This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

Authentication by Encrypted Negative Password

Wenjian Luo, *Senior Member, IEEE,* Yamin Hu, Hao Jiang, and Junteng Wang

***Abstract*—Secure password storage is a vital aspect in systems based on password authentication, which is still the most widely used authentication technique, despite its some security flaws. In this paper, we propose a password authentication framework that is designed for secure password storage and could be easily integrated into existing authentication systems. In our framework, first, the received plain password from a client is hashed through a cryptographic hash function (e.g., SHA-256). Then, the hashed password is converted into a negative password. Finally, the negative password is encrypted into an Encrypted Negative Password (abbreviated as ENP) using a symmetric-key algorithm (e.g., AES), and multi-iteration encryption could be employed to further improve security. The cryptographic hash function and symmetric encryption make it difficult to crack passwords from ENPs. Moreover, there are lots of corresponding ENPs for a given plain password, which makes precomputation attacks (e.g., lookup table attack and rainbow table attack) infeasible. The algorithm complexity analyses and comparisons show that the ENP could resist lookup table attack and provide stronger password protection under dictionary attack. It is worth mentioning that the ENP does not introduce extra elements (e.g., salt); besides this, the ENP could still resist precomputation attacks. Most importantly, the ENP is the first password protection scheme that combines the cryptographic hash function, the negative password and the symmetric-key algorithm, without the need for additional information except the plain password.**

***Index Terms*—Authentication, dictionary attack, lookup table attack, negative database, secure password storage.**

I. Introduction

OWING to the development of the Internet, a vast number of online services have emerged, in which password authentication is the most widely used authentication technique, for it is available at a low cost and easy to deploy [1], [2]. Hence, password security always attracts great interest from academia and industry [3]. Despite great research achievements on password security, passwords are still cracked since users' careless behaviors [4]. For instance, many users often select weak passwords [5], [6]; they tend to reuse same passwords in different systems [7]–[10]; they usually set their passwords using familiar vocabulary for its convenience to remember [11], [12]. In addition, system problems may cause password compromises. It is very difficult to obtain passwords from high security systems. On the one hand, stealing authentication data tables (containing usernames and passwords) in high security systems is difficult. On the other hand, when carrying out an online guessing attack, there is usually a limit to the number of login attempts [13]. However, passwords may be leaked from weak systems [14]. Vulnerabilities are constantly being discovered, and not all systems could be timely patched to resist attacks, which gives adversaries an opportunity to illegally access weak systems [15]. In fact, some old systems are more vulnerable due to their lack of maintenance. Finally, since passwords are often reused, adversaries may log into high security systems through cracked passwords from systems of low security.

After obtaining authentication data tables from weak systems, adversaries can carry out offline attacks [16]. Passwords in the authentication data table are usually in the form of hashed passwords [17]. However, because processor resources and storage resources are becoming more and more abundant, hashed passwords cannot resist precomputation attacks, such as rainbow table attack and lookup table attack [18].

Note that there is a trend of generalization of adversaries, because anyone could obtain access to information on vulnerabilities from vulnerability databases, such as the Open Source Vulnerability Database (OSVDB), National Vulnerability Database (NVD), and the Common Vulnerabilities and Exposures (CVE) [19], and then make use of these information to crack systems. Moreover, they could download and use attack tools without the need for very professional security knowledge. Some powerful attack tools, such as hashcat [20], RainbowCrack [21] and John the Ripper [22], provide a variety of functions, such as multiple hash algorithms, multiple attack models, multiple operating systems, and multiple platforms, which raises a higher demand for secure password storage.

In these situations, attacks are usually carried out as follows. First, adversaries precompute a lookup table, where the keys are the hash values of elements in a password list containing frequently-used passwords, and the records are the corresponding plain passwords in the password list. Next, they obtain an authentication data table from low security systems. Then, they search for the plain passwords in the lookup table by matching hashed passwords in the authentication data table and the keys in the lookup table. Finally, the adversaries log into higher security systems through cracked usernames and passwords, so that they could steal more sensitive information of users and obtain some other benefits. A considerable number of attacks are carried out in this way, so that adversaries could obtain passwords at a low cost, which is advantageous to their goals.

One of the main reasons for the success of the above lookup table attack is that the corresponding hashed password is determined for a given plain password. Therefore, the lookup table could be quickly constructed, and the size of the lookup table could be sufficiently large, which results in a high success rate of cracking hashed passwords.

Typical password protection schemes include hashed pass-

This study is supported in part by the National Natural Science Foundation of China (No. 61175045). (*Corresponding author: Wenjian Luo*)

Wenjian Luo, Yamin Hu, Hao Jiang, and Junteng Wang are with the School of Computer Science and Technology, the University of Science and Technology of China, Hefei 230027, Anhui, China. They are also with Anhui Province Key Laboratory of Software Engineering in Computing and Communication, the University of Science and Technology of China, Hefei 230027, Anhui, China.

Email: wjluo@ustc.edu.cn, huym@mail.ustc.edu.cn, jh9305@mail.ustc.edu.cn, wjt2013@mail.ustc.edu.cn.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

word, salted password and key stretching. Among these schemes, hashed password would be gradually eliminated for its vulnerability for precomputation attacks. Although salted password could resist precomputation attacks, it introduces an extra element (i.e., salt) and could not resist dictionary attack. In addition, salt tends to be implemented by mistake (such as salt reuse and short salt). Key stretching schemes, such as bcrypt [23], scrypt [24] and Argon2 [25] (the winner of Password Hashing Competition [26]), are used to defend against dictionary attack. Although key stretching schemes provide stronger password protection than salted password under dictionary attack, they impose an extra burden on programmers for configuring more parameters. In addition, they also use salt to resist precomputation attacks. Besides these schemes, some other password protection schemes were proposed. In [17], a scheme based on MD5 was proposed. It is a variant of salted password, where the salt is two random strings. Although it could resist lookup table attack and make dictionary attack difficult, it introduces many parameters, which makes it complicated and inconvenient to use. In [27], dynamic salt generation and placement are used to improve password security. Essentially, this scheme is also a variant of salted password, where the salt is a random string that is dependent on the original password. Consequently, it could resist lookup table attack, however it could not defend against dictionary attack and also introduces an extra element (i.e., salt). In [28], improved dynamic Key-Hashed Message Authentication Code function (abbreviated as d-HMAC) was proposed for password storage. It is also a variant of salted password, where the salt is the user's public key, and it introduces a secret key, which makes it inconvenient to use. In summary, although some new password protection schemes were proposed, they are similar to typical password protection schemes essentially. Therefore, in Section VI, without loss of generality, we only compare the typical password schemes with our scheme.

In this paper, a password protection scheme called Encrypted Negative Password (abbreviated as ENP) is proposed, which is based on the Negative Database (abbreviated as NDB) [29]–[32], cryptographic hash function and symmetric encryption, and a password authentication framework based on the ENP is presented. The NDB is a new security technique that is inspired by biological immune systems [29] and has a wide range of applications [33]–[36]. Symmetric encryption is usually deemed inappropriate for password protection. Because the secret key is usually shared by all encrypted passwords and stored together with the authentication data table, once the authentication data table is stolen, the shared key may be stolen at the same time [37]. Thus, these passwords are immediately compromised. However, in the ENP, the secret key is the hash value of the password of each user, so it is almost always different and does not need to be specially generated and stored. Consequently, the ENP enables symmetric encryption to be used for password protection. As an implementation of key stretching [38], multi-iteration encryption is introduced to further improve the strength of ENPs. Compared with the salted password scheme and key stretching, the ENP guarantees the diversity of passwords by itself without introducing extra elements (e.g., salt).

To summarize, the main contributions of this paper are as follows:

(1) We propose a password protection scheme called ENP, and we propose two implementations of the ENP: ENPI and ENPII, including their generation algorithms and verification algorithms. Furthermore, a password authentication framework based on the ENP is presented.
(2) We analyze and compare the attack complexity of hashed password, salted password, key stretching and the ENP. The results show that the ENP could resist lookup table attack without the need for extra elements and provide stronger password protection under dictionary attack.

The remainder of this paper is organized as follows. In Section II, we introduce related work, including the hashed password, salted password, key stretching, and the NDB. In Section III, we describe the proposed framework. In Section IV, we give two implementations of the ENP. In Section V, we analyze the attack complexity of the ENP. In Section VI, we compare hashed password, salted password and key stretching with the ENP and illustrate the advantages of the ENP. Our conclusions and future work follow in Section VII.

II. Related Work

A. Typical Password Protection Schemes

1) Hashed Password: The simplest scheme to store passwords is to directly store plain passwords. However, this scheme presents a problem that once adversaries obtain the authentication data table, all passwords are immediately compromised. To safely store passwords, a common scheme is to hash passwords using a cryptographic hash function [17], because it is infeasible to directly recover plain passwords from hashed passwords. The cryptographic hash function quickly maps data of arbitrary size to a fixed-size sequence of bits. In the authentication system using the hashed password scheme, only hashed passwords are stored. However, hashed passwords cannot resist lookup table attack [17]. Furthermore, rainbow table attack is more practical for its space-time tradeoff [18]. Processor resources and storage resources are becoming richer, which makes the precomputed tables used in the above two attacks sufficiently large, so that adversaries could obtain a higher success rate of cracking hashed passwords.

2) Salted Password: To resist precomputation attacks, the most common scheme is salted password [17]. In this scheme, the concatenation of a plain password and a random data (called salt) is hashed through a cryptographic hash function. The salt is usually generated at random, which ensures that the hash values of the same plain passwords are almost always different. The greater the size of the salt is, the higher the password security is. However, under dictionary attack, salted passwords are still weak. Note that compared with salted password, the ENP proposed in this paper guarantees the diversity of passwords without the need for extra elements (e.g., salt).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

3) Key Stretching: To resist dictionary attack, key stretching [38], which converts weak passwords to enhanced passwords, was proposed. Key stretching could increase the time cost required to every password attempt, so that the power of defending against dictionary attack is increased. In the ENP proposed in this paper, like key stretching, multi-iteration encryption is used to further improve password security under dictionary attack, and compared with key stretching, the ENP does not introduce extra elements (e.g., salt).

