An Improved Time-Based Encryption Key Rotation Scheme for Healthcare Databases

Ellie Pin-Yu Chen
Dept. of Computer Science and Information Engineering
National Chi Nan University
Puli Township, Taiwan
s110321053@ncnu.edu.tw

Quincy Wu
Dept. of Computer Science and Information Engineering
National Chi Nan University
Puli Township, Taiwan
solomon@ncnu.edu.tw

Meng-Shuo Shen
Puli Christian Hospital
Puli Township, Taiwan
4286@mail.pch.org.tw

***Abstract*— The practice of encrypting sensitive and critical data has led to the development of many mature cryptosystems that almost guarantee the confidentiality of encrypted data. However, many organizations still suffer security issues caused by poor key management. Some scenarios indicate that encryption keys have not been changed for several years. In that case, former employees still possess valid keys to access the system. Many systems adopt key rotation as countermeasures, but it is difficult to find a balance between security and efficiency. This paper shares the experience in Puli Christian Hospital with key rotation by proposing an improved algorithm with enhanced security and higher performance over a previous time-based key rotation system. It is demonstrated that the new mechanism is more suitable for healthcare database systems to store sensitive data.**

Keywords—AES, key rotation, Puli Christian Hospital, SHA-256, time-based one-time password

I. INTRODUCTION

Since computers were introduced, many organizations have stored business data in computer databases to facilitate business operation and management. Under the principle of defense in depth, sensitive data are stored in encrypted form on computers [1], so that an accidental data leakage will not immediately cause sensitive data to be disclosed. Key management plays a critical role in ensuring the security of databases that contain important and sensitive data, such as hospital employee salaries and patient diagnostics. A good encryption system can help hospitals prevent targeted and aggressive password cracking attacks, as well as sensitive data leakage.

Cryptosystem is a sophisticated computer system that encrypts and decrypts message. Plaintexts are the original messages. To ensure confidentiality, cryptographic algorithms are applied upon them to generate ciphertexts which can be safely stored in computer systems. A good cryptographic algorithm should guarantee that, without knowing the correct key for decryption, it will be difficult, or even computationally intractable, to decrypt the ciphertexts with brute force. Many famous cryptosystem has been proposed and applied widely in lots of fields, such as RSA [2] and AES [3].

However, choosing a strong cryptographic algorithm is not sufficient to guarantee data security. Key management is also a critical issue. Not all computer systems prompt users to manually type the key when data are being encrypted. This is to prevent the possible accidents that users make some typos and data are encrypted with a wrong key, which nobody knows the exact value. Then no one can ever decrypt the data. Therefore, in practice, many systems simply hardwire the encryption key in the computer program, to make sure data are always encrypted with a correct key. Nevertheless, this practice poses a potential risk that the key may be unchanged for tens of years. Attackers may be able to crack passwords which are kept the same for decades due to the fast growth of computing power. In this situation, key rotation is an important technique to improve data security [4].

Puli Christian Hospital is a prestigious hospital in central Taiwan which is famous for its innovative information technology that facilitates the daily operation of the hospital. It replaced its old constant-key system by introducing a time-based algorithm to rotate keys every day[5]. However, cryptanalysis shows that the new system has a potential vulnerability that for any user who had ever gotten a valid key to access the database in a single day, he would be able to apply a hash function (which is publicly known) a couple of times to obtain information that allows him to decrypt the ciphertext by brute force for less than 20 thousand tries, which takes within a second. Therefore, it is crucial to improve the current key rotation mechanism to ensure the confidentiality of encrypted data stored in the cryptosystem.

The remaining part of this paper is structured as follows. In Section II, the technical details of the previous time-based key rotation algorithm in Puli Christian Hospital and its defects are briefly described. The next section highlights the overview of the proposed improvement. Subsequently, the detailed algorithms and experimental results are presented in Section IV and Section V. The final section concludes with the contribution and future work of this cryptosystem.

