Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 406915, 7 pages
http://dx.doi.org/10.1155/2015/406915

Research Article

A New Countermeasure against Brute-Force Attacks That Use High Performance Computers for Big Data Analysis

Hyun-Ju Jo and Ji Won Yoon

Graduate School of Information Security, Korea University, Room 614, Future Convergence Building, Anam 5ga, Seongbuk-gu, Seoul 136-701, Republic of Korea

Correspondence should be addressed to Ji Won Yoon; jiwon_yoon@korea.ac.kr

Received 24 October 2014; Accepted 21 January 2015

Academic Editor: Sangjun Lee

Using high performance parallel and distributed computing systems, we can collect, generate, handle, and transmit ever-increasing amounts of data. However, these technical advancements also allow malicious individuals to obtain high computational power to attack cryptosystems. Traditional cryptosystem countermeasures have been somewhat passive in response to this change, because they simply increase computational costs by increasing key lengths. Cryptosystems that use the conventional countermeasures cannot preserve secrecy against various cryptanalysis approaches, including side channel analysis and brute-force attacks. Therefore, two new countermeasures have recently been introduced to address this problem: honey encryption and the structural code scheme. Both methods look different; however, they have similar security goals and they both feature distribution transforming encoders based on statistical schemes. We unify them into a statistical code scheme in this study. After explaining the statistical code scheme, we describe the structural code scheme, which has not been adopted as widely as the honey encryption.

1. Introduction

Currently, we can collect, process, analyze, and transmit massive amounts of data using high performance parallel and distributed computing systems. For instance, we can process and analyze a substantial amount of data in real time using the graphics processing units (GPUs) or the field-programmable gate arrays (FPGAs). These are specialized electronic circuits designed to rapidly manipulate vast amounts of data (commonly referred to as big data) and to accelerate their computation. However, this technology also allows malicious individuals to access high computational power to attack the cryptosystems. In other words, if malicious individuals can crack cryptosystems with GPUs or FPGAs, they can also find secret keys within a reasonable amount of time. Typically, attackers do not use sophisticated techniques to crack the cryptosystem but rather simple techniques such as *brute-force attacks*.

The *brute-force attack* is one of the simplest and strongest cryptanalysis methods. This attack searches for only one key, which returns meaningful plain text by computing all keys in the key space. From ancient Egypt to the present day, this cryptanalysis method has threatened many cryptosystems [1]. If attackers have sufficient time and computational power, they would be able to crack most cryptosystems by brute-force attack [2]. It is known that one-time pad (OTP) is the only secure cryptosystem against the brute-force attack.

Accordingly, modern cryptosystems have been developed with two different countermeasures. The first countermeasure is to increase computational complexity. If computational complexity is high, an attacker cannot crack the cryptosystem within a polynomial time. Therefore, cryptosystems protect secrecy by increasing the key length or the entropy using a pseudorandom number generator (PRNG). The second countermeasure, which has been recently invented [3, 4], protects secrecy by using statistical features. Two techniques from this countermeasure are utilized in this study: the honey encryption and the structural code scheme [4]. In this paper, we unify both approaches as the *statistical code scheme*, because both are based on statistical properties.

While conventional codes schemes, including the American Standard Code for Information Interchange (ASCII)

and Unicode, use deterministic encoders and decoders, the statistical code scheme uses statistical features to encode and decode words. While Juels and Ristenpart published a seminal article about the honey encryption in 2014 [4], a patent for a similar concept, the structural code schemes [5], was applied for that in 2013; this scheme was originally designed for statistical deniable encoders. Articles describing the structural code scheme were also published in the academic literature in 2014 [3].

As shown in Table 1, the statistical code scheme is divided into the honey encryption and the structural code scheme. Both countermeasures use statistical features against brute-force attacks; however, they have different designs. While honey encryption focuses on generating a similar distribution, the structural code scheme's focus is to build a similar structure from the learning system using Markov processes. Both schemes provide semantic security; however, the structural code scheme also provides syntactic security. In addition, while the honey encryption does not support natural language processing (NLP), the structural code scheme can handle natural languages using language models. However, unlike the honey encryption, the formal mathematical proof has not yet been provided for the structural code scheme. In this paper, we focus on the structural code scheme, which is relatively less popular than honey encryption in the fields of cryptography and security.

