Stealth: A Highly Secured End-to-End Symmetric Communication Protocol

Ripon Patgiri, *Senior Member, IEEE*

***Abstract*—Symmetric key cryptography is applied in almost all secure communications to protect all sensitive information from attackers, for instance, banking, and thus, it requires extra attention due to diverse applications. Moreover, it is vulnerable to various attacks, for example, cryptanalysis attacks. Cryptanalysis attacks are possible due to a single-keyed encryption system. The state-of-the-art symmetric communication protocol uses a single secret key to encrypt/decrypt the entire communication to exchange data/message that poses security threats. Therefore, in this paper, we present a new secure communication protocol based on Diffie-Hellman cryptographic algorithms, called Stealth. It is a symmetric-key cryptographic protocol to enhance the security of modern communication with truly random numbers. Additionally, it applies a pseudo-random number generator. Initially, Stealth uses the Diffie-Hellman algorithm to compute four shared secret keys. These shared secret keys are used to generate four different private keys to encrypt for the first block of the message for symmetric communication. Stealth changes its private keys in each communication, making it very hard to break the security protocol. Moreover, the four shared secret keys create additional complexity for the adversary to overcome, and hence, it can provide highly tight security in communications. Stealth neither replaces the existing protocol nor authentication mechanism, but it creates another security layer to the existing protocol to ensure the security measurement's tightness.**

***Index Terms*—Security, Security Protocol, Encryption, Cryptography, Symmetric-key Cryptography, Diffie-Hellman Cryptography, Random Number Generator, Computer Networking.**

I. Introduction

SYMMETRIC communication protocols are the most used security protocol, and it is able to provide high security due to the computation of a shared secret key. Adversaries require years to defeat such kinds of security measures, for instance, Diffie-Hellman cryptography [1]. Also, there are diverse variants of symmetric key exchange protocol to enhance communication security, for example, Elliptic-curve Diffie-Hellman cryptography [2] based on Diffie-Hellman, and Elliptic-curve cryptography [3]. Secure symmetric communication requires encryption/decryption methods to convert the plaintext to cipher; however, there are various possible attacks for symmetric key encryption algorithms [4]. Moreover, the encryption is performed using a single secret key, and therefore, attackers can find a pattern to reveal the secret key [5].

The shared secret key is used in encrypting a message for communication. A conventional communication system agrees upon a fixed shared secret key and exchanges the messages with two end-points. There is a high possibility to find a pattern to discover the shared secret key from a set of messages. Moreover, the brute-force method can always discover a shared secret key for encryption if the entire communication is performed using a single shared secret key. However, it may take many years or may take a few trials to reveal the secret key. Therefore, there is a high risk involved in such kind of communication. It is not impossible to reveal the shared secret key for the known-plaintext, chosen-plaintext, chosen-ciphertext, linear cryptanalysis, differential cryptanalysis, differential fault analysis, differential power analysis, differential timing analysis, and side-channel attacker [6]. Obviously, a cryptanalyst always tries to find a weakness and patterns to reveal the secret keys. Thus, symmetric-key cryptography poses a high risk of being attacked by adversaries. Therefore, there are two research questions, and these are as follows-

RQ1 Is it possible to change the secret keys in each symmetric communication without using extra communication overhead?

RQ2 Is it possible to protect the symmetric communication from cryptanalysist and brute-force attackers?

These two research questions pose a new challenge to secure symmetric communication. Therefore, in this paper, we propose a novel and highly secure communication protocol to secure from different kind of attackers in symmetric cryptography, called Stealth, which address the above research questions **RQ1** and **RQ2**. Stealth is a secure symmetric communication protocol for highly sensitive data communication to protect from adversaries. Therefore, it initially uses the Diffie-Hellman key exchange protocol to compute four shared secret keys. These shared secret keys are used to generate the four private keys for encryption/decryption. The encryption/decryption requires four private keys. In each communication, Stealth changes its private keys to ensure high security. Also, these private keys are generated using a pseudo-random number generator.

The key contribution of the paper is as follows-

- Stealth is a secure communication protocol based on Diffie-Hellman cryptography to defend against various attacks, and it enhances the Diffie-Hellman key exchange protocol without incurring extra communication. Stealth uses the same number of communication costs as well as the Diffie-Hellman algorithm. It also enhances the cryptography protocol, Advanced Encryption Standard (AES) [7], for better encryption or decryption.
- Stealth uses four private keys and changes its private keys in each communication. Therefore, the probability

Ripon Patgiri, Department of Computer Science & Engineering, National Institute of Technology Silchar, Assam, India-788010, ripon@cs.nits.ac.in, url: http://cs.nits.ac.in/rp/

Manuscript received Month 00, 20XX; revised Month 00, 20XX.

of capturing entire communication without knowing the private keys is $(\frac{1}{8^\beta})^m$ for β bit sized private keys and m communications.
- Stealth uses four shared secret keys to generate four private keys, and the probability of getting correct private keys is $\frac{1}{16^\beta}$.

Stealth works on the existing methodology for security, namely, authentication, key exchange protocol, and block cipher, and it does not replace the existing methods, but it adds an extra layer to protect the communication from various attackers. Stealth can defend diverse cryptanalysis attacks and brute force attacks. However, it does not address the issues of the man-in-the-middle (MITM) attack and DDoS attacks.

The paper is organized as follows- Section II highlights the preliminaries for the proposed work, called Stealth. Section III presents the proposed algorithm, Stealth, in-depth. Section V provides the experimental results of a pseudo-random number generator, called Stealth-PRNG. Stealth depends on the truly random number to secure each communication. Stealth-PRNG proves its randomness in NIST SP 800-22 statistical testing in Section V. Section IV analyzes the proposed work and discusses its security attacks. Also, it provides detailed mathematical analysis. Finally, this paper is concluded in Section VI.

II. Background

There is a diverse variant of symmetric cryptography algorithms [8]; namely, International Data Encryption Algorithm (IDEA) [9], Twofish, Serpent, Rijndael [7], Camellia, Salsa20, ChaCha20, Blowfish, CAST5, Kuznyechik, RC4, DES, 3DES, Skipjack, and Safer. These cryptography algorithms are used to convert plaintext to ciphertext using the shared secret key and vice-versa. Therefore, these cryptography algorithms require a method to compute the shared secret key. The famous key exchange protocol, the Diffie-Hellman algorithm, is used to compute shared secret keys. Also, it is enhanced using different methods, for instance, Elliptic-curve Diffie-Hellman algorithm [2] used to overcome the issue of the Logjam [10]. However, the Diffie-Hellman algorithm does not provide an authentication mechanism to protect man-in-the-middle (MITM) attacks. Therefore, secure symmetric communication requires an authentication mechanism, a key exchange protocol, and an encryption/decryption protocol. The authentication mechanism achieved using a digital signature to defeat the MITM attack.

A. Diffie-Hellman Key Exchange Protocol

TABLE I: Diffie-Hellman Key Exchange Protocol for three secret key generation.

$\mathbb{A}$	$\mathbb{E}$	$\mathbb{B}$
$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$
g	g	g
a		b
$\mathcal{A} = g^a \ mod \ \mathcal{P}$		$\mathcal{B} = g^b \ mod \ \mathcal{P}$
$\mathcal{B}$	$\mathcal{A}$, $\mathcal{B}$	$\mathcal{A}$
$\mathcal{SK} = \mathcal{B}^a \ mod \ \mathcal{P}$		$\mathcal{SK} = \mathcal{A}^b \ mod \ \mathcal{P}$

The Diffie-Hellman symmetric algorithm is used to compute shared secret keys between two end-points [1], [11]. Initially, $\mathbb{A}$ and $\mathbb{B}$ share two prime numbers (P, g) over the public channel as shown in Table I. The $\mathbb{A}$ and $\mathbb{B}$ uses a true random number generator to generate a random number a and b, respectively, which is independent of each other. $\mathbb{A}$ and $\mathbb{B}$ keeps the random numbers secret. $\mathbb{A}$ computes $\mathcal{A} = g^a \ mod \ \mathcal{P}$ and $\mathbb{B}$ computes $\mathcal{B} = g^b \ mod \ \mathcal{P}$. $\mathbb{A}$ shares computed value $\mathcal{A}$ to $\mathbb{B}$ and $\mathbb{B}$ shares the computed value $\mathcal{B}$ to $\mathbb{A}$ over public channel. Then, $\mathbb{A}$ and $\mathbb{B}$ can compute common shared secret key $\mathcal{SK}$. Thus, the Diffie-Hellman algorithm exchanges secret keys between two end-points.

The Elliptic-curve Diffie-Hellman (ECDH) cryptography can also be used to exchange shared secret keys over a public channel [2] as an alternative to the Diffie-Hellman algorithms. ECDH is based on Elliptic-curve cryptography [3], [12]. ECDH is an enhanced version of the conventional Diffie-Hellman Algorithm. However, both Diffie-Hellman and ECDH algorithms are prone to Man-in-the-middle (MITM) attacks. When $\mathbb{A}$ and $\mathbb{B}$ communicates, Mallory acts as $\mathbb{B}$ and seizes the entire communication with $\mathbb{A}$. Therefore, $\mathbb{A}$ and $\mathbb{B}$ are unable to communicate due to Mallory. The authentication mechanism is essential to defeating such kinds of attacks. In modern practice, a digital signature is used to defeat such kinds of attacks.

III. Stealth- The Proposed Systems

We propose a novel and highly secured symmetric communication protocol to implement hard secrecy which ensures high security, called Stealth. It is based on the Diffie-Hellman cryptography algorithm. Stealth adds extra complexity in the Diffie-Hellman algorithm to ensure end-to-end secure secret key sharing. Moreover, it depends on the existing AES algorithm. There are diverse symmetric cryptography algorithms [8]; however, our proposed systems implements the existing cryptography algorithms. Stealth is not only a symmetric communication protocol but also an enhancer of any symmetric cryptography algorithm to provide higher security than the conventional method. A sender $\mathbb{A}$ wish to send a message to $\mathbb{B}$ and both parties $\mathbb{A}$ and $\mathbb{B}$ are active the same time. The $\mathbb{A}$ and $\mathbb{B}$ use Diffie-Hellman algorithm to exchange four secret keys. These secret keys are used to generate four private keys to encrypting message. The three private keys are used to encrypt a block of message and a private key is used to generate a seed value. The second and the third private key are XORed with original message and the first private key is used to encrypt the block of message using AES. Each block of communication, the private keys are changed by both the sender $\mathbb{A}$ and the receiver $\mathbb{B}$. Both $\mathbb{A}$ and $\mathbb{B}$ execute the same function to produce same private keys. Otherwise, receiver $\mathbb{B}$ cannot decrypt the message from the sender $\mathbb{A}$. Similarly, the AES uses the first generated private key to encryption or decryption. The following subsections provide the detailed descriptions of our proposed system, Stealth.

