = \frac{\varphi(N)}{N} = \frac{N - p - q + 1}{N} > 1 - \frac{1}{p} - \frac{1}{q}$$

which is very close to 1, if p, q are large.

Eve's problem is the following: she knows only N and e. If she could factorize N to pq, then it would immediately give φ(N), leading to d fast. But there is no known method to factorize large numbers fast.

We illustrate [12] the RSA public key cryptosystem with a small numerical example. Of course, this example is not secure, since the numbers are so small that it would be easy for Eve to factor the modulus N. Secure implementations of RSA use moduli N with hundreds of digits.

3) RSA Key Creation

- Bob chooses two secret primes p = 1223 and q = 1987. Bob computes his public modulus
  N = p · q = 1223 · 1987 = 2430101
- Bob chooses a public encryption exponent e = 948047 with the property that
  gcd (e,(p − 1)(q − 1)) = gcd(948047, 2426892) = 1.

4) RSA Encryption

- Alice converts her plaintext into an integer
  m = 1070777 satisfying 1 ≤ m < N.
- Alice uses Bob's public key (N,e) = (2430101, 948047) to compute
  $c \equiv m^e \ (mod\ N)$, $c \equiv 1070777^{948047} \equiv 1473513 \ (mod\ 2430101)$
- Bob knows (p − 1)(q − 1) = 1222 · 1986 = 2426892, so he can solve
  ed ≡ 1 (mod (p − 1)(q − 1)), 948047 · d ≡ 1 (mod 2426892),
- Bob takes the ciphertext c = 1473513 and computes
  $c^d \ (mod\ N)$, $1473513^{1051235} \equiv 1070777 \ (mod\ 2430101)$

The value that he computes is Alice's message m = 1070777.

B. Primality Testing

To implement RSA [11], one needs large prime numbers, but how can we get them? To answer this question, we first cite the prime number theorem:

$$\lim_{n\to\infty} \frac{\pi(x)}{x/\log x} = 1,$$

where π(x) stands for the number of primes not exceeding x. Informally, we may say that up to x, there are approximately x/ log x prime numbers. Reformulating this, we may say that if we pick a random large integer n, then we expect a prime in the set

$$\{n, n+1, \ldots, n+2[\log n]\}.$$

So, we choose some n, and then we take the integers n, n + 1, . . . until we bump into a prime. The number of steps we have to take is estimated from above by the gap of consecutive primes. The prime number theorem suggests that this is something like log n on average. However, it is known that the prime gap around X can be larger than any constant multiple of log X (it can be bigger than 100 log X, say). Conjecturally, the gap is always smaller than a constant multiple of log2 X and the truth of this would suffice for a polynomial prime-generating algorithm. But this is just one part of the problem. Okay, assume there is a prime between n and n + 100 log2 n. How will we recognize it? Given a large number, how fast can we decide if it is prime or not? Of course, we can try any number up to √ n and if none of them divides n (except for 1 of course), then n is a prime. If n is large, this is awfully slow, so we need a better algorithm.

IX. CONCLUSION

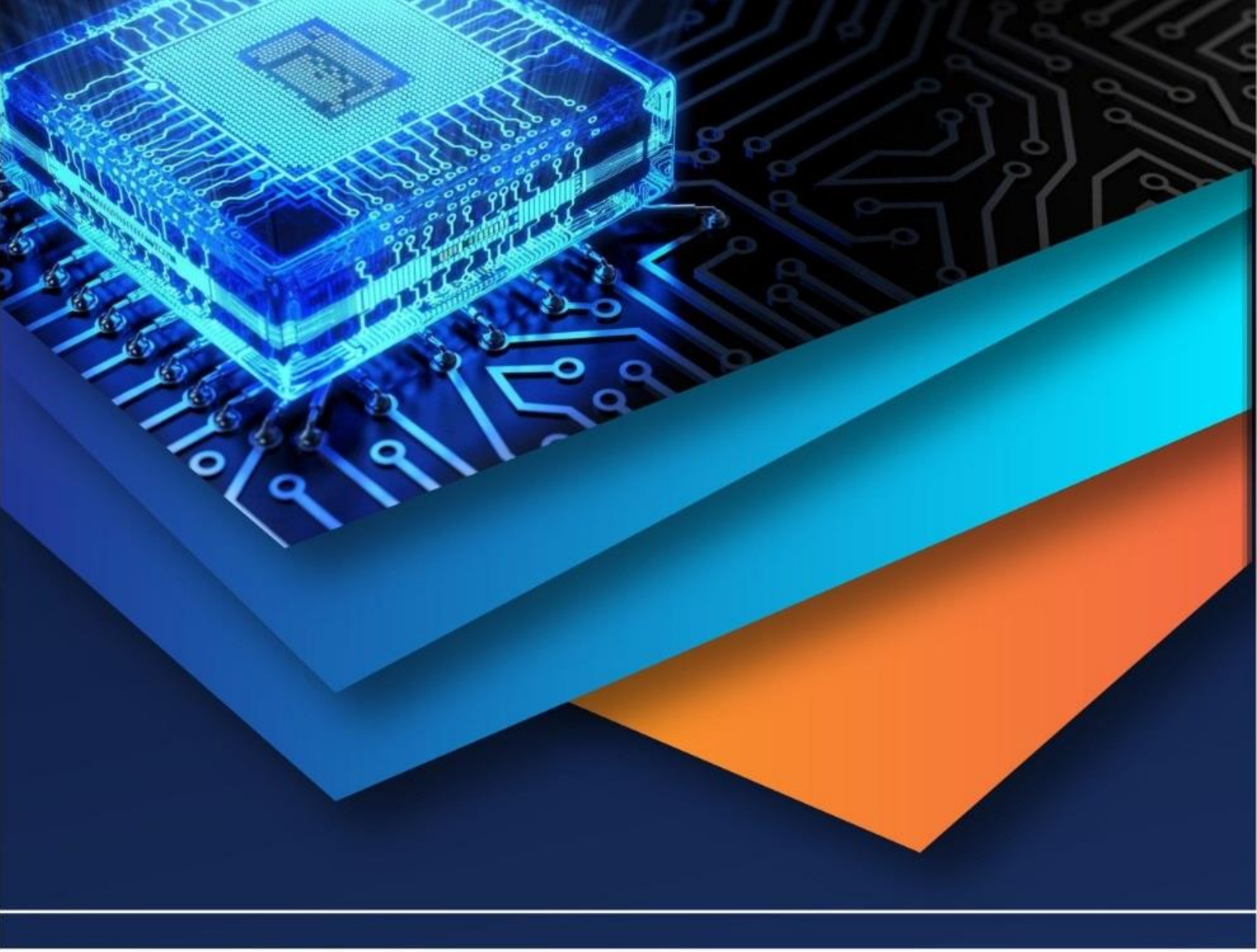
The current technological advancement generates a lot of data which is sensitive and more associated with an individual's information like Personal Identifiable Information (PII) or geolocation, etc. Securing this information is very important as it can be harmful if exposed to public networks. Cryptography is a means of securing this information and ensuring that only intended recipients can make use of it.

