Investigating offline password attacks: A comprehensive review of rainbow table techniques and countermeasure limitations

Fazal WAHAB*[1], Imran KHAN*[2], Ken SI[3],

[1] Corvit Systems Peshawar, Pakistan

[2, 3] Department of Informatics, University of Sussex, UK

fazalwahabstu@gmail.com, imran.khan@sussex.ac.uk, ken.si@sussex.ac.uk

***Corresponding Authors:**

Fazal WAHAB, Imran KHAN
fazalwahabstu@gmail.com, imran.khan@sussex.ac.uk

Abstract: The password is the most prevalent method of authentication and is essential for keeping data safe. Due to instances of the release of massive volumes of data records caused by database breaches, a sizable number of passwords have already been broken offline using the disclosed data in recent years. The rainbow table attack, the most practical offline password attack method, is systematically discussed in this study. The two improved rainbow table attack schemes – a novel time-memory tradeoff method using rainbow table sort proposed by Thing and Ying (TY attack) and an upgraded approach combining TY attack with differentiated points proposed by Li (Li attack) – are the main approaches of this article. Additionally, this article addresses the limitations of the current countermeasures for defending against offline password attacks from the perspectives of file encryption, password vaults, and hardware.

Keywords: Rainbow Table Attack, Offline Password Attack, TY Attack, Password Security.

1. Introduction

Personal and company information is increasingly being kept by Internet service suppliers or third-party information services due to the proliferation of cloud computing and big data. However, not all data service providers can be trusted with yours information. Hackers can gain access to the servers storing password credential files by exploiting system flaws, gaps in security, or human error. In the digital age, offline password attacks that target stolen or hacked hashed or encrypted password databases pose serious cybersecurity risks. Offline attacks involve the unauthorized acquisition of password hashes or encrypted passwords, which intruders then attempt to decode using a variety of methods, in contrast to online attacks that target active login systems (Aslam et al., 2023; Khan et al., 2023). In order to protect sensitive user data and improve cybersecurity protective measures, it is essential to comprehend the nature of offline password attacks and the mitigation techniques used to counteract them.

When launching an offline password attack, the adversary has full access to the file storing the encrypted form of passwords on the targeted system and can use any means to decrypt the passwords. The cracking procedure is undetectable by the target system (Yu & Huang, 2015). Sensitive user data has been regularly disclosed or published over the past few years due to the information service provider's weak security procedures. As a result of exposed password credential files or, in some cases, the plain text of passwords, offline password attacks can crack the passwords of a large number of users. The Verge reports that since 2012, up to 20,000 Facebook workers have had access to users' passwords stored in plain text on Facebook's servers. An estimated 1 billion Yahoo accounts were affected due to a data hack in 2013, as reported by The Guardian (Thielman, 2016). Users are now at great risk due to the exposure of files containing cipher text and even plaintext of their passwords. Due to the widespread practice of reusing passwords across multiple online services, hackers can potentially compromise a large number of credentials without the knowledge of users or the notice of companies by gaining access to relevant

https://doi.org/10.33436/v34i1y202408

files and carrying out offline password attacks. Research on methods for thwarting offline password attacks, such as protections to keep sensitive information from falling into the wrong hands and methods for authenticating users in the event of a breach, is now crucial (Kastrenakes, 2019).

Cybersecurity faces a significant challenge from offline password attacks that target encrypted or stolen password hashes. Dictionary attacks, rainbow table attacks and other methods are used by attackers to extract plaintext passwords from hashed data. Strong hashing algorithms, novel salts, and the enforcement of strict password regulations are all necessary for mitigating this threat (Thing & Ying, 2009). Organizations and individuals can more effectively protect sensitive data and uphold the integrity of their password security systems by understanding the techniques used by attackers and taking preventative action. Organizations and individuals can use a variety of security measures to prevent offline password attacks. The development of hashes can be slowed down using hashing algorithms like bcrypt or scrypt, which makes it more difficult and time-consuming for attackers to decipher passwords (Li et al., 2012). Additionally, by guaranteeing that even similar passwords generate different hashes, the use of distinct salts for each password can further strengthen security. A solid defense against offline attacks still relies on users being informed about strong password practices and having their passwords updated regularly.

This work is organized in such a way that a comprehensive literature overview of offline password attacks and their defences is presented in Section 2. Section 3 presents how a rainbow attack works and what it entails. Rainbow table strategies, with an emphasis on the TY attack and the Li attack, are covered in Section 4. Section 5 covered current file encryption, password vault, and hardware safeguards against offline password attacks. Finally, in Section 6, problems and restrictions inherent in current offline password attack methods, conclusion, limitations, and future work are presented.

2. Literature review

2.1. The development of password

A password is a secret piece of information, often a short string of characters, that users know and use for authentication, as described by Stallings (2011). Shift ciphers and substitution ciphers are examples of classical procedures for passwords (Stinson, 2005). Examples of such methods are the Caesar password and the Vigenere password. People started using rotor machines for encryption and decryption to increase the security and complexity of password systems. The well-known German cipher system Enigma, used throughout the Second World War, serves as an excellent example. The Communication Theory of Security Systems was published by Shannon (1949). He founded cryptographic theory and incorporated information theory into cryptography. Public key encryption was first presented by Diffie & Hellman (1976), ushering in a new era for cryptography (Diffie & Hellman, 1976). Numerous public key encryption algorithms have since been created. The RSA (Rivest et al., 1978 b) algorithm was developed by Rivest et al. (1978 a). It solved the open channel key distribution problem and enabled digital signatures. Public key algorithms also use El Gamal Encryption and ECC (Elliptic Curve Cryptography), in addition to the RSA technique. In 1993, the National Institute of Standards and Technology (NIST) released the Secure Hash Algorithm (SHA), which used the hash function and prevented the recovery of cyphertext or passwords. SHAs from more recent generations have been published. Hash encryption also uses the MD (Message Digest) algorithm and the SHA algorithm.

2.2. Offline password attacks

The authors (Yu & Huang, 2015) refer to an offline password attack as the process of obtaining the plaintext from the encrypted data utilizing only the password file and encryption method. Offline password attacks can be divided into three groups: dictionary, brute-force, and rainbow tables. A brute-force attack is one in which the attacker repeatedly tries every possible input combination. To identify brute-force attacks, Sperotto et al. (2009) suggested a Hidden Markov Model-based flow time series model. When conducting a dictionary attack, as opposed to a

brute-force attack, the attacker restricts the input space to words from a dictionary. To make dictionary attacks more effective, Weir et al. (2009) used a publicly available training set to construct highly probable password patterns.