B. Negative Database

In the NDB, the compression of the complement of a positive database (denoted as DB) is stored. As described in [30], $U = \{0,1\}^n$ denotes the universal set of n-bit sequences; $x \in U$ denotes an n-bit sequence; $DB = \{x_1, x_2, \cdots, x_m\}$ denotes a positive database that contains m entries; then NDB stores the compression (implemented using the wildcard '*') of $(U - DB)$.

Some concepts of NDB are given below. Every entry in an NDB contains three symbols: '0', '1', and '*'. The symbol '0' only match the bit 0, and the symbol '1' only match the bit 1; The symbol '*' can match either the bit 0 or 1. Every entry in an NDB consists of two kinds of positions: specified positions and unspecified positions. Positions where the symbols are '0' or '1' are called specified positions, while positions where the symbols are '*' are called unspecified positions. Accordingly, both '0' and '1' are specified symbols, and the '*' is the unspecified symbol. A sequence of bits is covered by one entry in an NDB; that is to say, the bits of the sequence are matched by the symbols of the entry at the specified positions. If a sequence of bits is covered by one entry in an NDB, we say that the sequence is covered by the NDB. If an NDB covers every entry in the (U-DB), we say that the NDB is complete; otherwise, it is incomplete. The NDB converted from a DB with only one entry is called the single NDB; otherwise, it is called the multiple NDB.

There are two types of NDB generation algorithms, one for single NDBs and one for multiple NDBs. In the first type, clause distribution control algorithm [39], 1-hidden algorithm [40], 2-hidden algorithm [40], q-hidden algorithm [41], hybrid algorithm [42], p-hidden algorithm [43], and K-hidden algorithm [44] were proposed successively. In the second type, the prefix algorithm [30], Randomize_NDB (abbreviated as RNDB) [30], multiple-solution algorithm [36] were proposed successively; certainly, these algorithms could also be used to generate single NDBs.

Among NDBs generated by these generation algorithms, under certain conditions, some are hard-to-reverse, and others are easy-to-reverse. In our work, we employ two easy-to-reverse complete single NDB generation algorithms (i.e., the prefix algorithm with permutation, see Algorithm A.1 in the Appendix, and the variant of the prefix algorithm, see Algorithm A.2 in the Appendix) to generate negative passwords.

An example is given to illustrate the complete single NDB in Table I. As represented in Table I, the database (i.e., DB) only contains one entry, and every entry in (U-DB) is covered by the NDB (i.e., the NDB is complete). Here, the NDB is generated by a variant of the prefix algorithm, which is the first part of the hybrid algorithm in [42]. Note that, in order to clearly represent various sequences, we use "the sequence of symbols" to clearly represent the concatenation of symbols (i.e., '0', '1', or '*'), "the sequence of bits" to clearly represent the concatenation of bits (i.e., 0 or 1), which may be converted to hex form for short, and "the sequence of characters" to clearly represent the concatenation of characters (for passwords, these characters usually include uppercase letters, lowercase letters, numerals and special characters). In addition, the sequence of symbols or characters is enclosed within a pair of double quotation marks, while the sequence of bits is not enclosed by any punctuation; similarly, the single symbol or character is enclosed within a pair of single quotation marks, while the single bit is not enclosed by any punctuation.

TABLE I
AN EXAMPLE OF THE COMPLETE SINGLE NDB.

DB	U-DB	NDB
0000	0001	"0*01"
	0010	"001*"
	0011	"01*0"
	0100	"01*1"
	0101	"100*"
	0110	"110*"
	0111	"1*10"
	1000	"1*11"
	1001	
	1010	
	1011	
	1100	
	1101	
	1110	
	1111	

As a form of negative representations of information, NDB [29]–[32] has been used in various fields, such as authentication [33], [34], [45], biometric recognition [35], [46], and information hiding [36]. The fields related to our work are introduced as follows. In [33] and [34], NDB was used to protect the original authentication data table as an additional protection layer. In [35] and [46], NDB was used to protect biometric data while supporting effective recognition. In [45], a one-time password authentication scheme based on NDB was proposed. In all of the above authentication methods, passwords are protected by the difficulty of reversing NDB. However, such reversal difficulty needs to be further improved. On the contrary, in the ENP proposed in this paper, the original plain passwords could be converted from ENPs; passwords are protected by cryptographic hash function and symmetric encryption; authentication is based on the comparison between the hash value of the plain password from a client and the hashed password corresponding to some ENP on the server.

III. THE PROPOSED FRAMEWORK

The proposed framework includes two phases: the registration phase and authentication phase. When adopting our

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

TABLE II
SOME MATCHES OF CRYPTOGRAPHIC HASH FUNCTIONS AND SYMMETRIC-KEY ALGORITHMS.

Cryptographic hash functions	Symmetric-key algorithms	#bits
MD5*	AES/IDEA/CAST-256/RC6	128
SHA-1*	CAST-256	160
SHA-224/SHA3-224	CAST-256	224
SHA-256/SHA3-256	AES/CAST-256/RC6	256
SHA-384/SHA3-384	RC5	384
SHA-512/SHA3-512	RC5	512

*Although MD5 and SHA-1 could not resist collision attack [47], they could still be used in our framework, since they still resist preimage attack [48]. Despite this, we recommend the use of SHA-2 (including SHA-224, SHA-256, SHA-384 and SHA-512) and SHA-3 (including SHA3-224, SHA3-256, SHA3-384 and SHA3-512).

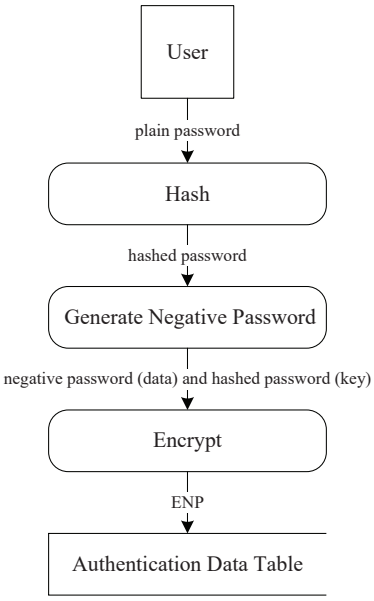

Fig. 1. The data flow diagram of the generation procedure of the ENP.

framework to protect passwords in an authentication data table, the system designer must first select a cryptographic hash function and a symmetric-key algorithm, where the condition that must be satisfied is that the size of the hash value of the selected cryptographic hash function is equal to the key size of the selected symmetric-key algorithm. For convenience, some matches of cryptographic hash functions and symmetric-key algorithms are given in Table II. In addition, cryptographic hash functions and symmetric-key algorithms that are not listed here could also be used in the ENP, which adequately indicates the flexibility of our framework. The proposed framework is based on the ENP; hence, for better understanding, the data flow diagram of the generation procedure of the ENP is shown in Fig. 1, and the data flow diagram of the verification procedure of the ENP is shown in Fig. 2.

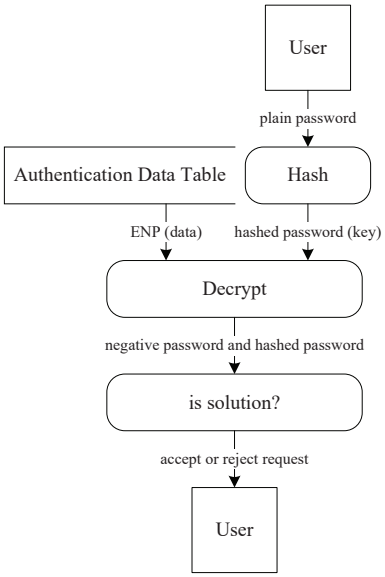

Fig. 2. The data flow diagram of the verification procedure of the ENP.

A. Registration Phase

The registration phase is divided into six steps.

(1) On the client side, a user enters his/her username and password. Then, the username and plain password are transmitted to the server through a secure channel;
(2) If the received username exists in the authentication data table, "The username already exists!" is returned, which means that the server has rejected the registration request, and the registration phase is terminated; otherwise, go to Step (3);
(3) The received password is hashed using the selected cryptographic hash function;
(4) The hashed password is converted into a negative password using an NDB generation algorithm (i.e., Algorithm A.1 or Algorithm A.2 in the Appendix);
(5) The negative password is encrypted to an ENP using the selected symmetric-key algorithm, where the key is the hash value of the plain password. Here, as an additional option, multi-iteration encryption could be used to further enhance passwords;
(6) The username and the resulting ENP are stored in the authentication data table and "Registration success" is returned, which means that the server has accepted the registration request.

B. Authentication Phase

The authentication phase is divided into five steps.

(1) On the client side, a user enters his/her username and password. Then, the username and plain password are transmitted to the server through a secure channel;
(2) If the received username does not exist in the authentication data table, then "Incorrect username or password!" is returned, which means that the server has rejected the authentication request, and the authentication phase is terminated; otherwise, go to Step (3);

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

(3) Search the authentication data table for the ENP corresponding to the received username;
(4) The ENP is decrypted (one or more times according to the encryption setting in the registration phase) using the selected symmetric-key algorithm, where the key is the hash value of the plain password; thus, the negative password is obtained;
(5) If the hash value of the received password is not the solution of the negative password (verified by Algorithm 1 or Algorithm 2), then "Incorrect username or password!" is returned, which means that the server has rejected the authentication request, and the authentication phase is terminated; otherwise, "Authentication success" is returned, which means that the server has accepted the authentication request.

C. Encrypted Negative Password

ENPs could be obtained through the following steps (see Fig. 1). The received plain password (i.e., a sequence of characters) from a client is first hashed using a cryptographic hash function. Next, the hashed password is converted into a negative password using an NDB generation algorithm (i.e., Algorithm A.1 or Algorithm A.2 in the Appendix). Then, the negative password is encrypted using a symmetric-key algorithm. Thus, the ENP is obtained. The solution of the negative password is the hash value of the received plain password.

In the above processing, each component (i.e., the cryptographic hash function, the symmetric-key algorithm, and the NDB generation algorithm) is indispensable. The cryptographic hash function converts plain passwords to hashed passwords; the fixed length property of resulting hashed passwords offers convenience for the subsequent encryption, since the length requirement for the secret key in the symmetric-key algorithm; and other properties (such as avalanche effect and collision resistance) are also crucial factors of employing the cryptographic hash function. The reasons for employing the symmetric-key algorithm are given below. The conversion from a hashed password to a negative password is not irreversible; therefore, if no encryption, when an adversary obtains a negative password, the adversary immediately obtains the corresponding hashed password, which makes the strength of the ENP equivalent to that of the hashed password essentially. However, when adopting encryption, the adversary does not know the key (i.e., the hashed password converted from the original plain password), so the adversary could not decrypt the ENP to get the negative password.