II. PREVIOUS WORK

A. Key Rotation

A direct improvement on key management is to periodically change a new key [6]. Any given key is only valid in a limited time interval, so former employees will not be able to access the system with former keys. This idea comes from the one-time password system [7]. This mechanism works well for authenticating users, since the number of users do not significantly vary as time goes by. However, in a database system, if the encryption key is changed periodically, generally this implies that all encrypted data must be retrieved from the database, decrypted with the old key, encrypted with the new key, and then stored back to the database. After the database accumulated many records containing encrypted data, each key rotation will inevitably impose heavy burden to the database system.

B. Previous proposed algorithm

Table I shows all notations in the previous time-based one-time password (TOTP) algorithm, which we shall call it Algorithm 1 for convenience. Let X denote the time interval to rotate keys. When the system starts up, a string of Shared Secret `S` and an integer `Count` should be specified. Other configurable parameters have default values. The default value of hash function is SHA-256, which is a one-way function preventing users from predicting the key in next time interval. T0 is the initial counter time and usually defaults to zero.

TABLE I. NOTATION OF THE TOTP ALGORITHM

Symbol	Meaning
S	Shared Secret
Count	An unsigned 32-bit integer (Its default value is randomly generated.)
T0	Unix time to start counting time steps (The default value of T0 is 0.)
i	The amount of time interval X after T0.
X	Time interval (e.g., 86400 seconds for a day)
Hash ()	Hash function (SHA-256, SHA-512)
K(*i*)	Decryption key valid at time interval *i*
$\oplus$	XOR encryption/decryption

The TOTP algorithm reuses the concept of time-based one-time passwords from RFC 6238 [8]. The Shared Secret `S` is passed to the hash function, along with value Count-*i*. In other words, in Step 3 of Algorithm 1, the second parameter in the Hash() function is the repeated times to hash the first parameter, which is the Shared Secret `S` in this algorithm. For example, Hash(S, 1) stands for applying the hash function on string `S` only once. Hash(S, 2) = Hash(Hash(S, 1), 1). More generally, Hash(S, N) = Hash(Hash(S, N-1), 1), which stands for applying the hash function N times. As *i* increases with time, the hash counter Count-*i* decreases, as shown in Table II. The number 19662, 19663, 19664, 19665 in the table are the number of elapsed days since the epoch time (00:00:00 UTC on January 1st, 1970). According to the irreversible characteristic of the hash chain, we can be assured that it is easy to derive past keys from a given key, but difficult to derive future keys.

Algorithm 1 – Encryption Algorithm of TOTP

Steps:

1. Calculate the amount of time intervals:

   i = (Current Unix time - T0) / X

2. Encrypt data:

   ciphertext = plaintext $\oplus$ Hash (S, Count)

3. Generate a key:

   K(i) = Hash (S, Count-i)

4. Dispatch K(i) to user

TABLE II. KEYS IN DIFFERENT TIME INTERVAL OF ALGORITHM 1

Date (2023)	Key
November 1st	Hash (S, Count - 19662)
November 2nd	Hash (S, Count - 19663)
November 3rd	Hash (S, Count - 19664)
November 4th	Hash (S, Count - 19665)

Algorithm 2 – Decryption Algorithm of TOTP

Steps:

1. The user supplies a key k.
2. Calculate the amount of time intervals:

   i = (Current Unix time - T0) / X

3. Decrypt data:

   h = Hash (K(i), i)

plaintext = ciphertext $\oplus$ h

If the user possesses the valid key k = K(*i*) = Hash (S, Count-*i*), then in Step 3 of Algorithm 2, Hash(k, *i*) = Hash(Hash(S, Count-*i*), *i*) = Hash(S, Count). Because the ciphertext generated in Step 3 of Algorithm 1 is plaintext ⊕ Hash (S, Count), we can obtain ciphertext ⊕ Hash (k, *i*) = plaintext ⊕ Hash (S, Count) ⊕ Hash (k, *i*) = plaintext ⊕ Hash (S, Count) ⊕ Hash (S, Count) = plaintext, which is the original plaintext.

