Hindawi
Security and Communication Networks
Volume 2017, Article ID 6760532, 9 pages
https://doi.org/10.1155/2017/6760532

Research Article

Protecting Private Data by Honey Encryption

Wei Yin,[1] Jadwiga Indulska,[2] and Hongjian Zhou[1]

[1]*North China Institute of Computing Technology, Beijing, China*
[2]*School of ITEE, The University of Queensland, Brisbane, QLD, Australia*

Correspondence should be addressed to Wei Yin; yinwei168@gmail.com

Received 10 July 2017; Revised 6 October 2017; Accepted 6 November 2017; Published 21 November 2017

Academic Editor: Leandros Maglaras

The existing password-based encryption (PBE) methods that are used to protect private data are vulnerable to brute-force attacks. The reason is that, for a wrongly guessed key, the decryption process yields an invalid-looking plaintext message, confirming the invalidity of the key, while for the correct key it outputs a valid-looking plaintext message, confirming the correctness of the guessed key. Honey encryption helps to minimise this vulnerability. In this paper, we design and implement the honey encryption mechanisms and apply it to three types of private data including Chinese identification numbers, mobile phone numbers, and debit card passwords. We evaluate the performance of our mechanism and propose an enhancement to address the overhead issue. We also show lessons learned from designing, implementing, and evaluating the honey encryption mechanism.

1. Introduction

Most people in China (as in any other country) are annoyed by junk text messages. The Internet users can also be affected by identity theft when criminals are using someone's identity. This can occur because some sensitive private data was not well protected and was then maliciously used by other parties causing damage to finances and reputation of the data owner.

When purchasing a product online, we are asked to provide our mobile phone number for the delivery purpose. When buying a train ticket in China, we need to fill in the identification card number. The commercial parties gather such sensitive private data. Some store them in a plaintext format. Some employ password-based encryption (PBE) [1]. However, the robustness of encryption depends on the key length. Although current encryption algorithms are considered secure, given enough time and computing power, they will be vulnerable to brute-force attacks. Also, the existing encryption mechanisms have a vulnerability; that is, when decrypting with a wrongly guessed key, they yield an invalid-looking plaintext message, while when decrypting with the right key, they output a valid-looking plaintext message, confirming that the ciphertext message is correctly decrypted.

Juels and Ristenpart [2] proposed the honey encryption concept to address this vulnerability and make the PBE encryption more difficult to break by brute-force. The honey term in the information security terminology describes a false resource. For example, honeypot [3] is a false server that attracts attackers to probe and penetrate. Honeyword [4] is a false username and password in the database. Once used for login, an intrusion is detected. Honey encryption can also address the previously mentioned vulnerability. Even when a wrong key is used for decryption, the system can yield a valid-looking plaintext message; therefore, the attacker cannot tell whether the guessed key is correct or not.

The innovation of honey encryption is the design of the distribution-transforming encoder (DTE). According to the probabilities of a message in the message space, it maps the message to a seed range in a seed space, then it randomly selects a seed from the range and XORs it with the key to get the ciphertext. For decryption, the ciphertext is XORed with the key and the seed is obtained. Then DTE uses the seed location to map it back to the original plaintext message. Even if the key is incorrect, the decryption process outputs a message from the message space and thus confuses the attacker.

The contribution of this paper is threefold. First, we design and implement the honey encryption system and apply the concept to three applications including Chinese identification numbers, mobile numbers, and passwords. These applications are based on uniformly distributed message spaces and the symmetric encryption mechanism. We

also extend honey encryption to applications with nonuniformly distributed message spaces and an asymmetric encryption mechanism (RSA). Second, we evaluate the performance of our honey encryption mechanism and propose an enhancement. Third, we discuss lessons learned from implementing and evaluating the honey encryption technique.

The rest of this paper is organised as follows. Section 2 presents the related work, followed by the discussion of the honey encryption concept in Section 3. The design and implementation are described in Section 4 and the use of honey encryption in three applications is shown in Section 5. The performance evaluation is discussed in Section 6 and a performance enhancement is proposed in Section 7. Section 8 discusses applications with nonuniformly distributed message spaces and use of asymmetric encryption in honey encryption. Section 9 describes the learned lessons followed by the conclusion in Section 10.

2. Related Work

Most of the systems with encryption use password-based encryption (PBE). These systems are susceptible to brute-force guessing attacks. Honey encryption [5] aims to address this vulnerability by not allowing attackers to gain much information from password guessing. For each possible key, the system outputs a valid-looking decrypted message. So it is hard to tell which one is the correct password. This way honey encryption can protect sensitive data in many applications.

Honey encryption deceives attackers that the incorrectly guessed key is valid. Many luring technologies that also use the term honey have been proposed in the last 20 years. Honeytokens [6] are decoys distributed over a system. If any decoy is used, this means that a compromise is taking place. For example, honeywords are passwords that are rarely used by normal users. Once a login attempt using a honeyword occurs, the system rises an alarm. Honeypots [3], Honeynet [7], and Honeyfarm [8] are luring systems that present many vulnerabilities. They are likely to become the targets for attackers. The objective of setting such systems is to study attackers' motivations, tools, and techniques.