The key objectives of our proposed systems are outlined below-

- To provide high security over conventional symmetric communications.

- To enhance Diffie-Hellman algorithms using unpredictable and cryptographically secure random number generator.
- To defeat cryptanalysis attacks which is a major challenge in secure symmetric communication.

To achieve above objectives, our proposed system has four assumptions which are outlined below-

- The $\mathbb{A}$ and $\mathbb{B}$ are valid entities, and both are active at a given time for communication.
- Digital signature is used to defeat man-in-the-middle (MITM) attacks. Therefore, Stealth assumes that there is no MITM attack.
- Stealth depends on the Diffie-Hellman algorithm, and therefore, we omit detailed analysis on Diffie-Hellman algorithms.
- Also, Stealth depends on the existing block cipher symmetric cryptography AES, and we skip the detailed analysis of the same.
- Our proposed system does not deal with DDoS attacks.

A. Diffie-Hellman Key Exchange Protocol for the Stealth

TABLE II: Diffie-Hellman Key Exchange Protocol for four secret key generation. The $\mathbb{A}$ is a sender, $\mathbb{E}$ is an attacker and $\mathbb{B}$ is a receiver. However, the sender can be receiver or vice-versa.

$\mathbb{A}$	$\mathbb{E}$	$\mathbb{B}$
$\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{T}, \beta$	$\mathcal{P}, \mathcal{Q}, \mathcal{R}, \beta$	$\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{T}, \beta$
e, f, g, h	e, f, g, h	e, f, g, h
a, b, c, d	Unknown	w, x, y, z
$\mathcal{A}_1 = e^a \bmod \mathcal{P}$, $\mathcal{A}_2 = f^b \bmod \mathcal{Q}$, $\mathcal{A}_3 = g^c \bmod \mathcal{R}$, $\mathcal{A}_4 = h^d \bmod \mathcal{T}$		$\mathcal{B}_1 = e^w \bmod \mathcal{P}$, $\mathcal{B}_2 = f^x \bmod \mathcal{Q}$, $\mathcal{B}_3 = g^y \bmod \mathcal{R}$, $\mathcal{B}_4 = h^z \bmod \mathcal{T}$
$\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$	$\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$, $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$	$\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$
$\mathcal{SK}_1 = \mathcal{B}_1^a \bmod \mathcal{P}$, $\mathcal{SK}_2 = \mathcal{B}_2^b \bmod \mathcal{Q}$, $\mathcal{SK}_3 = \mathcal{B}_3^c \bmod \mathcal{R}$, $\mathcal{SK}_4 = \mathcal{B}_4^d \bmod \mathcal{T}$	Unknown	$\mathcal{SK}_1 = \mathcal{A}_1^w \bmod \mathcal{P}$, $\mathcal{SK}_2 = \mathcal{A}_2^x \bmod \mathcal{Q}$, $\mathcal{SK}_3 = \mathcal{A}_3^y \bmod \mathcal{R}$, $\mathcal{SK}_4 = \mathcal{A}_4^z \bmod \mathcal{T}$

Stealth is a symmetric communication protocol. It depends on the Diffie-Hellman cryptography algorithm [1]. Diffie-Hellman cryptography requires a true random number that is kept secret. Similarly, Stealth uses Diffie-Hellman cryptography and uses eight true random numbers (a, b, c, d, w, x, y, z), eight prime numbers $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathcal{T}, e, f, g, \& h)$ and the bit sizes of the pseudo-random numbers β. Table II demonstrates the required parameters to calculate the secret keys. The eight true random numbers (a, b, c, d, w, x, y, z) are kept private. In Stealth, the $\mathbb{A}$ and $\mathbb{B}$ must be active at the given time for communication. Therefore, $\mathbb{A}$ calculates $\mathcal{A}_1 = e^a \bmod \mathcal{P}$, $\mathcal{A}_2 = f^b \bmod \mathcal{Q}$, $\mathcal{A}_3 = g^c \bmod \mathcal{R}$, $\mathcal{A}_4 = h^d \bmod \mathcal{T}$, and the $\mathbb{B}$ also calculates $\mathcal{B}_1 = e^w \bmod \mathcal{P}$, $\mathcal{B}_2 = f^x \bmod \mathcal{Q}$, $\mathcal{B}_3 = g^y \bmod \mathcal{R}$, $\mathcal{B}_4 = h^z \bmod \mathcal{T}$. The $\mathbb{A}$ shares $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, and $\mathcal{A}_4$ to $\mathbb{B}$, and the $\mathbb{B}$ shares $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$, and $\mathcal{B}_4$ to $\mathbb{A}$ over public channel. Let us assume that $\mathcal{SK}$ denotes shared secret key. The $\mathbb{A}$ calculates four secret keys $\mathcal{SK}_1 = \mathcal{B}_1^a \bmod \mathcal{P}$, $\mathcal{SK}_2 = \mathcal{B}_2^b \bmod \mathcal{Q}$, $\mathcal{SK}_3 = \mathcal{B}_3^c \bmod \mathcal{R}$, and $\mathcal{SK}_4 = \mathcal{B}_4^d \bmod \mathcal{T}$. Similarly, the $\mathbb{B}$ calculates the four secret keys $\mathcal{SK}_1 = \mathcal{A}_1^w \bmod \mathcal{P}$, $\mathcal{SK}_2 = \mathcal{A}_2^x \bmod \mathcal{Q}$, $\mathcal{SK}_3 = \mathcal{A}_3^y \bmod \mathcal{R}$, and $\mathcal{SK}_4 = \mathcal{A}_4^z \bmod \mathcal{T}$. Thus, the $\mathbb{A}$ and $\mathbb{B}$ computes the shared secret keys securely. These shared secret keys are used to compute the pseudo-random number generator to generate the private keys. A pseudo-random number generator generates the private keys. However, the a, b, c, d, w, x, y, and z are generated by a true random number generator. Initially, the shared (computed) secret keys are used to generate the private keys, and these privates keys are used to encrypt a block of message for communication.

B. Random Number Generation

Random number generators are essential for cryptography and many other applications. Therefore, the random number generator is classified into two key categories; namely, pseudo-random number generator (PRNG) [13] and true random number generator (TRNG) [14]. Both PRNG and TRNG can generate highly unpredictable and truly random numbers. Also, these algorithms produce random bits without following any patterns. Lacking pattern in bits creates hard to reproduce the given bits by the adversaries. However, PRNG uses the initial seed value as an input, while TRNG does not require any input. Therefore, a random number can be reproduced in PRNG for correct input.

Algorithm 1 Stealth-TRNG for generating the truly random numbers.

```
procedure GENSTEALTHTRNG(β)
    seed ← GETCPUCLOCK( )
    i ← 1
    while i < (β − 1) do        ▷ β is the required bit length
        P ← GETCPUCLOCK( )
        l1 ← LENGTH(P)
        Q ← GETCPUCLOCK( )
        l2 ← LENGTH(Q)
        N1 ← HASSHFUN(P, l1, seed)
        N2 ← HASSHFUN(Q, l2, N1)
        seed ← N2
        Bin[i] ← N1 ∧ 1    ▷ ∧ is a bitwise AND operator
        i ← i + 1
    end while
    Bin[i] ← 1 ▷ Producing odd number by padding 1 at the end.
    return Bin
end procedure
```

Stealth uses truly random numbers $(a, b, c, d, w, x, y,$ and $z)$, and these are generated by Algorithm 1. For instance, $a \leftarrow$ GENSTEALTHTRNG(128) assigns a 128 bits truly random number. The true random number generators are mainly dependent on hardware, for instance, FPGA [15], [16] or Quantum devices [17], [18], [19], [20]. These hardware-based true random number generators are quite faster than other conventional TRNG [21]. There are diverse TRNG algorithms based on various parameters to produce high quality truly random number; for instance, currents/voltages [22], [13], light [23], signal [24], [25], camera [26], etc.; however, Stealth-TRNG depends on the CPU Clock values. To the best of our

knowledge, our proposed TRNG is the first variant of true random number generator that utilizes CPU Clock and string hash function.

Moreover, PRNG requires an initial seed value to produce truly random numbers. This initial seed value is the attacking point for the attackers. Brute-force attackers can discover the initial seed value for the PRNG. Therefore, PRNG is weaker than TRNG, but Stealth uses PRNG to secure the communication and proves that it is much harder for brute-force attackers even if Stealth uses PRNG. The necessary conditions of PRNG to be used in Stealth are as follows-

- The PRNG should take input of initial key, seed value and the bit sizes.
- The PRNG should be able to reproduce same random number for the same initial key, the same seed value and the same bit sizes.
- The produced bit pattern should be highly unpredictable and does not follow any kind of patterns.

The PRNG must fulfill those above mentioned necessary conditions. Otherwise, it is not possible to produce high-quality random numbers. Also, it cannot produce the same private keys for $\mathbb{A}$ and $\mathbb{B}$. Moreover, PRNG should not depend on non-reproducible parameters such as CPU Clock value. Similar to TRNG, our proposed PRNG is the first variant of pseudo-random number generator that utilizes string hash function. The string hash function is used to mix the bits [27].

Algorithm 2 Stealth-PRNG for generating the pseudo-random numbers.

procedure GENSTEALTHPRNG($\mathcal{SK}$, $seed$, β)
  $i \leftarrow 1$
  $seed_1 \leftarrow seed$
  $seed_2 \leftarrow seed \oplus Prime\ number$
  while $i < (\beta - 1)$ **do** ▷ β is the required bit length
    $l \leftarrow$ LENGTH($\mathcal{SK}$)
    $N_1 \leftarrow$ HASSHFUN($\mathcal{SK}$, l, $seed_1$)
    $N_2 \leftarrow$ HASSHFUN($\mathcal{SK}$, l, $seed_2$)
    $seed_1 \leftarrow N_2$
    $seed_2 \leftarrow N_1$
    $key \leftarrow N_1 \oplus N_2$
    $\mathcal{SK} \leftarrow$ CONVERTINTOSTRING(key)
    $Bin[i] \leftarrow P \wedge 1$
    $i \leftarrow i + 1$
  end while
  $Bin[i] \leftarrow 1$ ▷ Producing odd number by padding 1 at the end.
  return Bin
end procedure

Algorithm 2 is a PRNG to generate highly unpredictable bits or numbers for Stealth. The GENSTEALTHPRNG( ) uses an input key, an initial seed value, and the bit sizes. Algorithm 2 can produce the same output for the same input parameters. It requires to produce the same private keys by both sender and receiver. Algorithm 2 is based on hash functions that mix bits and produce unpredictable bits. The GENSTEALTHPRNG( ) extract the least significant bit (LSB) and stores it in a binary array. Moreover, the input key $\mathcal{SK}$ changes in each iteration in the Algorithm 2. Here, the seed values are also changed in each iteration. Therefore, it produces highly unpredictable LSB (either 0 or 1). The hash functions mix the bits, and the function GENSTEALTHPRNG( ) uses murmur non-cryptographic string hash function [27]. Alternatively, there are many hash functions, namely, xxHash, SuperFastHash, CRC32, MD5, and SHA. However, the murmur hash function produces good random LSB bit compared to the other string hash functions. The algorithms iterate β times, and thus, its time complexity is $O(\beta \times l)$ where β is the bit size, and l is the length of the string. The length of the string l is constant, and therefore, the total time complexity is $O(\beta)$. Moreover, the bit size β is approximately 128 to 1024 bits. Therefore, we can rewrite the time complexity as $O(\beta) \approx O(1)$.