The key contributions of this paper can be summarized as follows:

(i) we claim that conventional cryptosystems can be cracked by *brute-force attacks* utilizing high performance computers used for big data analysis;

(ii) we explain why new countermeasures against *brute-force attacks* must be developed;

(iii) we propose that the structural code scheme and the honey encryption can be unified in the statistical code scheme;

(iv) we compare the characteristics of the honey encryption and the structural code scheme;

(v) we describe the algorithms of the structural code scheme in detail;

(vi) we evaluate and analyze the performance of structural code scheme.

The paper is organized as follows. In the following section, we explain the information interchange code, probabilistic language model, and Markov process. We will describe the details of the structural code scheme in Section 4. Next, we highlight the differences of the proposed scheme through a comparative analysis with ASCII and evaluate the performance in Sections 5 and 6. We then describe our conclusions and provide suggestions for future research.

2. Related Works

We mentioned two different types of countermeasures against the *brute-force attacks* in the introduction. Increasing the computational complexity using key length or PRNG is a strategy that has been used and studied for years; however, it shows limitations and weaknesses against *brute-force attacks*. For example, PSNR's lack of entropy decreases key space; therefore, the cryptosystem's key is revealed by a *brute-force attack* [6]. Goldberg's study suggests that the predictability of Netscape's seed value is related to low PRNG performance [7]. Debian OpenSSL, RSA Public key factoring, and Bitcoin's Elliptic Curve Digital Signature Algorithm also show vulnerability against *brute-force attacks* caused by low PRNG performance [8–10]. The same problem was also discovered in a fuzzy fingerprint vault system that protects fingerprints for user authentication [11].

Another factor that may lead to *brute-force attacks* is hardware with chips such as GPU and field-programmable gate arrays (FPGAs). Hardware with such chips can compute large amounts of data at once because they have advanced computing power through parallel processing. Therefore, although extending the key length increases computational complexity, hardware can solve it if sufficient time and efficient CPUs are available. In practice, data encryption standard (DES) was decrypted by the parallel processing capabilities of the DES cracking machine built by Electronic Frontier Foundation (EFF) [12].

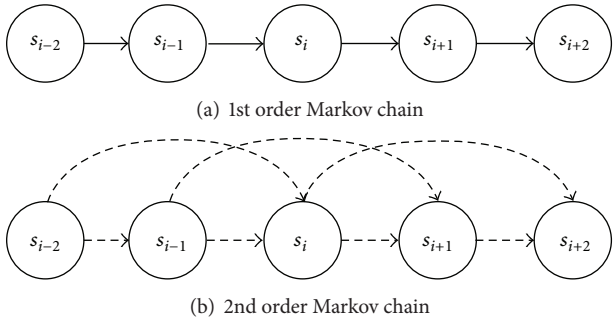
Given these factors, simply increasing the computational complexity is not sufficient to maintain secrecy. Hence, the statistical code scheme is presented as a new countermeasure. In 2013 the related work "Honeywords" appeared, and Honey encryption was presented in January 2014 [4, 13].

Table 1: Comparison of statistical code system subapproaches: the right column explains the structural code scheme (S-CS) and the left one shows the honey encryption (HE).

	Statistical code system	
	Honey encryption	Structural code system
Design	Distribution-based (fixed distribution)	Structure-based from learning (flexibly adaptive distribution)
Pros	Semantic security An explicit proof is provided	Semantic security Context sensitive NLP is available
Cons	Cannot support long messages Cannot support NLP	Cannot support long messages Formal proof has not been provided yet

3. Background

3.1. Code for Information Interchange. Information interchange code represents various computer data in electronic communication devices and accommodates the language and symbols of many countries. Although a variety of schemes already exists, the most representative code scheme is ASCII. As users encode and decode data, they use an information interchange code table that includes bit and sign information. Bit information corresponds to sign information, and this relation is a one-to-one function. Therefore, every user



(a) 1st order Markov chain

(b) 2nd order Markov chain

Figure 1: A simple Markov process. Top (a) shows a 1st order Markov chain only affected by one front value, and (b) shows a 2nd order Markov chain.

obtains the same result from the code table by deterministic features.

