SecretStore: A Secrecy as a Service model to enable the Cloud Storage to store user's secret data

Ripon Patgiri, *Senior Member, IEEE*
National Institute of Technology Silchar

***Abstract*—Data secrecy is a major concern in many domains. Nowadays, the data are kept in tight security with high privacy. Users do not want to share their secret information with anyone; however, the users' confidential data are not protected from the administrators. Administrators can read the users' data. Why should any Administrator read users' data? To address this issue, we propose a new secrecy protocol to store data secretly, named Secret Cloud Storage, SecretStore for short, to enable Secrecy as a Service model over the Cloud Computing paradigm. This article demonstrates how to protect users' data from any unintended users, including the data administrators. Moreover, we introduce tight security using the client-side symmetric cryptography method. In addition, we devise a forgetful private key to generate or regenerate a private key to encrypt or decrypt based on a secret word. We also show how to strengthen the weak password. Finally, we demonstrate how to implement the Secrecy as a Service model in Cloud Storage using highly unpredictable private keys.**

***Index Terms*—Cloud Storage, Secret store, Security, Privacy, Secrecy, Encryption, Cryptography, Cloud Service, Random Number.**

I. Introduction

Security and privacy are getting much attention due to emerging technologies [1], [2], for instance, Cloud Computing [3], [4], IoT [5], Big Data [6], Healthcare [7] etc., and these emerging technologies pose new challenges to overcome. Wang *et al.* [8] presents an empirical study on data security in Cloud Computing. Huang *et al.* [9] presents a framework, called SSTreasury+, to store encrypted data. Similarly, Wei *et al.* [10] provable and secure cloud storage techniques. Moreover, Itani *et al.* [11] presents privacy as a service model. Security, secrecy, and privacy (see Figure 2) become the paramount concern for users, vendors, and countries. Moreover, there are numerous issues in cloud security, and privacy [4]. However, secrecy is a prominent issue in cloud computing. Secrecy on password raises a question: "why should someone see any password in raw form or encrypted form [12]?" PassDB [12] suggests that a strict privacy protocol is required to implement in identity management systems [12]. The identity manager should not map user ID and password together [12]. Thus, an identity manager establishes secrecy. A similar question arises: why should others, including administrators read the clients' data? This question poses a new direction on data privacy. The cloud service providers store the clients' data, and the clients trust the cloud service providers. Figure 1 demonstrates the conventional Cloud Storage architecture, where the Storage as a Service model runs on users' trust. For instance, the cloud administrator can read the users' data, for instance, Google Drive, OneDrive, Dropbox, iCloud, to name a few. Therefore, it demands a new cloud computing model where user can store their secret data, and those secret data remains secret after uploading into the cloud storage. Moreover, it demands secure storage even if the attackers hack the cloud storage server.

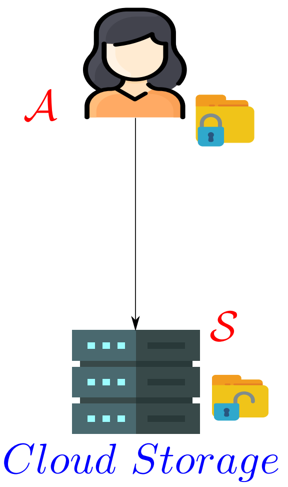

Fig. 1. Conventional Cloud Storage Architecture. User uploads the data and Server store its data. Server can decrypt the data if necessary.

Secrecy is maintaining secret information with intended users. It is classified into two categories: hard secrecy and soft secrecy. Hard secrecy ensures the users' data protection because they do not want to share the information with anyone as defined in Definition 1. In contrast, soft secrecy shares the user's secrecy with its intended users but not the administrator. as defined in Definition 2. Detailed description is given in Section II.

In this article, we address hard secrecy, which implements the "only me" philosophy to implement Secret Cloud Storage. We propose a novel Cloud Service model, called Secret Cloud Storage, SecretStore for short, which implements Secrecy as a Service and follows hard secrecy. It enables the users' to store their most sensitive data in Cloud without worrying about data secrecy and data privacy. Our proposed system, SecretStore, is a cloud storage model where data are stored at the server-side. SecretStore uses the existing cryptography technology for communication. The client converts its data to cipher form for storing at the server-side. The client uses

Ripon Patgiri, Department of Computer Science & Engineering, National Institute of Technology Silchar, Assam-788010, India, e-mail: ripon@cse.nits.ac.in and URL: http://cs.nits.ac.in/rp/
Manuscript received Month XX, 20XX; revised Month XX, 20XX.