C. Problems

As the adage goes, "No rose without a thorn, no sweet without a sour." After a field trial, it was soon observed that the above time-based one-time password system still has several serious deficiencies that must be improved before it can be utilized in practice.

- **Security**

The most concerning issue is the data security vulnerability in the database. The problem is not that the keys delivered to users are insecure, but that the data is encrypted by applying the hash function successively. Suppose the key is rotated once a day, and Mallory obtained one key (e.g., K(19662)) on November 1st of year 2023. He cannot know for certain what the key will be used on the next day (November 2nd), but he can try to decrypt the ciphertext with brute force, using the key Hash(K(19662), *j*), where *j* = 1, 2, 3, ..., 20000. Because any key k = Hash(S, Count-*j*) for some *j*, even if Mallory does not know the exact value of T0, there always exists some j such that Hash(k, j) = Hash(S, Count). Applying the XOR operation with Hash(S, Count) on the ciphertext allows Mallory to obtain the plaintext. Since the number of days since the epoch time is less than 20000, this cryptosystem can be cracked by brute force within 20000 tries!

- **Efficiency**

In Step 2 of Algorithm 1, we have to apply the hash function successively for `Count` times. When `Count` is a 32-bit unsigned integer, this step may run the hash function for billions of times, which in worst case took 50.6 minutes in [5]. In practice this is quite inefficient and thus unacceptable.

III. Proposed System

A. Motivation

To overcome the shortcomings of the TOTP algorithm, an initial recommendation is to increase the size of `Count` and *i*. If it takes 50 minutes to try all 2^{32} possible values of Count, a simple improvement to secure the system is to enlarge it as a 64-bit long long integer. Trying all 2^{64} possible values will take about 408,577 years, which is longer than human civilization since Ancient Egypt, and can be thought as sufficiently secure. However, this gives rise to another problem: the time it takes to hash the key (for Count-*i* times) to decrypt the plaintext is very long. Therefore, although enlarging the size of `Count` delays the time it takes attackers to crack the key, it also increases the time to decrypt ciphertexts. In light of the problems observed above, we propose an improvement to replace the one-way hash function in Algorithm 1 by a two-way encryption/decryption function. A formal description of the algorithm and details will be given in the next section. In this section, we provide some technical background.

B. Technical Background

- **Unix time:**

  Unix time, also known as POSIX time or epoch time, is the number of seconds that have elapsed since 00:00:00 UTC on January 1st, 1970.

- **XOR encryption:** XOR operation can be performed as bitwise addition modulo 2 on bit streams. By repeatedly applying the bitwise XOR operation of the key string on the data, we can quickly encrypt the data stream, the process is shown in Fig. 1. This proves to be a very efficient encryption mechanism in many popular encryption applications, such as RC5 [9] and Secure Real-time Transport Protocol (SRTP) [10].

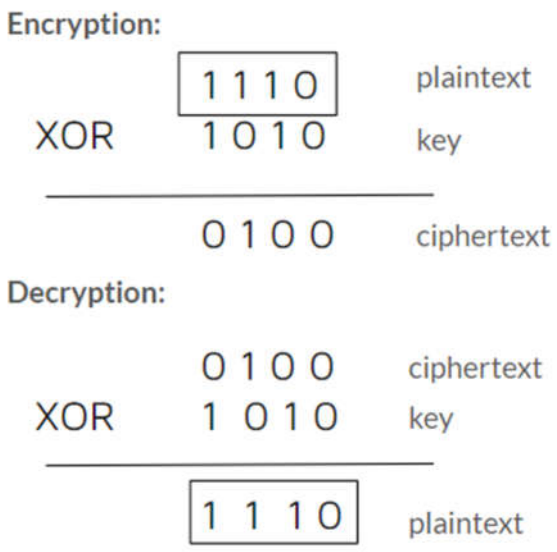

Fig. 1. Example of XOR encryption/decryption

- **AES cryptosystem:**

  Advanced Encryption Standard (AES), also known by its original name Rijndael [11], is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data. High speed and low RAM requirements were the major advantages of AES. As a result, AES performed well on a wide variety of hardware, from 8-bit smart cards to high-performance computers [12].