Honey encryption is also related to Format-Preserving Encryption (FPE) [9] and Format-Transforming Encryption (FTE) [10]. In FPE, the plaintext message space is the same as the ciphertext message space. In FTE, the ciphertext message space is different from the message space. Honey encryption maps a plaintext message to a seed range in the seed space. Since the message space and the seed space are different, the ciphertext message space is different from the message space.

While Vinayak and Nahala [11] apply the honey encryption concept to MANETs to prevent ad hoc networks from the brute-force attack, Tyagi et al. [5] adopt the honey encryption technique to protect credit card numbers and a simplified version of text messaging. Most of these data in [5, 11] are from uniformly distributed message spaces. However, genomic data usually has highly nonuniform probability distributions. The GenoGuard mechanism [1] incorporates the honey encryption concept to provide information-theoretic confidentiality guarantees for encrypted genomic data.

In [1, 5, 11], a fixed distribution-transforming encoder (DTE) is utilised for encryption and decryption, so it is only suitable for binary bit streams or integer sequences, not for images and videos. Yoon et al. [12] propose a visual honey encryption concept which employs an adaptive DTE so that the proposed concept can be applied to more complex domains including images and videos. Jaeger et al. [13] provide a systematic study of honey encryption in the low-entropy key settings. They rule out the ability to strengthen the honey encryption notions to allow known-message attacks when attackers can exhaust the key space.

Our paper focuses on applying the honey encryption technique to three new applications including citizens' identification numbers, mobile phone numbers, and debit card passwords. This data is vital private information that can cause serious damage to person's finance and/or reputation if stolen. Although honey encryption has been applied to a number of applications, due to the variety of message formats and probability features, the message space design needs to vary for new types of applications. The applications discussed in our paper are carefully selected, because later we will show that the protection honey encryption provides varieties for different applications: stronger for debit card passwords, weaker for mobile phone numbers.

In our comprehensive design and implementation of the honey encryption mechanisms for three different applications we cover small/large message spaces, uniformly/non-uniformly distributed message probabilities, and symmetric/asymmetric encryption mechanisms. As far as we know, our paper is the first one to study the performance of honey encryption. We discover the performance problem for large message spaces and present a performance optimisation for small message spaces. We also show that the capability of honey encryption to address the brute-force vulnerability could be lost if the message space has not been well designed and that this design needs to vary for different types of applications.

3. Honey Encryption Concept

Honey encryption protects a set of messages that have some common features (e.g., credit card numbers are such messages). A message set is called a message space. Before encrypting a message, we should determine the possible message space. All messages in the space must be sorted in some order. Then the probability of each message (PDF) that occurs in the space and the cumulative probability (CDF) of each message are needed. A seed space should be available for the distribution-transforming encoder (DTE) to map each message to a seed range in the seed space (n-bit binary string space). The DTE determines the seed range for each message according to the PDF and CDF of the message and makes sure that the PDF of the message is equal to the ratio of the corresponding seed range to the seed space. The n-bit seed space must be big enough so that each message can be mapped to at least one seed. A message can be mapped to multiple seeds and the seed is randomly selected.

Let us consider using honey encryption to encrypt the coffee types, as shown in Figure 1. The coffee message space M consists of Cappuccino, Espresso, Latte, and Mocha. These four messages are sorted alphabetically. Let us assume that 4/8

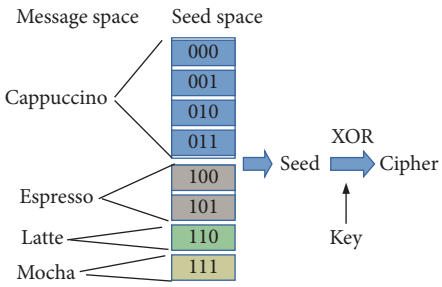


Figure 1: Honey encryption example.

people in Sydney like Cappuccino, 2/8 like Espresso, 1/8 like Latte and Mocha. The seed space is a 3-digit space. According to those probability statistics, we map these four messages to four ranges in the seed space. When encrypting Cappuccino, the DTE randomly selects a seed from the corresponding seed range. The seed is XORed with the key and the ciphertext is generated.

For decryption, the ciphertext is XORed with the key to obtain the seed. Then the DTE inversely maps the seed to the original plaintext message. In the encryption process, a message could have multiple mapping choices and the mapping is directional and random. However, since we sort plaintext messages in the message space and determine the seed range by the PDF and CDF of each message, it can be guaranteed that the seed ranges are arranged in the same order and the cumulative probability of the seed range in the seed space is equal to the cumulative probability of the message in the message space. Therefore, we establish an inverse_table that consists of mappings of the cumulative probability to the plaintext message. Finding the seed, we can determine the seed range. Finding the seed range, we can determine the cumulative probability shared by the seed range and the corresponding plaintext message. Then by looking up the cumulative probability in the inverse_table, we can find the original plaintext message and the ciphertext is decrypted.