C. Communication Protocol

Table III demonstrates the secure communication between $\mathbb{A}$ and $\mathbb{B}$. Initially, $\mathbb{A}$ and $\mathbb{B}$ establishes a connection. Let us consider, $\mathbb{A}$ would like to send a message $\mathcal{M}$ to $\mathbb{B}$, and thus, the connection is established through public key cryptography. Therefore, $\mathbb{A}$ and $\mathbb{B}$ can compute shared secret keys using Diffie-Hellman cryptography. Let us denote $\mathcal{PK}$ be the private key and $t^{\mathcal{A}}$ be the logical timestamp of $\mathbb{A}$. The $\mathbb{A}$ computes $\mathcal{PK}_1$ = GENSTEALTHPRNG($\mathcal{SK}_1, \mathcal{SK}_4, \beta$), $\mathcal{PK}_2$ = GENSTEALTHPRNG($\mathcal{SK}_2, \mathcal{SK}_4, \beta$), $\mathcal{PK}_3$ = GENSTEALTHPRNG($\mathcal{SK}_3, \mathcal{SK}_4, \beta$), $\mathcal{PK}_4$ = GENSTEALTHPRNG($\mathcal{PK}_1, \mathcal{PK}_2, \beta$), $\mathbb{M}_1 = \mathcal{M}_1 \oplus \mathcal{PK}_2$, $\mathbb{M}_1^m = \mathbb{M}_1 \oplus \mathcal{PK}_3$, and $\mathbb{M}_1^{cm} = Enc^{\mathcal{PK}_1}(\mathbb{M}_1^m)$. The $\mathbb{A}$ sends the cipher $(\mathbb{M}_1^{cm}, t_1^{\mathcal{A}})$ to $\mathbb{B}$. $\mathbb{B}$ receives $(\mathbb{M}_1^{cm}, t_1^{\mathcal{A}})$ from $\mathbb{A}$. The $\mathbb{B}$ computes $\mathcal{PK}_1$ = GENSTEALTHPRNG($\mathcal{SK}_1, \mathcal{SK}_4, \beta$), $\mathcal{PK}_2$ = GENSTEALTHPRNG($\mathcal{SK}_2, \mathcal{SK}_4, \beta$), $\mathcal{PK}_3$ = GENSTEALTHPRNG($\mathcal{SK}_3, \mathcal{SK}_4, \beta$), $\mathcal{PK}_4$ = GENSTEALTHPRNG($\mathcal{PK}_1, \mathcal{PK}_2, \beta$), $\mathbb{M}_1^m = Dec^{\mathcal{PK}_1}(\mathbb{M}_1^{cm})$. $\mathbb{M}_1 = \mathbb{M}_1^m \oplus \mathcal{PK}_3$, and retrieves original message $\mathcal{M}_1 = \mathbb{M}_1 \oplus \mathcal{PK}_2$. Stealth uses AES symmetric cryptography (block cipher) for encryption and decryption [7]. The secret keys $\mathcal{PK}_1$, $\mathcal{PK}_2$, and $\mathcal{PK}_3$ strengthen the symmetric key encryption. The generated secret key $\mathcal{PK}_1$ is used as a private key for encryption. Similarly, $\mathcal{PK}_2$, and $\mathcal{PK}_3$ are used as a mixer for mixing a block of the message and produces cipher. The mixing operation has to be performed before the encryption. It creates a cipher before encryption by the AES method. Therefore, to send the second block of message, $\mathbb{A}$ computes $\mathcal{PK}_5$ = GENSTEALTHPRNG($\mathcal{PK}_1, \mathcal{PK}_4, \beta$), $\mathcal{PK}_6$ = GENSTEALTHPRNG($\mathcal{PK}_2, \mathcal{PK}_4, \beta$), $\mathcal{PK}_7$ = GENSTEALTHPRNG($\mathcal{PK}_3, \mathcal{PK}_4, \beta$), $\mathcal{PK}_8$ = GENSTEALTHPRNG($\mathcal{PK}_5, \mathcal{PK}_6, \beta$), $\mathbb{M}_2 = \mathcal{M}_2 \oplus \mathcal{PK}_6$, $\mathbb{M}_2^m = \mathbb{M}_1 \oplus \mathcal{PK}_7$, $\mathbb{M}_2^{cm} = Enc^{\mathcal{PK}_5}(\mathbb{M}_2^m)$, and sends $(\mathbb{M}_2^{cm}, t_2^{\mathcal{A}})$ to $\mathbb{B}$. The $\mathbb{B}$ receives $(\mathbb{M}_2^{cm}, t_2^{\mathcal{A}})$ from $\mathbb{A}$ and computes $\mathcal{PK}_5$ = GENSTEALTHPRNG($\mathcal{PK}_1, \mathcal{PK}_4, \beta$), $\mathcal{PK}_6$ = GENSTEALTHPRNG($\mathcal{PK}_2, \mathcal{PK}_4, \beta$), $\mathcal{PK}_7$ = GENSTEALTHPRNG($\mathcal{PK}_3, \mathcal{PK}_4, \beta$), $\mathcal{PK}_8$ = GENSTEALTHPRNG($\mathcal{PK}_5, \mathcal{PK}_6, \beta$), $\mathbb{M}_2^m = Dec^{\mathcal{PK}_5}(\mathbb{M}_2^{cm})$, $\mathbb{M}_2 = \mathbb{M}_2^m \oplus \mathcal{PK}_7$, and retrieve the original block of the message $\mathcal{M}_2 = \mathbb{M}_2 \oplus \mathcal{PK}_6$. It shows that the private keys are not fixed, and it changes in each communication.

TABLE III: Communication Mechanism between $\mathbb{A}$ and $\mathbb{B}$. $\mathbb{B}$ receives all messages in order.

$\mathbb{A}$	$\mathbb{B}$
$\mathcal{PK}_1 = \text{GENSTEALTHPRNG}(\mathcal{SK}_1, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_2 = \text{GENSTEALTHPRNG}(\mathcal{SK}_2, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_3 = \text{GENSTEALTHPRNG}(\mathcal{SK}_3, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_4 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_2, \beta)$ $\mathbb{M}_1 = \mathcal{M}_1 \oplus \mathcal{PK}_2$ $\mathbb{M}_1^m = \mathbb{M}_1 \oplus \mathcal{PK}_3$ $\mathbb{M}_1^{cm} = Enc^{\mathcal{PK}_1}(\mathbb{M}_1^m)$ Send $(\mathbb{M}_1^{cm}, t_1^{\mathcal{A}})$ to $\mathbb{B}$	
	Receives $(\mathbb{M}_1^{cm}, t_1^{\mathcal{A}})$ from $\mathbb{A}$ $\mathcal{PK}_1 = \text{GENSTEALTHPRNG}(\mathcal{SK}_1, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_2 = \text{GENSTEALTHPRNG}(\mathcal{SK}_2, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_3 = \text{GENSTEALTHPRNG}(\mathcal{SK}_3, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_4 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_2, \beta)$ $\mathbb{M}_1^m = Dec^{\mathcal{PK}_1}(\mathbb{M}_1^{cm})$ $\mathbb{M}_1 = \mathbb{M}_1^m \oplus \mathcal{PK}_3$ $\mathcal{M}_1 = \mathbb{M}_1 \oplus \mathcal{PK}_2$
$\mathcal{PK}_5 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_6 = \text{GENSTEALTHPRNG}(\mathcal{PK}_2, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_7 = \text{GENSTEALTHPRNG}(\mathcal{PK}_3, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_8 = \text{GENSTEALTHPRNG}(\mathcal{PK}_5, \mathcal{PK}_6, \beta)$ $\mathbb{M}_2 = \mathcal{M}_2 \oplus \mathcal{PK}_6$ $\mathbb{M}_2^m = \mathbb{M}_1 \oplus \mathcal{PK}_7$ $\mathbb{M}_2^{cm} = Enc^{\mathcal{PK}_5}(\mathbb{M}_2^m)$ Send $(\mathbb{M}_2^{cm}, t_2^{\mathcal{A}})$ to $\mathbb{B}$	
	Receives $(\mathbb{M}_2^{cm}, t_2^{\mathcal{A}})$ from $\mathbb{A}$ $\mathcal{PK}_5 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_6 = \text{GENSTEALTHPRNG}(\mathcal{PK}_2, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_7 = \text{GENSTEALTHPRNG}(\mathcal{PK}_3, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_8 = \text{GENSTEALTHPRNG}(\mathcal{PK}_5, \mathcal{PK}_6, \beta)$ $\mathbb{M}_2^m = Dec^{\mathcal{PK}_5}(\mathbb{M}_2^{cm})$ $\mathbb{M}_2 = \mathbb{M}_2^m \oplus \mathcal{PK}_7$ $\mathcal{M}_2 = \mathbb{M}_2 \oplus \mathcal{PK}_6$

D. Shared Secret Key

Stealth uses Diffie-Hellman cryptography to compute shared secret keys. Stealth requires four secret keys, and these secret keys are computed to generate private keys for communication or data exchanging. The secret key $\mathcal{SK}_1$, $\mathcal{SK}_2$, $\mathcal{SK}_3$ and $\mathcal{SK}_4$ are shared using Diffie-Hellman algorithm. Also, these secret key $\mathcal{SK}_1$, $\mathcal{SK}_2$, and $\mathcal{SK}_3$ are used to generate initial private keys. However, the secret key $\mathcal{SK}_4$ is used as the initial seed value for the PRNG. In addition, the generated private keys are used to generate other private keys for next communication. This process continues to complete the communication.