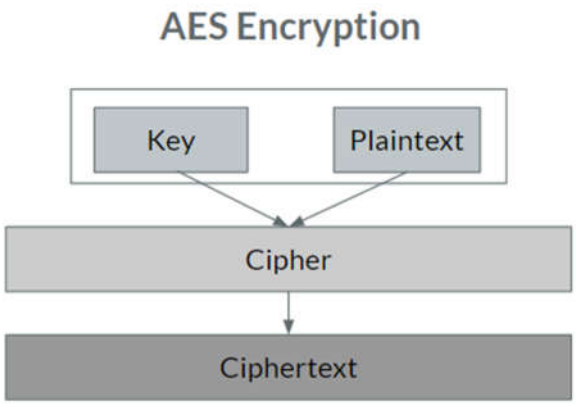

Fig. 2. AES encryption

IV. ALGORITHMS

A. Notations

There are three changes: (1) `Count` is enlarged to be a 64-bit long long integer; (2) Hash() only takes one parameter, so it is only applied once instead of multiple times; (3) in addition to the existed symbols in TOTP, new algorithms AES_enc() and AES_dec() are introduced to perform AES encryption and AES decryption, respectively.

TABLE III. NOTATION OF NEW ALGORITHM

Symbol	Meaning
S	Shared Secret
Count	An unsigned 64-bit long long (Its default value is random generated)
T0	Unix time to start counting time steps (The default value of T0 is 0.)
X	Time interval (e.g. 86400 seconds)
Hash ()	Hash function (SHA-256, SHA-512)
K(*i*)	Decryption key valid at time step *i*
AES_enc()	AES encryption function
AES_dec()	AES decryption function
⊕	XOR encryption/decryption

B. Functions

The proposed cryptosystem consists of three functions:

- **Encrypt data**:

$$\text{Ciphertext} = \text{plaintext} \oplus \text{Hash}(S||\text{Count}) \quad (1)$$

The notation || in (1) stands for concatenating the Shared Secret `S` with a private value `Count` which is only known by the cryptosystem. The system applies the hash function on this concatenated string to generate a hashed string. For example, if the hash function is SHA-256, the hashed string will be 32 bytes (256 bits). To store data into the database, the XOR operation is performed on the plaintext and the hashed string. If the plaintext is longer than the length of the hashed string, the hashed string is duplicated automatically to provide sufficient bit streams to apply the XOR operation.

- **Generate K(*i*):**

$$K(i) = \text{AES\_enc}(S, \text{Hash}(\text{Count-}i)) \quad (2)$$

A key is generated using the AES encryption function, with the Shared Secret `S` as the plaintext and the hash of Count-*i* as the cryptographic key to encrypt `S` and produce a string K(*i*). This cryptographic key is given to users, and it is valid over only a limited time interval (in a single day). Because `S` and `Count` are kept secret, attackers will be unable to derive future keys with the given K(*i*).

- **Decrypt data**:

$$\text{User\_S} = \text{AES\_dec}(K(i), \text{Hash}(\text{Count-}i)) \quad (3)$$
$$\text{plaintext} = \text{ciphertext} \oplus \text{Hash}(\text{User\_S}||\text{Count}) \quad (4)$$

To decrypt ciphertexts, a user sends K(*i*) and ciphertext to the system. The system calculates *i* at the moment, and calculates Count-*i*, Hash(Count-*i*), then use Hash(Count-*i*) as the decryption key to decrypt K(*i*) to obtain User_S. If the value of User_S is the same as the Shared Secret S, the plaintext will be revealed to the user after the XOR operation in (4). Otherwise, the user will receive a meaningless chaotic string. The `Count` remains confidential to every user throughout this process, and its protracted length in bits makes it difficult to guess using brute force attacks.