3.2. Probabilistic Language Model. A probabilistic language model represents a probability of a sequence of m words or strings given a stochastic process $\mathbf{S} = x_1x_2x_3, \ldots, x_m$. Each character of the words is realized (generated) from the stochastic process: $0 \leq p(x_i) \leq 1, i = 1, \ldots, m$. A discrete random process is a set of random variables that can be expressed in a discrete time order. The stochastic random process sp$(\cdot)$ has values in a state space $\mathbf{A} = \{s_1, s_2, \ldots, s_n\}$, which is a countable set. For instance, if we consider only integer numbers, then $\mathbf{A} = \{0, 1, 2, \ldots, 9\}$. In a similar manner, $\mathbf{A} = \{a, b, c, \ldots, z, A, B, \ldots, Z\}$ for characters. We can locate a natural language rule from the special features of sets that contain random variables per discrete time. For the probabilistic language model, we define the discrete time as a discrete section. A set of the word's probabilities in the discrete section is an n-gram model, and it indicates the probability of the subsequent n words. For example, a 1-gram model shows the probability of one character such as "a" or "b" and a 2-gram model indicates the probability of the subsequent two characters such as "aa" or "ab."

3.3. Markov Process. A Markov process is a process that satisfies a Markov property. This property indicates that the system can predict a future state depending on the current state. It is divided into a continuous-time Markov process and a discrete-time Markov process. In the Markov chain, discrete value $x_1, x_2, \ldots, x_m$ is a state and a set of states is named state space s_i. A Markov chain is defined by initial probability $p(x_1)$ and transition probability $T_{ab} = p(x_j = s_b \mid x_i = s_a)$. In the Markov chain, a chain will be connected to the front and the rear chains. Figure 1 shows the 1st order Markov process (a) and the 2nd order Markov process (b), respectively.

4. Structural Code Scheme

The structural code scheme is a new information interchange code designed to prevent *brute-force attacks*. The main purpose of this scheme is to create false plain text to protect the original plain text. While the false plain text consists of plain text decrypted with the wrong cryptography key, the correct plain text is plain text decrypted with the decryption key. Here, the false plain text is different from the correct plain text. This scheme produces semantically meaningful false plain text, so an attacker cannot determine whether the correct key was used. A probabilistic language model and a Markov process are used in the structural code scheme.

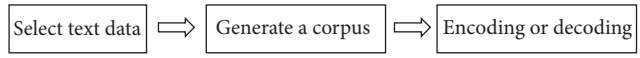


Figure 2: Structural code scheme steps.

Figure 2 shows the practical steps of the structural code scheme. First, a system user trains text data to obtain the word frequency, given an assumption that plain text follows the Markov process discussed in Section 3.3. The next step is to generate a corpus, which is a database for an encoding and decoding process [14]. The final step is to encode or decode user data with the corpus. We will explain the details of each step in the following paragraph.

4.1. Select Text Data. In the structural code scheme, we obtain the frequency of each character from the text data and use this frequency to create a probabilistic language model. Thus, the probabilistic language model reflects the properties of the text data. As we know, many studies on the frequency of characters in alphabets have already been conducted [15]. Interestingly, they revealed that the probabilities of "e" and "t" are high in English sentences. That is to say, when we generate a probabilistic language model from general text data, we will have high probabilities for "e" and "t." If we generate a language model using the play "Romeo and Juliet," the highest probability can be changed from "e" to Juliet's "j." In other words, if we locate "th" in the language model, it is highly probable that the next character will be "e" in a language model that includes the common word "the." In short, the probability depends on the properties of the text data. We always create a different probabilistic language model with this property when different data are used. After this step, we apply the kth order Markov process to create a corpus.