For Hellman's (1980) time-memory trade-off strategy, Oechslin (2003) initially proposed the rainbow table. The temporal complexity of a rainbow table attack is lower than that of a brute-force attack and lower than that of a dictionary attack. A rainbow table is a precomputed lookup table used in the rainbow table attack to convert a cryptographic hash of a password back to the original plaintext password. Attackers can utilize the reduction function and the hash function to calculate the hash chain of a given hash value to retrieve the related plaintext. The adversary can recover the plaintext by regenerating the hash chain of any value on this chain that appears in a rainbow table. In addition, based on the rainbow table attack, Thing & Ying (2009) offers a new memory-time trade-off password attack. This method selects an initial value from among a smaller, more manageable group of plaintexts. Pre-computing and memory optimization help lower the required amount of storage space. To boost the efficacy of cryptanalysis, Avoine et al. (2008) presented a checkpoint-based improvement to the rainbow table attack. Avoine & Carpent (2017) increased the effectiveness of rainbow table attacks by determining the appropriate width of each rainbow table and visiting them in accordance with a new decision function rather than in sequential order.

2.3. Defense mechanism against offline password attacks

Various strategies have been employed to counter offline password attacks. Some of the well-known mechanisms are as follows.

2.3.1. Defense mechanism from file-encryption perspective

Users' passwords can be fundamentally protected by new encryption algorithms that can encrypt files stored on the server and guarantee the availability of the data at the same time, preventing offline attackers from discovering the password's cyphertext. Privacy Homomorphism (Rivest et al., 1978 a) initially introduced encryption, which permits users to manipulate encrypted data without first decrypting it. The computations performed by the homomorphic method on the cyphertext after decoding are the same as those performed on the plaintext. Homomorphic Privacy, the sophisticated algorithm of the system requires an excessive amount of resources for encryption. Function Encryption permits servers to acquire only the outcome of a particular function on ciphertext and no other information to reduce the algorithm's cost. IBE (Identity-Based Encryption) was first proposed by Boneh & Franklin (2001). Then, to reconcile the tensity between accessibility and privacy, Sahai & Waters (2005) proposed a fuzzy identity-based encryption system. The shipper establishes a set of qualities, a threshold, and data encryption as part of their system. Only recipients with the required minimum number of specified attributes can decrypt the ciphertext. The Key-Policy Attribute-Based Encryption (KP-ABE) Goyal et al. (2006) proposed embeds the access structure in the private key and integrates fine-grained access control with ABE. It is based on fuzzy IBE. Searchable encryption is one sort of functional encryption that guarantees data accessibility through searchable cyphertext. Identity-driven Public Key Encryption with Keyword Search was introduced by Boneh et al. (2004). In contrast to contributors with public keys, who can only send encrypted data into the system, users with private keys can search for, access, and decrypt encrypted data in this system. The primary private key is comparable to the users' private key in encryption with keyword search, and the scheme's keyword is comparable to the identity in Identity-Based Encryption (IBE). The token used to search is comparable to the identity's private key.

2.3.2. Defense mechanism from a password manager perspective

The use of a password manager, which is also referred to as a password vault, is an extremely efficient method for protecting users' passwords. It is a type of software application that transforms a user's master password into a series of unique passwords that may be used for a variety of services. A new hash technique proposed by Blocki & Sridhar (2016) uses client-side key stretching to raise the computational cost of the key derivation function for incorrect master

password guesses. It aided in lowering legitimate user costs while raising attack costs. A new cracking-resistant approach for password vaults based on Natural Language Encoder was developed by Chatterjee et al. (2015), which uses a decoy password generator in the event of a failed master password decryption. This redirects attackers to the website instead.

2.3.3. Defense mechanism from hardware perspective

A new password system, proposed by Almeshekah et al. (2015) incorporates a security module that is embedded in the target system to prevent unauthorized access to files storing passwords. In the event that an attacker does not know the hardware-dependent function, this system's deception mechanism can generate bogus passwords and alert administrators to the attack.

3. Rainbow table attack

Offline password attacks can be either dictionary, rainbow table, or brute-force-based. For a brute-force attack to work, the key space must be traversed, which is a very time-consuming process. In contrast, the space complexity of a dictionary attack is extremely high because it requires the generation and storage of pairings of plaintexts and cipher texts before breaking. As users' passwords grow longer, both dictionary attacks and brute force attacks become impractical due to their respective space and time complexity. In response, researchers have turned their attention to the rainbow table attack, which combines the benefits of both.

3.1. Time-memory trade-off scheme

Time-memory trade-off schemes have been around since 1980, when (Hellman, 1980) first presented. That method starts by picking a key, k_1, at random, and then using the hash function H to determine its value. The new key k_2 is determined by applying a reduction function R to the cipher text.

$$k_2 = f(k_1) \tag{1}$$

Where $f = h°$

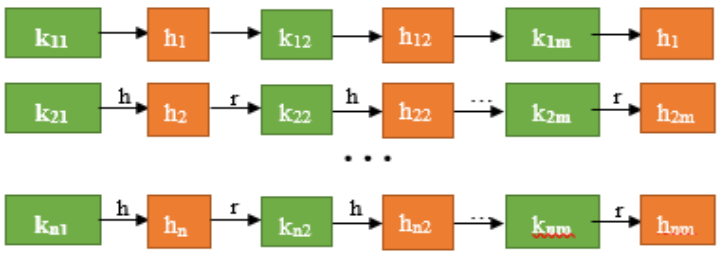

Figure 1. Hash Chain Table

The steps in Figure 1 to generate a hash chain through m iterations should be followed. The only information saved is the initial and final links in the chain (k1 to km) and (c1 to cm), respectively). It is possible to generate a table of hash chains by changing the value of k1. The given Hash value x should be decripted using r to obtain y_1. Then, h should be plugged in to y_1 to obtain x_1. The table should be checked to see if x_1 appears there. If so, the chain can be reconstructed and the original text located. If not, the steps again should be tried again. Multiple tables can be constructed and saved to increase the password-cracking success rate. Each table here makes use of a unique r-value reduction function. Instances of cracking can have varying degrees of success depending on the reduction function used. The success rate can be raised by trying out various reduction functions. In general, it is possible to lower the space complexity of the attack method by storing the estimated beginning and ending locations on the chain. However, additional operations involving comparison of the cipher text raise the temporal complexity. It is more practical than a dictionary attack or a brute-force attack because of the trade-off between time and memory.

3.2. Rainbow table

Hellman's time-memory trade-off framework places a premium on the ability to cross and merge chains. If the same value appears in two separate chains, the chains containing those values must be combined. Oechslin (2003) initially introduced the rainbow table technique, which employs a different reduction function in each chain and thereby improves the time-memory trade-off scheme. Figure 2 depicts the rainbow table's internal structure.

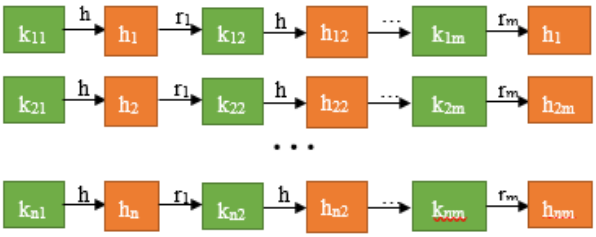

Figure 2. Rainbow Table

The rainbow table attack is conducted in three phases. First, a rainbow table is computed using m separate reduction functions (r_1 to r_m), and the initial and final values are saved. Then, an examination of each table from the beginning to the en dis required, followed by the merging of the crossing chains, their arrangement, and finally, their sorting. Finally, binary search is used to determine if there is a match in the terminal points, based on a given hash value of x. If so, the plaintext can be retrieved by recreating the Hash chain from the beginning. If not, r_m shaould be substituted for x to obtain y_1, and then h should be substituted for y_1 to obtain x_1. The existenc of a matching entry x_1 should be determined. In every other case, x should be replaced with r_{m-1} and proceed as before.