symmetric cryptography, which is described in later sections. The client's ciphered data is encrypted using the shared secret key between the client and server, and the encrypted data are sent to the server over insecure media. The server decrypts the data using the shared secret key. The server decrypts the encrypted data and retrieved ciphertext to store in its database. The server cannot retrieve the original data from the ciphertext because it is encrypted by the user's private key, which is not known by the server. Thus, the server stores encrypted data in its database, and the data are not allowed to access by anyone except the data owner itself. However, there is a significant challenge in maintaining private keys in our proposed system. A user cannot maintain its private keys permanently, like a server. A user can switch its platform frequently. Moreover, a user's device can be damaged or lost at any time. Therefore, we propose a regenerative private key method. A private key can be generated or regenerated using a secret word (password). However, the password-based solution has a weakness of weak passwords. Therefore, we demonstrate how to strengthen the weak password using a secret word and a number. This method provides highly unpredictable and cryptographically secured private keys. We illustrate the randomness of the generated private keys using the NIST SP 800-22 statistical test suite. Thus, our major challenge has been achieved to implement SecretStore. Therefore, SecretStore presents "Secret as a Service", "Secrecy as a Service" (SaaS), or "User Secrecy as a Service" (USaaS). To the best of our knowledge, there is no cloud storage available to store the users' secret data and protect it from all users, including administrators and attackers. Therefore, SecretStore is the first model to provide SaaS or USaaS, which prevents accessing the users' data from the administrators and the attacks. There are diverse data privacy works have been proposed; however, the data are controlled by the Cloud administrators. Therefore, it distinguishes state-of-the-art Cloud Storage service with our proposed model.

In this article, we discuss the essential techniques in Section II for our proposed systems, and Section III establishes the proposed method. Section IV presents the detailed analysis of our proposed system. Section V demonstrates the experimental evaluation of the proposed regenerative private key. Finally, we conclude the article in Section VII.

II. Preliminary

Client's data are compromised even if the service provider ensures the highest level of security in Cloud Computing. There are no security, privacy, and secrecy in client's data for administrators. Alternatively, the administrators are the valid "adversaries" [13]. Administrators can do what they want to do with client's data. These data have no security at all at the administrator-side. For example, Alice and Bob are chatting on a chat platform, and the administrator is recording their conversation. Here, the administrators are the valid Mallory while there is no valid or invalid Mallory. Mallory is an adversary at all. The cloud storage claims that the data are highly secured but only for communications and other users. It does not include the "valid and employed" Mallory at the server-side [13].

A. Secrecy

Definition 1. *Secrecy is data hiding from other users those who need not know.*

Definition 2. *Absolute security or hard secrecy is a security protocol that follows the "only me" philosophy, and the data are protected from adversaries, administrators, and any other entity.*

Definition 3. *Soft secrecy is a security protocol designed to maintain secrecy in a intended groups. It excludes adversaries, administrators, and any other entity from the intended parties.*

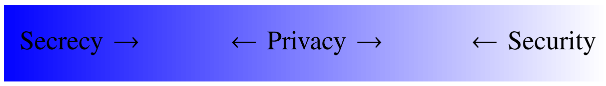

Fig. 2. The relation among security, privacy, and secrecy.

Secrecy is a state of being not shared any information with anyone [13]. Secrecy is classified into two key categories, namely, soft and hard secrecy which are defined in Definition 1, 2 and 3. Figure 2 shows the relation among security, privacy, and secrecy. The dark blue color represents hard secrecy on the left side of the bar, while the light color represents security without ensuring privacy and secrecy on the right side. In the middle of the bar, it represents privacy. The darkest shade of the secrecy represents the hard secrecy, and the lighter color toward the right side represents soft secrecy. Similarly, the darkest shade represents the highest security, and the lightest shade represents security without privacy and secrecy. Therefore, the lighter color of secrecy is equivalent to privacy, and the lighter color of privacy is equal to security. Alternatively, the darker color of privacy is secrecy.

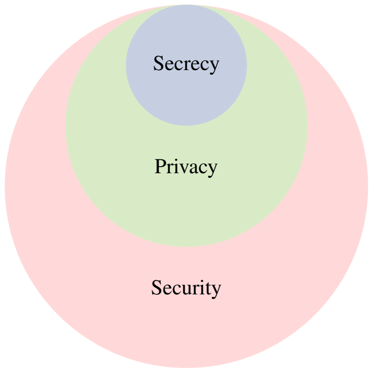

Fig. 3. Representation of the relation among Security, Secrecy, and Privacy using Venn diagram.

Figure 3 represents the relation among security, privacy, and secrecy using a Venn diagram. Secrecy is a proper subset of privacy and security. Likewise, privacy is a proper subset of security. Therefore, secrecy is the hardest security. Secrecy is classified into two categories, namely, soft and hard secrecy. Soft secrecy maintains information secret within a few members. A user decides who can see the data but not administrators. On the contrary, hard secrecy follows the philosophy of "only me" or "only by my choice". Hard secrecy never allows any unintended users to read its data, for instance, password data or other sensitive data. Therefore, secrecy is another challenge for SecretStore to establish between sender and receiver.

III. SecretStore- The proposed system

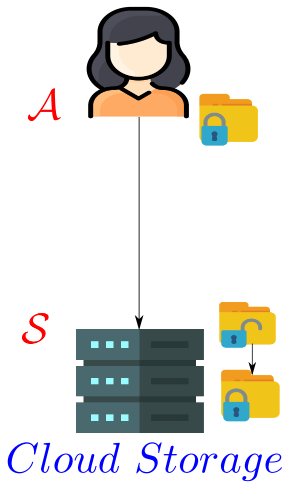

Fig. 4. The architecture of SecretStore. User uploads data using two-layered encryption where the server decrypts the first layer but cannot decrypts the second layer encryption. Therefore, the server stores the cipher data in its database.