The NDB generation algorithm is selected for converting a hashed password to the corresponding negative password; there are several reasons listed below.

(1) The NDB generation algorithm is a one-to-many mapping; simultaneously, it is reversible; additionally, while keeping the one-to-many relationship, it does not introduce extra elements (such as salt). Specifically, given a hashed password, there are lots of corresponding negative passwords; a negative password has one and only one corresponding hashed password; this conversion is done by the NDB generation algorithm itself, and not dependent on extra elements.
(2) The value space of negative passwords for a hashed password is big enough for resisting precomputation attacks (the analyses are presented in Section V).
(3) The NDB generation algorithms are simple and efficient. As shown in the pseudo-code of the NDB generation algorithms (i.e., Algorithm A.1 and Algorithm A.2, in the Appendix), these algorithms are easy to implement and analyze; thus, it helps achieve confidence on the use of the ENP; based on random permutation and inverse permutation, randomness is introduced to implement reversible one-to-many mapping, which is straightforward and efficient.

Before encryption, every entry in a negative password is encoded as the concatenation of two-bit pairs. The two-bit pairs have four forms: 00, 01, 10, and 11, where 00 denotes the symbol '0', 01 denotes the symbol '1', and both 10 and 11 denote the symbol '*'. For example, the sequence of bits 00101101 denotes the entry "0**1". Note that the symbol '*' is denoted by either 10 or 11 randomly instead of just 10 or 11, which ensures that any sequence of bits can be a legal negative password. As a result, under dictionary attack, adversaries cannot exclude any password in the password list based on the form of the negative password (i.e., not including 10 or 11).

Moreover, multi-iteration encryption could be introduced to further improve ENPs strength, which is an implementation of the key stretching technique. The greater the number of encryptions is, the more secure the ENPs are; however, the authentication speed decreases. The system designer must balance the speed of authentication against password security, and then selects a proper number of encryptions.

IV. Two Implementations of Encrypted Negative Password

In this section, we propose two implementations of the ENP, including their generation algorithms and verification algorithms. The first implementation is based on the prefix algorithm [30], and we call it ENPI; the second one is based on a variant of the prefix algorithm [42], and we call it ENPII.

A. ENPI

In ENPI, we employ the prefix algorithm [30] with permutation (i.e., Algorithm A.1 in the Appendix) to generate negative passwords (i.e., NDBs). Negative passwords generated by the prefix algorithm are deterministic and complete [30]. The conversation from a hashed password (i.e., a sequence of bits) to an negative password using the prefix algorithm is one-to-one; therefore, the random permutation operation is employed to make the conversation from a hashed password to a negative password one-to-many by randomly reordering the bits of the sequence. The permutation is usually written as a tuple. For instance, "abc" is permuted to "cab" by the permutation (3, 1, 2), since 'c' in "cab" is the 3rd element in "abc", 'a' is the 1st element, and 'b' is the 2nd element. A negative password in ENPI contains exactly m entries, where m is the size of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

the hashed password; thus, the permutation is represented by a tuple with m elements.

An example is shown in Table III to illustrate the diversity of ENPI, where two examples (represented in hex form) of ENPI converted from the plain password "password" are listed.

1) Generation Algorithm: In ENPI, Algorithm A.1 is employed to convert a hashed password to a negative password, where the input s is the hash value of the received plain password, and the output ndb is the resulting negative password. The generation procedure of ENPI is shown by an example in Table IV, where the plain password "password" is converted into an ENP through the processes of SHA-256, permutation, the prefix algorithm, inverse permutation, encoding, and AES encryption (the concatenation of sequences of the encoded negative password is encrypted through CBC mode); here, the permutation is (201, 237, 97, 67, 172, 140, 9, 190, 36, 187, 135, 178, 216, 222, 167, 230, 169, 49, 66, 214, 175, 98, 113, 34, 93, 11, 43, 111, 23, 40, 250, 63, 130, 215, 94, 90, 176, 37, 38, 83, 27, 7, 141, 99, 10, 177, 107, 131, 217, 101, 120, 254, 32, 242, 115, 174, 233, 58, 157, 15, 194, 46, 184, 253, 200, 168, 236, 148, 104, 2, 17, 124, 20, 202, 80, 6, 70, 192, 208, 118, 180, 226, 211, 203, 225, 209, 29, 65, 136, 228, 51, 8, 235, 96, 227, 60, 181, 26, 72, 129, 126, 77, 188, 127, 198, 210, 159, 1, 74, 3, 185, 103, 170, 114, 47, 244, 64, 78, 18, 25, 162, 155, 109, 76, 116, 164, 12, 108, 232, 246, 87, 21, 154, 161, 240, 42, 59, 134, 35, 142, 95, 143, 173, 71, 179, 88, 229, 147, 41, 238, 61, 158, 146, 138, 196, 212, 204, 197, 241, 182, 85, 30, 55, 186, 234, 91, 52, 13, 73, 28, 199, 218, 206, 163, 205, 56, 213, 39, 221, 82, 4, 224, 156, 193, 256, 248, 62, 219, 165, 48, 151, 54, 149, 231, 239, 132, 195, 102, 125, 75, 92, 57, 128, 105, 44, 69, 220, 19, 14, 112, 86, 255, 33, 166, 79, 53, 31, 24, 84, 100, 223, 117, 122, 5, 160, 153, 45, 191, 183, 121, 245, 207, 145, 133, 189, 123, 247, 50, 251, 144, 139, 68, 110, 81, 252, 137, 249, 171, 152, 106, 243, 119, 22, 150, 16, 89).

2) Verification Algorithm: After a user submits his/her username and plain password, the server first finds the corresponding ENP in the authentication data table according to the username. Next, the plain password is hashed and the ENP is decrypted. Then, the hashed password is verified to determine whether it is the solution of the negative password decrypted from the ENP.

The responsibility of the ENP verification algorithm is to verify whether a hashed password is the solution of a negative password. Before verifying the correctness, the ENP verification algorithm needs to verify whether the inputted negative password is legal. From Algorithm A.1, it can be inferred that a legal negative password in ENPI satisfies three conditions: (1) the ith entry has i specified positions, (2) for each entry, there is only one specified symbol that does not match the corresponding bit of the original hashed password, and these specified symbols are at different positions, and (3) the specified positions of any entry (except the first entry) cover that of its preceding one.

The pseudo-code of the ENPI verification algorithm is shown in Algorithm 1, where np_i is the ith entry in np; NUMBEROFSP(x) at Lines 3 and 8 counts the number of specified positions in x; INDEXOFSP(x) at Line 11 finds the index of the only specified position in x; TOBIT(sym) at Line 12 converts the symbol sym to the corresponding bit (i.e., '0' to 0 and '1' to 1; before conversion, the symbol sym has ensured to be '0' or '1'); conversely, TOSYMBOL(b) at Line 14 converts the bit b to the corresponding symbol (i.e., 0 to '0' and 1 to '1'). Satisfying the conditions at Lines 3, 8, and 14 illustrates that np is not an output of Algorithm A.1, i.e., np is a illegal negative password in ENPI, and the algorithm is terminated in advance. If the negative password is legal, when judging the legitimacy, the negative password is solved, and then we determine whether the solution and the hashed password are equal.

Algorithm 1 ENPI Verification Algorithm

Input: a hashed password $hashP$;
a negative password np
Output: true or false

1: $m \leftarrow$ LENGTH($hashP$)
2: **for** $i \leftarrow 1$ **to** m **with stepsize of** 1 **do**
3: **if** NUMBEROFSP(np_i) $\neq i$ **then**
4: **return** false
5: **end if**
6: **end for**
7: **for** $i \leftarrow 1$ **to** m **with stepsize of** 1 **do**
8: **if** NUMBEROFSP(np_i) $\neq 1$ **then**
9: **return** false
10: **end if**
11: $k \leftarrow$ INDEXOFSP(np_i)
12: $x[k] \leftarrow \neg$TOBIT($np_i[k]$)
13: **for** $j \leftarrow i+1$ **to** m **with stepsize of** 1 **do**
14: **if** $np_j[k] \neq$ TOSYMBOL($x[k]$) **then**
15: **return** false
16: **end if**
17: $np_j[k] \leftarrow$'*'
18: **end for**
19: **end for**
20: **if** $x = hashP$ **then**
21: **return** true
22: **else**
23: **return** false
24: **end if**

The verification procedure of ENPI is shown by an example in Table V, where an ENP (converted from the plain password "password") is converted into a sequence of bits (i.e., x) through the processes of AES decryption (the key is the hash value of "password"), decoding, and solving the negative password; the numbers 1, 2, $\cdots$, 256 correspond to i at Line 7 in Algorithm 1, and the np and x following the i are the current states of the negative password and the solution of the negative password after the ith pass in the loop at Lines 7–19 in Algorithm 1; and the final x is equal to the hash value of the plain password "password".

B. ENPII

By comparison with ENPI, more randomization is introduced into ENPII, which is based on a variant of the prefix

TABLE III
THE DIVERSITY OF ENPI.

