INFORMATION SECURITY ANALYSIS AND AUDIT

CODE: BCSE353E

SLOT: TB1

DATE: 14.07.2023

DIGITAL ASSIGNMENT – 3

PAPER ON SIDE CHANNEL ANALYSIS ATTACK

Madhavan Raman (21BCE1449)
Narasimhan H (21BCE1617)
Bala Adithiyan (21BCE1752)

Side Channel Analysis & Attack: Prevention using Symmetric key Cryptography

PROBLEM STATEMENT

This research article presents a proactive strategy to counter side-channel attack mechanisms utilized by hackers. These attacks are particularly prevalent as they represent the sole means to decrypt messages encrypted using the AES algorithm. The challenge with AES lies in its diverse complexity, making it susceptible to side-channel attacks.

ABSTRACT

Cryptography plays a vital role in securing data through encryption and decryption techniques. Encryption and decryption can be achieved using private and public key algorithms. The exchange of keys between parties can be facilitated by the Diffie-Hellman Algorithm, which is a feature of public key cryptography. However, private key (AES) cryptography is more vulnerable to penetration due to the use of identical keys at both ends. This susceptibility can be exploited through a technique known as Side-Channel Analysis. To counteract this attack, a novel algorithm is proposed that enhances data encryption by facilitating the exchange of private keys while ensuring key secrecy.

INTRODUCTION

The phenomenon of Side-Channel Attacks has emerged as a significant research area within applied cryptography. It has gained considerable momentum since the mid-1990s and is based on the fundamental concept of capturing unintended information leakage during the operation of a cryptographic system. This unintended leakage can be exploited to extract the encryption key with relatively low effort.

The importance of addressing side-channel attacks lies in the fact that developers of secure products must safeguard them against all potential attack vectors. In the case of side-channel attacks, the focus is not on questioning the mathematical security of cryptographic algorithms themselves, but rather on the vulnerability of their implementations.

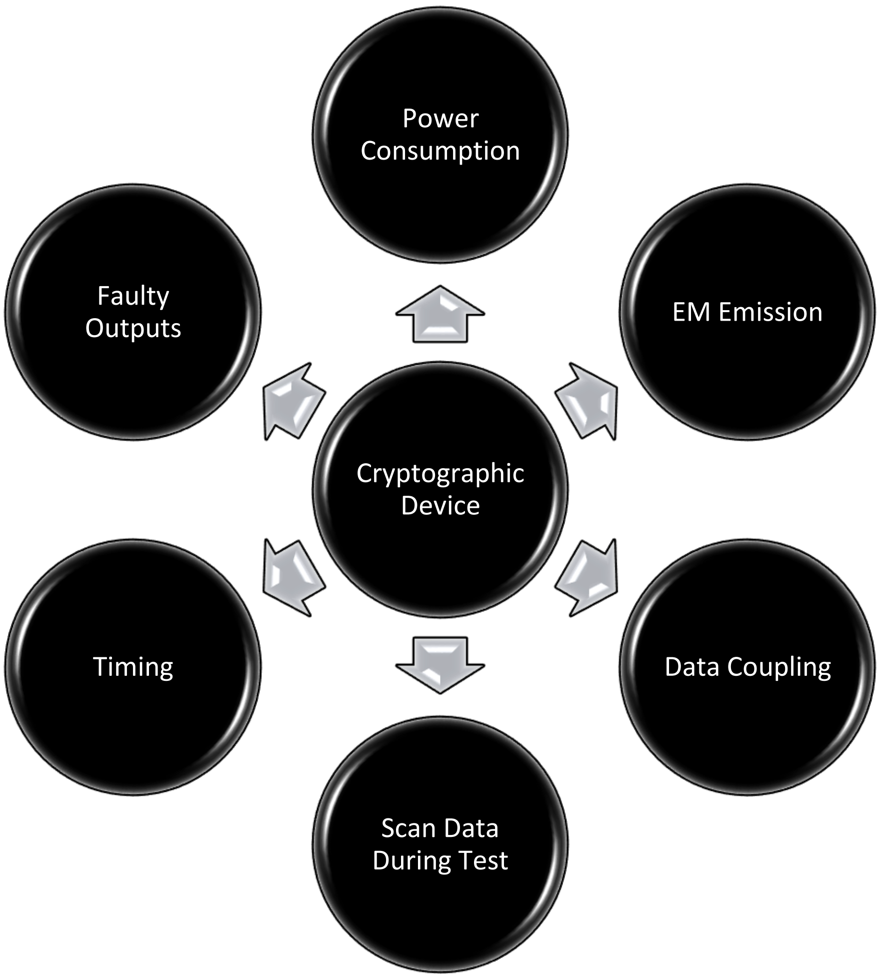


Fig-1: Typical Side – Channel Analysis

TIMING ANALYSIS

Timing analysis is the first published side-channel attack, introduced by Paul Kocher in 1995. This attack methodology focuses on breaking a cryptosystem by analysing the execution time of the cryptographic operation.

In this attack, the adversary aims to exploit the fact that the computation time for a private key operation is somehow dependent on the key being used. This is especially true for asymmetric key algorithms. One such example is the square and add exponentiation method.

By carefully observing the timing patterns and variances during the execution of cryptographic operations, an attacker can gain insights into the secret key, thereby compromising the security of the system.

$$\text{if } n == 1 \text{ then } Power(x,n) = x$$

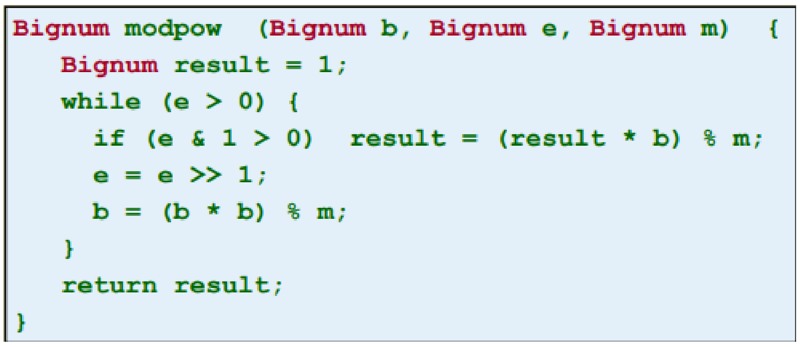
$$\text{else if } n\%2 == 0 \text{ then } Power(x,n) = Power(x^2, \frac{n}{2})$$

$$\text{else } Power(x,n) = x * Power(x^2, \frac{n-1}{2})$$

The algorithm mentioned in the context requires (n-1) multiplications for its simple implementation. However, the proposed algorithm significantly reduces the number of multiplications to O(log2n), resulting in a more efficient approach.

$$x^{13} = x^{1101} = x^{1*(2^3)+1*(2^2)+1*(2^0)} = x^8 * x^4 * x = (x^4)^2 * (x^2)^2 * x = ((x^2 * x)^2)^2 * x$$

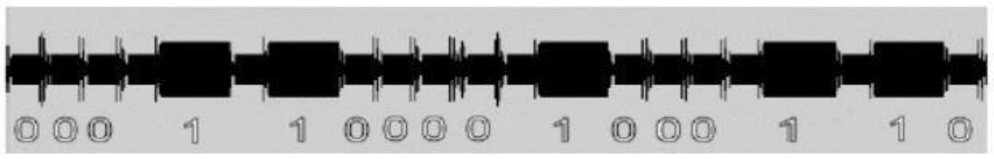
As observed, the algorithm in question presents a notable improvement by reducing the number of required multiplications to just 4, compared to the original implementation that would have necessitated 12 multiplications. This reduction in the number of multiplications signifies a significant enhancement in efficiency and computational complexity.