This paper covers the detailed overview of cryptography. In this paper, a comprehensive introduction to cryptography, mathematical foundations behind it, the different cryptographic algorithms and their comparative analysis with visual representations in graphs, necessary python code snippets embedded for the readers to generate results themselves are included.

REFERENCES

[1] Elliptic Curve Cryptography for Beginners. Mattrickard. https://matt-rickard.com/elliptic-curve-cryptography?fbclid=IwAR23d4HsavCaoh0Xj3JhxYhMsl4NDY6N8R7CPqJr9hdTGiCe2uXozGU6oXg
[2] Zero-knowledge proofs, explained. Shiraz Jagati. Retrieved Jul 31, 2023. https://cointelegraph.com/explained/zero-knowledge-proofs-explained?fbclid=IwAR2tMQ_abCFj3ESZPkckzfDngFGdst62fR2o5pqKW1t8rnd7vH6YcOs6S0g
[3] Barker, Elaine; Barker, William; Burr, William; Polk, William; Smid, Miles (July 2012). "Recommendation for Key Management" (PDF). NIST Special Publication 800-57. NIST. doi:10.6028/NIST.SP.800-57p1r3. Retrieved 19 August 2013.
[4] Pseudorandom number generators. Khan Academy. Retrieved 2016-01-11. https://www.khanacademy.org/computing/computer-science/cryptography/crypt/v/random-vs-pseudorandom-number-generators
[5] Huang, Andrew (2003). Hacking the Xbox: An Introduction to Reverse Engineering. No Starch Press Series. No Starch Press. p. 111. ISBN 9781593270292. Retrieved 2013-10-24. [...] the keystream generator [...] can be thought of as a cryptographic pseudo-random number generator (CPRNG).
[6] Dufour, Cédric. "How to ensure entropy and proper random numbers generation in virtual machines". Exoscale.
[7] ^ "/dev/random Is More Like /dev/urandom With Linux 5.6 - Phoronix". www.phoronix.com.
[8] An Intro to Fully Homomorphic Encryption for Engineers. Ravital. Retrieved Aug 30, 2021. https://blog.sunscreen.tech/an-intro-to-fully-homomorphic-encryption-for-engineers/
[9] https://www.geeksforgeeks.org/shamirs-secret-sharing-algorithm-cryptography/
[10] https://michael-khalfin.github.io/michael-khalfin-cv/Chinese%20Remainder%20Theorem%20and%20Cryptography.pdf
[11] Introduction to mathematical cryptography by P´eter Maga, Budapest Semesters in Mathematics
[12] An Introduction to Mathematical Cryptography; Jeffery Hoffstein, Jill Pipher, Joseph H. Silverman
[13] A Gentle Introduction to Number Theory and Cryptography, Notes For The Project Grad 2009, Luis Finotti
[14] A Review of Prime Numbers, Squaring Prime Pattern, Different Types of Primes and Prime Factorization Analysis, Prabhat Mahato, Aayush Shah, IJRASET, Volume – July 2023 III
[15] https://secuxtech.medium.com/history-of-cryptography-and-its-applications-b1aae0de93b0
[16] https://www.geeksforgeeks.org/cryptography-and-its-types/

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)