Example 1		Example 2	
1	286B51A725585A8673C12C0546527C12DBD56FDB4AC40D5D17D205577CB9D7EE A37A3D4529B5F1C21E5342C70023DB86AC086F3DDE42577500DBEE0E4AF8DE76	1	01178ABF97F3229B21BEB04D30816E6C060AF05E1F6BD563BC88B603050AA4B0 C0B30E044A904A3C3A56FEE4FA4342FA22A5FABE17DDC2208B46180FCAD48D60
2	E79E9A9B2D57F48A76F641B5769840719B54F659234B4B05A4755CA9F143B310 05F44FF189AAFEC194768236C421D314AD53064FE9A2B8F72711DEC58571F9A2	2	5F0E5902FAB912AF6CFC2328AFE96B27E04BC26887A58B774F64BFDF4B569866 40C35BC9FD4651F9FCDBA5BCF8F57D1E5EB3CE336CC890E9B97C63AE14F1CE2F
3	1A8E9764D3D953D7BBF630132507C4FF48F0DB1D082C180575AFDC69268A02E1 3DFC1CFAEF3CC637B4C0D1979EA848443C15637A9A618AD536EA36A4ED105CDB	3	0AC96B68AB8423B9D9C6115CFC12A81546C8F647A6D9C289946914B2221BCE5A D726CC4E076AFC98CCC20293EDC38E8E25B6F7104A1BB556C52B10911CEB7110
. . .	. . . . . .	. . .	. . . . . .
254	9785D274D99C13FA98511956835F7D33DEA711B6BB49DA3FD6C1AFAD2D517286 90A89E41FB18CC31740E11D5F5246AA8FC7F6F51919F421E6342051F2745A31A	254	29731731DAF4FDC07DC104C3A386D1D81429D910F889EB197A9F6C90BFFFFD22 9D79BE43BD02ABF616065C03D8C92015F866A98BD868662746D6815F389C48BD
255	9C4F7281500214BCF0A908EF3085BD85785C9C21DA2AEC31A42B527DF798F965 85BAC8816281A3F9C5511BC683E6AF87B4639B64D01135F38BB50AB8BAC23E4A	255	7451B1E4FD091B3F736CDB2697BCF6088E222A3094AD0DA1A85AEE952268B81D B1FF40D5B5F1D65F9C96A232392336EF857F2AF122FCF7A189B7B3A69EB03C78
256	31A5A3A25AB81B74F7C61E7CB0A71CE04CCF864350878843EF13A60B003674C4 59364AB5FE2AC5302DABE22FF74AE9C7DF2DCB279A9B5C68CDE8F500A630EC2C	256	4255789DA4736ACF9BAD51DB06A60DAACB6810BB4C6F526F5369253ED74D60A5 B04E7CE836E29ED1A2A596DE91FF52484122E878FB5C4443F24E9B3E6CF47A0F

TABLE IV
THE GENERATION PROCEDURE OF ENPI.

Plain password		“password”
Hashed password		0101111010001000010010001001100011011010001010000000001000111000101010001110100001110010101101111100011011100011000101001001001 11 0111001101100000000111101000011010110101010101011101111011101011000101010000100011110111101110010000111010001010101000010110110 00
Hashed password after π		00100011101010111011100110110010111111010100010101100110011000110011110000010010101001101100010110101011010010100000101110010000 10010010100010101111010101001101000101011111010011011000000110011001110000110010001010000000110110110000000110001111101111000100
Negative password before π'	1	“1*** **”
	2	“01** **”
	3	“000*** **”
	. . .	. . . . . .
	254	“00100011101010111011100110110010111111010100010101100110011000110011110000010010101001101100010110101011010010100000101110010000 100100101000101011110101010011010001010111110100110110000001100110011100001100100010100000001101101100000001100011111011110000**”
	255	“00100011101010111011100110110010111111010100010101100110011000110011110000010010101001101100010110101011010010100000101110010000 100100101000101011110101010011010001010111110100110110000001100110011100001100100010100000001101101100000001100011111011110001 1*”
	256	“00100011101010111011100110110010111111010100010101100110011000110011110000010010101001101100010110101011010010100000101110010000 10010010100010101111010101001101000101011111010011011000000110011001110000110010001010000000110110110000000110001111101111000101”
Negative password after π'	1	“** **1***”
	2	“** **0***********************************1*******************”
	3	“**0******************************* **0***********************************0*******************”
	. . .	. . . . . .
	254	“010111101000100*01001000100110001101101000101000000000100011100010101000111010000111001 01*1101111100011011100011000101001001001 11 0111001101100000000111001000011010110101010101011101111011101011000101010000100011110111101110010000111010001010101000010110110 00”
	255	“0101111010001001010010001001100011011010001010000000001000111000101010001110100001110 0101*1101111100011011100011000101001001001 11 0111001101100000000111101000011010110101010101011101111011101011000101010000100011110111101110010000111010001010101000010110110 00”
	256	“0101111010001000010010001001100011011010001010000000001000111000101010001110100001110010111101111100011011100011000101001001001 11 0111001101100000000111101000011010110101010101011101111011101011000101010000100011110111101110010000111010001010101000010110110 00”
Negative password after encoding	1	EEFBFBBBEABAAFBABAEEEFBEFBBBFEBAEFAEEFFABEEEAABFAFBFEEBAFAFEFEBEBEBAFBBFFEBBFFBBAEABFBFEABEAAEABBEFE6FEABAFEAABEAFAAEBBBBFBAEAFE
	2	BAABBFEFAEBEBEBBEBBFFEFBFBAFFEAFEFAFBFBBEAABBEEEFFBFAFFBAAEFAEABBEABEABFEEEEEBEFEBBEBAFAAAAABAABBBBE3AAAEEEABAAFAEBBBF6AEBBFAAFB
	3	EBAFAAFEABEFABFFAFBABABAAEAEFEBEFAFABBEEFABAEFEA2FABBEEBBABAABBFAFABBEBFEEEFFEEFAAFBFFABEBEBAABBBFBB3FAFAAEBEBEFABFABB3BBFEABAAE
	. . .	. . . . . .
	254	11544043104041405144044000101501110151005411D4554051501404410415150514000541005114444445455151140444010154551504015101111 0045140
	255	11544041104041405144044000101501110151005411945540515014044104151505140005510051144444454555151140444010154551504015101111 0045140
	256	11544040104041405144044000101501110151005411545540515014044104151505140005510051144444454555151140444010154551504015101111 0045140
ENP	1	286B51A725585A8673C12C0546527C12DBD56FDB4AC40D5D17D205577CB9D7EEA37A3D4529B5F1C21E5342C70023DB86AC086F3DDE42577500DBEE0E4AF8DE76
	2	E79E9A9B2D57F48A76F641B5769840719B54F659234B4B05A4755CA9F143B31005F44FF189AAFEC194768236C421D314AD53064FE9A2B8F72711DEC58571F9A2
	3	1A8E9764D3D953D7BBF630132507C4FF48F0DB1D082C180575AFDC69268A02E13DFC1CFAEF3CC637B4C0D1979EA848443C15637A9A618AD536EA36A4ED105CDB
	. . .	. . . . . .
	254	9785D274D99C13FA98511956835F7D33DEA711B6BB49DA3FD6C1AFAD2D51728690A89E41FB18CC31740E11D5F5246AA8FC7F6F51919F421E6342051F2745A31A
	255	9C4F7281500214BCF0A908EF3085BD85785C9C21DA2AEC31A42B527DF798F96585BAC8816281A3F9C5511BC683E6AF87B4639B64D01135F38BB50AB8BAC23E4A
	256	31A5A3A25AB81B74F7C61E7CB0A71CE04CCF864350878843EF13A60B003674C459364AB5FE2AC5302DABE22FF74AE9C7DF2DCB279A9B5C68CDE8F500A630EC2C

algorithm [30] (i.e., Algorithm A.2 in the Appendix), which has been proven to be complete [42] and is easy to solve. A negative password in ENPII contains exactly $m+4$ entries, where m is the size of the hashed password, and every entry has exactly three specified positions. Besides the permutation, more randomization is introduced during the conversion from an entry to an entry that only contains three specified positions.

1) Generation Algorithm: In ENPII, Algorithm A.2 is employed to convert a hashed password to a negative password, where the input s is the hash value of the received plain password, and the output ndb is the resulting negative password.

2) Verification Algorithm: In this subsection, we propose a method for verifying whether a hashed password is the solution of a negative password in ENPII.

Firstly, the ENPII verification algorithm verifies whether the inputted negative password is legal. From Algorithm A.2, it can be inferred that a legal negative password in ENPII

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

TABLE V
THE VERIFICATION PROCEDURE OF ENPI.