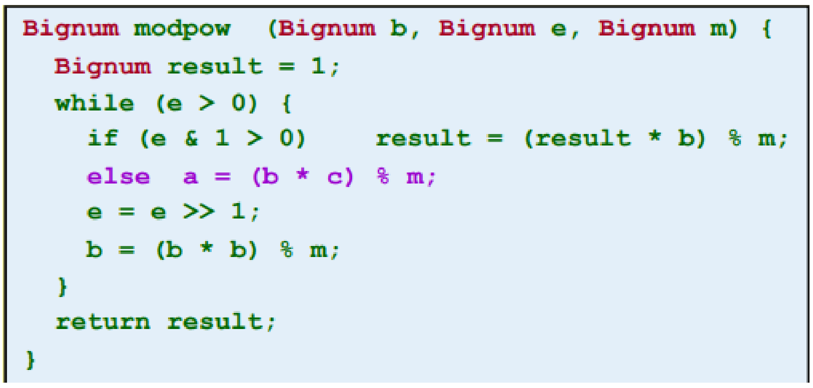


```
Bignum modpow  (Bignum b, Bignum e, Bignum m)  {
   Bignum result = 1;
   while (e > 0) {
     if (e & 1 > 0)  result = (result * b) % m;
     e = e >> 1;
     b = (b * b) % m;
   }
   return result;
}
```

Fig-2: Asymmetric algorithm's Pseudocode to compute power with less computation

Timing Analysis for Asymmetric Algorithm

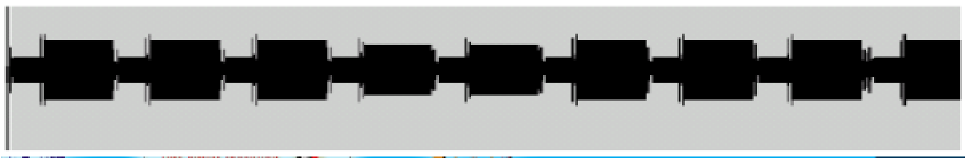




```
Bignum modpow  (Bignum b, Bignum e, Bignum m) {
  Bignum result = 1;
  while (e > 0) {
    if (e & 1 > 0)     result = (result * b) % m;
    else  a = (b * c) % m;
    e = e >> 1;
    b = (b * b) % m;
  }
  return result;
}
```

Fig-2: Symmetric algorithm's Pseudocode to compute power with less computation

Timing Analysis for Symmetric Algorithm



As the timing analysis can be performed by revealing number of 1's in key –

$\text{Time} = n * t_{square} + k * t_{mul}$

n : number of bits in the key

k: number of one bits in the key

t_{square} : time to compute square

t_{mul} : time to compute multiplication

We can make it as time – independent as a part of countermeasure

$\text{Time} = n * (t_{square} + t_{mul})$

POWER ANALYSIS

Power analysis is a highly effective form of side-channel attack, introduced by Paul Kocher et al. in 1998. This attack methodology focuses on analyzing the power consumed by a device during the execution of cryptographic operations.

By studying the power consumption patterns, an attacker can gain valuable information about the device's activities, including the processing of specific cryptographic operations. This information can then be exploited to extract sensitive key information, compromising the security of the system.

Power analysis attacks leverage the unintentional power variations that occur during different stages of the cryptographic computation, such as data-dependent power fluctuations or variations in power consumption based on specific key bits. By carefully analysing these power differentials, an attacker can uncover the cryptographic key being used, leading to a successful compromise of the system's security.

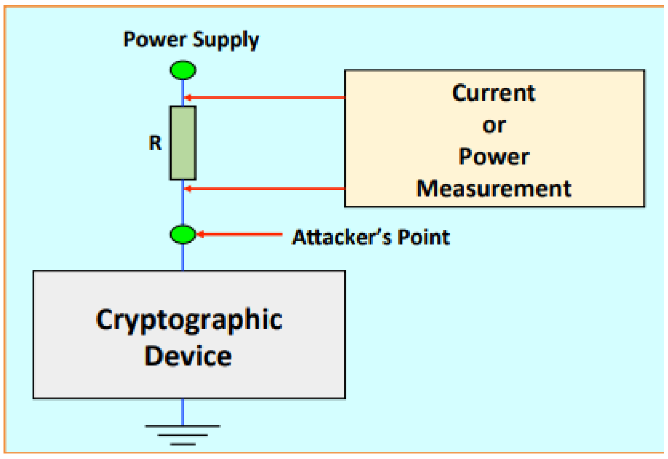


Fig-3: Data Acquisition Setup

The power analysis attack can be performed through two distinct methodologies known as Simple Power Analysis (SPA) and Differential Power Analysis (DPA).

SPA involves the direct use of power consumption measurements to extract information about the bits of the secret key. By visually examining the power consumption waveforms, an attacker can identify significant features such as rounds of DES/AES or distinguish between square and multiply operations in RSA exponentiation. They can also uncover smaller features like specific bit values.

While SPA attacks can be effective, they are relatively easier to defend against compared to more advanced techniques like DPA. Countermeasures against SPA attacks involve techniques such as power noise injection, randomizing power consumption, or introducing dummy operations to mask power differentials.

DPA, on the other hand, is a more sophisticated form of power analysis attack. It involves statistical analysis of power consumption measurements to extract key information. By analyzing power differentials across multiple computations, an attacker can uncover the secret key with higher precision.

Defending against DPA attacks typically requires more advanced countermeasures, such as implementing secure hardware designs, incorporating randomizing techniques, or employing advanced cryptographic algorithms resistant to power analysis attacks.

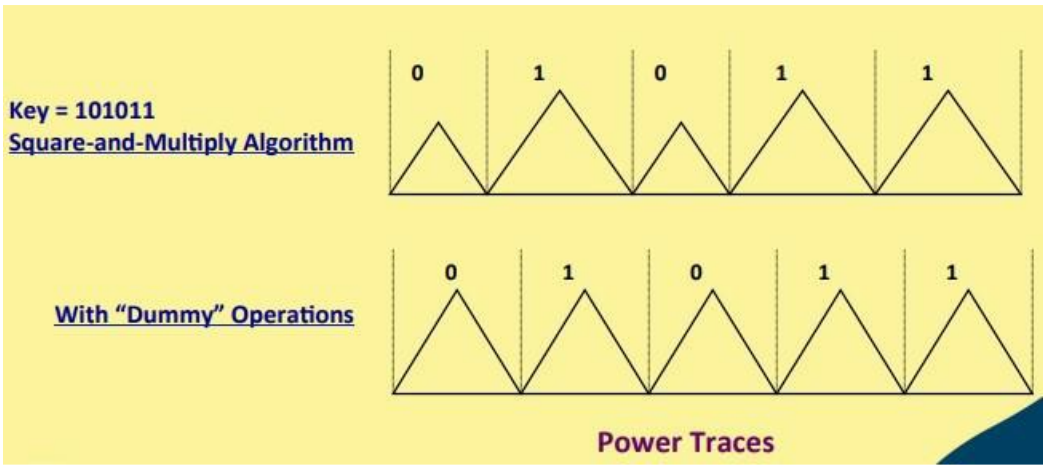


Fig-4: SPA Working Process Schematic

In Differential Power Analysis (DPA), the attack methodology becomes more complex compared to Simple Power Analysis (SPA). In DPA, the attacker partitions the data and related power consumption curves into two sets based on selected bits of the secret key. The attacker then takes the difference between the power consumption measurements of the two sets and examines these differences for potential peaks or variations.

By analyzing the power differentials, the attacker can identify patterns or correlations that reveal information about the secret key. These power differentials can provide valuable insights into the specific bits or operations being performed during the cryptographic computation.

Defending against DPA attacks requires more advanced countermeasures, such as implementing secure and tamper-resistant hardware designs, incorporating noise or randomization techniques to mask power differentials, or using cryptographic algorithms that are resistant to DPA attacks.