People store their data in cloud storage for secure and permanent storage purposes. Cloud storage ensures that data are available even if a disaster happens, and thus, cloud storage is the safest place to store clients' data. It also protects the data from attackers. However, the data are unprotected from the administrators as shown in Figure 1. Therefore, our proposed system, SecretStore, provides Secrecy as a Service (SaaS) or User Secret as a Service (USaaS) model. User (here, we analogously use user and client) can upload their data to the cloud for secure and permanent storage purposes. The data are encrypted at the client-side to ensure hard secrecy. The server stores the users' encrypted data, but the server cannot decrypt the raw data which is depicted in Figure 4. Therefore, the server stores the encrypted data in its database. Moreover, the server cannot scan viruses due to encrypted data by the user. The key objective of the proposed system is to provide tight security by applying access restrictions to all other users, including administrators and attackers. Therefore, a user can store the most sensitive data at SecretStore. It provides full secrecy with tight security on the data. Data can be accessed exclusively by the owner, and all other users (including the administrators and attackers) are restricted from accessing the data. Thus, we require a new methodology to store the encrypted data because encryption requires a private key at the client-side, and a user cannot maintain the private key permanently. Therefore, we propose a regenerative private key technique to encrypt the data at the client-side and upload the encrypted data in cloud storage.

A. *Our assumptions*

The server maintains the identity of its users through LDAP. However, LDAP maintains no secrecy. Therefore, recent work suggested that LDAP can be implemented by maintaining full anonymity [12]. User identity data are the most sensitive information, and these data should be taken utmost care to provide full confidentiality. We assume an identity management system, and these identity management systems ensure a user's validness. Therefore, we omit the detailed analysis of identity management systems and omit the detailed analysis of the connection between user and server. Our proposed system relies on the Elliptic-curve Diffie-Hellman (ECDH) key exchange protocol [14], and it is already a well-established protocol. Therefore, we omit the detailed analysis of the key exchange protocol. For symmetric cryptography, we use AES [15] and the detailed analysis is also omitted.

B. *Generation of private key*

The key storage is the grand challenge for the user because a user cannot store its key permanently in its own devices. The device may be damaged or lost at any time, and it is highly unpredictable. Even a user can switch its platform frequently. Thus, a user requires a regenerative private key. Therefore, the necessary condition for a private key is that it should be reproducible, unpredictable, and cryptographically secure. Initially, a user needs to compute its private keys using Algorithm 1 and convert it into a prime number. The IsPrime invokes AKS [16] algorithm to check whether a given number is prime or not. The primality check walks towards the nearest prime number using AKS algorithm. This requires a time complexity of $O(log^6 n)$ since prime numbers are not rare. A user would like to encrypt its data using two keys; then, the user needs to generate two private keys, i.e., a user can choose its level of encryption t.

Algorithm 1 Algorithm to generate pseudo-random key based on initial input.

```
1:  procedure GenKey($key$, $\beta$, $seed$)
2:      $j$ = Length($key$)
3:      while $i \geq \beta$ do
4:          $d$ = Murmur2($key, j, seed$)
5:          $seed = d$
6:          $e$ = Murmur2($key, j, seed$)
7:          $seed = e$
8:          $bin[i] = (d \wedge 1)$
9:      end while
10:     $P$ = ConvertToDecimal($bin$, $\beta$)
11:     $flag = false$
12:     while $flag = false$ do
13:         $flag$ = IsPrime($P$)
14:         $P = P + 1$
15:     end while
16: end procedure
```

Algorithm 1 can reproduce a previously generated private key for correct input. It generates an unpredictable and cryptographically secure random number. Therefore, Algorithm 1 takes three inputs, specifically, key, β and $seed$ where the key is a secret word, β is the bit length of the private key to be generated, and $seed$ is the initial value for the hash function.

Algorithm 1 iterates β times to generate a random number based on the initial input. The least significant bit (LSB) is extracted in each iteration. The LSB bits are recorded and used to produce a key. The key size may vary depending on the requirements, for instance, $16 \le \beta \le 2048$.

C. Insertion

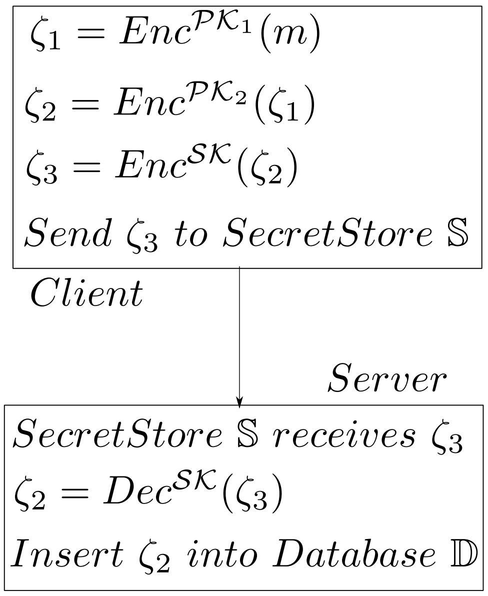
$\zeta_1 = Enc^{\mathcal{PK}_1}(m)$
$\zeta_2 = Enc^{\mathcal{PK}_2}(\zeta_1)$
$\zeta_3 = Enc^{\mathcal{SK}}(\zeta_2)$
Send ζ_3 to SecretStore $\mathbb{S}$
Client
Server
SecretStore $\mathbb{S}$ receives ζ_3
$\zeta_2 = Dec^{\mathcal{SK}}(\zeta_3)$
Insert ζ_2 into Database $\mathbb{D}$

Fig. 5. Insertion process of SecretStore as client-server model.