4. T-Y attack

In order to reduce rainbow table memory loss, Thing & Ying (2009) refined the construction of rainbow tables. The novel T-Y attack architecture deviates from the rainbow table in that, unlike the tables' pre-computation stage, it does not save all the plaintext.

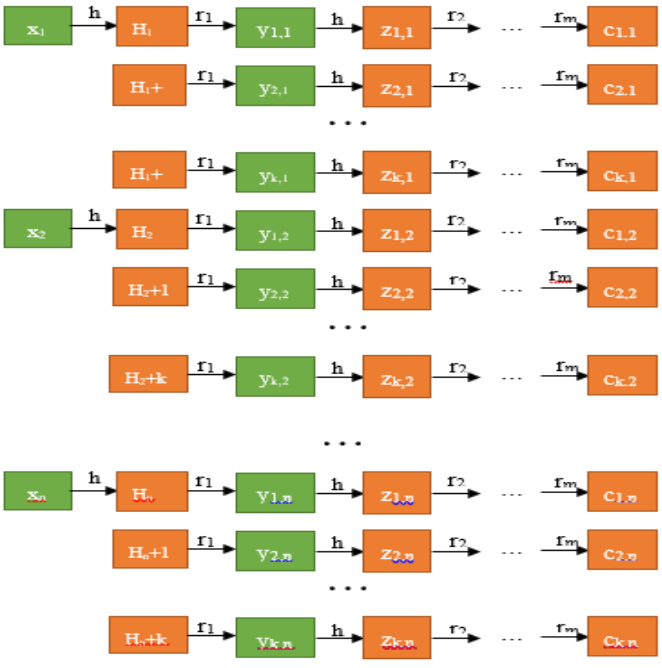

Figure 3. Table structure of T-Y Attack

To calculate the first block of a table, one can look at Figure 3 and assume that there are k+1 chains in the first block and that the first hash values of each chain are H, H+1, H+2,..., H+k. Consequently, the tracking of x1 and the numbers from c1 to cm is maintained. Each node in a T-Y attack is allocated a specific number of bits. Each block requires $l(k+2)$ bits of storage space since it keeps track of x as the starting point and k+1 as the final destination. For the same number of m chains as in the rainbow table, the number of blocks required for the T-Y Attack is 2m/(k+2). Thus, 2m/(k+2)*(n+k*n-k) bits of RAM are required for a T-Y Attack. Ting & Ying (2009) found that the minimum amount of total memory occurs at k=2m-2. The entire table consists of a single block. Ying & Thing (2011) used a technique for sorting passwords that involved inserting special characters into the endpoints of indexing the passwords so that their location could be determined even after sorting. T-Y attack can find the password by using the attack's endpoint, starting point, and special characters throughout the searching phase.

5. Li Attack

T-Y Attack was first implemented and evaluated in 2012 by Li et al. (2012). They discovered a linear relationship between the hash values H, H+1,...,H+k prior to the sorting process. The sorting procedure failed, and T-Y Attack was first implemented and evaluated in 2012 by Li et al. (2012). Pre-sorting hash values H, H+1,...,H+k were found to follow a linear progression. This order was lost in the sorting phase, and the unique character in T-Y Attack greatly increases the temporal complexity of the indexing procedure, making its practical implementation impossible. The TMTO (Time-Memory Trade-Off) strategy was enhanced by Li's combination of the T-Y attack and other systems based on distinguishing points. In Li Attack, a random X was picked as starting position. Due to the exponential rise in the temporal complexity of indexing caused by a unique character in T-Y attack, its practical implementation is unfeasible. The TMTO strategy was enhanced by Li's combination of the T-Y attack and other systems based on distinguishing points. In Li Attack, a random X was picked as starting position.

$$SP_0^0 = H(X+0) \oplus 0 \xrightarrow{F_1} o \xrightarrow{F_2} \cdots \xrightarrow{F_{l_0^0}} (0 \,\|\, \mathbb{R}_0^0) = EP_0^0 \tag{2}$$

$$SP_1^0 = H(X+1) \oplus 0 \xrightarrow{F_1} o \xrightarrow{F_2} \cdots \xrightarrow{F_{l_1^0}} (0 \,\|\, \mathbb{R}_1^0) = EP_1^0 \tag{3}$$

$$SP_{k_{1-1}}^0 = H(H+K_1-1) \oplus 0 \xrightarrow{F_1} o \xrightarrow{F_2} \cdots \xrightarrow{F_{l_{k_1-1}^0}} (0 \,\|\, \mathbb{R}_{k_1-1}^0) = EP_{k_1-1}^0 \tag{4}$$

$$SP_1^0 H(X+0) \oplus 1 \xrightarrow{F_i} o \xrightarrow{F_2} \cdots \xrightarrow{F_{l_1^0}} (1 \,\|\, \mathbb{R}_0^1) EP_0^1 \tag{5}$$

$$SP_1^1 = H(X+1) \oplus 1 \xrightarrow{F_1} o \xrightarrow{F_2} \cdots \xrightarrow{F_{l_1^1}} (1 \,\|\, \mathbb{R}_1^1) EP_1^1 \tag{6}$$

$$SP_{k_{1-1}}^0 = H(H+K_1-1) \oplus 1 \xrightarrow{F_1} o \xrightarrow{F_2} \cdots \xrightarrow{F_{l_{k_1-1}^1}} (0 \,\|\, \mathbb{R}_{k_1-1}^1) EP_{k-1}^1 \tag{7}$$

$$SP_0^{k_2-1} = H(X+0) \oplus (K_2-1) \xrightarrow{F_1} o \xrightarrow{F_2} \cdots \xrightarrow{F_{l_0^{k_2-1}}} (K_2-1 \,\|\, \mathbb{R}_0^{k_2-1}) + EP_0^{k_2-1} \tag{8}$$

$$SP_1^{k_2-1} = H(X+1) \oplus (K_2-1) \xrightarrow{F_1} o \xrightarrow{F_2} \cdots \xrightarrow{F_{l_1^{k_2-1}}} (K_2-1 \,\|\, \mathbb{R}_1^{k_2-1}) = EP_1^{k_2-1} \tag{9}$$

$$SP_{k_1-1}^{k_2-1} = H(K_1-1) \oplus (K_2-1) \xrightarrow{F_1} o \xrightarrow{F_2} \cdots \xrightarrow{F_{l_{k_1-1}^{k_2-1}}} (K_2-1 \,\|\, \mathbb{R}_{K_1-1}^{k_2-1}) = EP_{K_1-1}^{k_2-1} \tag{10}$$