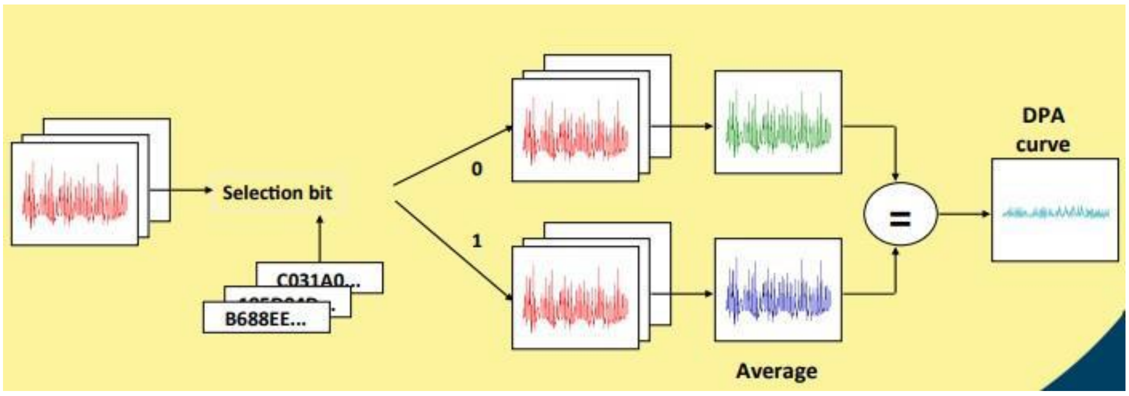


Fig-5: DPA Working Process Schematic

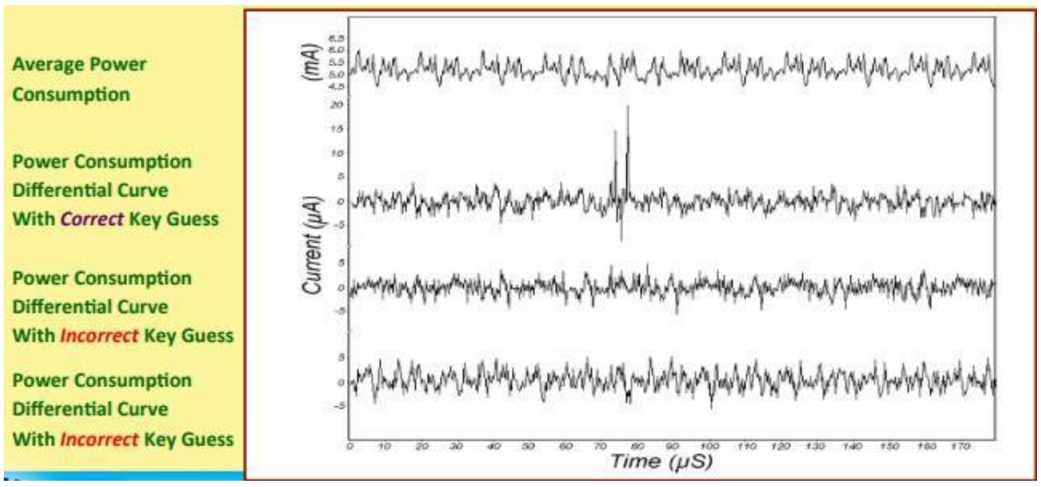


Fig-6: DPA After Observation

Indeed, there are several recommended countermeasures to mitigate the risk of power analysis attacks, including both Simple Power Analysis (SPA) and Differential Power Analysis (DPA). Here are some commonly employed techniques:

1. Adding noise generators: By introducing random noise into the power consumption, the correlation between power and sensitive data is disrupted, making it harder for attackers to extract meaningful information.

2. Generating interrupts: Introducing interrupts or perturbations during critical operations can mask power differentials, making it difficult for attackers to correlate power variations with specific cryptographic operations.

3. Internal randomization: Incorporating internal randomization techniques, such as random delays, random operations, or random data manipulations, can further obfuscate power differentials and thwart power analysis attacks.

These countermeasures aim to introduce uncertainties and disturbances into the power consumption patterns, making it more challenging for attackers to accurately correlate power variations with sensitive data or cryptographic operations.

It is important to note that implementing a combination of these countermeasures alongside secure hardware designs and robust cryptographic algorithms can enhance the overall resistance against power analysis attacks.

LITERATURE REVIEW

Title: Advanced Encryption Standard: Attacks and Current Research Trends

Abstract:

This paper aims to provide an overview of attacks that can be performed on the Advanced Encryption Standard (AES) and strategies to prevent them. AES is a symmetric key cryptography algorithm that includes block ciphers of 128-bit, 192-bit, and 256-bit lengths. The AES algorithm consists of ten rounds of encryption, each involving four stages: substitutive bytes, shift row, mix columns, and add round key. Hardware implementations of AES offer enhanced security and speed.

However, AES is vulnerable to various attacks, including side-channel attacks, known plaintext attacks (KPA), SQL injections, and cross-site scripting attacks. To mitigate these vulnerabilities, the paper discusses the use of randomization techniques as a preventive measure. For instance, splitting data into multiple parts can make it challenging for hackers to gather complete information.

Additionally, the paper explores other attack types and their corresponding countermeasures. To strengthen AES against attacks, a deep learning model is proposed. This model is trained, tested, and validated with the same MD5 key, AES-128 bit first round with S-box, and a classifier to identify malicious or attacked data. If an attack is detected, the model ignores it; otherwise, it proceeds with decryption.

In conclusion, the paper suggests that the security of AES can be enhanced by incorporating MD5 in key generation. Combining AES with Elliptic Curve Cryptography (ECC) can result in a more secure AES implementation capable of preventing common attacks. Moreover, models like XGBoost and Random Forest can be employed for predicting and preventing attacks.

Overall, this paper sheds light on the vulnerabilities of AES and offers insights into ongoing research trends, including the use of deep learning models and combining AES with other cryptographic techniques for improved security.

Title: Impact of Side Channel Attacks in Information Security

Abstract:

This paper examines the impact of side channel attacks on information security. Side channel attacks are techniques used by attackers to gain unauthorized access to sensitive data by monitoring power flow, power consumption, timing information, and electromagnetic radiation. Common types of side channel attacks include power analysis attack, simple power analysis (SPA), differential power analysis (DPA), timing attack, and acoustic attack.

Power analysis attacks are employed to gather information about the operations and parameters involved in a cryptographic algorithm. Simple power analysis aims to reveal the sequence of instructions executed during an encryption or decryption process. On the other hand, differential power analysis is a more potent technique that establishes correlations between ciphertexts to extract the secret key and make predictions.

Timing attacks exploit variations in execution time to deduce the secret key by observing different instruction execution durations. The paper discusses each type of side channel attack in detail, highlighting their characteristics and potential risks.

Countermeasures against side channel attacks are also explored. These include the duplication method, random code injection to counter power analysis attacks, techniques to prevent timing attacks, and measures to prevent fault attacks. Each countermeasure is described extensively, offering insights into their effectiveness in mitigating side channel vulnerabilities.

The paper concludes by emphasizing the need for future research on evolving side channel attacks targeting systems manufactured by various vendors. Understanding and addressing these emerging threats will be crucial for ensuring robust information security in an ever-evolving digital landscape.

Title: Key Management Using Combination of Diffie-Hellman Key Exchange with AES Encryption

Abstract:

This paper explores the use of a combination of Diffie-Hellman key exchange and AES encryption for secure key management. Diffie-Hellman key exchange is employed for key exchange, while AES is utilized for message encryption and decryption. However, Diffie-Hellman key exchange is susceptible to man-in-the-middle attacks and impersonation attacks.

To address these vulnerabilities, the paper proposes the inclusion of an authentication mechanism and two hash comparisons at the destination side. This approach helps eliminate replay attacks, man-in-the-middle attacks, and impersonation attacks. The working of Diffie-Hellman key exchange and AES encryption is explained in detail.

One common attack, the man-in-the-middle attack, occurs when a third party intercepts the public key and replaces it with their own, establishing a common key between the receiver and the third party. To counteract this, an enhanced Diffie-Hellman algorithm is introduced. This algorithm involves the sharing of two keys that are multiplied to derive the correct key. Since the attacker is unaware of the two keys being shared, this method mitigates the man-in-the-middle attack.

The paper further describes the six main processes involved in the application of Diffie-Hellman key exchange, including algorithm implementation, key distribution, private key storage, key usage from a database, key change, and key deletion. The three main stages, namely key generation, encryption, and decryption, are also outlined.

The avalanche effect, a measure of the effectiveness of AES, is calculated and found to have an average value of 49.9877%, indicating its strong cryptographic properties. In conclusion, this paper presents security methods for key management in Diffie-Hellman key exchange. The Diffie-Hellman protocol proves to be well-suited for scenarios with a large number of users on the cloud, where key management becomes challenging. The proposed scheme eliminates the overhead of key computation and management, enhancing security in key exchange processes.

Title: Design and Implementation of Advanced Encryption Standard (AES)

Abstract:

This research journal focuses on the design and implementation of the Advanced Encryption Standard (AES) algorithm. The importance of securing confidential data, such as corporate secrets, classified government information, and personal data to prevent identity theft, is emphasized. The AES algorithm can be implemented using both software and hardware.

Introduction:

In the digital world, there is a significant need for secure data transmission and protection against unauthorized access. Previous encryption techniques like DES and DES3 had limitations in terms of security issues and slower speed, which led to the development of the AES algorithm. AES offers faster encryption and decryption speeds compared to other algorithms. The implementation of AES can be done in software or hardware, with software being cost-effective and hardware offering higher security and performance speed. AES supports key sizes of 128, 192, or 256 bits, and the number of transformation rounds varies accordingly (10, 12, or 14 rounds). The transformation involves four functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey.

Literature Review:

The AES algorithm can be analyzed based on different implementation techniques:

A. Software implementation: AES can be implemented using software tools like C, HTML, Java, MATLAB, and Verilog/VHDL. It can run on various processors, but execution time varies based on the processor used. Software implementation provides lower development costs and easier parameter changes but may have slower execution times depending on the processor.