Figure 5 demonstrates the uploading of user data through a network. A user is asked to input t, depending on the $t \ge 1$ the private keys are generated. The message is converted into integer m for encryption. Let the private key be the $\mathcal{PK}_i$ generated by Algorithm 1 where $1 \le i \le t$. The user encrypts the message using t private keys as given in Equation (1).

$$\begin{aligned}\zeta_1 &= Enc^{\mathcal{PK}_1}(m)\\ \zeta_2 &= Enc^{\mathcal{PK}_2}(\zeta_1)\\ \zeta_3 &= Enc^{\mathcal{PK}_3}(\zeta_2)\\ &\vdots\\ \zeta_t &= Enc^{\mathcal{PK}_t}(\zeta_{(t-1)})\end{aligned} \tag{1}$$

Equation (1) converts the raw message to ciphertext. Now, the user and server compute the shared secret key using ECDH [14], and let it be $\mathcal{SK}$. The user's ciphertexts are encrypted using a shared secret key as given in (2). Therefore, the client encrypts the message using a shared secret key to send it to the server.

$$\zeta = Enc^{\mathcal{SK}}(\zeta_t) \tag{2}$$

The encrypted message ζ in Equation (2) is sent to the server. The server receives the encrypted message ζ and decrypts using shared secret $\mathcal{SK}$ as given in Equation (3).

$$\zeta_t = Dec^{\mathcal{SK}}(\zeta) \tag{3}$$

The server decrypts the data ζ, and retrieves ζ_t, which is also encrypted using several private keys. These encrypted data are stored in the server's database. The server cannot decrypt the ζ_t because the server does not have the private keys. Thus, the administrators cannot retrieve the original (raw) message.

D. Retrieval

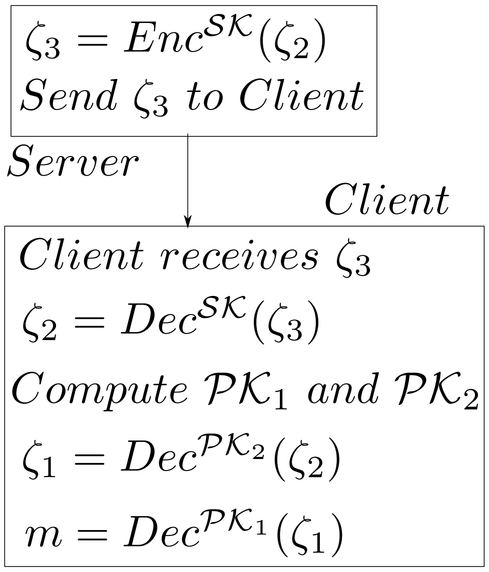
$\zeta_3 = Enc^{\mathcal{SK}}(\zeta_2)$
Send ζ_3 to Client
Server
Client
Client receives ζ_3
$\zeta_2 = Dec^{\mathcal{SK}}(\zeta_3)$
Compute $\mathcal{PK}_1$ and $\mathcal{PK}_2$
$\zeta_1 = Dec^{\mathcal{PK}_2}(\zeta_2)$
$m = Dec^{\mathcal{PK}_1}(\zeta_1)$

Fig. 6. Retrieval process of SecretStore as client-server model.

Figure 6 demonstrates the retrieval process (downloading) of data from a server by a client. A client stores its data in cloud storage, and the client issues a retrieval message on the data to read. Therefore, the user and server need to compute the shared secret key using ECDH. The server encrypts the data using the shared secret key given in Equation (4).

$$\zeta = Enc^{\mathcal{SK}}(\zeta_t) \tag{4}$$

The server sent the encrypted message ζ of Equation (4) to the client. The client receives ζ from the server and decrypts the encrypted code ζ using a shared secret key as given in Equation (5).

$$\zeta_t = Dec^{\mathcal{SK}}(\zeta) \tag{5}$$

Now, the user needs to decrypt the message using its private keys. Before decrypts, the client lookup the private keys in cache (local storage). If private keys are found, the client decrypts the message as given in Equation (6). Otherwise, the client regenerates all t private keys using Algorithm 1, and then the client can decrypt the incoming messages.

$$\begin{aligned}\zeta_{(t-1)} &= Dec^{\mathcal{PK}_t}(\zeta_t)\\ \zeta_{(t-2)} &= Dec^{\mathcal{PK}_{(t-1)}}(\zeta_{(t-1)})\\ \zeta_{(t-3)} &= Dec^{\mathcal{PK}_{(t-2)}}(\zeta_{(t-2)})\\ &\vdots\\ m &= Dec^{\mathcal{PK}_1}(\zeta_1)\end{aligned} \tag{6}$$

Thus, a user decrypts the raw message using its private keys. However, we suggest that t should not be too large. The large value of t slows down the cryptography process. The ideal value of t is 2; however, it can be increased to more than 2 if the security requirement is high.