4.2. Generate a Corpus. The second step in the structural code scheme is to create a corpus for encoding and decoding. A corpus is a structured set (or database) of natural language data for a specific study. First, we apply the kth order Markov process to a probabilistic language model. The probabilistic model in a countable set has a Markov chain characterized by adopting Markov process. The kth order Markov process predicts a future state, based on the number of k previous states, such that the $(k-1)$-gram language model can be expressed as a kth order Markov process. The full joint probability of the k-order Markov process is defined as

$$p(\mathbf{x}_{1:m}) = p(x_1, x_2, \ldots, x_m) = \prod_{i=1}^{m} p(x_i \mid \mathbf{x}_{1:i-1}) = \prod_{i=1}^{m} p(x_i \mid \mathbf{x}_{i-k:i-1}). \quad (1)$$

In (1), $p(x_1 \mid \mathbf{x}_{i:i-1}) = p(x_i \mid \mathbf{x}_{i-k:i-1})$. For instance, $p(x_3 = \text{“}e\text{”} \mid x_2 = \text{“}h\text{”}, x_1 = \text{“}t\text{”})$ is the 2nd order Markov probability that “e” follows “th.” Probabilistic values of (1) are the foundation of a corpus that will be used as an encoder and a decoder. Next, we introduce a function F_{corpus} that maintains the corpus structure:

$$F_{\text{corpus}}\left(x_i = s_j \mid \mathbf{x}_{i-k:i-1}\right) = L \int_{s_1}^{s_j} p\left(x_i \mid \mathbf{x}_{i-k:i-1}\right) dx_i. \quad (2)$$

The corpus is a multiplication of a cumulative mass function (CMF) value and a constant value L. In (2), F_{corpus} is a CMF value of a random variable, which is $0 \leq p(x) \leq 1$. When multiplying the constant value L by CMF, the corpus's scope will be changed as $0 \leq F_{\text{corpus}}(x) \leq L$. There are two reasons for multiplying by constant value L. The first reason is to represent all characteristics in the probabilistic language model. Using CMF, we assign a large amount of cumulative mass to high probability words and assign a smaller amount of cumulative mass to low probability words. Thus, large and small probabilities are reflected to the corpus directly. The second reason for multiplying by constant value L is a data preprocessing step for lossless decoding. While changing to the corpus, if the probability is extremely small or zero, then it could be missed in the original model of (2). That is, every single random variable should correspond to a symbol value; therefore, we modify (2) as follows:

$$F_{\text{corpus}}\left(x_i = s_j \mid \mathbf{x}_{j-k:j-1}\right) = \begin{cases} L \int_{s_1}^{s_j} p\left(x_i \mid \mathbf{x}_{i-k:i-1}\right) dx_i + \alpha \\ \qquad \text{if } p\left(x_i = s_j \mid \mathbf{x}_{i-k:i-1}\right) > \dfrac{1}{L}, \\ L\left[\int_{s_1}^{s_j} p\left(x_i \mid \mathbf{x}_{i-k:i-1}\right) dx_i - p\left(s_j \mid \mathbf{x}_{i-k:i-1}\right)\right] + 1, \\ \qquad \text{otherwise}, \end{cases} \quad (3)$$

where α can be estimated using $F_{\text{corpus}}(x_i = s_n \mid \mathbf{x}_{j-k:j-1}) = L$. Equation (3) specifies that the corpus will have a range from 1 to L, $1 \leq F_{\text{corpus}}(x_i = s_i \mid \mathbf{x}_{j-k:j-1}) \leq L$. When the probability value is smaller than $1/L$, we can maintain lossless encoding and decoding of data by setting a minimum value using (3). We prepared Figure 3 as an example of these steps.

Figure 3 shows the process of creating a corpus [15]. Figure 3(a) represents the 0th order Markov process probabilistic language model and its corresponding CMF value (when the first character is “t,” $p(x_1 = \text{“}t\text{”})$). Figure 3(b) represents the CMF value of the 2nd order Markov process (when the third character is “e” under the joint probability $p(x_3 = \text{“}e\text{”} \mid x_2 = \text{“}h\text{”}, x_1 = \text{“}t\text{”})$) and the corpus that results from multiplying by constant value L. As shown in Figure 3, the corpus has bit strings for direct use in encoding and decoding. In Figure 3 the probability of character “e” is highest when the first character is “t” and the second one is “h.”