B. Software Implementation Using Intel AES-NI Instruction Set: Intel introduced a set of processor-based instructions in 2010 to enhance AES encryption/decryption without the need for lookup tables. These instructions provide high-performance encryption and decryption. Advantages include hardware support for any mode of operation and key length, significantly faster execution compared to AES implementation, but it is limited to Intel family processors.

C. FPGA Implementation: Field Programmable Gate Array (FPGA) is a reconfigurable integrated circuit. FPGA implementation aims to reduce hardware, optimize area, and simplify the controller. Pseudorandom noise sequences are used to enhance data security. FPGA implementation offers reconfigurability, lower latency, higher throughput, and better security. However, it is expensive for mass production and consumes more power compared to ASIC.

D. Implementation Using ASIC: Application-Specific Integrated Circuit (ASIC) is a customized IC for specific applications, suitable for mass production. ASIC implementation provides higher throughput and requires less power, area, and timing. However, it has higher cost per unit, complex design implementation, and limited flexibility.

Conclusion:

Each implementation technique has its advantages and disadvantages. Software implementation is cost-effective and easy to implement but lacks security and speed. FPGA implementation offers high throughput but has higher latency, power consumption, and cost. ASIC implementation provides customization and higher throughput but is expensive and inflexible. The choice of implementation technique depends on specific requirements and trade-offs.

Title: AES Image Encryption

Abstract:

This research journal focuses on image encryption and decryption using the AES algorithm. With the increasing demand for image utility in various fields, the protection of confidential images from unauthorized access during transmission has become crucial. The proposed design employs an iterative approach and introduces a secret key to enhance security.

Introduction:

In the current world, technological advancements have led to the significance of image communication and its security. Confidential images are at a high risk and need protection from unauthorized users. Security is a core area of study, and conventional encryption techniques are commonly used to safeguard multimedia files. Reliable storage and transmission of digital images are required for various digital services, making image privacy a challenging task. AES techniques are implemented on images to prevent intrusive attacks. Unlike DES, AES can be applied to images. Image encryption involves processing input images to obtain top-quality images while keeping information secret.

Literature Review:

A. Cryptography:

Cryptography focuses on techniques for secure communication in unsecure conditions. It involves constructing and analyzing protocols to prevent third parties or the public from reading confidential messages. Cryptography finds applications in military communication, digital currencies, chip-based payment cards, and electronic commerce.

B. Encryption:

Encryption involves encoding information in a manner that only authorized parties can understand. It relies on cryptographic keys agreed upon by the sender and receiver. Secure encryption uses complex keys that are unlikely to be guessed by third parties.

C. Types of Encryption:

Encryption can be classified as symmetric and asymmetric encryption. Symmetric encryption uses a single key for both encryption and decryption, while asymmetric encryption uses a private key for decryption and a public key for encryption.

D. Data Encryption Reasons:

1. Privacy

2. Security

3. Data integrity

4. Authentication

5. Regulations

E. Drawbacks:

1. Key exhaustion

2. Attribution data

3. Key management at a large scale

F. AES:

AES is a symmetric block cipher chosen by the US government to protect classified information. It can be implemented using both hardware and software.

Conclusion:

Image steganography using AES provides security against intrusion attacks. It enables faster and more secure encryption and decryption processes, ensuring security in the transmission and storage of images.

Title: An Improved Cryptographic Scheme Using AES & RSA Algorithms for Maximum Security in File Encryption and Decryption

Abstract:

This research journal focuses on a cryptographic scheme that combines AES and RSA algorithms to enhance security in file encryption and decryption. The proposed scheme aims to utilize two different algorithms and employ multiple keys for encryption and decryption. The study examines cryptographic algorithms using secondary data obtained from related journals and conference papers.

Introduction:

The widespread use of online documents and their easy transferability and modification pose security risks such as insufficient data encryption, brute force attacks, and reliance on software for encryption. This research journal primarily focuses on the AES algorithm for symmetric encryption and the RSA algorithm for asymmetric encryption. While AES has proven utility, both AES and RSA have certain limitations, such as AES not utilizing its full encryption speed and RSA requiring high security and robust key management.

Literature Review:

Cryptography combines various features to address network security concerns faced by organizations and IT professionals. Decentralization of company operations and the prevalence of computer network communication have driven concerns about network security. Cryptography technology is heavily involved in network security decisions, as it plays a critical role in protecting data. The main categories of cryptography are based on the type of security keys used for data encryption and decryption, falling under asymmetric and symmetric encryption techniques.

In asymmetric encryption, the encryption key is public, while the decryption key is kept private. RSA encryption relies on the difficulty of factoring the product of two large prime integers, known as the factoring problem. However, RSA has limitations in terms of the amount of data that can be encrypted and its slower processing speed.

Conclusion:

The RSA encryption technique has limitations in terms of its capacity and processing speed for data encryption. To overcome these limitations, an improved cryptographic scheme is proposed, leveraging AES for symmetric encryption and RSA for encrypting AES-encrypted files. By combining the strengths of both algorithms, this scheme aims to provide maximum security in file encryption and decryption processes.

Title: Analysis and Implementation of AES & RSA for Cloud

Abstract:

This research journal focuses on the analysis and implementation of AES and RSA algorithms for cloud computing. Cloud computing has become an essential service in the IT corporate world, offering various services such as infrastructure, storage, and software. With the growing demand for cloud services, ensuring high security and data confidentiality is crucial. The author proposes a hybrid model that combines AES and RSA to achieve key privacy and data confidentiality in the cloud environment.

Introduction:

Cloud computing enables unlimited storage capacity and shared memory management over the network. It consists of deployment models and service models, with access types categorized as public, private, hybrid, and community. The service models in cloud computing include infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). Cloud providers like Google Cloud and Amazon Cloud offer large storage space and computing resources, eliminating the need for local devices. Security measures such as firewalls, virtual private networks, and other security tools ensure data security for users.

Literature Review:

Advance Encryption Standard (AES):

AES utilizes a substitution and permutation process, with transformation rounds specified based on the key size. Encryption involves adding the round key and applying usual rounds and a final round. Usual rounds consist of sub bytes, shift rows, mix columns, and add round key steps, while the final round includes sub bytes, shift rows, and add round key steps. The number of usual rounds depends on the key size. AES is resistant to various attacks, including brute-force, statistical, linear, and differential attacks, making it more secure than DES.

RSA Algorithm:

RSA is a popular asymmetric or public-key cryptography algorithm that works with dual keys. The sender's public key is used for encryption, while the receiver uses the secret key for decryption. RSA is widely used in e-commerce, VPNs, and email systems. It involves encryption, decryption, and key generation processes.

Hybrid Model:

The proposed hybrid model combines RSA and AES to enhance security and reduce time complexity in cloud computing. Registered clients can securely store their data on the cloud by logging in with credentials and undergoing authentication using two factors: username-password and OTP. The hybrid model leverages the strengths of RSA and AES algorithms to ensure stronger security and data protection.

Conclusion:

In conclusion, the hybrid model incorporating AES and RSA algorithms is recommended for encryption in cloud computing. This model enhances security strength and reduces time complexity, providing a robust solution for ensuring data confidentiality and privacy in cloud environments.

Title: Data Security using Advanced Encryption Standard (AES)

Abstract:

This research article focuses on the security of data using the Advanced Encryption Standard (AES) algorithm and the need for a modified version of AES to overcome its limitations. In the current digital world, data security and privacy are significant concerns, especially while using the internet. The AES algorithm was proposed to address these concerns and provide an additional layer of protection to ensure data encryption in the digital environment.

Introduction:

AES is a widely used encryption technique known for its speed and effectiveness compared to the Data Encryption Standard (DES). AES operates on bytes, whereas DES operates on bits. The variable key length of AES makes it a preferred choice over DES, which has a fixed key length. The Key Scheduling Algorithm is used to generate round keys from the initial key, which are utilized in the encryption rounds.

Literature Review:

NIST introduced AES in 1997, and subsequent research has focused on enhancing its security. Increasing the number of rounds in AES adds more processing power, making it more challenging for hackers. Research conducted in 2017 demonstrated that AES is superior to DES and Triple DES. There have been successful implementations of AES encryption and decryption in single chips, utilizing minimal resources while achieving maximum throughput.

In 2018, a modification to the RSA algorithm was made by combining it with AES technique using a USB device, enabling secure file transfer and secure cloud framework, albeit limited to text files. The research addresses six elements of AES, including key size, mode specificity, round key storage, round unraveling, SBOX implementation, and pipelining. The modified AES algorithm involves converting the message to Caesar's ciphertext, applying the standard AES algorithm with additional rounds, decrypting it back to ciphertext, and finally obtaining the original message.