Algorithm 2 Generating a strong password using user password and a secret number.

```
procedure GENPASSWORD($password,\ \eta$)
    $Alpha[23],\ Symbol[13],\ j = 0$
    for $password\ and\ \eta$ do
        $NewPassword = Symbol[password[j]\%13]$
        $NewPassword = Alhpa[password[j]\%23] + 32$
        $NewPassword = Symbol[\eta\%10]$
        $\eta = \frac{\eta}{10}$
        $NewPassword = Alhpa[password[j]\%23]$
        $NewPassword = password[j]$
    end for
end procedure
```

1) Issue of a weak password: Password-based solutions have an issue of weak passwords that the attacker can easily guess; however, most of the modern password-based solution asks for a combination of alphabet, number, and symbols for a new password. So, SecretStore follows the same rules for creating a new password with length of 8-32 which must have at least an alphabet, a special symbol, and a number. Still, there is a chance of creating a weak password, for instance, abc@1234. Therefore, we present a method to strengthen the weak password by Algorithm 2. Algorithm 2 uses two arrays, specifically, symbol array and alphabet array. For both the array, the size should be a prime number due to hashing. For instance, the size of the alphabet and symbol array can be $11, 13, 17, 19$, and 23, but both the array's size should not be equal. In the algorithm, we assume that a user remembers its password and a number η. The necessary condition for password length is $8 \leq l \leq 32$ and number size is $4 \leq \delta \leq 10$ digits. Therefore, a user needs to enter a password and a number. A user can enter the date of birth (eg., ddmmyyyy, mmddyyyy, yyyymmdd, ddmmyy, mmddyy, yyddmm, yymmdd, etc.), zip code, mobile number, year, or any number which is greater than three digits and easy to remember. Thus, the output of Algorithm 2 is ":p1EI:g0AE@t1AE!l0AE!i3@m6*t9@b1" "IEEE" and "19630101". Also, it produce a output "*u1AE_s2Ql_u0As#c2Je" for the input "Elsevier" and "2021". Similarly, for the input "ACM" and "1947", the output of Algorithm 2 is ":p7TA?k4VC*t9IM@b1". The output of Algorithm 2 is input into Algorithm 1 for generating the private keys. This procedure removes the weakness of the password-based solution. Most of the sensitive users use high quality secret word, for instance, "TIFS@!EEE:2o21" which makes easy to remember but difficult for adversaries.

IV. Analysis

Our proposed system works on regenerative private keys. Unlike a server, a user cannot maintain its private key because the user can change its platform. Moreover, a user device can be damaged or lost at any time, and therefore, it requires a regenerative private key which is highly unpredictable for its adversaries. We assume that a user can remember its secret word to generate or regenerate the private keys. Table I demonstrates the statistical tests on "IEEE2021" for key generation. If a user lost its private keys, the user can regenerate the private keys using the secret word. Now, a user wants more than one private keys to be generated by the Algorithm 1. Then, Algorithm 1 iterates β times to generate the first private key. The algorithm continues with the same key with different seed values for the second private key, and it iterates for another β times. Similarly, it continues for the third private key too. Thus, it is not required to maintain the private keys by the users; however, the private keys can be cached in the user's devices for faster processing.

A. Assurance of Tight Security

SecretStore provides tighter security than any state-of-the-art security protocol. The user encrypts the data before storing it in the cloud and then encrypted the ciphered data using a shared secret key to upload in the cloud. Therefore, SecretStore ensures its security even if the first layer of security is compromised, but the adversary cannot extract the raw data from the user. In any condition, the security is intact. Suppose an adversary gains access to the server and retrieves all the data from SecretStore. The adversary cannot retrieve the raw data even if the adversary can access the server. Moreover, the administrators cannot decrypt the stored data to misuse. Thus, hard secrecy is strictly maintained by SecretStore. In a conventional system, if an adversary is able to gain access to the server, then the adversary can easily read all those data from the server, for instance, wikileaks.org. Our proposed method prevents such kinds of attacks.

B. Analysis on weak password

Algorithm 2 is designed to strengthen the weak password. A user needs to input two secret codes; particularly, a password and a number. A user requires to remember both password and the number for later usage. The number must be greater than three digits. Therefore, a user can enter the date of birth in any format, zip code, mobile number, year, or any number to Algorithm 2 that can be easy to remember by the user. Let us assume that the password is easy to be guessed the attacker. In that case, the number plays a critical role where the attacker cannot guess the number. An attacker needs to uncover both password and a number for a particular user.

Theorem 1. *The probability of breaking the password and the secret number by Brute-force attacker is* $(\frac{1}{26^l} \times \frac{1}{10^\delta}) \approx 0$.