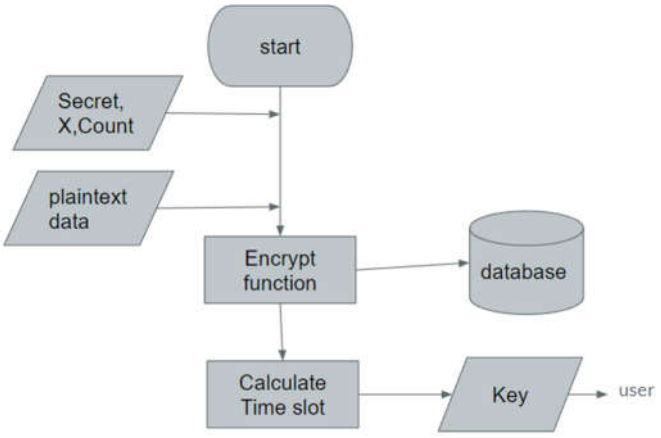

Fig. 3. Encryption flow chart

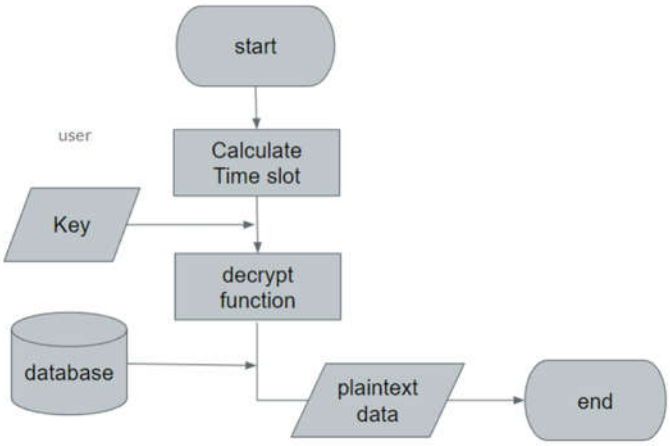

Fig. 4. Decryption flow chart

C. Algorithms

In Algorithm 3, the complete encryption process is outlined in five steps. After the required values are initialized, the system calculates *i*, which represents the number of time steps between the initial counter time T0 and the current Unix time. Then, the ciphertext is derived by performing XOR operation on the plaintext and the encryption key Hash(S||Count). Since the Shared Secret `S` and `Count` are initialized at Step 1 and kept unchanged, Step 3 will only be executed when the plaintext changed, i.e., when new data are inserted into the database. The fourth step generates the key K(*i*) by hashing Count-*i* and passing it to the AES encryption function with the Shared Secret `S`. This allows the key to be rotated on account of various values of *i*. Finally, the cryptosystem dispatches the key K(*i*) to user.

The decryption process is illustrated in Algorithm 4. When the user provides the key, the system calculates the value of *i* based on the current time. In the second step, the system computes User_S by performing the AES decryption function on the key provided by the user plus the string hashing Count-*i*. This string, User_S, will be used to decrypt the ciphertext in the next step. Given that the Shared Secret `S` and `Count` remain unchanged, the key for database encryption, Hash (S||Count), is also keeps unchanged. The ciphertext will be decrypted correctly only if the string User_S is the same as the original Shared Secret `S`. Performing XOR operation on ciphertext and Hash(User_S ||Count) obtains the correct result.

In contrast with the TOTP mechanism illustrated in Algorithm 1 and Algorithm 2, which changes the key for database encryption periodically, our system decrypts data by converting the key K(*i*) to the key for database encryption (Hash (S||Count)), which is shown in Step 2 and Step 3 of Algorithm 4. This design keeps the advantage of TOTP algorithm for rotating keys without re-encrypting data with the new key.