Conclusion:

The extended AES algorithm with bespoke configuration serves as the foundation of this work. As AES is copyright-free, users have the flexibility to modify the algorithm each time it is encrypted, ensuring enhanced security. AES is widely adopted by military organizations and top-secret government intelligence agencies for transmitting messages with high security. The research highlights the importance of continuous advancements and modifications in encryption algorithms to address evolving security threats and ensure robust data security.

Title: AES Algorithm for Data Sharing Encryption on Cloud Computing

Abstract:

This research article focuses on the use of the Advanced Encryption Standard (AES) algorithm for data sharing encryption in cloud computing. Cloud computing enables users to share resources, services, and data over a network. However, ensuring data security in the cloud is a significant challenge. To prevent unauthorized attacks and enhance data safety, this study proposes the use of the AES algorithm.

Introduction:

Data security is a critical concern in cloud computing, where data and information are stored and shared among users. Encryption and decryption using cryptographic algorithms are effective preventive measures in ensuring data security. Among these algorithms, AES is widely recognized as one of the most efficient and effective encryption techniques.

Literature Review:

The research methodology employed in this study includes planning, conducting a synthesis of evidence, and selecting relevant sources. The observations made during the research process highlight the importance of data encryption using AES in cloud computing. The comparison of data encryption techniques in cloud computing, data sources, systematic literature review (SLR) results, encryption algorithm evaluation, data sharing in cloud computing, and the characteristics and effectiveness of the AES algorithm were key areas explored.

Conclusion:

Based on the analysis of appropriate and accurate sources, it can be concluded that the AES algorithm is suitable for securing data sharing in cloud computing. Further research and in-depth exploration of the AES algorithm's capabilities and implementation are recommended. Utilizing the findings from the systematic literature review can provide valuable insights and suggestions for future research. This study contributes to the advancement of science and knowledge in the field of data security in cloud computing, particularly in utilizing the AES algorithm to prevent unauthorized attacks and enhance data privacy.

PROPOSED SYSTEM

The proposed system focuses on designing an algorithm that ensures secure communication and prevents side-channel attacks. It addresses the need for data security over public networks during communication. One such is Diffie Hellmann Algorithm.

The algorithm incorporates a public-private key pair for each sender and receiver. The public keys are distributed, and authentic duplicates are obtained. By using these keys, the parties can compute a shared secret, which serves as a symmetrical cipher for encryption and decryption.

In this paper, the Diffie-Hellman (D-H) key exchange concept is implemented in a client-server architecture. The client initiates the communication by sending a message, which is encrypted using a shared key derived from the client's public key and the server's private key. The message is then encoded into ciphertext. The server, using the same shared key, decodes the ciphertext to retrieve the original message. This algorithm ensures that data transmission occurs securely.

The advantages of this algorithm include secure data transmission, protection against side-channel attacks, and the utilization of a client-server architecture for efficient communication. By implementing the D-H key exchange, the algorithm establishes a secure shared key between the sender and receiver, enhancing the overall security of the system.

MODULES IN PROJECT AND DETAILED DESCRIPTION

The first module of the project involves the implementation of the Diffie-Hellman (D-H) algorithm. This algorithm is based on Elliptic Curve Cryptography (ECC) and plays a crucial role in establishing secure communication between two parties. ECC is known for its efficiency in providing strong security even with smaller key sizes.

The D-H algorithm allows two parties, typically a sender and a receiver, to securely exchange cryptographic keys over an insecure channel. It enables them to compute a shared secret key without transmitting it, making it resistant to eavesdropping attacks.

By implementing the D-H algorithm, the project ensures that the key exchange process is secure and efficient. It utilizes the properties of elliptic curves to achieve a high level of security with smaller key sizes. This module establishes a foundation for secure communication in the project and sets the stage for subsequent encryption and decryption processes.

Note: It is important to clarify that while D-H is an ECC-based algorithm, RSA is a separate encryption algorithm that is not based on ECC.

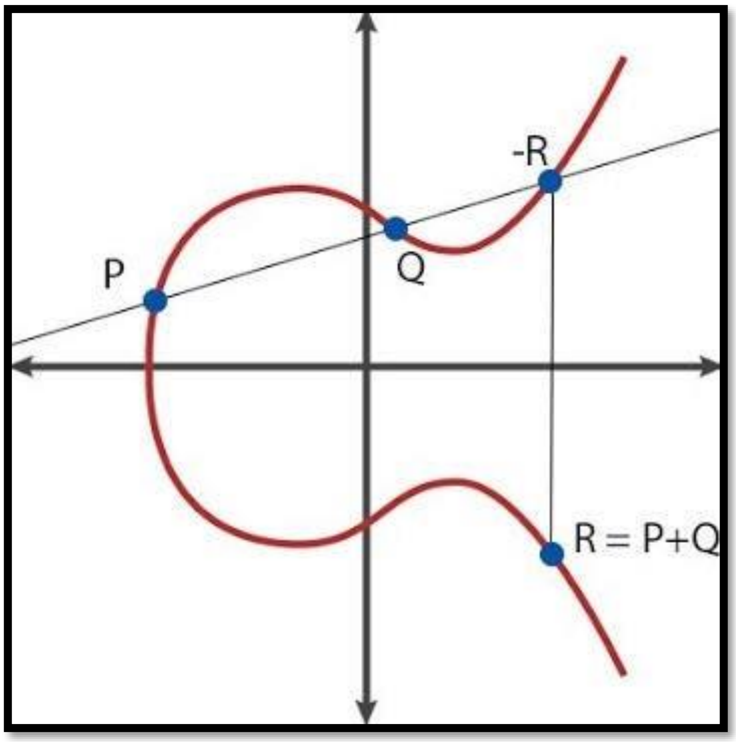


Fig-7: ECC polynomial curve for DH Algorithm

The main property of this ECC is that it has no cusps or intersections. It is represented graphically by the following polynomial equation:

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

where a, b are the coefficients of the polynomial. The algorithm is good if both the coefficients lie in the range 2-3 and the curve should be symmetric about the X-axis.

<u>Algorithm for designing D – H</u>

Let 'A' be the sender and 'B' be the receiver.

1. A and B agree on two large numbers 'g' and 'n' such that 1<g<n.
2. A chooses random x such that $X = g^x \; mod \; n$, sends X to B.
3. B chooses random y such that $Y = g^y \; mod \; n$, sends Y to A.
4. A compute $p = Y^x \; mod \; n$.
5. B compute $q = X^y \; mod \; n$.

Also, it is very important to understand that

$$p = q = g^{yx} \; mod \; n.$$

Encoding and Decoding Process (Client-Server Based)

This module focuses on the implementation of the encoding and decoding process in a client-server architecture for secure data transmission. The goal is to ensure that the transmitted data is protected and can only be understood by the intended recipient.

The process begins with the client side, where the message to be transmitted is encoded using appropriate encryption techniques. The result of this encryption process is the ciphertext, which is the encrypted version of the original message. Additionally, a shared key (referred to as 'p' in this context) is generated.

On the server side, the encrypted message and the shared key obtained from the client are received as inputs. The server then applies the decryption process using the shared key to decrypt the ciphertext and retrieve the original message.

This encoding and decoding process allows for secure communication between the client and the server, ensuring that the transmitted data remains confidential and cannot be understood by unauthorized parties.

The implementation of this module involves writing a Java program that demonstrates the working of the client and server sides during the data transmission process. Through appropriate encryption and decryption techniques, the client encodes the message, and the server decodes it using the shared key, enabling secure communication between the two parties.

Note: The exact encryption and decryption techniques used may vary depending on the specific algorithms and cryptographic protocols chosen for implementation.