4.3. Encoding and Decoding. In the encoding step, we select a character and search for the corresponding bit string in the corpus. Every CMF value has at least one bit string; therefore, we can represent every character. When we find several corresponding bit strings for one character such as “e” in Figure 3, we select the first or last bit string for convenience.

In Figure 3, we assume the prior probabilities of characters “t” and “h” and compute the conditional probability of “e” as the third character. Conditional probability values are also saved in our corpus as a cumulative mass function. Therefore, the prior probability affects the next character's probability in the corpus, and it plays a main role for maintaining syntactic functionality. We encode “t” with the corresponding bit string from the corpus. Next, we create another corpus, which is constructed of conditional probability values from when the previous character was “t.”

The decoding step locates the corresponding bit string from the corpus. The random bit string is likely to correspond to a character sequence having many low probabilities in the corpus. In other words, a character having a high probability will be a first letter, and conditional probability will create meaningful words. As a result, the attacker cannot distinguish this word (false plain text) from real plain text and therefore secrecy will increase.

Table 2: Notation.

Symbol	Term	Explanation
C	Ciphertext	Ciphertext of a secure communication
P_{correct}	Correct ciphertext	Sender's plain text
$K_{\text{enc,dec}}$	Correct key	Key for encryption and decryption
P_{wrong}	False plain text	Text has syntactical meaning
K_{wrong}	False key	Key used by the attacker

5. Comparison with a Current System

5.1. Scenario Comparison with ASCII System. Difference between ASCII and the structural code scheme will be clarified by means of a comparison scenario. Before describing the scenario, we explain its terms in Table 2.

The scenario assumes a secure communication that includes a sender, receiver, and an attacker. There are two scenarios labeled (a) and (b). Both scenarios assume the use of a cryptosystem. A sender and a receiver will exchange the correct plain text with the correct key during communication, and an attacker will obtain false plain text with a false key. Figure 4 demonstrates two scenarios: an ASCII code scheme (a) and the structural code scheme (b).

Figure 4(a) shows a secure communication process that uses the ASCII code scheme. The sender encodes correct plain text “secret” to a coded message “73 65 63 72 65 74” by using the ASCII encoder. The sender then encrypts the coded message with the correct key K_{enc}. Therefore, the ciphertext “88 76 2A F8 63 FD” will be sent in the communication process. The receiver obtains the ciphertext through the communication. Decrypting with shared key K_{enc} and decoding with the ASCII table will show the correct plain text to the receiver.