Li attack's offline preprocessing stage utilizes $H(X+i)+j(1\le i\le k_1, 1\le j\le k_2)$ to determine the $k_1 * k_2$ initial positions. Assuming the longest possible chain length of t_{max} , the terminal nodes can be determined by applying the t_{max} iteration of the reduction function $R_i(1\le i\le t_{max})$. Chain computations terminate when either the chain length or the value of the current first k_2 distinct points surpasses t_{max+j}. This phase involves the storage of chain indices, lengths, and termination locations.

$$S[0,0]=\left\{\mathbb{R}_0^0, l_0^0, 0\right\} \qquad 0<l_0^0\le t_{\max} \qquad (11)$$

$$S[1,0]=\left\{\mathbb{R}_1^0, l_1^0, 1\right\} \qquad 0<l_1^0\le t_{\max} \qquad (12)$$

$$\vdots$$

$$S[i,j]=\left\{\mathbb{R}_i^j, l_i^j, i\right\} \qquad 0<l_i^j\le t_{\max} \qquad (13)$$

$$S\left[K_1-1, K_2-1\right]=\left\{\mathbb{R}_{K_1-1}^{k_2-1}, l_{K_1-1}^{k_2-1}, K_1-1\right\} \qquad 0<l_{K_1-1}^{k_2-1} t_{\max} \qquad (14)$$

The above equations show tuples stored in Li Attack. Length of the (i, j)-th chain is given by R_i^j, where R_i^j is the remainder of bits at the ends of the chain. As part of the sorting process, tuples are written into a binary file. The lengths of these chains vary widely and allow categorization into groups of similar sizes. The data for each class can be organized in k_2 tables. If two tuples in two separate tables both have the same R_i^j , the one with the smaller value will be dropped, therefore, the (i, j)-th chain's offset in the j-th table is likely to be i. During the online search phase, the hash value Y must be calculated in order to reveal the plaintext. Discover the set of values where the most significant k_2 bits of each endpoint agree with those of Y_0. In addition, a match is found if R_i^j in that table corresponds to the remaining nk$_2$ bits of Y_0 and R_i^j is less than or equal to t_{max}. From the first link in the chain, the original plain text may be reconstructed. If Y doesn't appear there, we apply $R_{t_{max}-1}$ and $F_{t_{max}}$ to it, yielding $Y \xrightarrow{R_{t_{max}-1}} Y_1 \xrightarrow{F_{t_{max}}} Y_0$, and see if Y_0 or Y_1 is a close enough match. If no match is found, proceed as in the rainbow table algorithm and try again. Since the table's data is already ordered, performing a binary search to locate a match requires less time. In contrast, unlike T-Y attack, the chain length of a Li attack based on differentiated points varies, leading to more false alarms and a slower web search.

6. Countermeasures of offline password attack

In this part, the two most effective techniques to protect individuals from an offline attack is explored. Adding salt to Hash is one of the most common countermeasures used in industry to thwart Rainbow Table attacks. Hash schemes include PBKDF2, which is implemented in IOS. In the context of file encryption, other countermeasures include Function Encryption, Search Encryption, and Homomorphic Encryption. CryptDB, as an amalgamation of some of the preceding encryption algorithms, is the most practical encrypted database because it strikes a good balance between data security and system efficiency, making it resistant to offline attacks and protecting database files from being compromised. First, we'll cover the topic of salt-related Hash schemes, and then we'll go through CryptDB, a secure database.

6.1. Salt-Related Hash schemes

6.1.1. Adding Salt to Hash (Seasoning Hash with Salt)

Adding salt to hashing is the most widely used defense mechanism against rainbow attacks since it prevents passwords from being cracked in bulk. If a hash value is recovered without employing salt, the identical passwords used by other users can be recovered via rainbow tables and dictionaries. Figure 4 depicts the procedure of salting hashing and authentication.

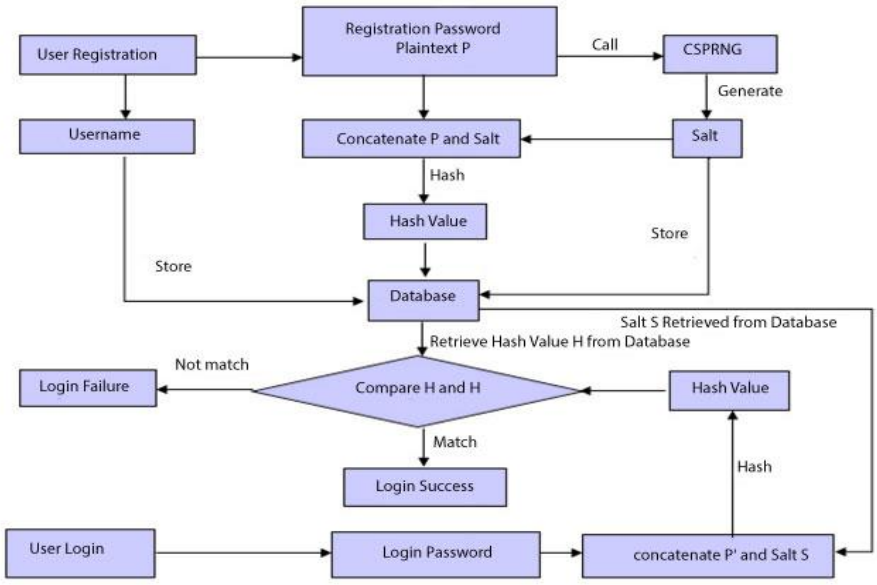

Figure 4. Adding Salt to Hashing

The server generates the user's registration password by hashing the plaintext of the password together with a salt. Instead of storing passwords in plaintext, databases record a salt and a hash value for each user. When a user logs in, the server first extracts their unique salt from the database, then combines the plaintext password for login with the salt before running it through a hashing process. The server then checks if the two hashes of the login and registration passwords match. In such a case, the login was successful. The password is never kept in plain text in the database at any stage of this procedure. Attackers that employ rainbow tables must first construct rainbow tables for each user, based on their passwords that have been salted randomly. Since each user's password has a unique hash value based on their salt, even if some passwords are recovered, the identical passwords used by other users will need to be cracked again. Figure 4 illustrates how making salt is crucial to the procedure. A rainbow table can quickly recover the hash value if the attacker knows the salt. A CSPRNG (Cryptographically Secure Pseudo-Random Number Generator) is required to generate a user-independent and random salt, such as java.security.SecureRandom. Using salted Hash to prevent the Rainbow Table attack is problematic for a few reasons. First, different salts should be used in different passwords. Second, the salt's length should be quite lengthy, making it prohibitively expensive to pre-calculate the Rainbow Table for all potential salts.