Algorithm 3 – Encryption of the Improved Algorithm

An administrator inputs Shared Secret, X, Count (optional), T0 (optional), Hash function (optional).

Steps:

1. Assign default values to T0, Count, Hash function if they are not specified
2. Calculate the amount of time intervals:

   i = (Current Unix time - T0) / X
3. Encrypt data:

   ciphertext = plaintext $\oplus$ Hash (S||Count)
4. Generate Key:

   K(i) = AES_enc (S, Hash(Count-i))
5. Dispatch K(i) to user

TABLE IV. KEYS IN DIFFERENT TIME INTERVAL OF ALGORITHM 3

Date (2023)	Key
November 1	AES_enc (S, Hash (Count - 19662))
November 2	AES_enc (S, Hash (Count - 19663))
November 3	AES_enc (S, Hash (Count - 19664))
November 4	AES_enc (S, Hash (Count - 19665))

Algorithm 4 – Decryption of the Improved Algorithm

A user gives the key K(i) and ciphertext to the system to decrypt.

Steps:

1. Calculate the amount of time intervals:

   i = (Current Unix time - T0) / X
2. The system calculates Count-i, and use it to get User_S:

   User_S = AES_dec(K(i), Hash(Count-i))
3. Decrypt data

   plaintext = ciphertext $\oplus$ Hash(User_S ||Count)
4. The user can see the result produced by Step 3.

V. EXPERIMENTAL RESULT

A. Implementation

In our implementation, both encryption and decryption are implemented as Python programs. We apply the modules `hashlib` of Python 3 to perform hash chain and `Crypto.Cipher` package [13] to perform AES function.

Let us assume the Shared Secret is a 160-bit ASCII string "12345678901234567890", and `Count` is the value of 10,000,000,000,000,000,000 (10 quintillion). In this example, the keys would be rotated once a day, so the Time Interval X = 24*60*60 = 86400. T0 = 0. SHA-256 is chosen as the hash function. The numbers of time interval *i* is calculated with the

same way as the TOTP algorithm. As shown in Table IV, 19662, 19663, 19664, 19665 are the first four days in November of 2023. As an example, we encrypt the plaintext of medical personnel's salaries in the hospital, as shown in Table V. After applying the function of formula (2), we can derive the rotated keys. In Table VI, only the leading 32 characters of rotated keys are shown, and the remaining part is shown as an ellipsis ("…").

Unlike the frequently changed key, the ciphertexts converted from the plaintext in Table V remain unchanged in the database, which is illustrated in Table VII.

TABLE V. THE EXAMPLE PLAINTEXT IN DATABASE

Name	Position	Salaries
Alice	pharmacist	50,000
Bob	nurse	60,000
Mallory	doctor	100,000
Diana	director	1,000,000

TABLE VI. KEYS IN DIFFERENT TIME INTERVAL OF THE IMPROVED ALGORITHM

Year (2023)	Key (hex form)
Nov. 1	zgYTr4kZYcyKiBj0h0pNsic9weF5J0LN…
Nov. 2	t0pTIX64KKxxQrETo8l41KSwRLfR2pag…
Nov. 3	YNcCI9iBCZrsOBR35u2rLNX6zTWuTgMj…
Nov. 4	lHS50jZ9HhGx36lNRl3u6ge+KYO5Uhoi…

TABLE VII. THE EXAMPLE CIPHERTEXT IN DATABASE

Whose salary	Ciphertext (hex form)	Ciphertext (decimal form)
Alice	a1cb08ccf79a090f	11658421736799078671
Bob	a1cb08ccf79a203f	11658421736799084607
Charlie	a1cb08ccf79b4cff	11658421736799161599
Diana	a1cb08ccf795881f	11658421736798783519