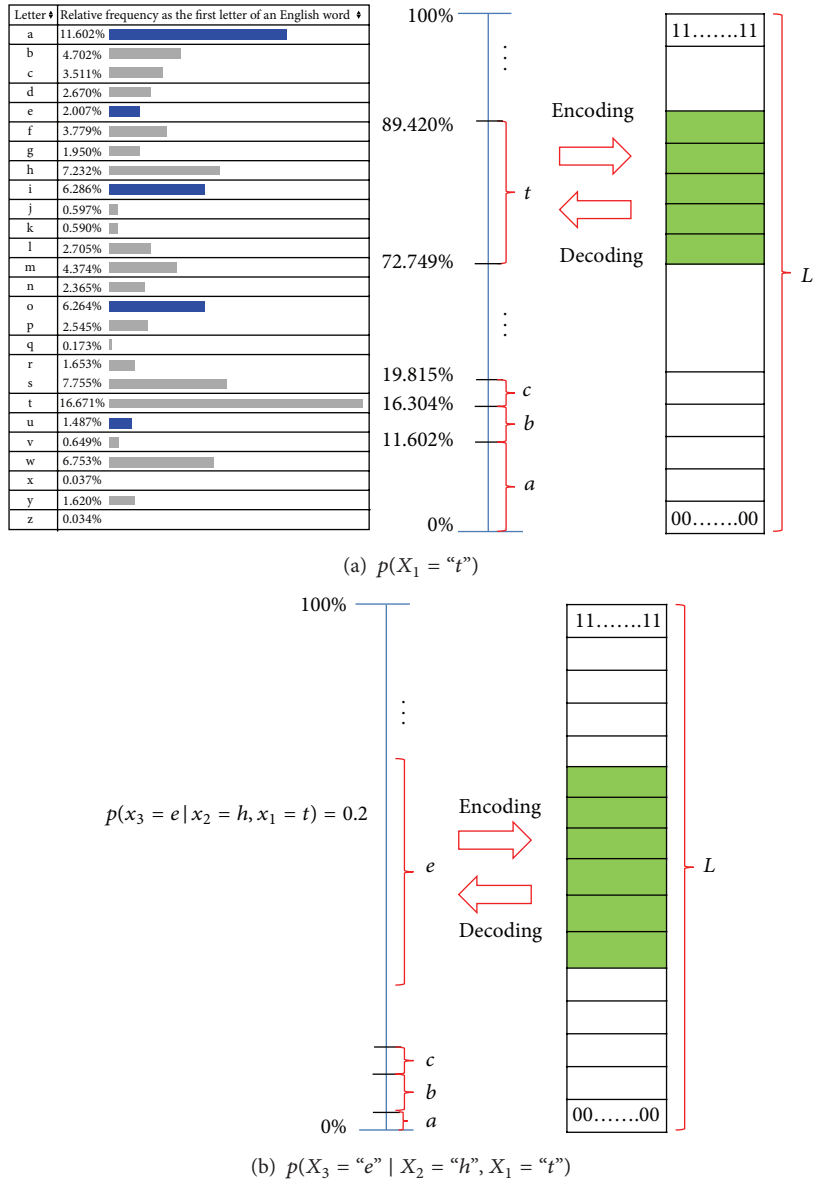


FIGURE 3: Cumulative function value of a language model and a corpus.

However, if there is an attacker in the middle of the communication, ASCII reveals a vulnerability. Attackers who can intercept the data in the communication will obtain ciphertext C. They decrypt the ciphertext with false key K_{wrong} and will obtain “29 71 25 6C 43 57” as a result. We can get “)q%lCW” as the result of ASCII decoding. This word is not syntactical; therefore, the attackers will realize that K_{wrong} is not correct for decryption. Thus, the ASCII code scheme is used as a measuring stick for success or failure.

Scenario (b) resolves this problem by using the structural code scheme. The communication between sender and receiver is the same; however, the attackers obtain different results from this scheme. This scheme returns the word “simple” as a result of false key K_{wrong} using the statistical code table. Although the attackers used false key K_{wrong}, they obtain false word P_{wrong}, which has syntactical meaning so they cannot distinguish correct plain text from false plain text and secrecy will be increased.

6. Experiments

6.1. Experiment Environment. We analyzed the probability of obtaining a false plain text in repeated trials to test the performance of the natural language format scheme. The main purpose of this experiment was to identify a final convergence value by increasing the number of repeated trials. Through this experiment, we can estimate the number of repeated trials that are necessary to obtain meaningful false plain texts.

First, we collected text data to set the corpus. In this study, we used the text of the play “Romeo and Juliet” (248 KB) [16]. We built a language model based on the text data