ENP		1	286B51A725585A8673C12C0546527C12DBD56FDB4AC40D5D17D205577CB9D7EEA37A3D4529B5F1C21E5342C70023DB86AC086F3DDE42577500DBEE0E4AF8DE76
		2	E79E9A9B2D57F48A76F641B5769840719B54F659234B4B05A4755CA9F143B31005F44FF189AAFEC194768236C421D314AD53064FE9A2B8F72711DEC58571F9A2
		3	1A8E9764D3D953D7BBF630132507C4FF48F0DB1D082C180575AFDC69268A02E13DFC1CFAEF3CC637B4C0D1979EA848443C15637A9A618AD536EA36A4ED105CDB
		. . .	. . . . . .
		254	9785D274D99C13FA98511956835F7D33DEA711B6BB49DA3FD6C1AFAD2D51728690A89E41FB18CC31740E11D5F5246AA8FC7F6F51919F421E6342051F2745A31A
		255	9C4F7281500214BCF0A908EF3085BD85785C9C21DA2AEC31A42B527DF798F96585BAC8816281A3F9C5511BC683E6AF87B4639B64D01135F38BB50AB8BAC23E4A
		256	31A5A3A25AB81B74F7C61E7CB0A71CE04CCF864350878843EF13A60B003674C459364AB5FE2AC5302DABE22FF74AE9C7DF2DCB279A9B5C68CDE8F500A630EC2C
Negative password after decryption		1	EEFBFBBBEABAAFBABAEEEFBEFBBBFEBAEFAEEFFABEEEAABFAFBFEEBAFAFEFEBEBEBAFBBFFEBBFFBBAEABFBFEABEAAEABBEFE6FEABAFEAABEAFAAEBBBBFBAEAFE
		2	BAABBFEFAEBEBEBBEBBFFEFBFBAFFEAFEFAFBFBBEAABBEEEFFBFAFFBAAEFAEABBEABEABFEEEEEBEFEBBEBAFAAAAABAABBBBE3AAAEEEABAAFAEBBBF6AEBBFAAFB
		3	EBAFAAFEABEFABFFAFBABABAAEAEFEBEFAFABBEEFABAEFEA2FABBEEBBABAABBFAFABBEBFEEEFFEEFAAFBFFABEBEBAABBBFBB3FAFAAEBEBEFABFABB3BBFEABAAE
		. . .	. . . . . .
		254	1154404310404140514404400010150111015100541 1D45540515014044104151505140005410051144444454551511404440101545515040151011110045140
		255	115440411040414051440440001015011101510054119455405150140441041515051400055100511444444545515114044401015455150401510111100451 40
		256	11544040104041405144044000101501110151005411545540515014044104151505140005510051144444454551511404440101545515040151011110045140
Negative password after decoding		1	“** **1***”
		2	“** **0***********************************1*******************”
		3	“**0******************************* **0***********************************0*******************”
		. . .	. . . . . .
		254	“010111101000100*010010001001100011011010001010000000010001110001010100011101000011100101*110111110001101110001100010100100100111 01110011011000000011100100001101011010101010101110111101110101100010101000100011110111101110010000111010001010101000010110 11000”
		255	“0101111010001001010010001001100011011010001010000000010001110001010100011101000011100101*110111110001101110001100010100100100111 01110011011000000011110100001101011010101010101110111101110101100010101000100011110111101110010000111010001010101000010110 11000”
		256	“01011110100010000100100010011000110110100010100000000100011100010101000111010000111001011110111110001101110001100010100100100111 01110011011000000011110100001101011010101010101110111101110101100010101000100011110111101110010000111010001010101000010110 11000”
***i* = 1**	***np***	1	“** **1***”
		2	“** **1*******************”
		. . .	. . . . . .
		255	“0101111010001001010010001001100011011010001010000000010001110001010100011101000011100101*110111110001101110001100010100100100111 011100110110000000111101000011010110101010101011101111011101011000101010*00100011110111101110010000111010001010101000010110 11000”
		256	“01011110100010000100100010011000110110100010100000000100011100010101000111010000111001011110111110001101110001100010100100100111 011100110110000000111101000011010110101010101011101111011101011000101010*00100011110111101110010000111010001010101000010110 11000”
	x		____ ____0____
***i* = 2**	***np***	1	“** **1***”
		2	“** **1*******************”
		. . .	. . . . . .
		255	“0101111010001001010010001001100011011010001010000000010001110001010100011101000011100101*110111110001101110001100010100100100111 011100110110000000111101000011010110101010101011101111011101011000101010*00100011110111101110010000111010001*1010100001011011000”
		256	“01011110100010000100100010011000110110100010100000000100011100010101000111010000111001011110111110001101110001100010100100100111 011100110110000000111101000011010110101010101011101111011101011000101010*00100011110111101110010000111010001*1010100001011011000”
	x		____ ____0___________________________________0___________________
. . .	. . . . . .		
***i* = 255**	***np***	1	“** **1***”
		2	“** **1*******************”
		. . .	. . . . . .
		255	“**************1*** **”
		256	“**1*************************************** **”
	x		0101111010001000010010001001100011011010001010000000010001110001010100011101000011100101_110111110001101110001100010100100100111 01110011011000000011110100001101011010101010101110111101110101100010101000100011110111101110010000111010001010101000010110 11000
***i* = 256**	***np***	1	“** **1***”
		2	“** **1*******************”
		. . .	. . . . . .
		255	“**************1*** **”
		256	“**1*************************************** **”
	x		01011110100010000100100010011000110110100010100000000100011100010101000111010000111001010110111110001101110001100010100100100111 01110011011000000011110100001101011010101010101110111101110101100010101000100011110111101110010000111010001010101000010110 11000

satisfies four conditions: (1) every entry has three specified positions, (2) the last six entries can be merged into two entries, (3) after merging, the last entry has one specified position; the penultimate entry has two specified positions; for each entry, there is only one specified symbol that does not match the corresponding bit of the original hashed password, and these specified symbols are at different positions, and (4) after merging, any specified position (except the one where the symbol does not match the corresponding bit of the original hashed password) of any entry (except the last one) must also be a specified position of its latter any entry.

The pseudo-code of the ENPII verification algorithm is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

shown in Algorithm 2, where NUMBEROFDS(x, y) at Lines 7 and 14 counts the number of different symbols between x and y (Note that '*' and '0' or '1' are different); MERGE(x, y) at Lines 10, 11, 12, and 17 constructs one entry that has the symbol '*' at the position where x (or y) has the symbol '0', and y (or x) has the symbol '1', and the same symbols at other positions with x and y (Before merging, we have ensured that x and y have different specified symbols at only one specified position and the same symbols at the other positions). Satisfying the conditions at Lines 3, 7, 14, 20, and 26 illustrates that np is not an output of Algorithm A.2, i.e., the np is a illegal negative password in ENPII, and the algorithm is terminated in advance. If the negative password is legal, when judging the legitimacy, the negative password is solved, and then the solution and hashed password are tested for equality.

Algorithm 2 ENPII Verification Algorithm

Input: a hashed password $hashP$;
a negative password np
Output: true or false

1: $m \leftarrow$ LENGTH$(hashP)$
2: **for** $i \leftarrow 1$ **to** $m+4$ **with stepsize of** 1 **do**
3: **if** NUMBEROFSP$(np_i) \neq 3$ **then**
4: **return** false
5: **end if**
6: **end for**
7: **if** NUMBEROFDS$(np_{m-1}, np_m) \neq 1$ **or** NUMBEROFDS$(np_{m+1}, np_{m+2}) \neq 1$ **or** NUMBEROFDS$(np_{m+3}, np_{m+4}) \neq 1$ **then**
8: **return** false
9: **else**
10: $np_{m-1} \leftarrow$ MERGE(np_{m-1}, np_m)
11: $np_{m+1} \leftarrow$ MERGE(np_{m+1}, np_{m+2})
12: $np_{m+3} \leftarrow$ MERGE(np_{m+3}, np_{m+4})
13: **end if**
14: **if** NUMBEROFDS$(np_{m+1}, np_{m+3}) \neq 1$ **then**
15: **return** false
16: **else**
17: $np_m \leftarrow$ MERGE(np_{m+1}, np_{m+3})
18: **end if**
19: **for** $i \leftarrow m$ **to** 1 **with stepsize of** -1 **do**
20: **if** NUMBEROFSP$(np_i) \neq 1$ **then**
21: **return** false
22: **end if**
23: $k \leftarrow$ INDEXOFSP(np_i)
24: $x[k] \leftarrow \neg$TOBIT$(np_i[k])$
25: **for** $j \leftarrow i-1$ **to** 1 **with stepsize of** -1 **do**
26: **if** $np_j[k] \neq$ TOSYMBOL$(x[k])$ **or** '*' **then**
27: **return** false
28: **end if**
29: $np_j[k] \leftarrow$ '*'
30: **end for**
31: **end for**
32: **if** $x = hashP$ **then**
33: **return** true
34: **else**
35: **return** false
36: **end if**

C. The performances of ENP

From the pseudo-code of Algorithm A.1, Algorithm 1, Algorithm A.2 and Algorithm 2, we could know that the time complexity of the generations of both ENPI and ENPII is $O(m^2)$, and the time complexity of the verifications of both ENPI and ENPII is also $O(m^2)$, where m is the length of the hashed password in the ENP. Since the length of the hashed passwords in the ENP is smaller, the generation and verification of the ENP are efficient.

V. ANALYSES

A. Notation

For convenience, we first introduce notation that will be used in this section and Section VI.

- N_d: the number of elements in a password list;
- N_p: the number of passwords to be cracked;
- T_h: the time spent on executing a cryptographic hash function;
- T_{ks}: the time spent on executing a key stretching algorithm;
- T_e: the time spent on executing the encryption of a symmetric-key algorithm;
- T_d: the time spent on executing the decryption of a symmetric-key algorithm;
- T_{m_hash}: the time spent on determining whether two hash values match;
- T_{m_ks}: the time spent on determining whether two passwords enhanced by a key stretching algorithm match;
- T_{m_NP}: the time spent on determining whether a hashed password matches a negative password, i.e., whether the hashed password is the solution of the negative password;
- l: the size of the salt (usually is sufficiently large);
- m: the size of the hash value (usually is 128, 160, 256, or 512 bits).

B. Attack Models

After obtaining an authentication data table, in order to crack the passwords in the table, adversaries may adopt brute force attack, dictionary attack, lookup table attack, reverse lookup table attack, rainbow table attack, or advanced dictionary attack (widely used in hashcat [20]). According to whether the precomputation technique is used, these attacks could be divided into two categories: the first includes lookup table attack and rainbow table attack, and the second includes brute force attack, dictionary attack, reverse lookup table attack, and advanced dictionary attack. In the first category, owing to the precomputation technique, both lookup table attack and rainbow table attack are effective methods to quickly crack lots of passwords, and the latter could attempt more possible passwords for its space-time tradeoff. In the second category, all four attacks simply attempt every possible

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

password in the password list, and the difference of these four attacks is the construction of the password list. The password list in brute force attack consists of all combinations of characters in a given character set; the password list in dictionary attack and reverse lookup table attack usually consists of frequently-used passwords; advanced dictionary attack exploits users' habits of constructing passwords, so that it could construct more complicated and effective password candidates. In addition, the reverse lookup table attack could simultaneously find passwords of several users for reusability of passwords. Without loss of generality, we select lookup table attack (from the first category) and dictionary attack (from the second category) to analyze the effectiveness of our scheme. The detailed descriptions of these two attacks are given below.

1) Lookup Table Attack: When carrying out lookup table attack, an adversary first prepares a password list, that usually consists of frequently-used passwords, concatenations of words in a vocabulary list, etc. Then, a lookup table is constructed, where the keys are encrypted passwords converted from elements in the password list by an encryption algorithm, and the records are the corresponding plain passwords (i.e., the elements in the password list). After these preparations are complete, the adversary steals an authentication data table in some way (assuming that the encryption algorithm used in this authentication data table is the same as that used in the prepared lookup table). Finally, for every encrypted password in the authentication data table, the adversary searches for the original plain password by matching the encrypted password and the keys in the lookup table. Note that the adversary could shorten the search time by adopting binary search algorithm or the data structure of hash table.

2) Dictionary Attack: When carrying out dictionary attack, an adversary first steals an authentication data table. Next, the adversary prepares the same password list with lookup table attack. Finally, for every encrypted password in the authentication data table, the adversary converts each element in the password list to ciphertext (assuming that the adversary knows the encryption algorithm used in the authentication data table) and determines whether the ciphertext matches the encrypted password. If a match is found, the adversary immediately obtains the original plain password (i.e., the current element in the password list).

C. Attack Complexity Analyses

In this subsection, we first explain that the ENP is able to resist lookup table attack. Then we analyze the attack complexity of the ENP under dictionary attack, which meanwhile serves as the basis for the next section.