In [2], Juels and Ristenpart discussed the robustness of honey encryption. First, the robustness may be compromised if the adversary has some side information about the message space. Second, if the key and message distribution are dependent or correlated, an attacker may be able to compare the decrypted message with the decryption key to identify a correct message. But even with those limitations, in the worst scenario, honey encryption security falls back to normal PBE security and therefore there is no drawback in using honey encryption.

4. Design and Implementation

We can design DTE as a common module that implements the encryption and decryption algorithms. For encryption, the DTE module takes in some parameters from the message space including the PDF and CDF probabilities of each message. Therefore, we abstract some interfaces for DTE to use when designing the message space. For decryption, the main task for DTE is to search the inverse_table and find the correct plaintext message. Therefore, the message space implementation should provide interfaces for probabilities and the inverse_table.

4.1. Message Space APIs. DTE maps the plaintext message to a seed in a seed range. The starting point of the seed range is determined by the CDF of the message, while the end point of the seed range is determined by the PDF of the message. Therefore, we define an interface for the message space containing functions including the cumulative_probability(mesg) function and the probability(mesg) function. These two functions accept a plaintext message as the parameter and output the CDF and PDF, respectively.

In decryption, DTE finds the plaintext message from the inverse_table by looking up the cumulative probability of the seed. The inverse_table is stored in a file. We define another function as get_inverse_table_file_name() in the message space interface. The function returns the filename of the inverse_table for DTE to look up and decrypt the ciphertext. If the inverse_table is not large, we can store the content in the memory when the system initiates. Then during decryption, the binary search method can be utilised to save time. However, if the inverse_table size exceeds the available system memory, DTE needs to read the inverse_table file line by line and find the plaintext by linear search.

4.2. DTE Implementation. DTE maps the plaintext message into a seed range, randomly selects a seed from the range, and XORs the seed with the key to output the ciphertext. The beginning of the seed range is determined by the CDF and the end of the seed range is determined by the PDF. The seed is randomly selected from the range.

When decrypting a ciphertext, the ciphertext is XORed with the key to obtain the seed. Then DTE determines the location of seed in the seed range. The location is corresponding to a probability value which lies between the CDF of the message and the CDF of the next message in the message space. Every line in the inverse_table contains a cumulative probability and its corresponding plaintext message. All lines are sorted by the cumulative probability. By searching the inverse_table, the DTE can find the plaintext message given the cumulative probability determined by the seed.

5. Applications

In this section, we apply the implemented honey encryption technique to private sensitive data including Chinese identification numbers, Chinese mobile phone numbers, and passwords. The code for DTE and the message space interface can be reused. However, the message space implementation should be customised and this is the focus of this section.

5.1. Identification Numbers. Identification numbers identify citizen's personal information and are widely used for authentication. Therefore they are used and stored by many commercial organisations. Stolen/leaked identification numbers can be misused by malicious users.

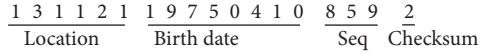


Figure 2: Identification number format.

The identification number consists of 18 digits, as shown in Figure 2. The first 6 digits are location symbols, identifying which suburb, city, and state in which persons were born. The 7th to 14th digits represent the birth date with the format of YYYYMMDD. For example, the birth date of a person born on 11th May 1985 will be presented as 19850511. The 15th to 17th is the sequence code. It uniquely identifies people in the same suburb who were born on the same day. Specifically, the 17th digit shows that the person is a male if it is odd. The 18th digit is the checksum. In this paper, we are not concerned with how exactly it is computed because it is deterministic. When generating the message space, we neglect this bit to simplify our implementation.

By gathering statistics, we found that China has 3519 location symbols. The 11th to 14th digits have 365 choices, as a year has 365 days. The sequence code has 999 choices, but in fact, this value rarely reaches 999. So the message space has $N = 3519 * Y * 365 * 999$ messages, where Y is the number of years considered. We assume each message has the same PDF of p, then $p = 1/N$. The user identification number in a commercial database should have a uniform distribution because a person has only one unique identification number and can only register in a database once (e.g., in the Taobao website, a large e-commerce company in China). Since we cannot obtain an identification number database from any business, we construct such a database with the assumption that each identification number has the same probability.

Therefore, the probability(mesg) function returns p for each message. We sort the message space incrementally. Then the CDF of a message depends on where the message is located in the message space. As every message has the PDF of p, the CDF of the message is i/N, where i is the message location in the message space. So when implementing the cumulative_probability(mesg) function, we first determine the location of the mesg, i, and then calculate the CDF.

In this implementation, we find that the size of the message space and the inverse_table file is much larger than the available memory space if we consider 100 years and 999 sequence code. To address this problem, we divide the identification number into four parts including location, year, month-day, seq, and checksum and store their possible values in different files. Therefore the identification number without the checksum can be viewed as a $L\_6Y\_4MD\_4S\_3$, where L_6 is a location code, Y_4 is a year, and MD_4 is a month and a day. We group a month and a day together because different months may have a different number of days.