E. Private Key

The secret keys are generated using the Diffie-Hellman algorithm. These secret keys are used to generate initial private keys $\mathcal{PK}_1$, $\mathcal{PK}_2$, $\mathcal{PK}_3$, and $\mathcal{PK}_4$ which are generated using the four secret keys. However, $\mathcal{PK}_4$ is generated using two private keys ($\mathcal{PK}_1$ and $\mathcal{PK}_2$). The secret key $\mathcal{SK}_4$ is replaced by $\mathcal{PK}_4$. Stealth uses these private keys to encrypt or decrypt the first block of the message. For second block of the message, the private keys $\mathcal{PK}_5$, $\mathcal{PK}_6$ and $\mathcal{PK}_7$ are generated using $\mathcal{PK}_1$, $\mathcal{PK}_2$, and $\mathcal{PK}_3$, respectively. The $\mathcal{PK}_4$ is used as a seed value to generate the private keys $\mathcal{PK}_5$, $\mathcal{PK}_6$ and $\mathcal{PK}_7$. The seed value $\mathcal{PK}_8$ is calculated using $\mathcal{PK}_5$ and $\mathcal{PK}_6$ to use as a seed value for the third block of the message. Thirdly, the private key $\mathcal{PK}_9$, $\mathcal{PK}_{10}$, and $\mathcal{PK}_{11}$ are calculated using $\mathcal{PK}_5$, $\mathcal{PK}_6$ and $\mathcal{PK}_7$, respectively. The $\mathcal{PK}_{12}$ is calculated using $\mathcal{PK}_9$ and $\mathcal{PK}_{10}$ to use as a seed value for the fourth block of the message and so on.

F. Encryption

Stealth uses AES [7] for encryption and decryption. Moreover, AES is a well-proven and well-practiced symmetric-key cryptography protocol. Conventionally, the plaintext and secret key are input into the AES and converted into ciphertext—however, Stealth input plaintext and convert these plaintext into ciphertext. The ciphertext is input into AES along with the private keys.

IV. ANALYSIS

Stealth is designed to provide higher security than conventional security systems in symmetric communication. It ensures security in each communication by providing an extra coating to the message. Stealth requires three private keys to be computed for a single communication and a seed value (also a private key). The private keys are changed at each communication. Also, the seed value is changed in each communication. This changing nature is unpredictable in each communication. The Diffie-Hellman algorithm's strength lies in random number generators. The true random number generator generates unpredictable bit patterns and cannot reproduce it by anyone. Therefore, Stealth uses true random number generator to generate the randomly chosen number for Diffie-Hellman. The pseudo-random number generator generates a truly random number based on initial inputs. It is used to

create the private keys because the private keys change in each communication. Stealth is applicable in block cipher but not stream cipher.

A. Shared Secret Key

The sharing of a secret key is fully dependent on the Diffie-Hellman key exchange algorithm in our proposed algorithm. Stealth requires four secret keys to be computed for sharing between two parties. In Stealth, the secret keys are not used to encrypt the message. However, these secret keys are used to generate private keys of the first communication, i.e., first block, and a seed value for the next communication. After the first communication, shared secret keys are not required.

Initially, Diffie-Hellman algorithm choose a random number, and therefore, Stealth requires eight random numbers, namely, $\mathbb{A}$ chooses a, b, c and d, and $\mathbb{B}$ chooses w, x, y, and z. The strength of the Diffie-Hellman lies within these random numbers. Therefore, Stealth uses a true random number generator to generate these random numbers. It is difficult to guess these random numbers by adversaries if these random numbers are generated using a true random number generator. True random number generator produces the bit patterns in the unpredictable sequence, and thus, attackers are unable to guess the exact number.

Theorem 1. *The probability of getting the four shared secret keys of Stealth by an attacker is $\frac{1}{16^\beta}$ where β is the bit size of the keys.*

Proof. Diffie-Hellman uses a single random number where the guessing probability of a particular random number is $\frac{1}{2^\beta}$ where the β is the bit size of the random number. The probability of not getting the exact random number is $(1-\frac{1}{2^\beta})$. For instance, $\beta = 32$, then the probability of not getting the random number is $(1 - \frac{1}{2^{32}} \approx 1)$ which means it is not easy for a conventional computer to break the security. However, it requires a powerful computing resources to break, and it also takes a huge time. Stealth uses four such random numbers, and these random numbers are independent events. Let, $Pr(a)$, $Pr(b)$, $Pr(c)$, and $Pr(d)$ be the guessing probability of a, b, c, and d, respectively. Generating the number a, b, c, and d are independent events. Therefore, the probability of guessing all random numbers is denoted in Equation (1).

$$\begin{aligned} Pr(a \cap b \cap c \cap d) &= Pr(a) \cap Pr(b) \cap Pr(c) \cap Pr(d) \\ &= \frac{1}{2^\beta} \times \frac{1}{2^\beta} \times \frac{1}{2^\beta} \times \frac{1}{2^\beta} \\ &= \frac{1}{2^{4\beta}} = \frac{1}{16^\beta} \approx 0 \end{aligned} \tag{1}$$

In Stealth, it is not possible to get the same secret keys by the adversaries. Alternatively, this probability applies in guessing direct secret keys. Equation (1) state that the adversary requires all four shared secret keys to breach the security of Stealth, i.e., it implies that it is impossible to breach the security if β is large enough. For instance, the bit size of β may vary from 16 bits to 1024 bits or more, depending on the security requirements. Let us assume that an adversary is able to get a shared secret key by any means, and it requires another more probability to overcome, as given in Equation (2).

$$\begin{aligned} Pr((a \cap b \cap c) \mid d) &= Pr(a \cap b \cap c) = Pr(a)\ Pr(b)\ Pr(c) \\ &= \frac{1}{2^{3\beta}} = \frac{1}{8^\beta} \approx 0 \end{aligned} \tag{2}$$

Equation (2) demonstrates that even though a shared secret key is revealed, there is still probability of not getting other three shared secret keys and it is $1 - \frac{1}{8^\beta}$. Thus, Stealth ensures high security in computing shared secret keys using Diffie-Hellman algorithm. □

Corollary 1. *The probability of not getting the four shared secret keys is $(1 - \frac{1}{16^\beta}) \approx 1$.*

B. Private Keys

The shared secret keys are the private keys in conventional symmetric cryptography, however, these are not the private keys in Stealth. The shared secret keys are used to generate the private keys for encryption and decryption, i.e., private-private cryptography. In this model, the private keys are not fixed, and it changes in each message. Moreover, a single private key is used in conventional communication, which is fixed. On the contrary, Stealth changes the private keys in each communication and uses four private keys. However, Stealth does not require any communication to generate these private keys after the first key exchange. The sender and receiver generates the same private keys without communication using a pseudo-random number generator.

Lemma 1. *The probability of getting correct private keys for a single communication is $\frac{1}{16^\beta}$ where the β is the bit size of the private keys.*

Proof. Theorem 1 has already proved that the probability of getting the correct shared secret keys which is $\frac{1}{16^\beta}$. Similarly, we can easily conclude that the probability of getting the correct private keys for communication is $\frac{1}{16^\beta}$. It is a probability of a single block of a message or a single communication between $\mathbb{A}$ and $\mathbb{B}$. Let us assume that the adversary is able to get the shared secret keys, and thus, the probability of getting all the private keys is 1, because the adversary can easily generate the private keys. However, if we assume that the adversary cannot get the shared secret keys, getting the private keys directly from the first message is $\frac{1}{16^\beta}$. Therefore, the probability of getting the next private keys is 1 for the second block of message and onward because the adversary can efficiently compute the second set of the private keys from the first set of the private keys. Similarly, getting correct set of the private keys for the second message or communication without knowing the first message's private keys is $\frac{1}{16^\beta}$. Thus, the adversary can compute the next set of the private keys, and the probability of getting the third set of the private keys is 1. Likewise, the probability of getting the private keys of the last message/communication without knowing the previous private keys is also the same. Generating private keys for the next event is not an independent event. Therefore, the wise way to attack Stealth is the first communication to capture

entire communication. Thus, the total probability of getting entire private keys is restricted to $\frac{1}{16^\beta}$. □

Lemma 1 states that the probability of capturing entire message is $\frac{1}{16^\beta}$. Then, why should Stealth change the private keys in each communication which introduces computation overhead? Let us assume that an adversary is able to get the correct private keys of a message with a probability of $\frac{1}{16^\beta}$. Now, the adversary can get the entire set of the private keys for the next communication. It means that the adversary cannot read the previous messages. Even though a part of communication is broken, the other (previous) part of the communication is still secured.

Theorem 2. *The probability of breaking m^{th} block of message and onward is $\frac{1}{16^\beta}((m-1) - \frac{1}{(16^\beta - 1)})$.*

Proof. The probability of breaking first block of message using private keys is $\frac{1}{16^\beta}$ and the breaking probability of remaining message is 1. Therefore, the total probability to break the first block of message is $\frac{1}{16^\beta}(1-1) = 0$. For the second block of message, the probability of breaking the message is $\frac{1}{16^\beta}$ and not able to break the first block of message is $(1 - \frac{1}{16^\beta})$. Therefore, the total probability of breaking is $\frac{1}{16^\beta}(1 - \frac{1}{16^\beta})$. For the third block of message, the probability of breaking the message is $\frac{1}{16^\beta}$ and the not able to break the previous messages is $(1 - \frac{1}{16^{2\beta}})$. Therefore, the total probability of breaking the message at the third block and onward is $\frac{1}{16^\beta}(1 - \frac{1}{16^{2\beta}})$. Similarly, the probability of breaking the fourth block of message is $\frac{1}{16^\beta}$ and the probability of not able to break the previous block of messages is $(1 - \frac{1}{16^{3\beta}})$. The total probability of breaking the block of message is $\frac{1}{16^\beta}(1 - \frac{1}{16^{3\beta}})$ and so on. The probability of breaking the last block of message is $\frac{1}{16^\beta}$ and the probability of not able to break the previous block of messages is $(1 - \frac{1}{16^{(m-1)\beta}})$. The total probability at the last block of message is $\frac{1}{16^\beta}(1 - \frac{1}{16^{(m-1)\beta}})$. Summing up all the total probability, it gives us

$$
\begin{aligned}
\mathbb{TP} &= \frac{1}{16^\beta}(1-1) + \frac{1}{16^\beta}(1 - \frac{1}{16^\beta}) + \frac{1}{16^\beta}(1 - \frac{1}{16^{2\beta}}) + \\
&\frac{1}{16^\beta}(1 - \frac{1}{16^{3\beta}}) + \ldots + \frac{1}{16^\beta}(1 - \frac{1}{16^{(m-1)\beta}}) \\
&= \frac{1}{16^\beta}((1-1) + (1 - \frac{1}{16^\beta}) + (1 - \frac{1}{16^{2\beta}}) \\
&+ (1 - \frac{1}{16^{3\beta}}) + \ldots + (1 - \frac{1}{16^{(m-1)\beta}})) \\
&= \frac{1}{16^\beta}((m-1) - (\frac{1}{16^\beta} + \frac{1}{16^{2\beta}} + \frac{1}{16^{3\beta}} + \cdots + \frac{1}{16^{(m-1)\beta}})) \\
&= \frac{1}{16^\beta}((m-1) - (\frac{1}{16^\beta} + \frac{1}{16^{2\beta}} + \frac{1}{16^{3\beta}} + \cdots + \infty)) \\
&= \frac{1}{16^\beta}((m-1) - \frac{1}{(16^\beta - 1)})
\end{aligned}
\tag{3}
$$