ARCHITECTURAL DIAGRAM FOR SYSTEM IMPLEMENTATION

Diffie – Hellman Algorithm

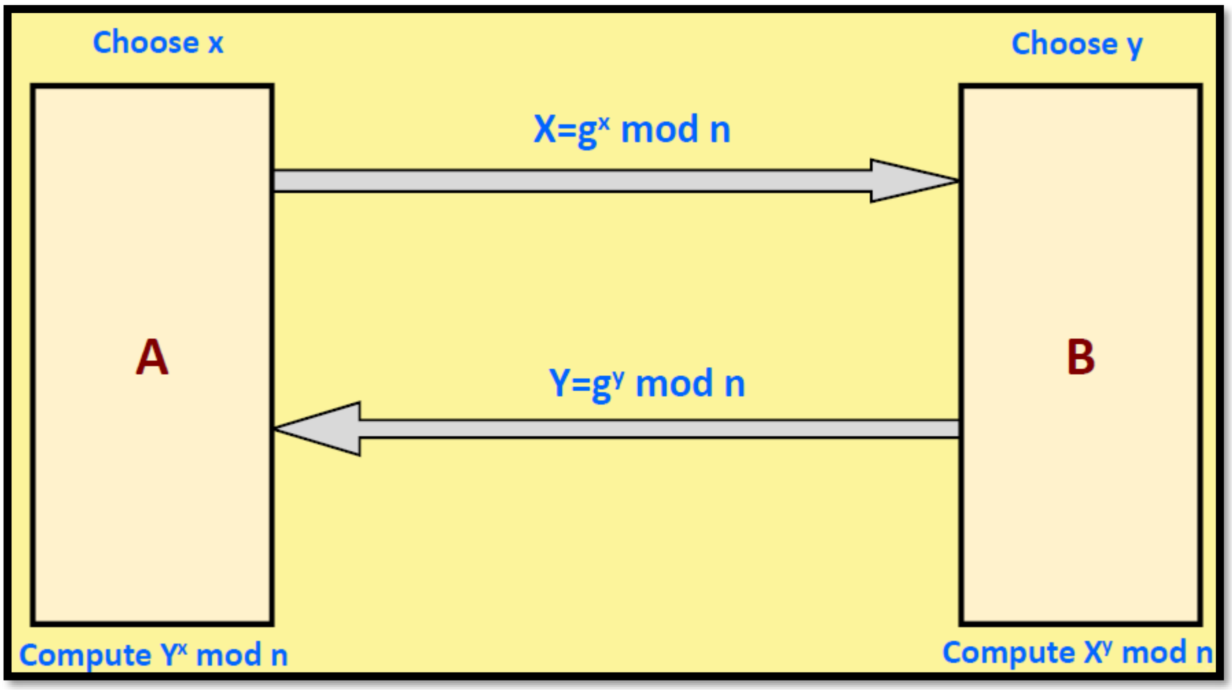


Fig-8: Architecture for DH Algorithm

Client – Server Architecture

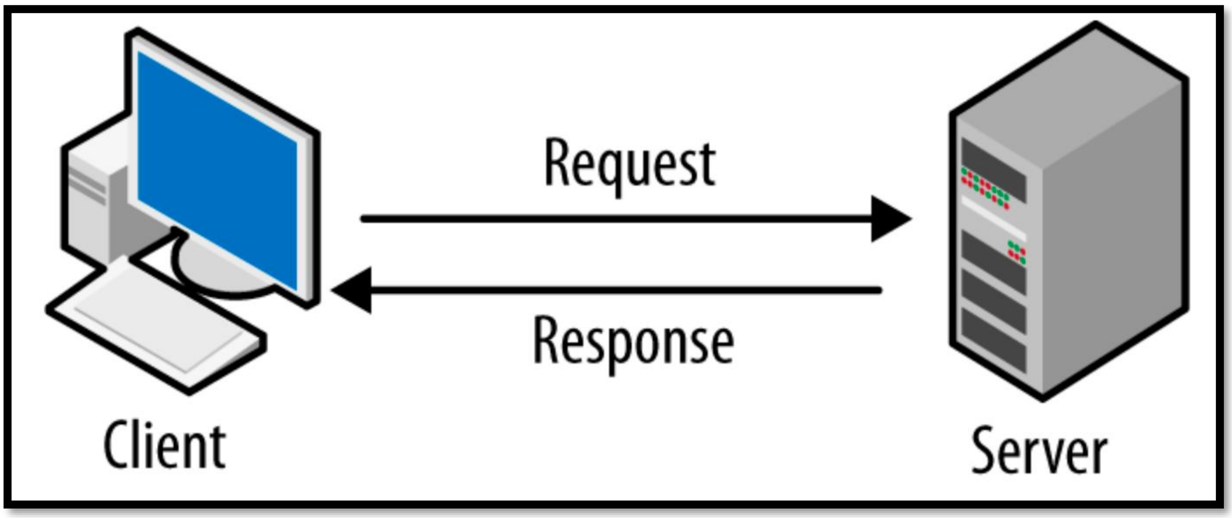


Fig-9: Architecture for Client – Server

PSEUDOCODE

Diffie – Hellmann

```
class ran {

    public static void main(String[] args) {
        int lowerBound = 1;  // Set the lower bound of the 
range
        int upperBound = 100;  // Set the upper bound of the 
range

        int G = generateRandomPrime(lowerBound, upperBound);
        Random rand = new Random();
        int n = rand.nextInt(G + 10);
        int a = rand.nextInt(100);
        int b = rand.nextInt(100);

        int x = modPow(G, a, n);
        int y = modPow(G, b, n);
        int ka = modPow(y, a, n);
        int kb = modPow(x, b, n);

        System.out.printf("%d\n%d", ka, kb);
    }

    public static int generateRandomPrime(int lowerBound, int 
upperBound) {
        Random random = new Random();
        int g;
        boolean isPrime;

        do {
            g = random.nextInt(upperBound - lowerBound + 1) + 
lowerBound;
            isPrime = true;

            // Check if the random number is prime
            for (int i = 2; i <= Math.sqrt(g); i++) {
                if (g % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        } while (!isPrime);
        return g;
    }
    public static int modPow(int base, int exponent, int
modulus) {
        int result = 1;
        base = base % modulus;

        while (exponent > 0) {
            if (exponent % 2 == 1) {
                result = (result * base) % modulus;
            }

            exponent = exponent >> 1;
            base = (base * base) % modulus;
        }

        return result;
    }
}
```

Encoding at the Client – Side

```
class HelloWorld {
    public static String decToBinary(int val){
        String s = new String();
        while(val!=0){
            int a = val%2;
            s = a+s;
            val = val/2;
        }
        //System.out.println(s);
        return s;
    }
    public static String complement(String s){
        String ans = new String();
        for(int i = 0;i<s.length();i++){
            if(Character.compare(s.charAt(i),'1')==0){
                ans = ans+"0";
            }
            else{
                ans = ans+"1";
            }
        }
        return ans;
    }
    public static String randomString(){
        String ans = new String();
        Random rand = new Random();
        for(int i = 0;i<7;i++){
            int num = rand.nextInt(2);
            ans = num + ans;
        }
        return ans;

    }
    public static String keyGenerator(){
        String ans = new String();
        Random rand = new Random();
        for(int i = 0;i<6;i++){
            int num = rand.nextInt(6);
            ans = num+ans;
        }
        return ans;

    }
    public static String columnShifter(String s,String rand){
        for(int i = 0;i<rand.length();i = i+2){

                    int ind1 = (int)rand.charAt(i)-48;
                int ind2 = (int)rand.charAt(i+1)-48;
                char temp = s.charAt(ind1);
                s =
s.substring(0,ind1)+s.charAt(ind2)+s.substring(ind1+1);
                s =
s.substring(0,ind2)+temp+s.substring(ind2+1);


        }
        return s;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Vector <String> v = new Vector<String>();
        String str = sc.nextLine();
        String s = new String();
        String rand = new String();
        String a = new String();
        String key = keyGenerator();
        for(int i = 0;i<str.length();i++){
            int val = (int)str.charAt(i);
            s = decToBinary(val);
            s = complement(s);
            rand = randomString();
            s = columnShifter(s,key);
            //System.out.println(a);
            v.add(rand);
            v.add(s);

        }
        String newstr = new String();
        int n = 0;
        int rem = 5;
        if(v.size()%5==0){
             n = v.size()/5;
        }
        else{
            n = v.size()/5;
            rem = v.size()%5;
        }

        System.out.println(newstr);
        System.out.println(v);
        System.out.println(key);

    }
}
```