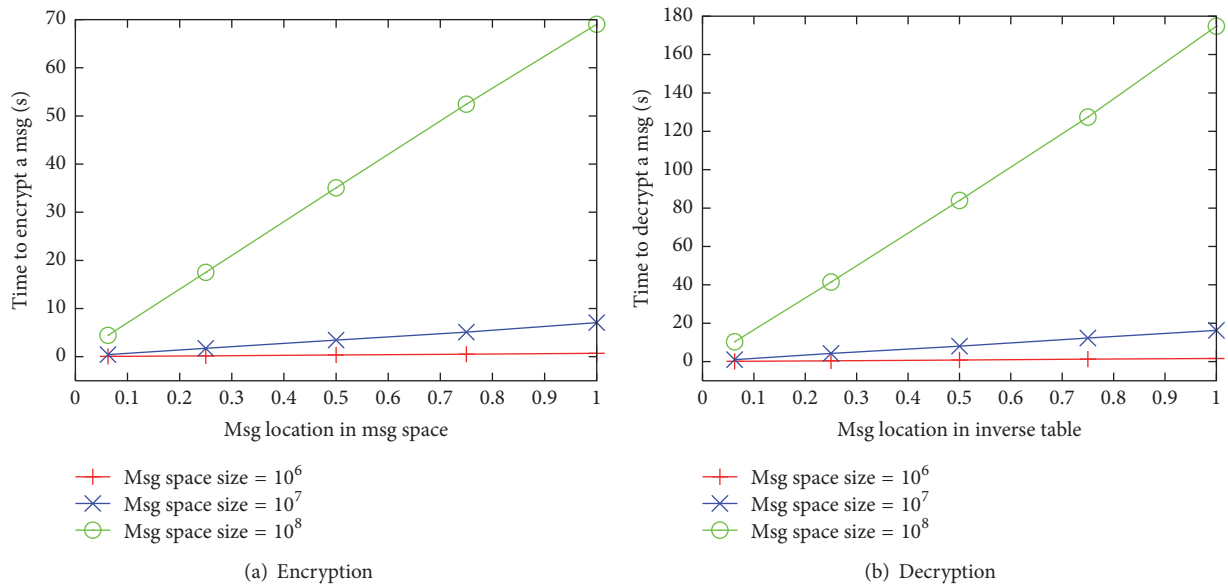
In this way, we construct a message space considering all people in China who were born between 1917 and 2016 and that the sequence code lies between 1 and 200. DTE reads these four files into the memory and stores these values in four different lists. To encrypt a message, DTE joins the four parts to make an identification number and compares it to the message to be encrypted to determine the location of the message. Then the cumulative probability is calculated and the message is encrypted. For decryption, the system XORs the ciphertext with the key to obtain the seed. Then DTE calculates the probability, P_s, dividing the seed by the seed space size. Afterwards DTE begins to make identification numbers and their cumulative probabilities. By comparing P_s with those cumulative probabilities, DTE finally finds the original plaintext message.

Although we solve the memory problem by storing the identification numbers in different files, we find that the time to honey encrypt and decrypt a message in such a large message space is very high. In addition, if taking into account a nonuniformly distributed message space, it would be more complicated, since different messages have different PDFs and CDFs that cannot be calculated by determining the message location.

5.2. Mobile Numbers. Nowadays, mobile phone numbers are combined with the debit cards for financial transaction purposes and therefore mobile numbers should be well protected. The mobile phone number consists of 11 digits. The first 3 digits represent the operator code. There are three operators, China Mobile, China Unicom, and China Telecom in China. Each operator has been assigned a number of operator codes. The 4th to 7th digits are the location code of the mobile phone. The 8th to 11th digits are random. We implement the honey encryption technique for the Beijing Unicom numbers. For Beijing Unicom, the first 7 digits have 2872 choices, and the last 4 digits have 10^4 choices, so the message space consists of $N = 2872 * 10^4$ messages. Let us assume that each number has the same PDF of p, then the probability(mesg) function returns probability $p = 1/N$. Each mobile number in a commercial database is unique for a particular telecommunication company in China. So we assume a uniform distribution of these numbers. The cumulative_probability(mesg) function returns i/N, where i stands for the location of the message in the message space.

We define the MobileNumber class and implement the mentioned three functions for the message space interface. Given a mobile number, DTE outputs a random seed, which is XORed with the key to get the ciphertext. For a specific mobile number, using the system to encrypt it multiple times, the system outputs different ciphertexts due to the randomness in the seed generation process. When the ciphertext is XORed with the key, we can obtain the seed. Then the seed is used by DTE to get the correct plaintext of the mobile number.