□

Theorem 3. *The probability of being able to capture the entire communication by the attacker without knowing the keys is $\left(\frac{1}{8^\beta}\right)^m$ where m is the number of communication.*

Proof. Let us assume that an attacker is able to attack a particular block without knowing the private keys, and the probability is $\frac{1}{2^{3\beta}}$, but it does not mean that the attackers can also decrypt the next block. Therefore, the probability of breaking the second block is $\frac{1}{8^\beta}$, and so on. Similarly, the probability of breaking the last block of the message is also the same. These events are independent of each other because the communication is broken without knowing the private keys. Therefore, the total probability of breaking the entire communication is given in Equation (4).

$$
\begin{aligned}
Pr(\mathcal{M}_1 \cap \mathcal{M}_2 \cap \ldots \mathcal{M}_m) &= Pr(\mathcal{M}_1)\ Pr(\mathcal{M}_2) \\
&\quad Pr(\mathcal{M}_3) \ldots Pr(\mathcal{M}_m) \\
&= \frac{1}{8^\beta} \times \frac{1}{8^\beta} \times \frac{1}{8^\beta} \times \ldots \times \frac{1}{8^\beta} \\
&= \left(\frac{1}{8^\beta}\right)^m \approx 0
\end{aligned}
\tag{4}
$$

Therefore, Stealth provides tight coating of messages' block. □

Corollary 2. *The probability of not being able to capture entire communication by the attacker without knowing the keys is $1 - \left(\frac{1}{8^\beta}\right)^m \approx 1$.*

Equation (4) shows that the attackers cannot easily break the entire communication. The bit size β is 128, 256, or 512 in modern practices. Thus, the probability of capturing entire communication using such kind of attack is almost zero. Therefore, the attacker should attack the first block of communication to get the four private keys or the Diffie-Hellman algorithm.

C. Time Complexity

The time complexity of Stealth depends on various factors, particularly generating prime numbers, generating a random number, and encryption/decryption time complexity. Stealth requires a prime number generator, random number generator, and expensive operations like power and modulus operation to compute the first shared secret keys. Hence, it is costlier (slower) than the conventional Diffie-Hellman algorithm. Now, the time complexity of generating a prime number is $O(\beta + \tau \log^6 n)$ where β is the time complexity to generate the random bits by a random number generator, $\log^6 n$ is the time complexity of primality test of AKS algorithm [28], and the τ is the walking time towards the nearest prime numbers. First, Stealth generates a random number and check it for primality. If the generated random number is not a prime number, the number is incremented by one. Then, repeat the process until a prime number is met. This walking time is τ. The β is not a large number to be considered for asymptotic notation, for instance, it can be 1024. Also, the prime numbers are not rare, and hence, τ is not large for asymptotic notation. Therefore, the total time complexity to generate a prime number is $O(\log^6 n)$. But β and τ slow down the performance of the Stealth. In this case, communication cost is excluded. Therefore, the total time complexity of Stealth for computing the first shared secret keys are $8\times O(\beta+\tau \log^6 n)+4\times O(\beta)+O(\epsilon) \approx O(\log^6 n)$, and it is a one time cost. The time complexity

for a single communication depends on the random number generator and encryption/decryption. Therefore, the random number generator requires $O(\beta)$ time complexity and let the time complexity of encryption/decryption is $O(\epsilon)$. Therefore, the total time complexity of the first message/communication is $O(\beta) + O(\epsilon) \approx O(1)$. Thus, the overall time complexity depends on the total number of blocks, and hence, it is $O(m + \log^6 n)$ where m is the total communicated blocks.

D. Communication

Table IV demonstrates the communication between $\mathbb{A}$ and $\mathbb{B}$ where $\mathbb{B}$ receives all messages at once. In this case, this communication is similar to Table III and the computation for private keys are in chronological order. The $\mathbb{B}$ can receive the messages in a different order in Table V. The $\mathbb{B}$ receives the second block of message first, and the first block of message at second. In this case, $\mathbb{B}$ need to compute $\mathcal{PK}_1$, $\mathcal{PK}_2$, $\mathcal{PK}_3$, $\mathcal{PK}_4$, $\mathcal{PK}_5$ and $\mathcal{PK}_6$, and stores $\mathcal{PK}_1$, $\mathcal{PK}_2$ and $\mathcal{PK}_3$ for the first block of message to decrypt. The $\mathbb{B}$ cannot compute $\mathcal{PK}_4$, $\mathcal{PK}_5$ and $\mathcal{PK}_6$ without computing $\mathcal{PK}_1$, $\mathcal{PK}_2$, $\mathcal{PK}_3$, i.e., the secret keys are computed in serial order. Therefore, it requires maintaining a logical timestamp to know the messaging order shown in Table V. Message can arrive in any order, and it does not create any issue. Specifically, there is m block of the message, and $\mathbb{B}$ receives the m^{th} first, i.e., the $\mathbb{B}$ receives the message in descending order of sender, $\mathbb{A}$. Then, $\mathbb{B}$ computes all the private keys for the first block of the message to the last block of the message, and stores these generated private keys in sequential order to decrypt the incoming message sent from $\mathbb{A}$. Then, $\mathbb{B}$ can decrypt the last block of the message. The message decryption is interdependent on each other.

Let us consider, $\mathbb{A}$ and $\mathbb{B}$ are communicating and exchanging messages with each other, for instance, chat servers. In this case, $\mathbb{A}$ and $\mathbb{B}$ need to maintain their logical timestamp to get the order of message and calculate the right private keys. Table VI demonstrates the active message exchanging between two parties. The $\mathbb{A}$ and $\mathbb{B}$ exchange the messages. Thus, both $\mathbb{A}$ and $\mathbb{B}$ need to compute the private keys in serial order. The private keys of both $\mathbb{A}$ and $\mathbb{B}$ are the same for the first block of the message. Similarly, the private keys are same for the second block of the message both $\mathbb{A}$ and $\mathbb{B}$, and so on. To decrypt the message from $\mathbb{A}$, $\mathbb{B}$ does not need to recalculate the private keys. Similarly, $\mathbb{A}$ does not need to recalculate the private keys to decrypt the message from $\mathbb{B}$. This implies that $\mathbb{A}$ and $\mathbb{B}$ derives the same privates keys at a given event.

E. Communication and Computation overhead

Stealth maintains the Diffie-Hellman communication protocol and introduces no communication overheads. But there is computation overhead in encryption/decryption. Initially, Stealth computes eight prime numbers, four random numbers. It also performs extra costly operations. Therefore, it is slower than the Diffie-Hellman algorithm to compute shared secret keys. Moreover, the sender or receiver computes private keys in each communication. This process incurs computational overhead. Apparently, the security cannot be compromised at any cost, and thus, these extra computational costs are justifiable.

F. Attacks

The bitwise XOR has interesting properties, and Stealth exploits these properties. XOR produces zero for same values, for instance, $C \oplus C = 0$. Stealth uses XOR to create a cipher. Let, C_1 be the plaintext, and C_2 and C_3 be the key. Let, $\zeta = C_1 \oplus C_2 \oplus C_3$. It requires two keys to retrieve any one key from ζ. For example, $C_1 = \zeta \oplus C_2 \oplus \mathbb{C}_3$, $C_2 = \zeta \oplus C_1 \oplus \mathbb{C}_3$ or $C_3 = \zeta \oplus C_1 \oplus \mathbb{C}_2$. It is highly vulnerable for cryptanalyst in one-keyed XOR-cipher. For instance, $\zeta = C_1 \oplus C_2$. The ζ can be attacked by known-plaintext, chosen-plaintext by cryptanalyst, or frequency analysis. Therefore, three or more keyed XOR operations create difficulties for a cryptanalyst to decode the original message. Therefore, private keys can be increased from four to five for tighter security. However, it can create all ones or zero if many keys are XORed.

1) Brute-force attack: Brute-force attackers attack the encrypted code by an exhaustive search method. It is not impossible to attack any encrypted code, however, it takes many years to decode. Let us assume that an attacker is able to decrypt a code in Stealth using an exhaustive search. However, the attacker requires two extra keys to decode the code. Brute-force attackers need to break the first encryption and retrieved the cipher. Again, the attacker requires two other keys to decipher the coded message. Therefore, brute-force attack does not work on Stealth.

2) Cryptanalysis: Cryptanalyst uses analysis of an encrypted code and discovers the patterns. There are many types of possible cryptanalysis, particularly the known-plaintext attack (KPA), chosen-plaintext attack (CPA) and chosen ciphertext attack (CCA), differential cryptanalysis (DCA), and linear cryptanalysis. For instance, a one-keyed XOR cipher is easy for deciphering by the cryptanalyst. However, Stealth uses a proven existing encryption method and extra XOR operations. Even if the adversaries break the encryption, the encrypted code is still secure due to additional XOR operation. Additionally, it requires two keys to decipher the encrypted code. Moreover, Stealth changes its keys in each block. Therefore, this cryptanalysis attack does not apply in Stealth.

3) Dictionary Attacks: The dictionary attack is famous attacks in password guessing systems where attacker builds a dictionary of possible words to capture the communication. Stealth convert the plaintext into cipher text, then the converted ciphertext into encrypted text. Moreover, it changes private keys for encryption. Therefore, it is not possible to build a dictionary to reveal the original messages by the attackers.

V. Experimental Results

We have evaluated Algorithm 1 and Algorithm 2 for Stealth-TRNG and Stealth-PRNG, respectively, in Ubuntu Desktop environment. The configuration of the experimental environment is is as follows- Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, 8GB RAM, 1TB HDD, Ubuntu 18.04.5 LTS and GCC version 7.5.0. We have generated 10M bits to be tested in NIST SP 800-22 [29], [30]. We have used “5483651” as an input key, two seed values (98899, 104723), and a bit size of 10M for Algorithm 2. The input key can be any number or string. However, Algorithm 1 is not reproducible due to true random number and CPU Clock values.

TABLE IV: Communication Mechanism between $\mathbb{A}$ and $\mathbb{B}$. $\mathbb{B}$ receives all messages in at once.