Decoding at the Server – Side

```
class HelloWorld {
    public static String reverse(String s){
        String ans = new String();
        for(int i = s.length()-1;i>=0;i--){
            ans = ans+s.charAt(i);
        }
        return ans;
    }
    public static String columnShifter(String s,String rand){
        for(int i = 0;i<rand.length();i = i+2){

                 int ind1 = (int)rand.charAt(i)-48;
                int ind2 = (int)rand.charAt(i+1)-48;
                char temp = s.charAt(ind1);
                s =
s.substring(0,ind1)+s.charAt(ind2)+s.substring(ind1+1);
                s =
s.substring(0,ind2)+temp+s.substring(ind2+1);


        }
        return s;
    }
    public static String complement(String s){
        String ans = new String();
        for(int i = 0;i<s.length();i++){
            if(Character.compare(s.charAt(i),'1')==0){
                ans = ans+"0";
            }
            else{
                ans = ans+"1";
            }
        }
        return ans;
    }
    public static int binToDec(String s){
        int ans = 0;
        for(int i = 0;i<s.length();i++){
            if(Character.compare((char)s.charAt(i),(char)'1')=
=0){
                //System.out.println("working");
                ans = ans+(int)Math.pow(2,6-i);
            }
        }
        return ans;
    }
    public static void main(String[] args) {
        Vector<String> v = new Vector<String>();
        v.add("0010100");
        v.add("0001101");
        v.add("0111000");
        v.add("1000011");
        v.add("0101111");
        v.add("0001111");
        v.add("1100111");
        v.add("0000101");
        v.add("1111111");
        v.add("1001011");
        v.add("0100101");
        v.add("1000101");
        v.add("0001111");
        v.add("0001101");
        String key = "414005";
        //String s = reverse("hello");

        //System.out.println(s);
        key = reverse(key);

        for(int i = 0;i<v.size();i++){
            //System.out.println(v.get(i));
            if(i%2!=0){
                String s = new String();
                s = v.get(i);
                s = reverse(s);
                s = columnShifter(s,key);
                s = complement(s);
                int ans = binToDec(s);
                //System.out.println(s);
                System.out.println((char)ans);
            }
        }
    }
}
```

Server – Side Implementation

```
class Crypt_c{
    public static void main(String [] args){
        try{
            ServerSocket ss = new ServerSocket(8888);
            Socket s = ss.accept();
            // Initializing buffered reader to take input from
the client
            BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
            // Initializing printstream To Send a message to
the client
            PrintStream ps = new
PrintStream(s.getOutputStream());
            // Scanner is used to take input from the compiler
            Scanner sc = new Scanner(System.in);
            // Getting input message
            Vector<String> v = new Vector<String>();
            int n= 14;
            for(int i = 0;i<n;i++){
                String str;
                str = br.readLine();
                v.add(str);
            }

            for(int i = 0;i<n;i++){
                System.out.println(v.get(i));
            }

            System.out.println("got");
        }
        catch(Exception e){
            System.out.println(e);
        }
    }
}
```

Client – Side Implementation

```
class Crypt_c{
    public static void main(String [] args){
        try{
            Socket s = new Socket("localhost",8888);
            // Initializing buffered reader to take input from
the server
            BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
            // Initializing printstream To Send a message to
the server
            PrintStream ps = new
PrintStream(s.getOutputStream());
            // Scanner is used to take input from the compiler
            Scanner sc = new Scanner(System.in);
            // Input
            Vector<String> v = new Vector<String>();
            v.add("0010100");
            v.add("0111000");
            v.add("0001101");
            v.add("1000011");
            v.add("0101111");
            v.add("0001111");
            v.add("1100111");
            v.add("0000101");
            v.add("1111111");
            v.add("1001011");
            v.add("0100101");
            v.add("1000101");
            v.add("0001111");
            v.add("0001101");
            // Sending input
            for(int i = 0;i<v.size();i++){
                ps.println(v.get(i));
            }
            System.out.println("Sent");
            // String str;
            // ps.println("hello");

        }
        catch(Exception e){
            System.out.println(e);
        }
    }
}
```

SAMPLE CODE

ENCODING

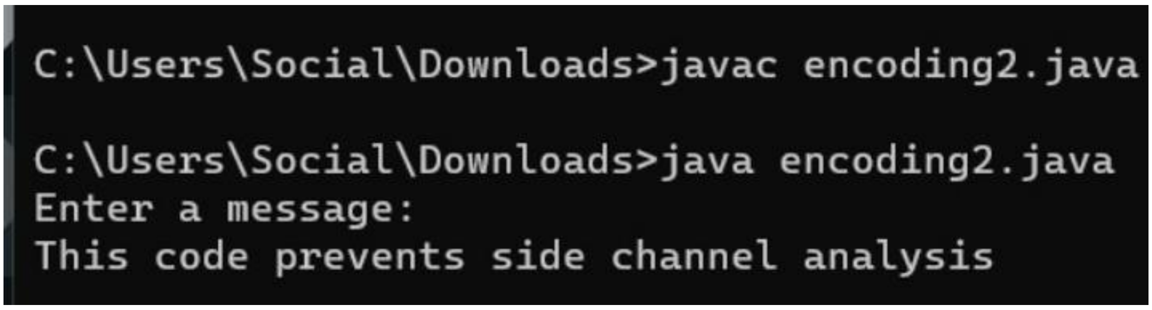
```
import java.net.*;
import java.io.*;
import java.util.*;
import java.security.SecureRandom;

class HelloWorld {
    public static int strToInt(String s) {
        int ans = 0;
        for (int i = 0; i < s.length(); i++) {
            ans = ans + ((int) s.charAt(i) - (int) '0') *
(int) Math.pow(10, s.length() - i - 1);
        }
        return ans;
    }

    public static String convert(int ka) {
        StringBuilder s = new StringBuilder();
        while (ka != 0) {
            int x = ka % 10;
            if (x > 5) {
                x = x - 5;
            }
            s.insert(0, x);
            ka = ka / 10;
        }
        return s.toString();
    }

    public static int generateRandomPrime(int lowerBound, int
upperBound) {
        SecureRandom random = new SecureRandom();
        int g;
        boolean isPrime;

        do {
            g = random.nextInt(upperBound - lowerBound + 1) +
lowerBound;
            isPrime = true;

            // Check if the random number is prime
            for (int i = 2; i <= Math.sqrt(g); i++) {
                if (g % i == 0) {
                    isPrime = false;
                    break;
                }
```

```java
            }
        } while (!isPrime);
        return g;
    }

    public static int modPow(int base, int exponent, int
modulus) {
        int result = 1;
        base = base % modulus;

        while (exponent > 0) {
            if (exponent % 2 == 1) {
                result = (result * base) % modulus;
            }

            exponent = exponent >> 1;
            base = (base * base) % modulus;
        }

        return result;
    }

    public static String decToBinary(int val) {
        StringBuilder s = new StringBuilder();
        while (val != 0) {
            int a = val % 2;
            s.insert(0, a);
            val = val / 2;
        }
        return s.toString();
    }

    public static String complement(String s) {
        StringBuilder ans = new StringBuilder();
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == '1') {
                ans.append('0');
            } else {
                ans.append('1');
            }
        }
        return ans.toString();
    }

    public static String randomString() {
        StringBuilder ans = new StringBuilder();
        SecureRandom rand = new SecureRandom();
        for (int i = 0; i < 7; i++) {
            int num = rand.nextInt(2);
            ans.insert(0, num);
        }
        return ans.toString();
    }

    public static String keyGenerator() {
        StringBuilder ans = new StringBuilder();
        SecureRandom rand = new SecureRandom();
        for (int i = 0; i < 6; i++) {
            int num = rand.nextInt(6);
            ans.insert(0, num);
        }
        return ans.toString();
    }

    public static String columnShifter(String s, String rand)
{
        StringBuilder result = new StringBuilder(s);
        for (int i = 0; i < rand.length(); i = i + 2) {
            int ind1 =
Character.getNumericValue(rand.charAt(i));
            int ind2 = Character.getNumericValue(rand.charAt(i
+ 1));
            char temp = result.charAt(ind1);
            result.setCharAt(ind1, result.charAt(ind2));
            result.setCharAt(ind2, temp);
        }
        return result.toString();
    }

    public static String reverse(String s) {
        return new StringBuilder(s).reverse().toString();
    }

    public static void main(String[] args) {
        try {
            Socket s = new Socket("localhost", 8888);
            // Initializing buffered reader to take input from
the server
            BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
            // Initializing printstream To Send a message to
the server
            PrintStream ps = new
PrintStream(s.getOutputStream());
            // Scanner is used to take input from the compiler
            Scanner sc = new Scanner(System.in);
```

```java
            int lowerBound = 100;
            int upperBound = 999;
            StringBuilder key = new StringBuilder();
            for (int i = 0; i < 2; i++) {
                int G = generateRandomPrime(lowerBound,
upperBound);
                ps.println(G);
                SecureRandom rand = new SecureRandom();
                int n = rand.nextInt(G + 10);
                ps.println(n);
                int a = rand.nextInt(100);
                int x = modPow(G, a, n);
                ps.println(x);
                String string = br.readLine();
                int y = strToInt(string);
                int ka = modPow(y, a, n) % 900 + 100;
                key.insert(0, convert(ka));
            }

            Vector<String> v = new Vector<>();
            System.out.println("Enter a message: ");
            String str = sc.nextLine();

            for (int i = 0; i < str.length(); i++) {
                int val = (int) str.charAt(i);
                String s1 = decToBinary(val);
                s1 = complement(s1);
                String rand = randomString();
                s1 = columnShifter(s1, key.toString());
                s1 = reverse(s1);
                v.add(rand);
                v.add(s1);
            }

            for (int i = 0; i < v.size(); i++) {
                ps.println(v.get(i));
            }
            ps.println("0");
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

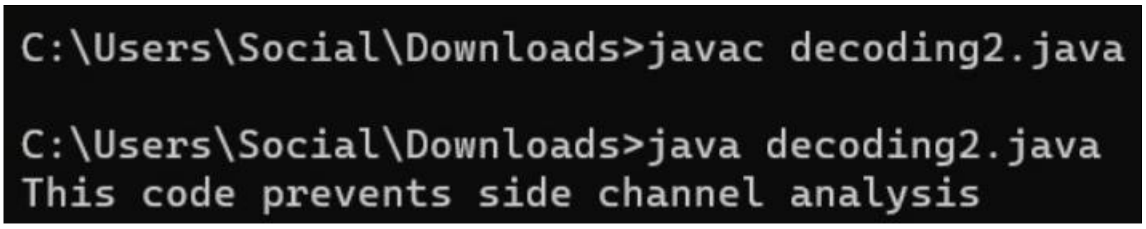
DECODING