5.3. Passwords. Passwords usually consist of uppercase and lowercase letters, digits, and symbols. Many users use weak passwords. For example, the debit card uses a 6-digit password for withdrawing money from the ATM machine. Honey encryption can help to protect such passwords from brute-force attacks. The 6-digit password space consists of $N = 10^6$ messages, ranging from 000000 to 999999. We sort the message space incrementally and assume each message has equal probability, so the probability(mesg) function returns PDF = $1/N$. Different people may choose the same password



(a) Encryption (b) Decryption

Figure 3: Time to encrypt/decrypt a message.

therefore the distribution may not be uniform. However, for simplicity, we assume uniformly distributed messages and therefore the cumulative_probability(mesg) function returns i/N. We will discuss nonuniformly distributed message spaces in Section 8.1.

We implement the password class by defining the three functions in the message space interface. When combined with DTE, the system can encrypt and decrypt correctly. Also when using a wrong key to decrypt a message, the system outputs a message that does not indicate that the key is not correct.

6. Evaluation

The platform for evaluating our honey encryption system is the Toshiba Portege-M800 laptop. The processor is Intel Core 2 Duo 2.0 Hz. The memory has a 3 GB RAM. The operating system is Ubuntu Kylin 16.04. The goal of experiments is to study the time taken to encrypt and decrypt a message. In order to make it easy to increase the size of the message space for multiple times, we choose the password message space for evaluation and increase the size from 10^6 to 10^8.

6.1. Time to Encrypt a Message. For encryption in a large message space, DTE should read the message space file line by line, calculate the PDF and CDF, determine the seed range, and randomly select a seed from the range. Finally, the chosen seed is XORed with the key to obtain the ciphertext.

We extend the message space size from 10^6 to 10^8 and conduct an evaluation. The time to encrypt a message is measured and displayed in Figure 3(a). The x-axis presents the message location in the message space. For example, 0.25 stands for the message that is located at 25% of the message space. The y-axis represents the time taken to encrypt a message. It can be observed from the figure that the encryption time increases as the location of the message moves deeper. This is because the encryption algorithm reads the message space line by line until it finds the message to get the probabilities. The larger the message space, the more time it needs for encryption because the most time-consuming work in this encryption is reading and processing the message space. For the message space that contains 10^6 or 10^7 messages, the time is reasonable, but for a message space of 10^8 messages, the maximum time to encrypt a message can be as high as 70 s, which is too high.

6.2. Time to Decrypt a Message. During the decryption process, DTE first XORs the key with the ciphertext and obtains the seed. Then it determines the location of the seed in the seed space. Using the location information, it looks up the inverse_table and gets the corresponding plaintext message.

We measure the time to decrypt a message in three message spaces, ranging from 10^6 to 10^8, and display these statistics in Figure 3(b). The x-axis stands for the location of a plaintext message in the inverse_table, and the y-axis represents the time to decrypt the message. As shown from the figure, the decryption time increases as the plaintext message location in the inverse_table goes deeper. This is because the decryption algorithm reads line by line the inverse_table file until it finds the plaintext message. The larger the inverse_table, the slower the decryption process because the most time-consuming part in this decryption is to process the inverse_table file. When the inverse_table size is 10^6, the time to decrypt a message is acceptable, but when the inverse_table size is 10^8, the time can reach 160 s. Comparing Figures 3(a) and 3(b) it can be seen that the time to decrypt and encrypt a message is different because the message space file only contains a message in one line, but the inverse_table

file contains a message and its cumulative probability for each line. Thus processing the latter takes more time.

7. Enhancement

For a large message space, the decryption algorithm needs to read the inverse_table file line by line and find the correct plaintext message using the calculated cumulative probability. For a small message space, we can read the whole inverse_table into the memory and use the binary search method to find the corresponding plaintext message in the decryption process.

For a large message space, the encryption algorithm needs to read the message space file and determine the message's PDF and CDF. But if the message space is incrementally sorted like the password message space, the value of the message, V, has a relationship with its location, A, in the message space; that is, $A = V + 1$. Also, the cumulative probability is related to the message location in the message space; that is, CDF = A/N, where N is the number of messages in the message space. Therefore, instead of searching the message space file for CDF, we can calculate the CDF. It should be noted that not all message spaces have such features. Taking the identification number, for example, the CDF of a message is not related to the value of the message itself.

We improve the encryption and decryption algorithm and evaluate their performance. Figure 4(a) shows the encryption time of the enhanced encryption algorithm. For 10^6 and 10^7 message spaces, no matter where the message is located, the encryption time is only around 136 microseconds. The lines for both 10^6 and 10^7 message spaces overlap. This means the encryption time is independent of the message space size.

Figure 4(b) shows the time taken for the enhanced decryption algorithm. No matter where the message is located in the message space, the decryption time is around 45 microseconds. The lines for both 10^6 and 10^7 message spaces overlap, which means the decryption time is independent of the message space size. This may be because the difference of the binary search algorithm in the 10^6 and 10^7 message spaces is not significant. We tried the 10^8 message spaces to verify this case, but the system fails due to memory error because the available memory is too small to hold the inverse_table file containing 10^8 messages.