1) Lookup Table Attack: If an adversary intends to crack ENPs using lookup table attack, because there are lots of corresponding ENPs for a given plain password, the adversary must first compute all possible ENPs for every element in the password list. Thus, we first calculate the number of possible ENPs converted from a plain password in the cases of both ENPI and ENPII.

In the case of ENPI, the random permutation causes the diversity of ENPs. Hence, the number of ENPs converted from a plain password is equal to the number of permutations, which is calculated by

$$N_{ENPI} = m!. \tag{1}$$

Besides the random permutation, the randomization during encoding '*' further improves the diversity of ENPs. However, for the simplification of analyses, this randomization is not counted. The same simplification is performed for ENPII.

In the case of ENPII, the conversion from a hashed password to a negative password in ENPII is considered to include two steps below.

(1) The hashed password is converted into a negative password in ENPI;
(2) The resulting negative password is converted into a negative password in ENPII.

Both Step (1) and Step (2) cause diversity of ENPs. Therefore, the number of ENPs converted from a plain password is calculated by

$$\begin{aligned} N_{ENPII} = &\left\{ \binom{m-1}{1} * \binom{m-2}{1} * \binom{m-2}{1} \right\} \\ &* \binom{m-2}{1} * \prod_{i=2}^{m-1} \binom{i}{2} * m!, \end{aligned} \tag{2}$$

where the first part (i.e., $\left\{ \binom{m-1}{1} * \binom{m-2}{1} * \binom{m-2}{1} \right\} * \binom{m-2}{1} * \prod_{i=2}^{m-1} \binom{i}{2}$) is the number of the negative passwords in ENPII converted from a negative password in ENPI, and the second part (i.e., $m!$) is the number of negative passwords in ENPI converted from a hashed password. Each part of Equation (2) is caused by the randomness in the code of Algorithm A.2, which we expand on below. The $\binom{m-1}{1} * \binom{m-2}{1}$ in $\left\{ \binom{m-1}{1} * \binom{m-2}{1} * \binom{m-2}{1} \right\}$ is caused by the randomness in the code at Line 20; the last $\binom{m-2}{1}$ in $\left\{ \binom{m-1}{1} * \binom{m-2}{1} * \binom{m-2}{1} \right\}$ is caused by the randomness in the code at Line 24; the subsequent $\binom{m-2}{1}$ is caused by the randomness in the code at Line 15; the $\binom{i}{2}$ is caused by the randomness in the code at Line 7.

To simplify Equation (2), we substitute Equation (3) and $\binom{m}{1} = m$ into Equation (2). Thus, we obtain Equation (4).

$$\prod_{i=2}^{m-1} \binom{i}{2} = \frac{(m-1)!(m-2)!}{2^{m-2}} \tag{3}$$

$$\begin{aligned} N_{ENPII} =& \{(m-1) * (m-2) * (m-2)\} \\ &* (m-2) * \frac{(m-1)!(m-2)!}{2^{m-2}} * m! \\ =& \frac{(m-2)^3 (m-1)!(m-1)!m!}{2^{m-2}} \\ =& \frac{[(m-2)m!]^3}{2^{(m-2)} m^2} \end{aligned} \tag{4}$$

In conclusion, under lookup table attack, the size of the lookup table (i.e., space complexity) is $(N_d * m!)$ when adopting ENPI to protect passwords or $(N_d * \frac{[(m-2)m!]^3}{2^{(m-2)} m^2})$ when adopting ENPII. Note that the size of the lookup table

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

increases quickly with the size of the hashed password. The size of the hash password is usually 128, 160, 256, or 512, so that the size of the lookup table is too big to be precomputed for the limits of storage resources. Therefore, the ENP is able to resist lookup table attack.

2) Dictionary Attack: If adversaries intend to crack ENPs using dictionary attack, for each ENP in the authentication data table, they verify every password in the password list. They first hash the password in the password list to a hashed password. Next, they decrypt the ENP, where the key is the hashed password. Then, they determine whether the hashed password is the solution of the negative password using Algorithm 1 or Algorithm 2; the success indicates that the adversaries crack this ENP. From the above procedure, we could conclude that the time complexity of cracking passwords is $O(N_d * N_p * (T_h + T_d + T_{m\_NP}))$; when multi-iteration encryption is employed, the time complexity is $O(N_d * N_p * (T_h + n * T_d + T_{m\_NP}))$, where n is the number of decryptions.

VI. Discussion

In order to further highlight the advantages of our scheme, in this section, we analyze and compare the attack complexity of our scheme with that of typical password storage schemes (i.e., hashed password, salted password and key stretching) under lookup table attack and dictionary attack.

A. Attack Complexity Analyses

1) Hashed Password: Hashed password [17] is a widely used scheme to protect passwords in an authentication data table. Hashed passwords can be calculated by

$$hashP = \text{HASH}(p_{plain}), \tag{5}$$

where HASH is a cryptographic hash function, such as SHA-256, p_{plain} is a plain password, and $hashP$ is the hash value of p_{plain}.

In the hashed password scheme, the passwords in the authentication data table are the hash values of plain passwords. Hashed passwords could be easily cracked by precomputation attacks, and the reasons are as follows. For a given plain password, the corresponding hashed password is determined, so that adversaries could precompute the hash values of all elements in the password list; these hashed passwords constitute the keys of the lookup table, and the records are the corresponding elements in the password list. Therefore, the time complexity of precomputing the lookup table is $O(N_d * T_h)$, and the size of the lookup table is N_d. The lookup table could be reused. After constructing the lookup table, when a linear search algorithm is adopted to search hashed passwords in the lookup table, the time complexity of cracking passwords is $O(N_d * N_p * T_{m\_hash})$. In addition, smart adversaries could sort the hashed passwords in the lookup table, then adopt binary search algorithm to shorten the search time; in this case, the time complexity of cracking passwords is $O(\log(N_d) * N_p * T_{m\_hash})$. Furthermore, through the data structure of hash table, the time complexity of cracking passwords could be decreased to $O(1 * N_p * T_{m\_hash})$. Therefore, hashed passwords are vulnerable under lookup table attack.

2) Salted Password: In order to resist lookup table attack, salt is introduced to improve the security of hashed passwords [17]. Salted passwords can be calculated by

$$hashP_{salt} = \text{HASH}(p_{plain}||salt), \tag{6}$$

where "||" is a concatenation operator, and $hashP_{salt}$ is the salted password. Note that the salt can also be on the left of p_{plain}.

In the case of salted passwords, the parameter of the cryptographic hash function is the concatenation of a plain password and a salt. The size of the salt is usually sufficiently large, and the salt usually is generated at random; hence, the salted passwords of the same plain passwords are almost always different. When the size of salt is l, the value space of the salt is 2^l; hence, if an adversary intends to crack passwords using lookup table attack, the size of the lookup table is $N_d * 2^l$. Note that the size of the lookup table increases exponentially with the size of the salt. When the size of the salt is sufficiently large, the size of the lookup table is too big to precompute the lookup table for the limits of storage resources. Therefore, salted passwords are able to resist lookup table attack. Also, since the lookup table cannot be prepared in advance, the adversary cannot reduce the time complexity by adopting binary search algorithm or hash table.

However, after an adversary gathers salted passwords and their corresponding salts, the adversary may adopt dictionary attack to crack passwords. For a given salted password and its salt, the adversary must hash every concatenation of each element in the password list and the salt, and then determine whether the result matches the original salted password. Since the number of concatenations is N_d for a given salted password, the time complexity of cracking all passwords using dictionary attack is $O(N_d * N_p * (T_h + T_{m\_hash}))$.

3) Key Stretching: Key stretching is the technique that makes passwords more secure by increasing the time used to test every password in the password list. The relevant algorithms include bcrypt [23], scrypt [24] and Argon2 [25], etc. Generally, the function that enhances passwords using the key stretching algorithm could be expressed as

$$ksP_{salt} = \text{KS}(\text{“}cost\ \ factor\text{”}, p_{plain}, salt), \tag{7}$$

where KS is a key stretching algorithm; the "*cost factor*" denotes parameters used to control the usage of computing resources (including processor and memory) in each run of the key stretching algorithm; ksP_{salt} is the password enhanced by the key stretching algorithm.

As with the salted password scheme, salt is also used in the key stretching scheme to increase the diversity of passwords. Besides adding some randomness through salt, one of the most significant features of the key stretching scheme is the controllability for the usage of computing resources in every run. Hence, the generation of the lookup table under the key stretching scheme could be more difficult than that under the salted password scheme. Consequently, the key stretching scheme could better resist lookup table attack.

In addition, the controllability for the usage of computing resources is also important for resisting dictionary attack. Under the key stretching scheme, the time complexity of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

TABLE VI
THE COMPARISONS OF ATTACK COMPLEXITY.

Schemes	Lookup table attack		Dictionary attack	
	Time complexity*	Space complexity	Time complexity	Space complexity
Hashed password	$O(N_d * N_p * T_{m\_hash})$	$O(N_d)$	$O(N_d * N_p * (T_h + T_{m\_hash}))$	$O(1)$
Salted password	$O(N_d * 2^l * N_p * T_{m\_hash})$	$O(N_d * 2^l)$	$O(N_d * N_p * (T_h + T_{m\_hash}))$	$O(l)$
Key stretching	$O(N_d * 2^l * N_p * T_{m\_ks})$	$O(N_d * 2^l)$	$O(N_d * N_p * (T_{ks} + T_{m\_ks}))$	$O(l)$
ENPI	$O(N_d * m! * N_p * T_{m\_NP})$	$O(N_d * m!)$	$O(N_d * N_p * (T_h + [n*]T_d + T_{m\_NP}))$	$O(m^2)$
ENPII	$O(N_d * \frac{[(m-2)m!]^3}{2^{(m-2)}m^2} * N_p * T_{m\_NP})$	$O(N_d * \frac{[(m-2)m!]^3}{2^{(m-2)}m^2})$	$O(N_d * N_p * (T_h + [n*]T_d + T_{m\_NP}))$	$O(m^2)$

*The time complexity under lookup table attack is based on linear search algorithm, and considers that the lookup table has been generated.

cracking passwords using dictionary attack is $O(N_d * N_p * (T_{ks} + T_{m\_ks}))$; the value of T_{ks} is controllable and could be large; and the larger T_{ks} (i.e., the higher resource cost) in every password attempt makes it more difficult to crack passwords using dictionary attack.