$\mathbb{A}$	$\mathbb{B}$
$\mathcal{PK}_1 = \text{GENSTEALTHPRNG}(\mathcal{SK}_1, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_2 = \text{GENSTEALTHPRNG}(\mathcal{SK}_2, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_3 = \text{GENSTEALTHPRNG}(\mathcal{SK}_3, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_4 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_2, \beta)$ $\mathbb{M}_1 = \mathcal{M}_1 \oplus \mathcal{PK}_2$ $\mathbb{M}_1^m = \mathbb{M}_1 \oplus \mathcal{PK}_3$ $\mathbb{M}_1^{cm} = Enc^{\mathcal{PK}_1}(\mathbb{M}_1^m)$ Send $(\mathbb{M}_1^{cm}, t_1^{\mathcal{A}})$ to $\mathbb{B}$	
$\mathcal{PK}_5 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_6 = \text{GENSTEALTHPRNG}(\mathcal{PK}_2, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_7 = \text{GENSTEALTHPRNG}(\mathcal{PK}_3, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_8 = \text{GENSTEALTHPRNG}(\mathcal{PK}_5, \mathcal{PK}_6, \beta)$ $\mathbb{M}_2 = \mathcal{M}_2 \oplus \mathcal{PK}_6$ $\mathbb{M}_2^m = \mathbb{M}_1 \oplus \mathcal{PK}_7$ $\mathbb{M}_2^{cm} = Enc^{\mathcal{PK}_5}(\mathbb{M}_2^m)$ Send $(\mathbb{M}_2^{cm}, t_2^{\mathcal{A}})$ to $\mathbb{B}$	
	Receives $(\mathbb{M}_1^{cm}, t_1^{\mathcal{A}})$ from $\mathbb{A}$ $\mathcal{PK}_1 = \text{GENSTEALTHPRNG}(\mathcal{SK}_1, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_2 = \text{GENSTEALTHPRNG}(\mathcal{SK}_2, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_3 = \text{GENSTEALTHPRNG}(\mathcal{SK}_3, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_4 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_2, \beta)$ $\mathbb{M}_1^m = Dec^{\mathcal{PK}_1}(\mathbb{M}_1^{cm})$ $\mathbb{M}_1 = \mathbb{M}_1^m \oplus \mathcal{PK}_3$ $\mathcal{M}_1 = \mathbb{M}_1 \oplus \mathcal{PK}_2$
	Receives $(\mathbb{M}_2^{cm}, t_2^{\mathcal{A}})$ from $\mathbb{A}$ $\mathcal{PK}_5 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_6 = \text{GENSTEALTHPRNG}(\mathcal{PK}_2, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_7 = \text{GENSTEALTHPRNG}(\mathcal{PK}_3, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_8 = \text{GENSTEALTHPRNG}(\mathcal{PK}_5, \mathcal{PK}_6, \beta)$ $\mathbb{M}_2^m = Dec^{\mathcal{PK}_5}(\mathbb{M}_2^{cm})$ $\mathbb{M}_2 = \mathbb{M}_2^m \oplus \mathcal{PK}_7$ $\mathcal{M}_2 = \mathbb{M}_2 \oplus \mathcal{PK}_6$

TABLE V: Communication Mechanism between $\mathbb{A}$ and $\mathbb{B}$. The $\mathbb{B}$ receives all messages in different order.

$\mathbb{A}$	$\mathbb{B}$
$\mathcal{PK}_1 = \text{GENSTEALTHPRNG}(\mathcal{SK}_1, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_2 = \text{GENSTEALTHPRNG}(\mathcal{SK}_2, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_3 = \text{GENSTEALTHPRNG}(\mathcal{SK}_3, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_4 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_2, \beta)$ $\mathbb{M}_1 = \mathcal{M}_1 \oplus \mathcal{PK}_2$ $\mathbb{M}_1^m = \mathbb{M}_1 \oplus \mathcal{PK}_3$ $\mathbb{M}_1^{cm} = Enc^{\mathcal{PK}_1}(\mathbb{M}_1^m)$ Send $(\mathbb{M}_1^{cm}, t_1^{\mathcal{A}})$ to $\mathbb{B}$	
$\mathcal{PK}_5 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_6 = \text{GENSTEALTHPRNG}(\mathcal{PK}_2, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_7 = \text{GENSTEALTHPRNG}(\mathcal{PK}_3, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_8 = \text{GENSTEALTHPRNG}(\mathcal{PK}_5, \mathcal{PK}_6, \beta)$ $\mathbb{M}_2 = \mathcal{M}_2 \oplus \mathcal{PK}_6$ $\mathbb{M}_2^m = \mathbb{M}_1 \oplus \mathcal{PK}_7$ $\mathbb{M}_2^{cm} = Enc^{\mathcal{PK}_5}(\mathbb{M}_2^m)$ Send $(\mathbb{M}_2^{cm}, t_2^{\mathcal{A}})$ to $\mathbb{B}$	
	Receives $(\mathbb{M}_2^{cm}, t_2^{\mathcal{A}})$ from $\mathbb{A}$ $\mathcal{PK}_1 = \text{GENSTEALTHPRNG}(\mathcal{SK}_1, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_2 = \text{GENSTEALTHPRNG}(\mathcal{SK}_2, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_3 = \text{GENSTEALTHPRNG}(\mathcal{SK}_3, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_4 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_2, \beta)$ $\mathcal{PK}_5 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_6 = \text{GENSTEALTHPRNG}(\mathcal{PK}_2, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_7 = \text{GENSTEALTHPRNG}(\mathcal{PK}_3, \mathcal{PK}_4, \beta)$ $\mathcal{PK}_8 = \text{GENSTEALTHPRNG}(\mathcal{PK}_5, \mathcal{PK}_6, \beta)$ $\mathbb{M}_2^m = Dec^{\mathcal{PK}_5}(\mathbb{M}_2^{cm})$ $\mathbb{M}_2 = \mathbb{M}_2^m \oplus \mathcal{PK}_7$ $\mathcal{M}_2 = \mathbb{M}_2 \oplus \mathcal{PK}_6$
	Receives $(\mathbb{M}_1^{cm}, t_1^{\mathcal{A}})$ from $\mathbb{A}$ $\mathbb{M}_1^m = Dec^{\mathcal{PK}_1}(\mathbb{M}_1^{cm})$ $\mathbb{M}_1 = \mathbb{M}_1^m \oplus \mathcal{PK}_3$ $\mathcal{M}_1 = \mathbb{M}_1 \oplus \mathcal{PK}_2$

TABLE VI: Communication Mechanism between $\mathbb{A}$ and $\mathbb{B}$. Both are sending messages to each other.

$\mathbb{A}$	$\mathbb{B}$
$\mathcal{PK}_1 = \text{GENSTEALTHPRNG}(\mathcal{SK}_1, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_2 = \text{GENSTEALTHPRNG}(\mathcal{SK}_2, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_3 = \text{GENSTEALTHPRNG}(\mathcal{SK}_3, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_4 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_2, \beta)$ $\mathbb{M}_1 = \mathcal{M}_1^{\mathcal{A}} \oplus \mathcal{PK}_2$ $\mathbb{M}_1^{m\mathcal{A}} = \mathbb{M}_1^{\mathcal{A}} \oplus \mathcal{PK}_3$ $\mathbb{M}_1^{cm\mathcal{A}} = Enc^{\mathcal{PK}_1}(\mathbb{M}_1^{m\mathcal{A}})$ Send $(\mathbb{M}_1^{cm\mathcal{A}}, t_1^{\mathcal{A}})$ to $\mathbb{B}$	$\mathcal{PK}_1 = \text{GENSTEALTHPRNG}(\mathcal{SK}_1, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_2 = \text{GENSTEALTHPRNG}(\mathcal{SK}_2, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_3 = \text{GENSTEALTHPRNG}(\mathcal{SK}_3, \mathcal{SK}_4, \beta)$ $\mathcal{PK}_4 = \text{GENSTEALTHPRNG}(\mathcal{PK}_1, \mathcal{PK}_2, \beta)$ $\mathbb{M}_1^{\mathcal{B}} = \mathcal{M}_1^{\mathcal{B}} \oplus \mathcal{PK}_2$ $\mathbb{M}_1^{m\mathcal{B}} = \mathbb{M}_1^{\mathcal{B}} \oplus \mathcal{PK}_3$ $\mathbb{M}_1^{cm\mathcal{B}} = Enc^{\mathcal{PK}_1}(\mathbb{M}_1^{m\mathcal{B}})$ Send $(\mathbb{M}_1^{cm\mathcal{B}}, t_1^{\mathcal{B}})$ to $\mathbb{A}$
Receives $(\mathbb{M}_1^{cm\mathcal{B}}, t_1^{\mathcal{B}})$ from $\mathbb{B}$ $\mathbb{M}_1^{m\mathcal{B}} = Dec^{\mathcal{PK}_1}(\mathbb{M}_1^{cm\mathcal{B}})$ $\mathbb{M}_1^{\mathcal{B}} = \mathbb{M}_1^{m\mathcal{B}} \oplus \mathcal{PK}_3$ $\mathcal{M}_1^{\mathcal{B}} = \mathbb{M}_1^{\mathcal{B}} \oplus \mathcal{PK}_2$	Receives $(\mathbb{M}_1^{cm\mathcal{A}}, t_1^{\mathcal{A}})$ from $\mathbb{A}$ $\mathbb{M}_1^{m\mathcal{A}} = Dec^{\mathcal{PK}_1}(\mathbb{M}_1^{cm\mathcal{A}})$ $\mathbb{M}_1^{\mathcal{A}} = \mathbb{M}_1^{m\mathcal{A}} \oplus \mathcal{PK}_3$ $\mathcal{M}_1^{\mathcal{A}} = \mathbb{M}_1^{\mathcal{A}} \oplus \mathcal{PK}_2$

TABLE VII: P-values and success rates of Algorithms 1 for 32, 64 and 128 bits in NIST SP 800-22.

Test name	32 bits		64 bits		128 bits	
	P-value	Pass rate	P-value	Pass rate	P-value	Pass rate
Approximate Entropy	0.468595	31/32	0.350485	64/64	0.875539	128/128
Frequency	0.468595	31/32	0.911413	62/64	0.739918	127/128
Block Frequency	0.122325	32/32	0.100508	64/64	0.723129	126/128
Cumulative sums	0.178278, 0.407091	32/32, 32/32	0.012043, 0.162606	62/64, 62/64	0.350485, 0.654467	128/128, 127/128
Runs	0.253551	31/32	0.035174	64/64	0.134686	127/128
Longest runs	0.739918	31/32	0.378138	61/64	0.585209	127/128
Rank	0.739918	32/32	0.671779	64/64	0.015065	128/128
FFT	0.035174	32/32	0.324180	64/64	0.422034	127/128
Non-overlapping Template	0.976060	32/32	0.991468	64/64	0.941144	128/128
Overlapping Template	0.468595	31/32	0.500934	61/64	0.452799	128/128
Random Excursions	0.437274	11/11	0.834308	15/15	0.739918	12/12
Random Excursions Variant	0.834308	11/11	0.637119	15/15	0.911413	12/12
Serial	0.671779, 0.804337	32/32, 32/32	0.378138, 0.637119	64/64, 64/64	0.017912, 0.517442	127/128, 128/128
Linear complexity	0.407091	32/32	0.195163	64/64	0.772760	127/128
Universal	0.534146	31/32	0.148094	64/64	0.407091	127/128

TABLE VIII: Comparison of Algorithms 2 for 32, 64 and 128 bits in NIST SP 800-22.