```
import java.net.*;
import java.io.*;
import java.util.*;
import java.security.SecureRandom;

class HelloWorld {
    public static int strToInt(String s) {
        int ans = 0;
        for (int i = 0; i < s.length(); i++) {
            ans = ans + ((int) s.charAt(i) - (int) '0') *
(int) Math.pow(10, s.length() - i - 1);
        }
        return ans;
    }

    public static String convert(int ka) {
        StringBuilder s = new StringBuilder();
        int count = 0;
        while (ka != 0) {
            int x = ka % 10;
            if (x > 5) {
                x = x - 5;
            }
            s.insert(0, x);
            ka = ka / 10;
            count++;
        }
        return s.toString();
    }

    public static int generateRandomPrime(int lowerBound, int
upperBound) {
        SecureRandom random = new SecureRandom();
        int g;
        boolean isPrime;

        do {
            g = random.nextInt(upperBound - lowerBound + 1) +
lowerBound;
            isPrime = true;

            // Check if the random number is prime
            for (int i = 2; i <= Math.sqrt(g); i++) {
                if (g % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        } while (!isPrime);
        return g;
    }

    public static int modPow(int base, int exponent, int
modulus) {
        int result = 1;
        base = base % modulus;

        while (exponent > 0) {
            if (exponent % 2 == 1) {
                result = (result * base) % modulus;
            }

            exponent = exponent >> 1;
            base = (base * base) % modulus;
        }

        return result;
    }

    public static String reverse(String s) {
        StringBuilder ans = new StringBuilder();
        for (int i = s.length() - 1; i >= 0; i--) {
            ans.append(s.charAt(i));
        }
        return ans.toString();
    }

    public static String columnShifter(String s, String rand)
{
        StringBuilder result = new StringBuilder(s);
        for (int i = 0; i < rand.length(); i = i + 2) {
            int ind1 =
Character.getNumericValue(rand.charAt(i));
            int ind2 = Character.getNumericValue(rand.charAt(i
+ 1));
            char temp = result.charAt(ind1);
            result.setCharAt(ind1, result.charAt(ind2));
            result.setCharAt(ind2, temp);
        }
        return result.toString();
    }

    public static String complement(String s) {
        StringBuilder ans = new StringBuilder();
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == '1') {
                ans.append('0');
            } else {
                ans.append('1');
            }
        }
        return ans.toString();
    }

    public static int binToDec(String s) {
        int ans = 0;
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == '1') {
                ans = ans + (int) Math.pow(2, 6 - i);
            }
        }
        return ans;
    }

    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(8888);
            Socket s = ss.accept();
            // Initializing buffered reader to take input from
the client
            BufferedReader br = new BufferedReader(new
InputStreamReader(s.getInputStream()));
            // Initializing printstream To Send a message to
the client
            PrintStream ps = new
PrintStream(s.getOutputStream());
            // Scanner is used to take input from the compiler
            Scanner sc = new Scanner(System.in);
            int lowerBound = 100;
            int upperBound = 999;
            StringBuilder key = new StringBuilder();

            for (int i = 0; i < 2; i++) {
                String str3 = br.readLine();
                int G = strToInt(str3);
                SecureRandom rand = new SecureRandom();
                String str2 = br.readLine();
                int n = strToInt(str2);
                int b = rand.nextInt(100);
                int y = modPow(G, b, n);
                String str = br.readLine();
                int x = strToInt(str);
                ps.println(y);
                int kb = modPow(x, b, n) % 900 + 100;
                key.insert(0, convert(kb));
            }

            Vector<String> v = new Vector<>();
            int x = 1;
            try {
                while (x == 1) {
                    String fstr = br.readLine();
                    if (fstr.equals("0")) {
                        break;
                    }
                    v.add(fstr);
                }
            } catch (Exception e) {
                x = 0;
            }

            key = new StringBuilder(reverse(key.toString()));
            StringBuilder fans = new StringBuilder();
            for (int i = 0; i < v.size(); i++) {
                if (i % 2 != 0) {
                    String s1 = reverse(v.get(i));
                    s1 = columnShifter(s1, key.toString());
                    s1 = complement(s1);
                    int ans = binToDec(s1);
                    if ((char) ans == '@') {
                        ans = ' ';
                    }
                    fans.append((char) ans);
                }
            }
            System.out.println(fans.toString());

        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```



```
C:\Users\Social\Downloads>javac encoding2.java

C:\Users\Social\Downloads>java encoding2.java
Enter a message:
This code prevents side channel analysis
```

Fig-10: Encoding Output



```
C:\Users\Social\Downloads>javac decoding2.java

C:\Users\Social\Downloads>java decoding2.java
This code prevents side channel analysis
```

Fig-11: Decoding Output

Key Points to Note

1. Use of SecureRandom: The implementation utilizes the java.security.SecureRandom class instead of java.util.Random for generating secure random numbers. This ensures that the random numbers used in the cryptographic operations are not predictable and provide stronger security.

2. Minimizing Timing Variations: Unnecessary print statements or operations that could introduce timing variations have been removed. Timing variations can potentially leak sensitive information and compromise the security of the algorithm. By minimizing such variations, the implementation aims to enhance the overall security of the system.

3. Use of StringBuilder: The implementation utilizes the StringBuilder class instead of string concatenation when building strings. StringBuilder is more efficient in terms of memory usage and performance, especially when dealing with large strings or frequent string manipulations.

4. Constant-Time Operations: The code structure has been revised to avoid operations that may introduce timing variations. Achieving constant-time execution is crucial in cryptography to prevent potential side-channel attacks. However, it is important to note that achieving complete constant-time execution requires a thorough analysis of the entire codebase and may involve more complex modifications beyond the mentioned key points.

These considerations demonstrate a focus on improving security and performance aspects within the implementation of the AES and RSA algorithms for data encryption and decryption.

CONCLUSION

This research article focuses on addressing the issue of side-channel analysis attacks in cryptographic algorithms, particularly the AES algorithm. Side-channel attacks exploit vulnerabilities in the implementation of algorithms, such as timing analysis and power analysis, to extract sensitive information. The proposed approach aims to prevent such attacks by implementing time-independent operations and incorporating noise generators to mask power consumption.

The article also reviews various research papers and studies related to AES attacks, the impact of side-channel attacks on information security, key management using Diffie-Hellman key exchange, design and implementation of AES, AES image encryption, improved cryptography using AES and RSA, AES and RSA implementation for cloud purposes, data security using AES, and data sharing using AES. These studies highlight different aspects and applications of cryptography and emphasize the importance of secure data transmission and storage.

The Diffie-Hellman key exchange algorithm is highlighted as a secure approach for sharing keys between parties. It is implemented in a client-server architecture to ensure secure data transmission. The client encodes the message using a shared key, and the server decodes the message to retrieve the original content.

Overall, the research article provides insights into the importance of cryptography and the measures taken to enhance security, specifically in relation to side-channel analysis attacks. It emphasizes the need for continuous research and improvements in cryptographic algorithms to stay ahead of potential security threats.

BIBLIOGRAPHY

Data and Computer Communications -- W. Stallings.

Data Communication and Networking -- B. A. Forouzan

TCP/IP Protocol Suite -- B. A. Forouzan

UNIX Network Programming -- W. R. Stallings

Introduction to Computer Networks and Cybersecurity -- C-H. Wu and J. D. Irwin

Cryptography and Network Security: Principles and Practice -- W. Stallings

Literature Reviews from published papers