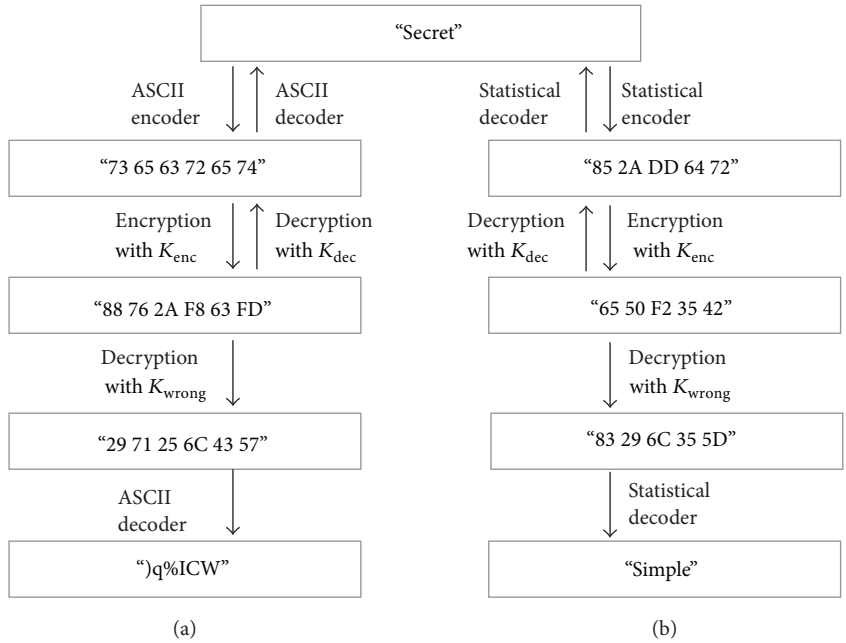


Figure 4: Processing steps in the two communication scenarios. (a) is an ASCII code scheme and (b) is the structural code scheme. The figure shows the differences between the ASCII coder and the proposed coder in a secure data transmission.

and converted the language model to a corpus for encoding and decoding. We used MATLAB 2013a to conduct the experiment. The code used in the experiment was composed of a training portion, computation portion, and an encoding, decoding portion.

In the experiment, we analyzed how many repeated trials were required to build syntactically meaningful false plain text. The number of repeated trials used in the experiment was 5, 10, 20, and 100. There were two standards for setting the repeated trial number. First, the repeated trial number had to be larger than a minimum number, which indicates the number of repeated trials required to obtain meaningful false plain text. Second, we set the maximum number of repeats to 100 because an experimenter had to check the grammatical results manually. With these standards, we selected random numbers between 5 and 100.

6.2. Analysis of Experiment. Table 3 shows the results of the experiment. All ten plain texts $P_{correct}$ were selected from words related to a keyword "Cipher." The left column shows ten plain text encryption techniques used in the test. We counted the number of syntactically meaningful false plain texts. The right column shows the results of the experiments. It shows the frequency rate of false but meaningful plain texts from the repeated trials. In the experiment, we only counted words that were composed of meaningful dictionary characters. For example, if the result "ah, thou shall" is composed of meaningless words, we conclude that it is not the false plain text. However, if the result "romeo and the" is composed of only dictionary words, we judged that it is the false plain text. The last line of Table 3 shows the convergence value of the probabilities. This value indicates the minimum number of repeated trials that are required to obtain meaningful false plain text. The result 0.38 shows that at least four repetitions were required. Therefore, users must perform at least four repeated trials to obtain false plain text P_{wrong}, which will be used for confronting the attack. These false plain texts are based on training data processed by a probabilistic language model. Therefore, if the user trained a movie script, the false plain text will look like a line from a movie script. The attackers will be unable to differentiate between false plain text and real plain text, thus enabling the user to maintain secrecy against brute-force attacks.

Table 3: This table shows the results of our test. Using the text set (Romeo and Juliet), we tested the statistical code scheme with different numbers of training repetitions. We found semantically or synthetically meaningful text and checked the related probabilities (repetitions = 5, 10, 20, 50, and 100).

Plain text	Number of proper false plain texts				
	5	10	20	50	100
Secret message	0.4	0.3	0.25	0.28	0.38
Autokey cipher	0.8	0.3	0.4	0.36	0.38
Encryption software	0.6	0.6	0.65	0.36	0.43
Pretty good privacy	0.2	0.4	0.45	0.44	0.36
Steganography	0.4	0.4	0.45	0.26	0.41
Telegraph code	0.6	0.4	0.4	0.42	0.29
Famous ciphertexts	0.4	0.7	0.4	0.44	0.42
Cryptography classification	0.4	0.3	0.3	0.48	0.38
Kish cipher	0.6	0.4	0.25	0.4	0.35
Coding	0.8	0.2	0.35	0.52	0.44
Average of probability	**0.52**	**0.4**	**0.38**	**0.396**	**0.384**

7. Discussion

We have described and tested the new countermeasure. However, this study focused on text data only. As a result, readers may conclude that they can only preserve secrecy when they communicate using text data. However, we found that this scheme can be applied to various formats. We will expand training targets from text data to other data formats in subsequent studies. With this goal, we will determine methods to secure various data formats in addition to text data.