Proof. Let us assume that the password length is l. The probability of breaking the password using Brute-force (BF) attacker is $\frac{1}{26^l}$ where $8 \leq l \leq 32$. Let the digit δ represents the digits in the number. Now, the probability of breaking the number using BF attacker is as low as $\frac{1}{10^\delta}$ where $4 \leq \delta \leq 10$. Then the total probability of breaking the secret code is given in Equation (7).

$$Pr(BF) = Pr(password) \cap Pr(\eta) \tag{7}$$

The secret number and the password are independent events; therefore, the probability is given in Equation (8).

$$\begin{aligned} Pr(BF) &= Pr(password)Pr(\eta) \\ &= \frac{1}{26^l} \times \frac{1}{10^\delta} \end{aligned} \tag{8}$$

For the lowest case, the password's size is $l = 8$ and secret number size is $\delta = 4$, then the probability becomes $\frac{1}{26^8 \times 10^4} \approx 0$. Similarly, the highest case, the probability becomes $\frac{1}{26^{32} \times 10^{10}} \approx 0$. □

Corollary 1. *The probability of not able to break the secret code of SecretStore is* $(1 - \frac{1}{26^l} \times \frac{1}{10^\delta}) \approx 1$

A dictionary attack is another issue in password-based solutions. The attacker collects a massive amount of possible passwords to break the security. Similarly, a dictionary attacker has to constructs an enormous amount of most used numbers to break the security of Algorithm 2. Thus, it adds another complexity for the attackers to break the Algorithm 2. However, most of the users may use year. An attacker may construct a dictionary of the user's date of birth, phone number, year (1950-2021), and zip code, since these are the most common to use in Algorithm 2 but a user may pick any number. Moreover, birthday attack is also an issue of password-based solutions. Birthday attackers try to find two password collisions; however, the secret number can create a strong deterrence to such kinds of attacks. Therefore, our proposed solution provides a good defense on such kind attacks even if a user chooses a weak password or a weak number.

1) Brute-force attacks: A brute-force attack is the most common attack which accomplishes the attack by performing an exhaustive search in the keyspace. It can break almost any kind of security, but it may take many years; however, it is a severe attack. Therefore, we propose many levels of encryption to defeat such kinds of attacks. For instance, a client encrypts two times by its private key at $t = 2$ and encrypts it again using a shared secret key. Therefore, it is not possible to attack our proposed system by the brute-force attack since the attacker has to break three security layers at $t = 2$.

2) Cryptanalysis attacks: There are various attacks in computer networking, for instance, DDoS. However, our proposed system follows a symmetric communication protocol; therefore, we do not consider many attacks which are not applicable in our proposed system; for instance, MITM and DDoS attacks are out of the scope of the proposed system.

A cryptanalysis attack is an attack based on the ciphertext analysis and reveals the secret keys or retrieves the plaintext. It applies in most symmetric cryptography protocols where studying the ciphertext gives a secret key or plaintext pattern. Therefore, it is essential to protect against such kinds of attacks. There are many kinds of Cryptanalysis attacks, particularly ciphertext-only, known-plaintext, chosen-ciphertext or chosen-plaintext, adaptive chosen-plaintext, related-key attack, frequency analysis, index of coincidence, Boomerang, differential cryptanalysis, linear cryptanalysis, etc. attacks, which are the most commonly known in symmetric cryptography. These attacks are possible one-keyed symmetric cryptography; however, it also takes many years to break the one-keyed cryptography. Our proposed system depends on t key to encrypt before being sent to the receiver, and then, the ciphertext is encrypted using a shared secret key. The attacker has to break the first layer of security, and then the attacker can break the t layer of encryption. This system provides a tight coating over a plaintext such that the adversaries cannot retrieve the original message even if the attacker can break the first layer of security.

3) Dictionary attacks: Diction attacks is accomplished by creating dictionary, i.e., collecting huge set of text to capture the communication. The collected text are used build dictionary of ciphertext, and therefore, it becomes easy to break the password-based security. However, it is almost impossible to attack our proposed system using dictionary based attack. On the contrary, our proposed system relies on password-based private-key generation as shown in Algorithm 1. Therefore, it is wise-way to attack the private key generation system rather than the direct attacking on the ciphertext.

4) Probability: Let l be the length of the password, β be the length of private keys, and γ be the length of a shared secret key. The probability of breaking guessing the correct password is $\frac{1}{62^l}$ where the total sample space is 62 characters, including the upper and lowercase letters and ten digits excluding special symbols. Therefore, the probability of not getting correct password is $(1 - \frac{1}{62^l})$. It is a probability of a brute-force attack. However, the dictionary-based password attack is much simpler, and it is the weakness of all password-based solutions. Therefore, the password is strictly composed using at least a capital letter, a small letter, a digit and a special symbol, and a length of at least eight. This restriction makes it difficult for dictionary-based attackers. Let us assume that the probability of guessing a password is 1. Thus, an adversary can generate the private keys; however, the adversary has to break the security of shared secret key encryption. The probability of breaking the shared secret key is $\frac{1}{2^\gamma}$. The total probability of breaking the entire security is given in Equation (9).

$$\begin{aligned} Total\ probability &= Pr(Password) \cap Pr(SecretKey) \\ &= \frac{1}{62^l} \times \frac{1}{2^\gamma} \end{aligned} \tag{9}$$

Since the password breaking and encrypting the code using the shared secret keys are independent events. The probability of getting entire private keys without knowing the password is given in Equation (10).

$$Total\ probability = \frac{1}{2^\beta} \tag{10}$$

Since the private keys are dependent on each other, and if an adversary gets the first private key, it is easy to capture entire private keys. However, if an adversary wishes to break security directly from the ciphertext, then the total probability is given in Equation (11).

$$Total\ probability = \frac{1}{62^l} \times \frac{1}{2^\gamma} \times \frac{1}{2^\beta} \tag{11}$$

Equation (11) gives the complexity to break our proposed solutions. Now, the adversary is attacking the private keys without any order, then, Equation (12) gives the total probability.

$$Total\ probability = \frac{1}{62^l} \times \frac{1}{2^\gamma} \times \frac{1}{2^{\beta t}} \tag{12}$$

Therefore, the total probability of breaking the proposed security is given in Equation (12). The total probability of not breaking the security of the proposed system is given in Equation (13).

$$Total\ probability = 1 - \left(\frac{1}{62^l} \times \frac{1}{2^\gamma} \times \frac{1}{2^{\beta t}}\right) \qquad (13)$$

Thus, Equation (13) gives the security tightness of our proposed system. Above analysis shows that it is quite difficult to break the security of our proposed solution for the adversaries due to multiple layer of encryption.