B. Security

In this subsection, we compare the previous TOTP algorithms and the improved algorithms based on Table VIII, which focuses on their encryption parameters. To break the key of the original TOTP algorithm with brute force, an attacker only has to perform at most 2^{32}= 4,294,967,296 hash operations. According to the calculation in Table IX, we can see it takes 0.7725 microseconds per hashing. Therefore, from formula (5) we can see that all the 2^{32} hash operations can be accomplished in less than 55 minutes. In contrast, in Algorithm 4 if attackers want to guess all possible values of `Count`, which is a 64-bit long long integer, there are 2^{64} possibilities. Even if the attacker has a powerful computer with 7GHz CPU clock rate, and it takes only 1 CPU clock cycle to compute a hash function, searching the whole key space will take totally 2^{64} / ($7x10^9$) = 2635249153 seconds, which is approximately 83 years. This is sufficiently secure, as many secret data will be declassified after 50 years.

$$0.7725 * 2^{32} = 3317862236\ (\mu s) = 55\ (min) \quad (5)$$

$$\frac{2^{64}}{7*10^9*86400*365} = 83.56\ (years) \quad (6)$$

TABLE VIII. COMPARING ALGORITHM 1 AND ALGORITHM 3

	Algorithm 1	Algorithm 3
Encryption	plaintext ⊕ Hash (S, Count)	plaintext ⊕ Hash (S‖Count)
Key(*i*)	Hash(S, Count-*i*)	AES_enc(S, Hash(Count-*i*))
Count	32-bit int	64-bit long long

C. Efficiency

To assess the operational performance of these algorithms, we examine the encryption time and key generation time in worst case of Algorithm 1 and Algorithm 3. The experiments were run on a computer with CPU speed 3.2 GHz. The hash library is `hashlib` of Python 3, and AES function is the `Crypto.Cipher` package[13]. The result of operation time of Hash function and AES function are shown in Table IX, and the comparison of key generation time and encryption time is respectively shown in Table X and Table XI, and Fig.5 shows the trend of key generation time.

With these measurements, we can infer the performance of the two algorithms. According to Table X and Table XI, it is obvious that Algorithm 3 is much faster than Algorithm 1. Furthermore, according to Fig.5, we can see the rise of value of `Count` leads to the steadily growing difference of key generation time. In Algorithm 1, the hash function is invoked for a large number of times for both encryption and key generation. On the contrary, Algorithm 3 only requires a single invocation of the hash function to encrypt data and one invocation of the AES encryption function to generate the key. In other words, key generation time and encryption time of Algorithm 3 is constant, regardless of the value of `Count`. On the contrary, in Algorithm 1 the key generation time and encryption time are both growing along with the rise of value of `Count`. Hence, it is easy to foresee that the performance between these two algorithms will be even more significant when `Count` is a 64-bit long long integer in both two algorithms. As the number of rotated keys is limited and determined by the value of `Count`, a larger value of `Count` provides better security to this cryptosystem.

TABLE IX. RUNNING TIME OF FUNCTIONS IN ALOGORITHM 1 AND ALGORITHM 3

	Operation Time
Hash function	50.6 minutes / 4 billion SHA-256 hashing = 0. 7725 μs/hashing
AES encryption	0.17 second / 4000 AES Encryption = 0.0425 ms/encryption

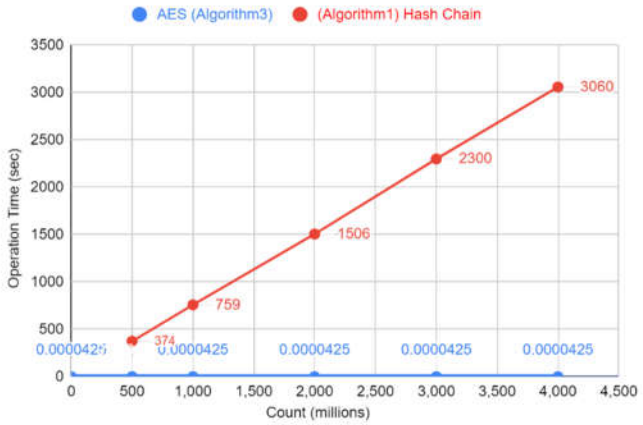

Fig. 5. Key generation time of Algorithm 1 and Algorithm 3 with different `Count`