B. Comparisons

In order to clearly compare the attack complexity of hashed password, salted password, key stretching and the ENP, the time complexity and space complexity under lookup table attack and dictionary attack are listed in Table VI.

1) Under Lookup Table Attack: Table VI shows that, under lookup table attack, the space requirement of hashed password could be easily satisfied, and the time complexity is also low, so it is the most vulnerable password protection scheme among these schemes. However, it is difficult to meet the space requirements of salted password, key stretching, and the ENP; consequently, they could resist lookup table attack. In addition, the space complexity of the ENP (both ENPI and ENPII) is larger than that of salted password and key stretching when $l = m$, which makes the ENP better protected against lookup table attack than salted password and key stretching. It is worth mentioning that the ENP could guarantee the diversity of encrypted passwords by itself; that is to say, our scheme could resist lookup table attack without introducing extra elements (e.g., salt).

2) Under Dictionary Attack: By comparison with lookup table attack, nothing (such as the lookup table) needs to be precomputed and stored under dictionary attack; the only thing that adversaries need to do is to attempt every password in the password list for cracking passwords. Hence, our focus for attack complexity analyses under dictionary attack is on the time complexity. From Table VI, we could see that the time complexity of the ENP is obviously higher than that of hashed password and salted password (since $T_h + [n*]T_d + T_{m\_NP}$ is larger than $T_h + T_{m\_hash}$); consequently, the ENP has higher security than hashed password and salted password under dictionary attack.

The thing that we could do to defend against dictionary attack is to increase the time used to test every possible password, and then make dictionary attack ineffective. The ENP does that through multi-iteration encryption (i.e., increasing the n in the time complexity of cracking ENPs using dictionary attack; thus, at the time of cracking ENPs, adversaries need to execute the decryption the equal number of times with the encryption; the greater the number of encryption times is, the more difficult it is to crack ENPs using dictionary attack), while key stretching does that through raising higher resource demands (i.e., increasing the T_{ks}), including processor resources and memory resources. In addition, the latest key stretching algorithms, such as Argon2 [25], employ the memory-hard function [49] to defend against hardware approaches based on FPGAs, GPUs, or ASICs. Since the flexibility of our scheme, the key stretching algorithm could replace the cryptographic hash function in our scheme as a component to make the ENP feasible to resist hardware approaches. Moreover, by comparison with the key stretching scheme, the ENP takes advantage of symmetric encryption to protect passwords, which further improves the strength of passwords; and the ENP does not introduce salt to resist precomputation attacks, which is the unique advantage of our scheme.

C. Advantages

1) No Dependence on Salt: Adding salts is a widely used method to resist precomputation attacks, however, it tends to be implemented by mistake (such as salt reuse and short salt), which poses a potential security exposure and puts higher requirement on programmers. In addition, programmers need to pay attention to the storage and usage of salts. Conversely, the ENP only needs to select a pair of cryptographic hash function and symmetric-key algorithm without the need for extra elements (such as salt), which indicates that our scheme is programmer-friendly.

2) Resistance to Lookup Table Attack: Given a plain password, there are lots of corresponding ENPs, i.e., the ENPs converted from the same password are almost always different, which makes it effectively resist lookup table attack. Furthermore, for the same passwords of a user in different systems, because the corresponding ENPs are almost always different, even if an adversary obtains two ENPs in different authentication data tables from different systems, the adversary cannot

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

determine whether the original plain passwords corresponding to the two ENPs are the same.

3) Resistance to Dictionary Attack: In the ENP, multi-iteration encryption is used to defend against dictionary attack by making every password attempt consume more time. Even though hardware approaches are used, the passwords generated by our scheme are still strong, since our scheme could not only increase the number of encryptions, but also replace the cryptographic hash function with a key stretching algorithm.

VII. Conclusions and Future Work

In this paper, we proposed a password protection scheme called ENP, and presented a password authentication framework based on the ENP. In our framework, the entries in the authentication data table are ENPs. In the end, we analyzed and compared the attack complexity of hashed password, salted password, key stretching and the ENP. The results show that the ENP could resist lookup table attack and provide stronger password protection under dictionary attack. It is worth mentioning that the ENP does not need extra elements (e.g., salt) while resisting lookup table attack.

In the future, other NDB generation algorithms will be studied and introduced to the ENP to further improve password security. Furthermore, other techniques, such as multi–factor authentication and challenge–response authentication, will be introduced into our password authentication framework.

Appendix

In this part, we provide the details of the prefix algorithm [30] with permutation (used in ENPI) and a variant of the prefix algorithm (used in ENPII) that is the first part of the hybrid algorithm [42]. The purpose of writing this appendix is to present a self-contained paper for better understanding.

The pseudo-code of the prefix algorithm with permutation is shown in Algorithm A.1, and the pseudo-code of the variant of the prefix algorithm is shown in Algorithm A.2. For both Algorithm A.1 and Algorithm A.2, the input s is a sequence of bits, and the output ndb is the resulting negative database (NDB) that contains many sequences of symbols (i.e., '0', '1', or '*'). In the ENP proposed in this paper, the hash value (represented as a sequence of bits; as the input of Algorithm A.1 or Algorithm A.2) of the plain password received from a client is converted to a negative password (as the output of Algorithm A.1 or Algorithm A.2) by Algorithm A.1 or Algorithm A.2.

In Algorithm A.1, $\pi(s)$ at Line 2 denotes a random permutation for the input s, and $\pi'(s)$ at Line 13 is its inverse permutation; $permutedBits$ at Line 2 denotes the sequence of bits permuted from the input s; LENGTH(x) at Line 3 counts the number of bits in x; CREATESEQUENCEOFSYMBOLS(m) at Line 5 returns a sequence of symbols of length m that only consists of the symbol '*'; TOSYMBOL(b) at Lines 7 and 9 converts the bit b to the corresponding symbol (i.e., 0 to '0' and 1 to '1'); APPEND(ndb, x) at Line 13 appends x to the end of the ndb list. Note that, at the beginning of the algorithm, s is randomly permuted (i.e., $\pi(s)$); accordingly, the entries to append to ndb must be permuted in reverse (i.e., $\pi'(s)$).

Algorithm A.1 Prefix Algorithm with Permutation

Input: a sequence of bits s
Output: an NDB ndb

1: $ndb \leftarrow$ an empty list
2: $permutedBits \leftarrow \pi(s)$
3: $m \leftarrow$ LENGTH$(permutedBits)$
4: **for** $i \leftarrow 1$ **to** m **with stepsize of** 1 **do**
5: $\quad x \leftarrow$ CREATESEQUENCEOFSYMBOLS(m)
6: $\quad$ **for** $j \leftarrow 1$ **to** $i-1$ **with stepsize of** 1 **do**
7: $\quad\quad x[j] \leftarrow$ TOSYMBOL$(permutedBits[j])$
8: $\quad$ **end for**
9: $\quad x[i] \leftarrow$ TOSYMBOL$(\neg(permutedBits[i]))$
10: $\quad$ **for** $j \leftarrow i+1$ **to** m **with stepsize of** 1 **do**
11: $\quad\quad x[j] \leftarrow$ '*'
12: $\quad$ **end for**
13: $\quad$ APPEND$(ndb, \pi'(x))$
14: **end for**
15: **return** ndb

Algorithm A.2 Variant of the Prefix Algorithm

Input: a sequence of bits s
Output: an NDB ndb

1: $ndb \leftarrow$ an empty list
2: $permutedBits \leftarrow \pi(s)$
3: $m \leftarrow$ LENGTH$(permutedBits)$
4: **for** $i \leftarrow m$ **to** 3 **with stepsize of** -1 **do**
5: $\quad x \leftarrow$ CREATESEQUENCEOFSYMBOLS(m)
6: $\quad x[i] \leftarrow$ TOSYMBOL$(\neg(permutedBits[i]))$
7: $\quad j \leftarrow$ RAND$(1, i-1)$; $k \leftarrow$ RAND$(1, i-1)$ $(k \neq j)$
8: $\quad x[j] \leftarrow$ TOSYMBOL$(permutedBits[j])$
9: $\quad x[k] \leftarrow$ TOSYMBOL$(permutedBits[k])$
10: $\quad$ APPEND$(ndb, \pi'(x))$
11: **end for**
12: $x \leftarrow$ CREATESEQUENCEOFSYMBOLS(m)
13: $x[1] \leftarrow$ TOSYMBOL$(permutedBits[1])$
14: $x[2] \leftarrow$ TOSYMBOL$(\neg(permutedBits[2]))$
15: $j =$ RAND$(3, m)$
16: $x[j] \leftarrow$ '0'; APPEND$(ndb, \pi'(x))$
17: $x[j] \leftarrow$ '1'; APPEND$(ndb, \pi'(x))$
18: $x \leftarrow$ CREATESEQUENCEOFSYMBOLS(m)
19: $x[1] \leftarrow$ TOSYMBOL$(\neg(permutedBits[1]))$
20: $j \leftarrow$ RAND$(2, m)$; $k \leftarrow$ RAND$(2, m)$ $(k \neq j)$
21: $x[j] \leftarrow$ '0'; $x[k] \leftarrow$ '0'; APPEND$(ndb, \pi'(x))$
22: $x[j] \leftarrow$ '0'; $x[k] \leftarrow$ '1'; APPEND$(ndb, \pi'(x))$
23: $x[k] =$ '*'
24: $k \leftarrow$ RAND$(2, m)$ $(k \neq j)$
25: $x[j] \leftarrow$ '1'; $x[k] \leftarrow$ '0'; APPEND$(ndb, \pi'(x))$
26: $x[j] \leftarrow$ '1'; $x[k] \leftarrow$ '1'; APPEND$(ndb, \pi'(x))$
27: **return** ndb

In Algorithm A.2, more randomization is introduced at Lines 7, 15, 20, and 24; RAND(x, y) at Lines 7, 15, 20, and 24 returns a random integer value which is uniformly drawn from the interval $[x, y]$. The code at Lines 4–11 generates the 1st to $(m-2)$th entries, the code at Lines 12–17 generates the $(m-1)th$ to mth entries, and the code at Lines 18–26

generates the $(m+1)$th to $(m+4)$th entries.