8. Other Applications

We discussed honey encryption for identification numbers, mobile phone numbers, and debit card passwords in Section 5. In those applications, we assume the message space forms a uniform distribution. We also combine honey encryption with a simple symmetric encryption mechanism, XOR, for encryption and decryption. In this section we discuss the implementation of honey encryption with a nonuniformly distributed message space. We also extend honey encryption to an asymmetric encryption mechanism, RSA.

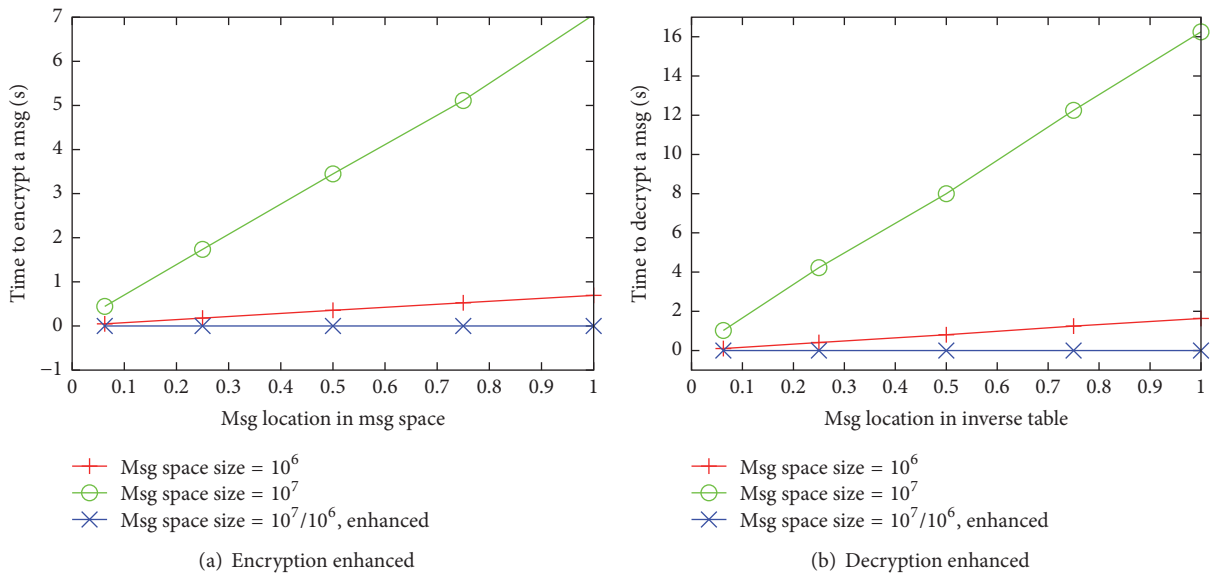
8.1. Application with Nonuniform Distribution. The coffee example in Section 3 has a nonuniform distribution. Let us consider applying honey encryption to a similar application but with a message space of 10^6 messages. We store the PDF of each message into a binary search tree and the CDF of each message into another. The probability(mesg) and cumulative_probability(mesg) functions calculate the corresponding probabilities by looking up these two trees. For decryption, another binary search tree is utilised to store the cumulative probability and plaintext message pairs. The time to encrypt and decrypt a message is shown in Figure 5. The encryption time is almost twice as the decryption time, because, for encryption, the system looks up two trees while, for decryption, it needs only one tree lookup. For the 50% location, the decryption and encryption time is almost 0. This is because the corresponding message is the root of the tree. The time increases from the center to two sides, because the message location goes deeper in the tree. This application shows that PDF and CDF probabilities can be stored in various data structures if the message space has a nonuniform distribution.

8.2. Application with Asymmetric Key Encryption. So far, we have combined a symmetric encryption mechanism, XOR, with the honey encryption technique to produce a ciphertext from a given seed. The symmetric key scheme assumes that the involved parties can store the symmetric key securely and the key is distributed to both parties by a secure channel. In this section, we extend the honey encryption mechanism to a public key encryption mechanism, RSA, to mitigate this limitation.

The encryption process can be easily integrated with RSA. The 1024-bit public and private keys are generated by the RSA algorithm. For encryption, the plaintext message is mapped to a seed by the DTE encode() process and then the seed is encrypted by an RSA public key to generate the ciphertext. For decryption, the RSA private key is used to decrypt the ciphertext to obtain the seed and then the seed goes through the DTE decode() process to obtain the plaintext message.

When decrypting with a wrong private key, RSA encounters an error instead of outputting a valid-looking seed. To solve this problem, the system captures the exception when a wrong decryption key is utilised and outputs a random seed from the seed range to confuse the attacker. If the decryption key is the correct private key, the system can call the RSA decryption function without any exception and obtain the right seed. The seed then is mapped to a cumulative probability corresponding to a plaintext message. We implemented the asymmetric key honey encryption mechanism for the debit card password application (enhanced version) with a message space size of 10^6.