V. Experimental Results

We have conducted a series of rigorous tests to validate the randomness of the generated number in the Ubuntu desktop computer. The computer configuration is as follows- Intel Core i7-7700 CPU @ 3.60GHz × 8, Ubuntu 18.04.5 LTS, 8GB RAM, 1TB HDD, and GCC Version 7.5.0. This experimentation is essential analysis is required to test the randomness of the generated private keys. The generated private keys are reproducible and highly random, as shown in Table I. In the experimental evaluation, we have used "IEEE2021" as a secret word and input it into Algorithm 1. The output is tested in NIST SP 800-22 statistically tests for the approximation entropy, frequency, block frequency, cumulative sums, runs, longest runs, rank, FFT, non-overlapping template, overlapping template, random excursions, random excursions variant, serial, universal, and linear complexity tests [17], [18].

We have generated 10M random bits and tested 32bits, 64 bits and 128 bits stream at NIST SP 800-22 test suite. The test results are drawn in Table. The necessary condition for P-value is ≥ 0.01 to be accepted as random; otherwise, it cannot be accepted as random. The pass rate of the test indicates the successful test percentage. In this test, the P-value and pass rate are equally important to consider for randomness. Higher P-value and pass rate ensure high randomness in generated private keys. It indicates that there are no patterns in the generated bits. Moreover, it also indicates that it is cryptographically secure due to highly randomness of the generated bits.

The highest P-value of 32bits, 64 bits and 128 bits stream for the secret word for "IEEE2021" are 0.976060, 0.985035 and 0.985035, respectively, with a 100% success rate. The lowest P-value of 32 bits, 64 bits and 128 bits stream for the secret word for "IEEE2021" are 0.066882, 0.018879 and 0.031497, respectively. The lowest success rates of 32 bits, 64 bits, and 128 bits stream for the secret word for "IEEE2021" are 0.96875, 0.96875, and 0.984375, respectively. Thus, this statistical test proves the randomness of proposed private keys.

VI. Discussion

Secrecy is an urgent requirement to be implemented. Current state-of-the-art cloud storage technology does not implement hard secrecy. On the contrary, the administrator can easily read the users' data and misuse their data. Users' data are not safe from the administrators, and thus, it is required to remove the administrators from the valid and intended users list. However, administrators are valid and intended users by default, but it should be valid and intended users. There is no difference between the administrators and the attackers if they read the users' data without permission. Moreover, administrators should not read any data of users. Why should they read the data of a user? There is no sufficient reason for reading the users' data by the administrators except the recommender systems. Therefore, it is time to remove the administrators from the list of the valid and intended users.

The key weakness of our proposed system lies within the password-based solution. Even though we provide the strengthen mechanism of users' passwords, users may create a weak password that will be easy for adversaries to guess. However, there is another layer of security to protect, i.e., shared secret keys. Our proposed system is quite valuable for storing the most sensitive data in Cloud Storage. Moreover, many users do not want their data to be read by anyone except themselves. It applies to everyone. Many people have secret data to store permanently, but they cannot store it due to valid and employed Mallory in the cloud. Moreover, there is a chance of data leakage, such as WikiLeaks [19], [20]. Also, there are many techniques available on data leakage prevention, but these works do not consider administrators as a Mallory. Therefore, it creates a difference between SecretStore and state-of-the-art data leakage prevention techniques. Our proposed system ensures tight security against data leakage. Another weakness of our proposed system is the many layers of encryption. It slows down the cryptography process. People use low-powered computing devices; therefore, multiple encryptions create computation overhead on such devices. The computation overhead is justifiable when it comes to sensitive data.

VII. Conclusion

This article has presented a novel cloud storage called SecretStore to protect users' data from unintended users. It features "Secret as a Service" or "User Secrecy as a Service". We have also demonstrated how to prevent administrators from accessing users' data. This process ensures that a user can store its most sensitive data in the cloud. There are no possibilities of data leakage. We have mathematically analyzed the probability of breaking the security of the proposed system. Our proposed solution provides client-side cryptography, and thus, a user encrypts its data using forgetful private keys. The encrypted user data is encrypted again using shared secret keys to transmit over the public media. We have demonstrated how to generate a private key or regenerate a private key using secret words. Also, we have demonstrated how to strengthen the weak password for generating private keys to create a strong deterrence against password attackers. We use ECDH to compute a shared secret key between user and server to transmit the encrypted message. The server receives the encrypted message and decrypts ciphertext. This ciphertext is stored in the server's database. Thus, we have proved that the user's data cannot be read by the administrator, preventing data misuse. Moreover, we have demonstrated the randomness of generated private keys experimentally using NIST SP 800-22 statistical test suites. To the best of our knowledge, this is the first system that provides strict secrecy of users' data and protects access from any unintended users, including administrators and attackers.