6.2. PBKDF2: Slow Hash Algorithm

Nevertheless, the risk of rainbow table password recovery in bulk can be considerably reduced by using a random salt. If an attacker has access to a large enough cluster of machines, they can retrieve a limited number of passwords using rainbow tables. PBKDF2 (Password-Based Key Derivation Function 2) is a key-stretching method that significantly raises the cost of an attack by mixing salt with the Hash algorithm. To enhance the expense of an attack, the slow Hash Algorithm based on PBKDF2 uses a large number of iterations in conjunction with salt. Figure 5 depicts the steps of the program.

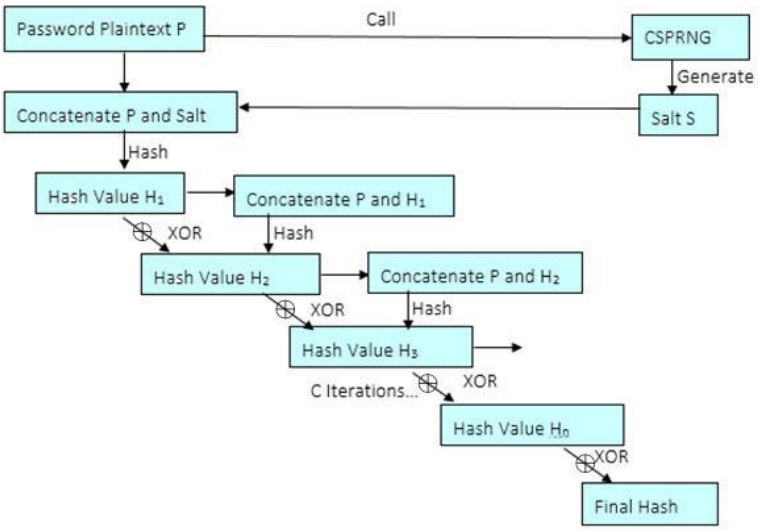

Figure 5. PBKDF2 Algorithm

The procedure starts with the plaintext of the password, adds a salt and applies the Hash function to generate a Hash value, and then repeats this process until the maximum number of iterations is reached. Notable among these is the reiteration count. C sets the bounds on how many times the PRF must be iterated until a sufficient master key is produced. C is what sets how long it takes to produce the password's key. Users can increase the security of their passwords by increasing the number of iterations beyond what they are comfortable with. The general public is advised to employ at least 1,000 iterations. Iteration times of up to 10,000 are possible for particularly vital passwords when access to powerful computing resources is available.

6.3. CryptDB: Encrypted Database

Encrypting files on databases to prevent data decryption is a crucial defense against password attacks because offline password attacks rely on password files. Encrypted database queries involving hundreds of operations using SQL might be difficult. File encryption systems like Privacy Homomorphic Encryption and Searchable Encryption have very high costs and make query operation exceedingly difficult, making it challenging to guarantee data confidentiality while maintaining the high performance of complicated SQL queries. High-performance Database Management System (DBMS) is necessary for industry applications to enhance the user experience. To achieve these goals, researchers at MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) created CryptDB in 2011 (Popa et al., 2012). This innovative database management system makes it possible to efficiently run SQL queries on encrypted data. CryptDB is a relational DBMS that offers verifiable security without requiring users to put their faith in the DBMS or the DBA. All data in the CryptDB system is encrypted, and the original DBMS may still execute SQL statements without needing to decode the data first. Initially, the SQL statement will be rebuilt at the front-end node. After that, the information contained in the statement will be encrypted and decrypted. The SQL statement's operators will be rewritten in some cases. The modified statement will be run immediately thereafter. CryptDB can function on untrusted DBMS since the database encryption key is only available to the front-end node and not the DBMS server.

There are three key problems that must be fixed before SQL operations can be carried out on DBMS servers housing encrypted data. To begin with, the system has to be able to process a larger number of SQL statements (which can include CRUD activities like insert, delete, update, and query). Secondly, there must be an assurance of safety. Thirdly, the completed system must be reasonably efficient and useful in everyday life. CryptDB employs the following methods to meet the demands of supporting SQL statement queries that involve many operations and strategies and necessitate both a high security level and great efficiency. Each and every SQL statement can be broken down into its component parts-equality tests, order comparisons, aggregates, and joins.

To secure the data in the database, CryptDB employs a SQL-aware encryption technique that allows for the aforementioned four procedures. CryptDB is efficient in large part because the majority of its encryption methods employ symmetric encryption. CryptDB employs the customizable query-based encryption architecture known as the onion layer model to provide the highest security level possible in light of the varying security levels of SQL-aware encryption schemes that support various functions. The encryption method that satisfies semantic security is at the outermost layer of the onion, while the scheme with a lower security level but a more critical function is closer to the onion's core. By implementing UDFs, DBMS may be used without any changes. In the first phase, the onion layer paradigm is used to encrypt data successively deeper into the system, with the last layer providing semantic security. To provide users with decrypted cipher text in response to their queries, CryptDB will perform the following procedures.

6.3.1. CryptDB system design

In the same way that SQL statements are executed on plaintext in conventional databases, the CryptDB technology enables the DBMS server to run encrypted text SQL queries. As can be seen in Figure 6, CryptDB is composed primarily of two parts: the client's trusted front end and the server's untrusted DBMS. Since the front end is aware of the overall database's architecture and the layer to which the encrypted data on the server belongs, it has access to the master key. CryptDB's encrypted data, encryption hierarchy, and a few supporting tables are accessible to the untrusted. To facilitate front-end operations on the servers' encrypted text, the servers also implement some UDFs tailored for CryptDB. The most recent onion layer and the MK (Master Key) encrypt the database structure, and the server additionally retains an encrypted state of the front end.

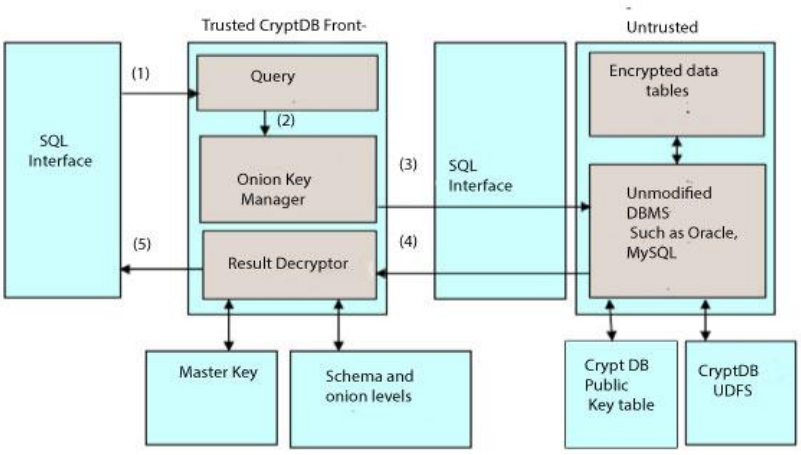

Figure 6. CryptDB System Structure

Figure 6 depicts the steps of a CryptDB query.

Step 1. The QRE (Query Rewriter/Encryptor) module receives the query request initiated by the user. Depending on the needs of the operation, QRE will encrypt the table and column names and use the master key to encrypt the constants in the SQL queries.