We evaluate the performance of honey encryption with the RSA extension, as shown in Figure 6. It is observed from Figure 6 that the encryption time for RSA is four times higher than the symmetric key encryption mechanism and the difference is more significant for the decryption time. For the symmetric key encryption mechanism, the decryption time is only 46 microseconds. However, it is around 0.045 s for RSA. As RSA is computationally expensive, honey encryption based on RSA inherits this drawback.



(a) Encryption enhanced (b) Decryption enhanced

Figure 4: Time to encrypt/decrypt a message.

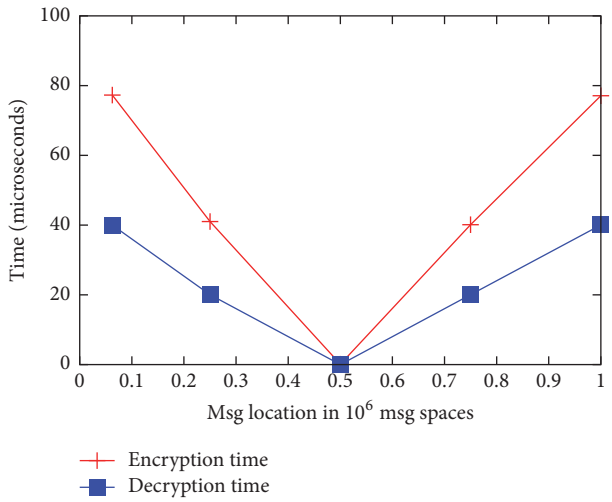


Figure 5: Performance in application with a nonuniform distributed message space.

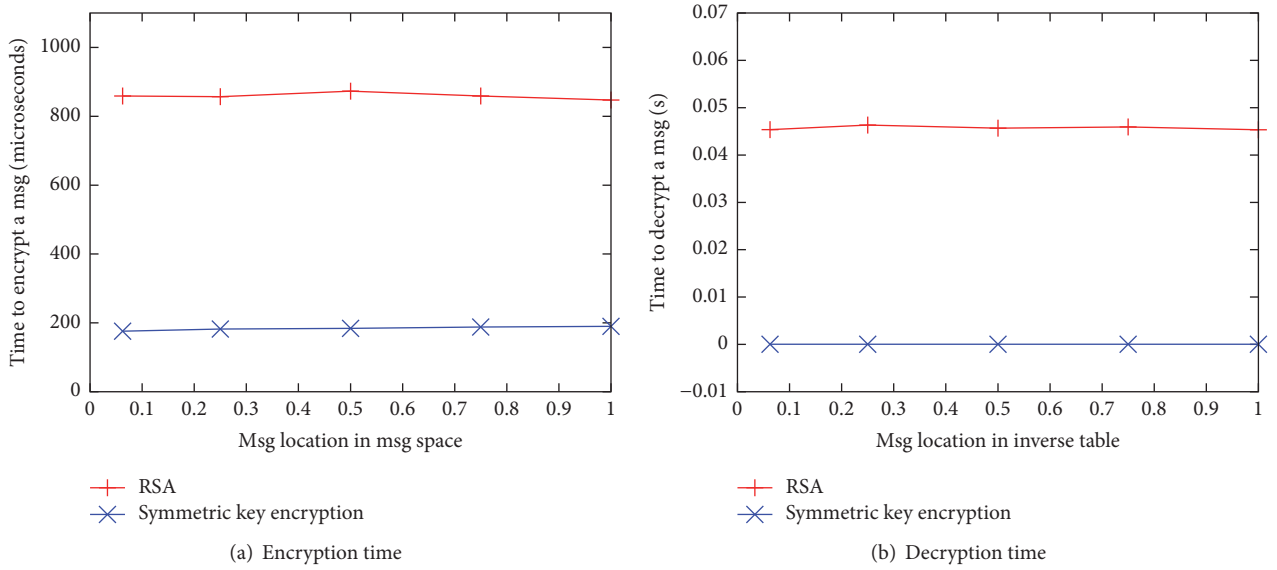
9. Lesson Learned

In our research on honey encryption, we find that it is an effective countermeasure for brute-force attacks. However, we also discover the following limitations.

(1) Honey encryption is suitable for a small, not large, message space as the overhead of processing a large message space is very high. In this mechanism, DTE needs to read the message space and inverse_table file line by line for encryption and decryption if the message space is larger than the available system memory. Having these files in the memory will speed up the search (e.g., by using the binary search method) for decryption. Therefore, we claim that honey encryption is suitable for encryption/decryption with a small message space; otherwise, the encryption/decryption systems should have advanced hardware configurations.

(2) The message space should be carefully designed, or honey encryption cannot well address the brute-force vulnerability. Although a plaintext derived by DTE from a wrongly guessed key looks like a correctly decrypted ciphertext, attackers can use other methodologies to confirm whether the guessed key is incorrect if the message space has not been carefully designed. In the mobile phone number case, the attacker can dial the mobile number to check whether the number is a correct one. For the identification database in a women's hospital, the deciphered identification number should not belong to a man. Identification numbers that belong to 0-to-4-year-old babies should be less likely in a database in an e-commerce company. In a middle school, most students should not be less than 12 years old and older than 19 years old. As the decryption process outputs a message from the message space, this message should not have any fingerprint that could be used by attackers to identify the correctness of the message.