Test name	32 bits		64 bits		128 bits	
	P-value	Pass rate	P-value	Pass rate	P-value	Pass rate
Approximate Entropy	0.016990	32/32	0.468595	62/64	0.788728	127/128
Frequency	0.949602	31/32	0.931952	63/64	0.337162	127/128
Block Frequency	0.949602	31/32	0.299251	63/64	0.392456	127/128
Cumulative sums	0.468595, 0.035174	31/32, 31/32	0.232760, 0.602458	63/64, 63/64	0.324180, 0.128379	126/128, 126/128
Runs	0.671779	31/32	0.213309	64/64	0.834308	128/128
Longest runs	0.253551	31/32	0.862344	64/64	0.110952	126/128
Rank	0.534146	32/32	0.500934	64/64	0.066882	128/128
FFT	0.407091	32/32	0.671779	64/64	0.100508	128/128
Non-overlapping Template	0.991468	32/32	0.995711	64/64	0.941144	128/128
Overlapping Template	0.468595	31/32	0.378138	63/64	0.253551	128/128
Random Excursions	0.637119	13/13	0.739918	12/12	0.162606	15/15
Random Excursions Variant	0.437274	13/13	0.739918	12/12	0.637119	15/15
Serial	0.350485, 0.804337	32/32, 32/32	0.911413, 0.949602	62/64, 63/64	0.819544, 0.517442	127/128, 128/128
Linear complexity	0.213309	32/32	0.350485	64/64	0.311542	125/128
Universal	0.100508	32/32	0.804337	63/64	0.484646	126/128

Table VII and Table VIII demonstrate the P-values and the success rate of random number testing on NIST SP 800-22 [29], [30] for Algorithm 1 and Algorithm 2, respectively. NIST SP 800-22 provides statistical testing of the randomness of given bits. We have generated 10M random bits and tested them using NIST SP 800-22 test. It provides approximate entropy, frequency, block frequency, cumulative sums, runs, longest runs, rank, FFT, non-overlapping template, overlapping template, random excursions, random excursions variant, serial, linear complexity, and universal testing for bits' randomness. The deciding factor of P-value is ≥ 0.01, otherwise, the given bits are not random. It may contain certain patterns that can easily be identified and discover the generated numbers' pattern by adversaries. Table VII and Table VIII shows the corresponding P-values, and it shows quite satisfactory results on the randomness test for the generated bits by Algorithm 1 and Algorithm 2, respectively. The minimum success rate of Algorithm 1 is 0.96875, 0.96875 and 0.984375 for 32,

64 and 128 bit streams, respectively. The maximum success rate of Algorithm 1 is 1 for all bit streams. The lowest P-value of Algorithm 1 are 0.035174, 0.012043, 0.017912 in 32, 64 and 128 bit streams, respectively. The highest P-values are 0.976060, 0.991468 and 0.941144 in 32, 64, and 128 bit streams respectively. Also, Algorithm 2 test's success rates are as low as 0.96875, 0.96875, and 0.9765625 in 32, 64, and 128 bit streams, respectively. The highest success rate is 1 (100%) for all bit streams. The lowest P-values of 32, 64, and 128 bit streams are 0.016990, 0.213309, and 0.066882, respectively. The highest P-values of 32, 64, and 128 bit streams are 0.991468, 0.995711, and 0.941144, respectively.

A. Comparison for randomness

Stealth-PRNG produces random numbers, and it passes its statistical test. However, it is compared with other state-of-the-art random number generators for randomness. Tables IX, X and XI compare the Stealth-TRNG and Stealth-PRNG with Erozan *et al.* [22], Koyuncu *et al.* [31], Jiang *et al.* [21], Johnson *et al.* [32], Wieczorek and Golofit [33] and Yeoh *et al.* [26]. Stealth-TRNG shows the highest success rate of 1 and the lowest success rate of 0.984375. The maximum P-value is 0.949602 and minimum is 0.015065. Similarly, Stealth-PRNG exhibits the highest success rate of 1 and the lowest success rate of 0.9765625. The highest P-value of Stealth-PRNG is 0.941144 and the lowest PRNG is 0.066882. The highest success rate of Erozan *et al.* [22], Jiang *et al.* [21], Johnson *et al.* [32], Wieczorek and Golofit [33] and Yeoh *et al.* [26] are 1, 1, 1, 0.99 and 0.995, respectively. Likewise, the lowest success rate of Erozan *et al.* [22], Jiang *et al.* [21], Johnson *et al.* [32], Wieczorek and Golofit [33] and Yeoh *et al.* [26] are 0.96, 0.973684211, 0.8, 0.98, and 0.984, respectively. The highest P-values of Erozan *et al.* [22], Koyuncu *et al.* [31], Jiang *et al.* [21], Johnson *et al.* [32], Wieczorek and Golofit [33] and Yeoh *et al.* [26] are 0.494555, 0.99834, 0.795464, 0.9114, 0.92, and 0.9966685, respectively. Similarly, the lowest P-value of Erozan *et al.* [22], Koyuncu *et al.* [31], Jiang *et al.* [21], Johnson *et al.* [32], Wieczorek and Golofit [33] and Yeoh *et al.* [26] are 0.055361, 0.01101, 0.000954, 0.0043, 0.08, and 0.122325, respectively. Therefore, the overall randomness of Stealth-TRNG and Stealth-PRNG are entirely satisfactory as compared to state-of-the-art random number generators. However, hardware-based random number generators are relatively faster than Stealth-PRNG.

P-value determines the randomness of the generated bits. A high P-value indicates high-quality randomness. Both PRNG 2 and TRNG 1 generate highly unpredictable, truly random numbers; however, a random number can be reproduced generated by PRNG if we know the initial input but not true for the TRNG. TRNG produces a highly unpredictable random number that is not possible to reproduce by any means.

VI. Conclusions

In this article, we have demonstrated secured symmetric communication between two endpoints, called Stealth. Stealth provides dynamic security in symmetric communication. It neither replaces any existing methodology of key exchange protocol nor encryption method but it creates another security layer to protect from various kinds of attacks. Also, Stealth is not designed to deal with DDoS attacks and MITM. Stealth creates a secure coating on a raw message for communication. Initially, it depends on Diffie-Hellman algorithms to compute the shared secret keys. These shared secret keys are altered for the blocks of the message to communicate. Stealth uses existing version of AES cryptography. Stealth creates another layer to provide tight security for secured symmetric communication. However, it adds additional computational overhead to the system, but security is intact. We have also demonstrated PRNG and TRNG for Stealth, called Stealth-PRNG and Stealth-TRNG, which provides a truly random number to protect from various attackers. Both Stealth-PRNG and Stealth-TRNG are tested in NIST SP 800-22 for randomness and are able to pass all the 15 statistical testings for randomness. We have also compared Stealth-PRNG and Stealth-TRNG with state-of-the-art random number generators, and are able to outperform the existing algorithm in randomness. In addition, we have also explored security measurement mathematically. The probability of getting correct shared secret key is $\frac{1}{16^\beta}$ and getting the correct private key is $\frac{1}{16^\beta}$. Also, the probability of getting all the correct messages or seize the entire communication without knowing the keys is $(\frac{1}{8^\beta})^m$ for m blocks of a message in communication, for instance, block cipher. The patterns of the blocks of a message cannot be revealed at any cost due to the different private keys in each communication, which protects from the cryptanalysis attacks.

References

[1] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[2] R. for Pair-Wise Key Establishment Schemes UsingDiscrete Logarithm Cryptography, "Elaine barker and lily chen and allen roginsky and miles smid," Accessed on January 2021 from https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-56ar.pdf, 2007.

[3] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology — CRYPTO '85 Proceedings*, H. C. Williams, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 417–426.

[4] S. G. Stubblebine and C. A. Meadows, "Formal characterization and automated analysis of known-pair and chosen-text attacks," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, pp. 571–581, 2000.

[5] Y. Zhu, "Attack pattern discovery in forensic investigation of network attacks," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 7, pp. 1349–1357, 2011.

[6] J. Black and H. Urtubia, "Side-channel attacks on symmetric encryption schemes: The case for authenticated encryption." in *USENIX Security Symposium*, 2002, pp. 327–338.

[7] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.

[8] L. Bossuet, M. Grand, L. Gaspar, V. Fischer, and G. Gogniat, "Architectures of flexible symmetric key crypto engines—a survey: From hardware coprocessor to multi-crypto-processor system on chip," *ACM Comput. Surv.*, vol. 45, no. 4, Aug. 2013. [Online]. Available: https://doi.org/10.1145/2501654.2501655

[9] D. Khovratovich, G. Leurent, and C. Rechberger, "Narrow-bicliques: Cryptanalysis of full idea," in *Advances in Cryptology – EUROCRYPT 2012*, D. Pointcheval and T. Johansson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 392–410.

[10] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann, "Imperfect forward secrecy: How diffie-hellman fails in practice," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 5–17. [Online]. Available: https://doi.org/10.1145/2810103.2813707

TABLE IX: Comparison of Stealth-TRNG and Stealth-PRNG with other algorithms in successful randomness testing in NIST SP 800-22.