Step 2. OKM (Onion Key Manager) is a module that receives and processes queries. In order to decrypt the columns that need to be modified in the UPDATE statement, this module provides the corresponding onion key through analysis and then invokes UDF to carry out the operation.

Step 3. The OKM module sends the modified query statement to the untrusted back-end server so that it can be executed.

Step 4. The DBMS sends back the query's results.

Step 5. The RD (Result Decryptor) module is used to decrypt the ciphertext result before sending it back to the user.

6.3.2. SQL-Aware encryption

CryptDB utilizes several pre-existing encryption schemes plus a custom-built encryption primitive for the SQL JOIN operation to enable the execution of SQL commands on the ciphertext. This is because CryptDB performs the same action on all column rows using the same key. CryptDB derives its necessary security features from interchangeable encryption techniques. These are the algorithms used for encryption:

(i) RND (Random) Encryption

When compared to IND-CCA2, RND encryption is the most secure option. It has semantic security and is immune to adaptively selected plaintext attacks. Semantic information is lost, and the cipher text becomes indistinguishable from random numbers in the cipher text space. Once encrypted, two identical numbers look very different from one another. However, it is impossible to do accurate computations using RND-encrypted cipher text. For RND encryption, CryptDB employs the AES-CBC algorithm in its UFE mode.

(ii) Deterministic Encryption (DET)

When compared to RND encryption, DET encryption is less safe. Since the encrypted form of two identical plaintexts is the same, decrypting it reveals the original plaintext. The server can execute SELECT statements with restrictions such as COUNT, JOIN, DISTINCT, GROUP BY, etc., and perform equivalency tests. DET encryption can be implemented in several different ways; one example is:

$$DET_K(v) = RND_{K_1}(v) HMAC - SHA1_{K_2} \quad (15)$$

Where K_1 and K_2 are generated from K, and K is derived from the master key MK used by the table and column. The server checks for equality between values by comparing their HMAC-SHA1 hash values.

(iii) Order-Preserving Encryption (OPE)

OPE permits the order of plaintext to be stored in the cipher text without revealing any other information about the plaintext. If $x < y$, then for any key K, $OPE_K(x) < OPE_K(y)$. By using OPE to encrypt a column of data, the server can do range queries and operations like ORDER BY, MIN, MAX, and SORT.

(iv) Homomorphic Encryption (HOM)

HOM is secure up to an IND-CCA standard and can withstand an Adaptive Chosen Plaintext attack. The server can operate on the encrypted text in the same way it works on the plaintext. The front end receives the results of the calculations and decrypts them. The HOM of addition is more efficient, despite the fact that the HOM of most operations is too inefficient to be useful in practice. Paillier Encryption is a CryptDB feature that allows for HOM insertion (Paillier, 1999). Multiplying a ciphertext by itself yields the ciphertext of the plaintext sum in the Paillier method, as shown by the equation $HE_K(x) * HE_K(y) = HE_K(x+y)$. The server first invokes UDF to conduct Paillier multiplication on the additional homomorphic-encrypted columns, then consults the public key database of CryptDB to locate K, and finally computes the plaintext and associated cipher text. In addition, other operations like averaging a column's value or incrementing a column's value can be carried out with the help of the HOM of addition. The ciphertext of any plaintext is always 2048 bits in length, which is a drawback of the HOM of addition. When the HOM of addition is excessive, the cipher text space grows quite large (Shah et al., 2023; Ying & Thing, 2011).

(v) SEARCH (Search Encryption)

CryptDB employs a SEARCH encryption protocol that permits keyword search on ciphertext data (Amanatidis et al., 2007; Liu et al., 2023), which is necessary for the implementation of keyword search functions like LIKE in SQL. SEARCH can also find duplicates within the same column.

(vi) Join and OPE-JOIN (JOIN)

Since DET encryption employs unique keys for each column, a separate encryption layer is required to provide join operations between fields. The server can use the token received from the front end in conjunction with the JOIN operation to compare the values of two columns. The server can find pairs of equal values in distinct columns by using the JOIN operation, which also supports DET and SE.

6.3.3. Adjustable Query-Based encryption

The core of CryptDB is a dynamically configurable query-based encryption module that controls how much data is accessible in the untrusted server. Data is RND-encrypted if no query operation is required to be performed. Data will be DET-encrypted if the column will only be used for equivalency detection. However, the system must dynamically and adaptively select the encryption scheme in light of the possible query statements. CryptDB's onion encryption layer paradigm illustrated in Figure 7 is a means to this end. In onion encryption, one piece of data undergoes many transformations so that it can be queried in a variety of ways. The most secure forms of encryption, like RND and HOM, are located at the very edge of the network. OPE encryption is less secure than other methods, but it has more uses. Numbers can be stored in DET, OPE, and HOM onion databases that are generated by CryptDB. There is no need to make a HOM onion layer for string. CryptDB's front-end anonymizes the database's structure to stop the service from gleaning information about the database from things like table and column names.

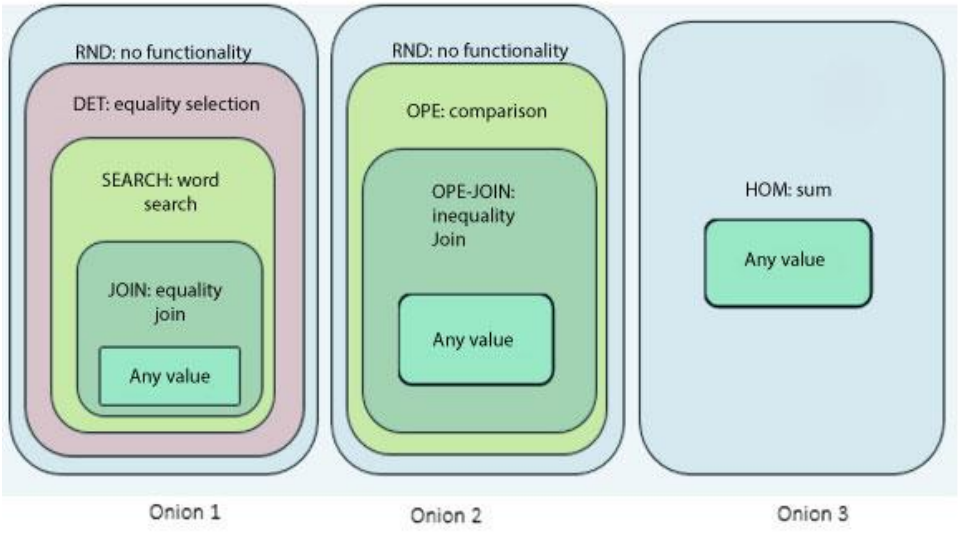

Figure 7. Onion layer model

The ciphertext data for CryptDB is displayed in Table 1. Each piece of information can be encrypted with as many as three onions (for integers) or two onions (for other data types).