(3) The capability of protecting sensitive private data provided by honey encryption varies for different applications. The decryption process outputs a message from the message space, no matter whether the key is correct or not. This feature could leak some valid messages and this may have different impact on different applications. Taking the identification number, for example, a malicious user can still get some valid identification numbers from the system, but the attacker may not be able to get the corresponding name of the identification holder. So the possibility for the attacker to maliciously use the identification to commit crimes is limited. For text message spammers, the leakage of mobile phone numbers is enough for sending spam. For the debit card passwords, no matter what message DTE outputs, it is useless as the message is a meaningless number. Therefore, the value of honey encryption may vary for different applications.



(a) Encryption time

(b) Decryption time

Figure 6: Time to encrypt/decrypt a message.

(4) The implementation of honey encryption must be customised for different applications as the message space varies. To apply honey encryption to a specific application, it requires the developer to design/customise the message space and inverse_table.

10. Conclusion

Private data should be well protected to avoid loss due to leakage and misuse. The existing password-based encryption (PBE) methods used to protect private data are vulnerable in face of brute-force attacks, as the attacker can determine whether the guessed key is correct or not by looking at the output of the decryption process. The honey encryption technique is a countermeasure for such a vulnerability. In this paper, we discussed the honey encryption concept and we also designed and implemented a honey encryption mechanism for Chinese identification numbers, mobile phone numbers, and debit card passwords. Applications with uniformly or nonuniformly distributed message spaces and with symmetric or asymmetric key encryption mechanisms are designed and implemented. The performance of our honey encryption mechanism was evaluated and an enhancement was proposed to address the overhead issue.

Finally, we discussed the lessons learned from our experience of designing, implementing, and evaluating the honey encryption mechanism. Specifically, we have the following observations. (1) Honey encryption is suitable for a small, not large, message space as the overhead of processing a large message space is very high. (2) The message space should be carefully designed for each application, or honey encryption may not properly address the brute-force vulnerability. (3) The capability of protecting sensitive private data provided by honey encryption varies for different applications. (4) The implementation of honey encryption must be customised for different applications as the message spaces vary.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work is partially supported by the NSFC project 61702542 and the China Postdoctoral Science Foundation project 2016M603017.

References

[1] Z. Huang, E. Ayday, J. Fellay, J.-P. Hubaux, and A. Juels, "GenoGuard: Protecting genomic data against brute-force attacks," in *Proceedings of the 36th IEEE Symposium on Security and Privacy (SP '15)*, pp. 447–462, Washington, DC, USA, May 2015.

[2] A. Juels and T. Ristenpart, "Honey encryption: security beyond the brute-force bound," in *Advances in Cryptology-EUROCRYPT 2014*, pp. 293–310, 2014.

[3] P. Owezarski, "A near real-time algorithm for autonomous identification and characterization of honeypot attacks," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2015*, pp. 531–542, Singapore, April 2015.

[4] A. Juels and R. L. Rivest, "Honeywords: making password-cracking detectable," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '13)*, pp. 145–160, ACM, November 2013.

[5] N. Tyagi, J. Wang, K. Wen, and D. Zuo, "Honey encryption applications," in *Computer and Network Security*, pp. 1–16, 2015.

[6] L. Spitzner, *Honeytokens: The Other Honeypot*, 2003.

[7] I. S. Kim and M. H. Kim, "Agent-based honeynet framework for protecting servers in campus networks," *IET Information Security*, vol. 6, no. 3, pp. 202–211, 2012.

[8] P. Jain and A. Sardana, "Defending against internet worms using honeyfarm," in *Proceedings of the International Information*

Technology Conference (CUBE '12), pp. 795–800, Pune, India, September 2012.

[9] B. Mihir, T. Ristenart, P. Rogaway, and T. Stegers, "Format-preserving encryption," in *Selected Area in Cryptography*, pp. 295–312, 2009.

[10] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Protocol misidentification made easy with format-transforming encryption," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 61–72, ACM, November 2013.

[11] P. P. Vinayak and M. A. Nahala, "Avoiding brute force attack in manet using honey encryption," *International Journal of Science and Research*, vol. 4, no. 3, pp. 83–85, 2015.

[12] J. W. Yoon, H. Kim, H.-J. Jo, H. Lee, and K. Lee, "Visual honey encryption: application to steganography," in *Proceedings of the 3rd ACM Information Hiding and Multimedia Security Workshop*, pp. 65–74, ACM, Portland, Ore, USA, June 2015.

[13] J. Jaeger, T. Ristenpart, and Q. Tang, "Honey encryption beyond message recovery security," in *Advances in Cryptology–EUROCRYPT 2016*, pp. 758–788, 2016.