TABLE X. WORST-CASE KEY GENERATION TIME

	Key generation Time
Algorithm 1(32-bit integer)	50.6 minutes
Algorithm 3(64-bit long long)	425 microseconds

TABLE XI. WORST-CASE ENCRYPTION TIME

	Encryption Time
Algorithm 1(32-bit integer)	50.6 minutes
Algorithm 3(64-bit long long)	0.7725 microseconds

VI. CONCLUSIONS AND FUTURE WORK

This paper presents an enhanced time-based key rotation algorithm for database encryption. The proposed algorithm preserves the beneficial aspects of previous design, such as the ability to periodically rotate keys and the ability to encrypt data only once when data are inserted into the database. This makes it well-suited for database systems containing lots of encrypted data. Additionally, the proposed algorithm addresses the weakness and performance deficiencies of the previous time-base key rotation algorithm. It will be deployed to replace the old constant-key cryptosystem to provide better protection for sensitive data stored in the system.

ACKNOWLEDGEMENT

This research was partially supported by the joint funding of Puli Christian Hospital and National Chi Nan University under the grant number 112-PuChi-AIR-005.

REFERENCES

[1] G. Yendamury and N. Mohankumar, "Defense in depth approach on AES cryptographic decryption core to enhance reliability," 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), Toronto, ON, Canada, 2021, pp. 1-7, doi: 10.1109/IEMTRONICS52119.2021.9422567.

[2] M. Shand and J. Vuillemin, "Fast implementations of RSA cryptography," Proceedings of IEEE 11th Symposium on Computer Arithmetic, Windsor, ON, Canada, 1993, pp. 252-259, doi: 10.1109/ARITH.1993.378085.

[3] M. L. Akkar and C. Giraud, "An implementation of DES and AES, secure against some attacks,". In Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), pp.309-318, Paris, France, May 14-16, 2001. https://doi.org/10.1007/3-540-44709-1_26

[4] A. Everspaugh, K. Paterson, T. Ristenpart, and S. Scott, "Key rotation for authenticated encryption," In Proceedings of Advances in Cryptology (CRYPTO 2017), pp.98-129, Santa Barbara, USA, August 20-24, 2017. https://doi.org/10.1007/978-3-319-63697-9_4

[5] Ellie Pin-Yu Chen and Quincy Wu, "Time-based secure key management and rotation of healthcare databases," accepted and to appear in 2023 International Conference on Innovation, Communication and Engineering (ICICE), Bangkok Thailand, November 9-13, 2023.

[6] L. Bracciale, P. Loreti, E. Raso, and G. Bianchi, "TooLate: cryptographic data access control for offline devices through efficient key rotation," In Proceedings of the 2th Workshop on CPS&IoT Security and Privacy (CPSIoTSec '21), pp.57-62, New York, USA, November 2021. https://doi.org/10.1145/3462633.3483982

[7] N. Haller, C. Metz, P. Nesser, and M. Straw, "A one-time password system", IETF RFC 2289, February 1998.

[8] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "TOTP: time-based one-time password algorithm", IETF RFC 6238, May 2011.

[9] R. Baldwin, R. Rivest, "The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS algorithms", IETF RFC 2040, October 1996.

[10] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, "The secure real-time transport protocol (SRTP)", IETF RFC 3711, March 2004.

[11] FIPS PUB 197, "Advanced encryption standard (AES)," National Institute of Standards and Technology, U.S. Department of Commerce, November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[12] Tony Le Bourne, "ACM Ryzen 7 1700X review - AES Encryption Performance," *Vertez*, P.7, February 4, 2017.

[13] PyCryptodome, https://pycryptodome.readthedocs.io/