Table 1. Plaintext and Cyphertext after Encryption

ID	Name	C1-Onion1	C1-Onion2	C1-Onion2	C2-Onion1	C2-Onion2
23	Alice	0x2b82ae	0xcb9e4	0x234e4	0x7d35a3	0xd101e3
		Plaintext		Ciphertext		

The front end encrypts the same columns across all levels of each onion using the same key. Separate columns, onion layers, and data tables using unique encryption keys. The master key MK is used as a starting point for deriving these encryption keys. $K_{t,c,l} = PRP_{MK}(\text{table } t, \text{column } c, \text{onion } o, \text{layer } l)$ where PRP is a pseudorandom permutation like AES, is an example of an encryption key for a data table, column, and onion encryption level. The

outermost layer of each onion is encrypted using the strongest encryption method. In Figure 7, we can see that RND encryption is used for the last layer of Onion1 and Onion2, while HOM encryption is used for the final layer of Onion3. Depending on the computing requirements, the Onion Key Management (OKM) Module decides whether to remove a layer of the onion from the front-end node when it gets a SQL statement from the user's application. If the associated encryption layer needs to be removed, the OKM module will communicate the necessary decryption key to the server, which will then remove the layer of encryption. Instead of altering the DBMS, CryptDB uses its own UDF to do the necessary onion-stripping (decryption) procedure. For instance, the OKM module will transmit the following operations to the server and cause the UDF DECRYPT_RND to be called on the server side to decode the cyphertext from the second column of Onion2 in Table 1 to the OPE encryption layer:

UPDATE tableName SET C2-Onion2= DECRYPT_RND (K, C2-Onion2)

6.3.4. Rewrite query execution

The front-end node will obfuscate, encode, and rewrite a SQL query before passing it to a potentially malicious DBMS for execution. To protect the identity of the user doing the query, the front end modifies the table and column names. The names of the columns in Onion1 in Figure 7 are then anonymized by the front end before the JOIN operation is implemented. When the query is complete, the constants will determine the encryption algorithm to use based on the layer of the onion where the matching column is stored in the DBMS. The existence of a database table called Students that contains only the identifier and the full name of each student is assumed. At startup, each column is encrypted using the onion model depicted in Figure 7. With RND or HOM as the final layer of encryption, even the server cannot deduce any information from the ciphertext data. The following query request is generated:

SELECT * FROM Students WHERE name=’Bob’

Since an equal join on the name column is necessary for the SQL query, it must be decrypted to the DET layer. The following SQL is executed by the front end to remove the layer:

UPDATE tableName SET C2-Onion1= DECRYPT_RND($K_{1,2,RND}$, C2-Onion1)

The following SQL is then run:

SELECT C1-Onion1, C2-Onion1 FROM tableName WHERE C2-Onion1= 0x7d35a3

Where 'Bob' is encrypted to 0x7d35a3 with key K1,2, DET. The front end can then decrypt the cyphertext after obtaining the result. Front-end employs the same approach to rewrite SQL queries like DELETE, INSERT, and UPDATE. The front end of an INSERT statement encrypts the value to be inserted according to the onion level of the column to which the data is being entered. There is no more action necessary for the DELETE statement. The UPDATE statement is executed in a manner analogous to that of the INSERT statement.

7. Conclusion

The digital landscape is continuously challenged by offline password attacks. Making successful security plans requires an understanding of the techniques used by attackers, such as dictionary attacks and rainbow table attacks. Implementing secure password hashing techniques, utilizing distinctive salts, and encouraging users to use strong passwords are all necessary for mitigating this issue. Fortifying password security and staying one step ahead of attackers are continual challenges as technology develops. Nevertheless, companies and people may strengthen their defenses against offline password attacks and shield critical data from unwanted access by taking preventative actions and making a commitment to cybersecurity best practices. Due to the exposure of massive volumes of data records as a result of database breaches, a large number of passwords have been cracked offline using the disclosed data in recent years. This research discusses the rainbow table attack, the most effective offline password attack method. In this research, two updated rainbow table attack strategies are compared and contrasted. Thing and

Ying's new time-memory tradeoff method using rainbow table sort (TY attack), and Li's upgraded strategy, which combines TY attack with differentiated points (Li attack). This article also summarizes the efficacy of hardware, password vaults, and file encryption in protecting against offline password attacks and discusses their limits.

Limitations

Rainbow tables have historically been successful at breaking passwords, but they have a number of drawbacks, such as the need for storage, susceptibility to salted passwords, decreased effectiveness against complex passwords, incompetence for continuously hashed passwords, resource-intensiveness, and susceptibility to changing security countermeasures. Attackers must therefore take into account these restrictions and modify their strategies when trying to hack passwords.

Future Work

The development of novel authentication systems, password security improvements, and user education will be the main areas of future effort in the field of offline password attacks. To develop a safer digital environment, addressing the constantly changing spectrum of cyber hazards demands a multidisciplinary strategy that includes technical breakthroughs, user interaction, and legal and ethical considerations.

REFERENCES

Almeshekah, M. H., Gutierrez, C. N., Atallah, M. J. & Spafford, E. H. (2015) ErsatzPasswords: Ending Password Cracking and Detecting Password Leakage. In *Proceedings of the 31st Annual Computer Security Applications Conference, 7-11 December, 2015, Los Angeles, CA, USA.*

Amanatidis, G., Boldyreva, A. & O'Neill, A. (2007) Provably-Secure Schemes for Basic Query Support in Outsourced Databases. In *Data and Applications Security XXI, 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security, 8-11 July, 2007 Redondo Beach, CA, USA.*

Aslam, N., Khan, I. U., Bader, S. A., Alansari, A., Alaqeel, L. A., Khormy, R. M., Alkubaish, Z. A. & Hussain, T. (2023) Explainable Classification Model for Android Malware Analysis Using API and Permission-Based Features. *Computers, Materials & Continua*. 76(3), 3167-3188. doi: 10.32604/cmc.2023.039721.

Avoine, G. & Carpent, X. (2017) Heterogeneous Rainbow Table Widths Provide Faster Cryptanalyses. In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2-*6 April, 2017, Abu Dhabi, United Arab Emirates.* pp. 815-822.

Avoine, G., Junod, P. & Oechslin, P. (2008) Characterization and Improvement of Time-Memory Trade-Off Based on Perfect Tables. *ACM Transactions on Information and System Security*. 11(4), 1-22. doi: 10.1145/1380564.1380565.

Blocki, J. & Sridhar, A. (2016) Client-CASH: Protecting Master Passwords against Offline Attacks. *In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, 30 May 2016-3 June 2016, Xi'an, China.*

Boneh, D. & Franklin, M. (2001) Identity-Based Encryption from the Weil Pairing. In: Kilian, J. (eds.) *Advances in Cryptology — CRYPTO 2001*, CRYPTO 2001. Lecture Notes in Computer Science, 2139. Springer. pp. 231-229.

Boneh, D., Di Crescenzo, G., Ostrovsky, R. & Persiano, G. (2004) Public Key Encryption with Keyword Search. In: Cachin, C., Camenisch, J.L. (eds.) *Advances in Cryptology - EUROCRYPT 2004. EUROCRYPT 2004. International Conference on the Theory and Applications of*

Cryptographic Techniques, Interlaken, Switzerland, 2-6 May, 2004. Lecture Notes in Computer Science, 3027. Springer. doi: 10.1007/978-3-540-24676-3_30.