TABLE I
P-VALUES AND SUCCESS RATES OF ALGORITHMS 1 FOR 32, 64 AND 128 BITS STREAM FOR THE WORD "IEEE2021" IN NIST SP 800-22 STATISTICAL TESTS.

Test name	32 bits		64 bits		128 bits	
	P-value	Pass rate	P-value	Pass rate	P-value	Pass rate
Approximate Entropy	0.066882	32/32	0.232760	64/64	0.031497	128/128
Frequency	0.602458	32/32	0.299251	62/64	0.311542	126/128
Block Frequency	0.602458	32/32	0.048716	64/64	0.804337	
Cumulative sums	0.804337	32/32	0.213309	63/64	0.324180	127/128
Runs	0.949602	32/32	0.804337	64/64	0.095617	128/128
Longest runs	0.350485	31/32	0.888137	63/64	0.162606	126/128
Rank	0.407091	32/32	0.437274	63/64	0.253551	126/128
FFT	0.739918	32/32	0.253551	64/64	0.602458	127/128
Non-overlapping Template	0.976060	32/32	0.985035	64/64	0.985035	128/128
Overlapping Template	0.407091	31/32	0.468595	64/64	0.287306	126/128
Random Excursions	0.162606	15/15	0.534146	16/16	0.637119	13/13
Random Excursions Variant	0.637119	15/15	0.066882	16/16	0.437274	13/13
Serial	0.976060	32/32	0.253551	64/64	0.819544	126/128
Linear complexity	0.468595	32/32	0.500934	63/64	0.500934	127/128
Universal	0.178278	32/32	0.018879	64/64	0.534146	126/128

REFERENCES

[1] X. Shu, D. Yao, and E. Bertino, "Privacy-preserving detection of sensitive data exposure," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 5, pp. 1092–1103, 2015.

[2] B. Jiang, M. Seif, R. Tandon, and M. Li, "Context-aware local information privacy," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2021.

[3] M. Du, Q. Wang, M. He, and J. Weng, "Privacy-preserving indexing and query processing for secure dynamic cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 9, pp. 2320–2332, 2018.

[4] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 843–859, 2013.

[5] M. Sun and W. P. Tay, "On the relationship between inference and data privacy in decentralized iot networks," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 852–866, 2020.

[6] Y. Sun, Q. Liu, X. Chen, and X. Du, "An adaptive authenticated data structure with privacy-preserving for big data stream in cloud," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3295–3310, 2020.

[7] L. Rajabion, A. A. Shaltooki, M. Taghikhah, A. Ghasemi, and A. Badfar, "Healthcare big data processing mechanisms: The role of cloud computing," *International Journal of Information Management*, vol. 49, pp. 271–289, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268401217304917

[8] Z. Wang, N. Wang, X. Su, and S. Ge, "An empirical study on business analytics affordances enhancing the management of cloud computing data security," *International Journal of Information Management*, vol. 50, pp. 387–394, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268401218302603

[9] K.-Y. Huang, G.-H. Luo, and S.-M. Yuan, "Sstreasury+: A secure and elastic cloud data encryption system," in *2012 Sixth International Conference on Genetic and Evolutionary Computing*, 2012, pp. 518–521.

[10] L. Wei, H. Zhu, Z. Cao, X. Dong, W. Jia, Y. Chen, and A. V. Vasilakos, "Security and privacy for storage and computation in cloud computing," *Information Sciences*, vol. 258, pp. 371–386, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025513003320

[11] W. Itani, A. Kayssi, and A. Chehab, "Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures," in *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, 2009, pp. 711–716.

[12] R. Patgiri, S. Nayak, and S. K. Borgohain, "Passdb: A password database with strict privacy protocol using 3d bloom filter," *Information Sciences*, vol. 539, pp. 157 – 176, 2020.

[13] R. Patgiri, "Whisper: A curious case of valid and employed mallory in cloud computing," in *To be appeared in the confernece proceedings of the 8th IEEE International Conference on Cyber Security and Cloud Computing (IEEE CSCloud 2021), June 26-28, Washington DC, USA*, 2021, pp. 1–6.

[14] 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.

[15] FIPS, "Specification for the advanced encryption standard (aes)," Federal Information Processing Standards Publication 197, 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[16] M. Agrawal, N. Kayal, and N. Saxena, "PRIMES Is in P," *Ann. Of Math.*, vol. 160, no. 2, pp. 781–793, Sep 2004. [Online]. Available: http://www.jstor.org/stable/3597229

[17] 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.

[18] 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.

[19] S. Alneyadi, E. Sithirasenan, and V. Muthukkumarasamy, "A survey on data leakage prevention systems," *Journal of Network and Computer Applications*, vol. 62, pp. 137–152, 2016.

[20] Q. Zhao, C. Zuo, G. Pellegrino, and L. Zhiqiang, "Geo-locating drivers: A study of sensitive data leakage in ride-hailing services." in *NDSS Symposium 2019*, February 2019, pp. 1–15. [Online]. Available: https://publications.cispa.saarland/2757/