Test name	Stealth-TRNG		Stealth-PRNG		Erozan *et al.*[22]		Koyuncu *et al.*[31]	
	P-value	Pass rate	P-value	Pass rate	P-value	Pass rate	P-value	Pass rate
Approximate Entropy	0.875539	128/128	0.788728	127/128	–	–	0.15224	Successful
Frequency	0.739918	127/128	0.337162	127/128	0.202268	96/100	0.72184	Successful
Block Frequency	0.723129	126/128	0.392456	127/128	0.213309	100/100	0.06380	Successful
Cumulative sums	0.350485	128/128	0.324180	126/128	0.428568	96/100	0.56254	Successful
Runs	0.134686	127/128	0.834308	128/128	0.171867	99/100	0.06380	Successful
Longest runs	0.585209	127/128	0.110952	126/128	–	–	0.19640	Successful
Ranks	0.015065	128/128	0.066882	128/128	–	–	0.99834	Successful
FFT	0.422034	127/128	0.100508	128/128	0.474986	98/100	0.12786	Successful
Non-overlapping Template	0.949602	128/128	0.941144	128/128	–	–	0.69314	Successful
Overlapping Template	0.452799	128/128	0.253551	128/128	0.055361	99/100	0.90598	Successful
Random Excursions	0.739918	12/12	0.162606	15/15	–	–	0.86541	Successful
Random Excursions Variant	0.911413	12/12	0.637119	15/15	–	–	0.35789	Successful
Serial	0.517442	128/128	0.819544	127/12	0.494555	100/100	0.87105	Successful
Linear complexity	0.772760	127/128	0.311542	125/128	0.249284	97/100	0.01101	Successful
Universal	0.407091	127/128	0.484646	126/128	–	–	0.02262	Successful

TABLE X: Comparison of Stealth-TRNG and Stealth-PRNG with other algorithms in successful randomness testing in NIST SP 800-22.

Test name	Stealth-TRNG		Stealth-PRNG		Jiang *et al.*[21]		Johnson *et al.*[32]	
	P-value	Pass rate	P-value	Pass rate	P-value	Pass rate	P-value	Pass rate
Approximate Entropy	0.875539	128/128	0.788728	127/128	0.00983	75/76	0.7399	1.0
Frequency	0.739918	127/128	0.337162	127/128	0.477737	74/76	0.9114	1.0
Block Frequency	0.723129	126/128	0.392456	127/128	0.768138	75/76	0.9114	1.0
Cumulative sums	0.350485	128/128	0.324180	126/128	0.426525	74/76	0.3505	1.0
Runs	0.134686	127/128	0.834308	128/128	0.042413	75/76	0.0089	0.92
Longest runs	0.585209	127/128	0.110952	126/128	0.042413	76/76	0.7400	1.0
Rank	0.015065	128/128	0.066882	128/128	0.094936	76/76	0.0043	1.0
FFT	0.422034	127/128	0.100508	128/128	0.739918	75/76	0.0089	1.0
Non-overlapping Template	0.949602	128/128	0.941144	128/128	–	11052/11248	0.0043	1.0
Overlapping Template	0.452799	128/128	0.253551	128/128	0.5929591	75/76	0.0213	0.8
Random Excursions	0.739918	12/12	0.162606	15/15	–	360/368	–	–
Random Excursions Variant	0.911413	12/12	0.637119	15/15	–	818/828	–	–
Serial	0.517442	128/128	0.819544	127/128	0.795464	76/76	0.5341	1.0
Linear complexity	0.772760	127/128	0.311542	125/128	0.350485	76/76	0.9114	1.0
Universal	0.407091	127/128	0.484646	126/128	0.000954	76/76	–	–

TABLE XI: Comparison of Stealth-TRNG and Stealth-PRNG with other algorithms in successful randomness testing in NIST SP 800-22.

Test name	Stealth-TRNG		Stealth-PRNG		Wieczorek and Golofit [33]		Yeoh *et al.*[26]	
	P-value	Pass rate	P-value	Pass rate	P-value	Pass rate	P-value	Pass rate
Approximate Entropy	0.875539	128/128	0.788728	127/128	0.49	0.98	0.647530	0.995
Frequency	0.739918	127/128	0.337162	127/128	0.55	0.99	0.516113	0.988
Block Frequency	0.723129	126/128	0.392456	127/128	0.08	0.99	0.928857	0.993
Cumulative sums	0.350485	128/128	0.324180	126/128	0.27	0.98	0.572847	0.989
Runs	0.134686	127/128	0.834308	128/128	0.92	0.99	0.122325	0.991
Longest runs	0.585209	127/128	0.110952	126/128	0.34	0.99	0.291091	0.985
Rank	0.015065	128/128	0.066882	128/128	0.68	0.99	0.530120	0.995
FFT	0.422034	127/128	0.100508	128/128	0.82	0.99	0.858002	0.990
Non-overlapping Template	0.949602	128/128	0.941144	128/128	0.81	0.99	0.743915	0.99
Overlapping Template	0.452799	128/128	0.253551	128/128	0.21	0.99	0.502247	0.984
Random Excursions	0.739918	12/12	0.162606	15/15	0.48	0.98	0.292960	0.9863
Random Excursions Variant	0.911413	12/12	0.637119	15/15	0.34	0.99	0.9966685	0.9893
Serial	0.517442	128/128	0.819544	127/128	0.79	0.99	0.402962	0.988
Linear complexity	0.772760	127/128	0.311542	125/128	0.44	0.99	0.433590	0.984
Universal	0.407091	127/128	0.484646	126/128	0.72	0.99	0.373625	0.984

[11] E. Bresson, O. Chevassut, and D. Pointcheval, "Provably secure authenticated group diffie-hellman key exchange," *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 3, p. 10–es, Jul. 2007. [Online]. Available: https://doi.org/10.1145/1266977.1266979

[12] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[13] M. Bakiri, C. Guyeux, J.-F. Couchot, and A. K. Oudjida, "Survey on hardware implementation of random number generators on fpga: Theory and experimental analyses," *Computer Science Review*, vol. 27, pp. 135 – 153, 2018.

[14] D. Evtyushkin and D. Ponomarev, "Covert channels through random number generator: Mechanisms, capacity estimation and mitigations," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 843–857.

[15] M. Alcin, I. Koyuncu, M. Tuna, M. Varan, and I. Pehlivan, "A novel high speed artificial neural network–based chaotic true random number generator on field programmable gate array," *International Journal of*

Circuit Theory and Applications, vol. 47, no. 3, pp. 365–378, 2019.
[16] İsmail Koyuncu and A. Turan Özcerit, "The design and realization of a new high speed fpga-based chaotic true random number generator," *Computers & Electrical Engineering*, vol. 58, pp. 203 – 214, 2017.
[17] X. Lin, S. Wang, Z.-Q. Yin, G.-J. Fan-Yuan, R. Wang, W. Chen, D.-Y. He, Z. Zhou, G.-C. Guo, and Z.-F. Han, "Security analysis and improvement of source independent quantum random number generators with imperfect devices," *npj Quantum Information*, vol. 6, no. 1, pp. 1–8, 2020.
[18] M. Stipcevic and R. Ursin, "An on-demand optical quantum random number generator with in-future action and ultra-fast response," *Scientific Reports*, vol. 5, no. 1, pp. 1–8, 2015.
[19] Y. Liu, Q. Zhao, M.-H. Li, J.-Y. Guan, Y. Zhang, B. Bai, W. Zhang, W.-Z. Liu, C. Wu, X. Yuan, H. Li, W. J. Munro, Z. Wang, L. You, J. Zhang, X. Ma, J. Fan, Q. Zhang, and J.-W. Pan, "Device-independent quantum random-number generation," *Nature*, vol. 562, no. 7728, p. 548–551, 2018.
[20] M. Avesani, D. G. Marangon, G. Vallone, and P. Villoresi, "Source-device-independent heterodyne-basedquantum random number generator at 17 gbps," *Nature Communications*, vol. 9, no. 1, p. 5365, 1–7.
[21] H. Jiang, D. Belkin, S. E. Savel'ev, S. Lin, Z. Wang, Y. Li, S. Joshi, R. Midya, C. Li, M. Rao, M. Barnell, Q. Wu, J. J. Yang, and Q. Xia, "A novel true random number generator based on a stochastic diffusive memristor," *Nature Communications*, vol. 8, no. 1, pp. 1–9, 2017.
[22] A. T. Erozan, G. Y. Wang, R. Bishnoi, J. Aghassi-Hagmann, and M. B. Tahoori, "A compact low-voltage true random number generator based on inkjet printing technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 6, pp. 1485–1495, 2020.
[23] K. Lee, S. Lee, C. Seo, and K. Yim, "Trng (true random number generator) method using visible spectrum for secure communication on 5g network," *IEEE Access*, vol. 6, pp. 12 838–12 847, 2018.
[24] P. Bierhorst, E. Knill, S. Glancy, Y. Zhang, A. Mink, S. Jordan, A. Rommal, Y.-K. Liu, B. Christensen, S. W. Nam, M. J. Stevens, and L. K. Shalm, "Experimentally generated randomness certified by the impossibility of superluminal signals," *Nature*, vol. 556, no. 7700, p. 223–226, 2018.
[25] C. Camara, P. Peris-Lopez, H. Martin, and M. Aldalaien, "ECG-RNG: a random number generator based on ecg signals and suitable for securing wireless sensor networks," *Sensors*, vol. 18, no. 9, pp. 1–15, 2018.
[26] W.-Z. Yeoh, J. S. Teh, and H. R. Chern, "A parallelizable chaos-based true random number generator based on mobile device cameras for the android platform," *Multimedia Tools and Applications*, vol. 78, no. 12, pp. 15 929–15 949, 2019.
[27] A. Appleby, "Murmurhash," Retrieved on December 2020 from https://sites.google.com/site/murmurhash/, 2008.
[28] M. Agrawal, N. Kayal, and N. Saxena, "Primes is in p," *Annals of Mathematics*, vol. 160, no. 2, pp. 781–793, 2004.
[29] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Booz-allen and hamilton inc mclean va, Tech. Rep., 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf
[30] L. E. Bassham III, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks *et al.*, *SP 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications*. National Institute of Standards & Technology, 2010. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final
[31] İ. Koyuncu, M. Tuna, İ. Pehlivan, C. B. Fidan, and M. Alçın, "Design, fpga implementation and statistical analysis of chaos-ring based dual entropy core true random number generator," *Analog Integrated Circuits and Signal Processing*, vol. 102, no. 2, pp. 445–456, 2020.
[32] A. P. Johnson, R. S. Chakraborty, and D. Mukhopadyay, "An improved dcm-based tunable true random number generator for xilinx fpga," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 4, pp. 452–456, 2017.
[33] P. Z. Wieczorek and K. Gołofit, "True random number generator based on flip-flop resolve time instability boosted by random chaotic source," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 4, pp. 1279–1292, 2018.

Ripon Patgiri (ripon@cse.nits.ac.in) is currently working as an Assistant Professor in the Department of Computer Science & Engineering, National Institute of Technology Silchar, Assam-788010, India. He receives his Master Degree and PhD degree from Indian Institute of Technology Guwahati and National Institute of Technology Silchar, respectively. His research interests are Bloom Filter, Network Security, Privacy, and Networking. He has published several research papers in reputed journals, conferences and books. Also, he has published several edited books. He is a senior member of IEEE, member of EAI, associate member of IETE, and life member of ACCS. He serves as a General Chair, Program Chair and Organizing Chair of several conferences. URL: http://cs.nits.ac.in/rp