Chatterjee, R., Bonneau, J., Juels, A. & Ristenpart, T. (2015) Cracking-Resistant Password Vaults Using Natural Language Encoders. In: *36th IEEE Symposium on Security and Privacy, 18-20 May, 2015, San Jose, CA, USA.*

Diffie, W. & Hellman, M. (1976) New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6), 644–654. doi: 10.1109/TIT.1976.1055638.

Fazal, W., Imran, K. et al. (2023) An investigation of cyber-attack impact on consumers' intention to purchase online. *Decision Analytics Journal*, 100297.

Goyal, V., Pandey, O., Sahai, A. & Waters, B. (2006) Attribute-based encryption for fine-grained access control of encrypted data. In: *Proceedings of the 13th ACM conference on Computer and communications security, 3 November, 2006, Virginia, Alexandria, USA.*

Hellman, M. E. (1980) A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*. 26(4), 401-406.

Kastrenakes, J. (2019) Facebook Stored Hundreds of Millions of Passwords in Plain Text. *The Verge.* https://www.theverge.com/2019/3/21/18275837/facebook-plain-text-password-storage-hundreds-millions-users [Accessed 20th January 2024].

Khan, H. U., Sohail, M., Nazir, S., Hussain, T., Shah, B. & Ali, F. (2023) Role of Authentication Factors in Fin-Tech Mobile Transaction Security. *Journal of Big Data*. 10(138), 1-37. doi: 10.1186/s40537-023-00807-3.

Li, Z., Lu, Y., Wang, W., Zhang, B. & Lin, D. (2012) A New Variant of Time Memory Trade-Off on the Improvement of Thing and Ying's Attack. In: Chim, T.W., Yuen, T.H. (eds.) *Information and Communications Security (ICICS) 2012*. Lecture Notes in Computer Science, 7618. Springer. pp. 311-320.

Liu, X., Zhao, Y., Xu, T., Wahab, F., Sun, Y. & Chen, C. (2023) Efficient False Positive Control Algorithms in Big Data Mining. *Applied Sciences*. 13(8), 5006. doi: 10.3390/app13085006.

Oechslin, P. (2003) Making a Faster Cryptanalytic Time-Memory Trade-Off. In *The 23rd Annual International Cryptology Conference, CRYPTO 2003, 17-21 August, 2003, Santa Barbara, California, USA.* Lecture Notes in Computer Science, 2729. pp. 617-630.

Paillier, P. (1999) Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *International conference on the theory and applications of cryptographic techniques, 2-6 May, 1999, Prague, Czech Republic.*Springer. pp. 223-238.

Popa, R. A., Redfield, C. M. S., Zeldovich, N. & Balakrishnan, H. (2012) CryptDB: processing queries on an encrypted database. *Communications of the ACM*. 55(9), 103-111. doi:10.1145/2330667.2330691.

Rivest, R. L., Adleman, L. & Dertouzos, M. (1978a) On data banks and privacy homomorphisms. *Foundations of Secure Computation*. 4(11), 169-180.

Rivest, R.L., Shamir, A. & Adleman, L. (1978b) A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*. 21(2), 120-126.

Sahai, A., Waters, B. (2005) Fuzzy Identity-Based Encryption. In: Cramer, R. (eds.) *Advances in Cryptology – EUROCRYPT 2005.* EUROCRYPT 2005. Lecture Notes in Computer Science, 3494. Springer. pp. 457-473. doi: 10.1007/11426639_27.

Shah, A., Ali, B., Wahab, F., Ullah, I., Amesho, K. T. T & Shafiq, M. (2023) Entropy-based grid approach for handling outliers: a case study to environmental monitoring data. *Environ Sci Pollut Res Int.* 30(60), 125138–125157. doi: 10.1007/s11356-023-26780-1.

Shannon, C. E. (1949) Communication Theory of Secrecy Systems. *Bell System Technical Journal.* 28(4), 656-715.

Sperotto, A., Sadre, R., de Boer, P. T. & Pras, A. (2009) Hidden Markov Model Modeling of SSH Brute-Force Attacks. In: Bartolini, C., Gaspary, L. P. (eds.). *Integrated Management of Systems, Services, Processes and People in IT. DSOM 2009*. Lecture Notes in Computer Science, 5841. Springer. pp. 164–176.

Stallings, W. (2011) *Cryptography and Internetwork Security: Principles and Practice*. Englewood Cliffs. New Jersey, Prentice Hall.

Stinson, D. R. (2005) *Cryptography: Theory and Practice*. Boca Raton, Florida, USA, CRC Pres (Chapman & Hall).

Thielman, S. (2016) Yahoo Hack: 1bn Accounts Compromised by Biggest Data Breach in History. *The Guardian.* https://www.theguardian.com/technology/2016/dec/14/yahoo-hack-security-of-one-billion-accounts-breached [Accessed 10th January 2024].

Thing, V. L. & Ying, H.-M. (2009) A Novel Time-Memory Trade-Off Method for Password Recovery. *Digital Investigation.* 6, S114-S120. doi: 10.1016/j.diin.2009.06.004.

Weir, M., Aggarwal, S., de Medeiros, B. & Glodek, B. (2009) Password Cracking Using Probabilistic Context-Free Grammars. In: *2009 30th IEEE Symposium on Security and Privacy, 17-20 May 2009, Oakland, CA, USA*. IEEE Computer Society, NW Washington, DC, United States. pp. 391-405. doi: 10.1109/SP.2009.8.

Ying, H.-M. & Thing, V. L. L. (2011) A Novel Rainbow Table Sorting Method. In CYBERLAWS 2011: *The Second International Conference on Technical and Legal Aspects of the e-Society, 23-28 February, 2011, Gosier, Guadeloupe, France.*

Yu, F. & Huang, Y. (2015) An Overview of Study of Password Cracking. *In: 2015 International Conference on Computer Science and Mechanical Automation (CSMA), 23-25 October, 2015, Hangzhou, China.* pp. 25-29.

Fazal WAHAB received his MS degree in the discipline of Software Engineering from the Software Collage, Northeastern University, Shenyang China in 2020. He is currently a Ph.D. scholar in the Department of Computer Science and Technology at the same university. His research interests include Cybersecurity (IoT, MANET), Deep Learning, Machine Learning and Computer Networking. He has authored and co-authored many research articles in reputed journals.

Imran KHAN is currently a Computer Science lecturer in the Department of Informatics at the University of Sussex. He received his MSc degree from Liverpool JMU and a Ph.D. in Information Systems (specializing in knowledge-sharing systems) from the University of the West of Scotland. His research interests include Computer Security, Machine Learning, Blockchain technology, and Cloud Computing.

Ken SI received his master's degree from the University of Sussex in the discipline of informatics. His research interest includes network and information security and computer networking.

* * *