We compared our proposed scheme with American Standard Code for Information Interchange (ASCII) scheme because it is the most well-known information interchange scheme in modern computers. As one knows, there are many other encoding schemes such as Unicode and Chinese Character Code for Information Interchange (CCCII). Note that our proposed structural code scheme is generic in all information interchanges because it uses a frequency-based corpus which can be adaptively and systematically constructed in each information interchange.

8. Conclusion

To date, increasing computational complexity has been the only countermeasure against brute-force attacks. However, simply increasing the computational complexity does not guarantee the security of cryptosystems in an era of ever-increasing threats. Vulnerabilities are caused by the weakness of PRNGs and by attackers' ability to access advanced hardware. Therefore, we explored improved methods to confront brute-force attack.

In this paper, we presented a statistical code system and analyzed it as a new countermeasure. It comprises two categories: the honey encryption and the structural code scheme. Our study focused on the structural code scheme; thus, we explained it in detail. We described the details of the procedure and tested the performance of this recommended method. Unlike the honey encryption, this scheme uses the syntactical features of natural language to exchange information and generates false plain text to confuse the attacker. In other words, if we use this countermeasure to encode and decode our secret data, we can protect our secrecy more effectively, even in natural language.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research has been mainly supported by H2101-14-1001. In addition, this research is partially supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (NRF-2013R1A1A1012797) and by the Special Research Funds for New Faculty, Korea University.

References

[1] *Cryptozine: A Brief History of Cryptography*, 2008.

[2] Wikipedia, Brute-force attack—Wikipedia, The Free Encyclopedia, 2014.

[3] H.-J. Jo and J. W. Yoon, "Poster: statistical coding scheme for the protection of cryptographic systems against brute-force attack," in *Proceedings of the 35th IEEE Symposium on Security and Privacy*, San Jose, Calif, USA, May 2014, http://www.ieee-security.org/TC/SP2014/program-posters.html.

[4] A. Juels and T. Ristenpart, "Honey encryption: security beyond the brute-force bound," in *Advances in Cryptology—EUROCRYPT 2014*, vol. 8441 of *Lecture Notes in Computer Science*, pp. 293–310, Springer, Berlin, Germany, 2014.

[5] H. K. Noh and J. W. Yoon, Efficient deniable cryptosystem through a deniable encoding method, 2013.

[6] E. Barker and A. Roginsky, "Transitions: recommendation for transitioning the use of cryptographic algorithms and key lengths," NIST Special Publication 800:131A, 2011.

[7] I. Goldberg and D. Wagner, "Randomness and the Netscape browser," *Dr. Dobb's Journal*, vol. 21, no. 1, pp. 66–71, 1996.

[8] Debian Security Advisory, 2008, http://www.debian.org/.

[9] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter, "Ron was wrong, Whit is right," *IACR Cryptology ePrint Archive*, vol. 2012, article 64, 2012.

[10] R. Chirgwin, *Android Bug Batters Bitcoin Wallets*, 2013.

[11] P. Mihailescu, "The fuzzy vault for fingerprints is vulnerable to brute force attack," *CoRR*, http://arxiv.org/abs/0708.2974.

[12] J. Gilmore, *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*, 1998.

[13] 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.

[14] T. McEnery, *Corpus Linguistics: An Introduction*, Edinburgh University Press, 2001.

[15] Wikipedia, Letter frequency—Wikipedia, The Free Encyclopedia, 2014.

[16] textdata, http://www.textfiles.com/.