REFERENCES

[1] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "Passwords and the evolution of imperfect authentication," *Communications of the ACM*, vol. 58, no. 7, pp. 78–87, Jun. 2015.

[2] M. A. S. Gokhale and V. S. Waghmare, "The shoulder surfing resistant graphical password authentication technique," *Procedia Computer Science*, vol. 79, pp. 490–498, 2016.

[3] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Proceedings of 2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 689–704.

[4] A. Adams and M. A. Sasse, "Users are not the enemy," *Communications of the ACM*, vol. 42, no. 12, pp. 40–46, Dec. 1999.

[5] E. H. Spafford, "Opus: Preventing weak password choices," *Computers & Security*, vol. 11, no. 3, pp. 273–278, 1992.

[6] Y. Li, H. Wang, and K. Sun, "Personal information in passwords and its security implications," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2320–2333, Oct. 2017.

[7] D. Florencio and C. Herley, "A large-scale study of web password habits," in *Proceedings of the 16th International Conference on World Wide Web*. ACM, 2007, pp. 657–666.

[8] R. Shay, S. Komanduri, A. L. Durity, P. S. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin, and L. F. Cranor, "Designing password policies for strength and usability," *ACM Transactions on Information and System Security*, vol. 18, no. 4, pp. 13:1–13:34, May 2016.

[9] D. Wang, D. He, H. Cheng, and P. Wang, "fuzzyPSM: A new password strength meter using fuzzy probabilistic context-free grammars," in *Proceedings of 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Jun. 2016, pp. 595–606.

[10] H. M. Sun, Y. H. Chen, and Y. H. Lin, "oPass: A user authentication protocol resistant to password stealing and password reuse attacks," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 2, pp. 651–663, Apr. 2012.

[11] M. Zviran and W. J. Haga, "Password security: An empirical study," *Journal of Management Information Systems*, vol. 15, no. 4, pp. 161–185, 1999.

[12] P. Andriotis, T. Tryfonas, and G. Oikonomou, "Complexity metrics and user strength perceptions of the pattern-lock graphical authentication method," in *Proceedings of Human Aspects of Information Security, Privacy, and Trust*. Springer International Publishing, 2014, pp. 115–126.

[13] D. P. Jablon, "Strong password-only authenticated key exchange," *SIGCOMM Computer Communication Review*, vol. 26, no. 5, pp. 5–26, Oct. 1996.

[14] J. Jose, T. T. Tomy, V. Karunakaran, A. K. V, A. Varkey, and N. C. A., "Securing passwords from dictionary attack with character-tree," in *Proceedings of 2016 International Conference on Wireless Communications, Signal Processing and Networking*, Mar. 2016, pp. 2301–2307.

[15] A. Arora, A. Nandkumar, and R. Telang, "Does information security attack frequency increase with vulnerability disclosure? an empirical analysis," *Information Systems Frontiers*, vol. 8, no. 5, pp. 350–362, Dec. 2006.

[16] R. Song, "Advanced smart card based password authentication protocol," *Computer Standards & Interfaces*, vol. 32, no. 5, pp. 321–325, 2010.

[17] M. C. Ah Kioon, Z. S. Wang, and S. Deb Das, "Security analysis of MD5 algorithm in password storage," in *Proceedings of Instruments, Measurement, Electronics and Information Engineering*. Trans Tech Publications, Oct. 2013, pp. 2706–2711.

[18] P. Oechslin, "Making a faster cryptanalytic time-memory trade-off," in *Proceedings of Advances in Cryptology - CRYPTO 2003*. Springer Berlin Heidelberg, 2003, pp. 617–630.

[19] S. Noel, M. Elder, S. Jajodia, P. Kalapa, S. O'Hare, and K. Prole, "Advances in topological vulnerability analysis," in *Proceedings of 2009 Cybersecurity Applications Technology Conference for Homeland Security*, Mar. 2009, pp. 124–129.

[20] "Hashcat," https://hashcat.net/hashcat/.

[21] "RainbowCrack," http://project-rainbowcrack.com/.

[22] "John the Ripper," http://www.openwall.com/john/.

[23] N. Provos and D. Mazières, "A future-adaptive password scheme," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. USENIX Association, 1999, pp. 32–32.

[24] "RFC 7914: The scrypt Password-Based Key Derivation Function," https://tools.ietf.org/html/rfc7914.

[25] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: New generation of memory-hard functions for password hashing and other applications," in *Proceedings of 2016 IEEE European Symposium on Security and Privacy*, Mar. 2016, pp. 292–302.

[26] "Password Hashing Competition," https://password-hashing.net/.

[27] S. Boonkrong and C. Somboonpattanakit, "Dynamic salt generation and placement for secure password storing," *IAENG International Journal of Computer Science*, vol. 43, no. 1, pp. 27–36, 2016.

[28] M. Najjar, "Using improved d-HMAC for password storage," *Computer and Information Science*, vol. 10, no. 3, pp. 1–9, Jul. 2017.

[29] F. Esponda, E. S. Ackley, S. Forrest, and P. Helman, "Online negative databases," in *Proceedings of Artificial Immune Systems*. Springer Berlin Heidelberg, 2004, pp. 175–188.

[30] F. Esponda, S. Forrest, and P. Helman, "Enhancing privacy through negative representations of data," Department of Computer Science, University of New Mexico, Tech. Rep., 2004.

[31] F. Esponda, "Negative representations of information," Ph.D. dissertation, University of New Mexico, Albuquerque, NM, USA, 2005.

[32] F. Esponda, S. Forrest, and P. Helman, "Negative representations of information," *International Journal of Information Security*, vol. 8, no. 5, pp. 331–345, Oct. 2009.

[33] D. Dasgupta and R. Azeem, "An investigation of negative authentication systems," in *Proceedings of the 3rd International Conference on Information Warfare and Security*, Apr. 2008, pp. 117–126.

[34] D. Dasgupta and S. Saha, "A biologically inspired password authentication system," in *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*. ACM, 2009, pp. 41:1–41:4.

[35] D. Zhao, W. Luo, R. Liu, and L. Yue, "Negative iris recognition," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 112–125, Jan. 2018.

[36] R. Liu, W. Luo, and L. Yue, "Hiding multiple solutions in a hard 3-SAT formula," *Data & Knowledge Engineering*, vol. 100, pp. 1–18, 2015.

[37] S. Boonkrong, "Security of passwords," *Information Technology Journal*, vol. 8, no. 2, pp. 112–117, Jul.–Dec. 2012.

[38] J. Kelsey, B. Schneier, C. Hall, and D. Wagner, "Secure applications of low-entropy keys," in *Proceedings of Information Security*. Springer Berlin Heidelberg, 1998, pp. 121–134.

[39] W. Barthel, A. K. Hartmann, M. Leone, F. Ricci-Tersenghi, M. Weigt, and R. Zecchina, "Hiding solutions in random satisfiability problems: A statistical mechanics approach," *Physical review letters*, vol. 88, p. 188701, Apr. 2002.

[40] D. Achlioptas, H. Jia, and C. Moore, "Hiding satisfying assignments: Two are better than one," *Journal of Artificial Intelligence Research*, vol. 24, no. 1, pp. 623–639, Nov. 2005.

[41] F. Esponda, E. S. Ackley, P. Helman, H. Jia, and S. Forrest, "Protecting data privacy through hard-to-reverse negative databases," *International Journal of Information Security*, vol. 6, no. 6, pp. 403–415, Oct. 2007.

[42] R. Liu, W. Luo, and X. Wang, "A hybrid of the prefix algorithm and the q-hidden algorithm for generating single negative databases," in *Proceedings of 2011 IEEE Symposium on Computational Intelligence in Cyber Security*, Apr. 2011, pp. 31–38.

[43] R. Liu, W. Luo, and L. Yue, "The p-hidden algorithm: hiding single databases more deeply," *Immune Computation*, vol. 2, no. 1, pp. 43–55, Mar. 2014.

[44] D. Zhao, W. Luo, R. Liu, and L. Yue, "A fine-grained algorithm for generating hard-toreverse negative databases," in *Proceedings of 2015 International Workshop on Artificial Immune Systems*, Jul. 2015, pp. 1–8.

[45] D. Zhao and W. Luo, "One-time password authentication scheme based on the negative database," *Engineering Applications of Artificial Intelligence*, vol. 62, pp. 396–404, 2017.

[46] J. Bringer and H. Chabanne, "Negative databases for biometric data," in *Proceedings of the 12th ACM Workshop on Multimedia and Security*. ACM, 2010, pp. 55–62.

[47] J. Liang and X. Lai, "Improved collision attack on hash function MD5," *Journal of Computer Science and Technology*, vol. 22, no. 1, pp. 79–87, Jan. 2007.

[48] Y. Sasaki and K. Aoki, "Finding preimages in full MD5 faster than exhaustive search," in *Proceedings of Advances in Cryptology - EUROCRYPT 2009*. Springer Berlin Heidelberg, 2009, pp. 134–152.

[49] D. Boneh, H. Corrigan-Gibbs, and S. Schechter, "Balloon hashing: A memory-hard function providing provable protection against sequential attacks," in *Proceedings of Advances in Cryptology – ASIACRYPT 2016*. Springer Berlin Heidelberg, 2016, pp. 220–248.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2844854, IEEE Transactions on Information Forensics and Security

Wenjian Luo received his BS and PhD degrees from the Department of Computer Science and Technology, University of Science and Technology of China, Hefei, China in 1998 and 2003, respectively. He is currently an associate professor of the School of Computer Science and Technology, University of Science and Technology of China. His current research interests include machine learning and data mining, information security and data privacy, computational intelligence and applications.

Yamin Hu received his BE degree from the College of Information Engineering, Northwest A&F University, Yangling, China in 2016. He is currently working toward a master's degree at the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China. His current research interests include information security and data privacy.

Hao Jiang received his BS degree from the Department of Computer Science and Technology, North China Electric Power University, Baoding, China in 2014. He is currently working toward a PhD degree at the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China. His current research interests include information security, data privacy, and computational intelligence.

Junteng Wang received his BE degree from the School of Information Science and Technology, University of Science and Technology of China, Hefei, China in 2017. He is currently working toward a master's degree at the School of Computer Science and Technology, University of Science and Technology of China. His current research interests include information security and data privacy.