Black-Box Testing of Cryptographic Algorithms Based on Data Characteristics

Haoling Fan[1,2,3], Lingjia Meng[1,2,3], Fangyu Zheng[1,2,3(✉)], Mingyu Wang[1,2,3], and Bowen Xu[1,2,3]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
{fanhaoling,menglingjia,zhengfangyu,wangmingyu,xubowen}@iie.ac.cn
[2] Data Assurance and Communications Security Center, Chinese Academy of Sciences, Beijing 100093, China
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract. Serving communications security, identity authentication, etc., cryptographic algorithms constitute the cornerstone of cyberspace security. During the past decades, cryptanalysts have proved that many once prevailing cryptographic algorithms (e.g., MD4, MD5, 3DES, RC4) are no longer secure now. However, insecure cryptographic algorithms are still widely deployed in practice, seriously endangering the security of cyberspace. The reasons for this dilemma are many-fold, one of which is difficult to detect the algorithms used in the legacy binaries. Most of the existing detecting methods of cryptographic algorithms, either require source code analysis (i.e., white-box testing) or depend on the dynamic execution information (i.e., dynamic testing), narrowing the testing scope where the source codes of commercial software are not provided and the running environment may be difficult to deploy. In this paper, we propose a method of static black-box testing of cryptographic algorithms, which can identify a specific algorithm based on the corresponding data characteristics. We have implemented the testing method and used it to check 150 binaries of three types, including cryptographic libraries, commonly-used programs that use cryptographic algorithms, and general-purpose Github projects without cryptographic algorithms. The empirical results demonstrate that 80.6% of the insecure cryptographic algorithm are implemented in the test files that contain the cryptographic algorithms. The false negative rate and false positive rate were 2.10% and 1.68% using our method. Moreover, we found that the insecure cryptographic algorithms (i.e., MD4, SHA-1) is still exist in some popular software, e.g., MbedTLS and 7-Zip.

Keywords: Black-box testing · Data characteristics · Algorithm identification

This work is supported in part by the National Natural Science Foundation of China No. 61902392 and CCF-Tencent Open Fund under Grant RAGR20210131. The corresponding author is Fangyu Zheng.

J. Lin and Q. Tang (Eds.): AC3 2022, LNICST 448, pp. 153–169, 2022.
https://doi.org/10.1007/978-3-031-17081-2_10

1 Introduction

The proliferation of the Internet has given rise to remote conferences, e-commerce, especially after the pandemic of COVID-19. Cryptography plays a vital role in cyberspace security, ensuring the confidentiality and the integrity of information, preventing information from being tampered and forged.

Built upon computational complexity theory, the practical cryptographic algorithms are not absolutely-secure.

We consider the reasons for the insecurity of cryptographic algorithms, including short keys, back doors, collision in hash algorithms, and so on. The widely-used insecure cryptographic algorithms are shown in Table 1.

Table 1. Insecure cryptographic algorithms

Cryptographic algorithm	Insecure reference
MD2	MD2 was obsoleted in RFC 6149 [24]
MD4	In 2011, RFC 6150 stated that RFC 1320 (MD4) is historic (obsoleted) [22]
MD5	Approved the information RFC 6151 to update the security considerations in MD5 [25]
SHA-1	NIST formally deprecated the use of SHA-1 in 2011 [23]
DES/3DES	NIST has deprecated DES and 3DES for new applications in 2017, and for all application by 2023 [18]
RC2	It is vulnerable to brute force attacks, the 40-bit cryptographic algorithm is outdated [11]
RC4	RC4 was banned from all versions of TLS in 2015 by RFC7465 [16]
RSA[a]	Insecure padding mode and insufficient key length [5]
ECC/DH	Based on unsafe elliptic curves [7]
Dual_EC_DRBG	NIST removed Dual_EC_DRBG from the pseudorandom number generator algorithm standard [21]

[a] When we use black-box testing tool to identify RSA as an insecure cryptographic algorithm, we can only detect the existence of a specific implementation of RSA, and further analysis is required for its padding mode and key length.

Any insecure cryptographic algorithm would seriously affect the overall security of the system. However, out of our expectation, such great risks do not seem to raise worldwide attention. Disabled cryptographic algorithms are still used: 25% of websites in 2011 are still using insecure MD5 [1], and 2015 statistics show that about 30% of network communications are still protected by disabled RC4 [2]. Outdated cryptographic algorithms in cryptographic libraries are abused, Lazar et al. found that 83% of cryptographic vulnerabilities are caused by misuse of cryptographic vaults [14], and Egele et al. found that about 88% of mobile applications (sampled from Google Play) involve a considerable degree of misuse of the cryptographic library [8]. Detection is the first step required to address the issue of insecure cryptographic algorithms and implementations. A straightforward method

for cryptographic algorithm identification is code review. The manual detection method of cryptographic algorithm is time-consuming and laborious, and requires personnel with professional skills. Meanwhile, some software can not release source codes, and thus the code review method cannot apply. In order to detect cryptographic algorithms of closed-source software, researchers perform binary file detection, including dynamic analysis and static analysis. Static methods, e.g., Cipher hunting [10], CryptoGuard [20], CrySL [13], do not execute the program but require disassembly or decompilation of the file under test to check for the existence of an implementation of a cryptographic algorithm. The advantage of static analysis is that the program does not required actual executions. However, static analysis may fail to trigger some cryptographic algorithms in the code, resulting in a higher false negative rate and false positive rate. The methods of dynamic analysis include K-Hunt [15], Reformat [26], SMV-Hunter [9]. Dynamic analysis requires an encryption algorithm to be triggered at runtime for detection, and dynamic detection is relatively time-consuming.

In order to solve the problems of the existing detection methods, in this paper, we propose a cryptographic algorithm detection method that does not require open source or an execution environment, has a low false negative rate and false positive rate, and is highly efficient. According to this method, we have implemented a black-box testing tool for cryptographic algorithms based on characteristic data. This paper makes the following main contributions:

- **Characteristic data classification.** We perform artificial data characteristic extraction on the standard of cryptographic algorithms and implementations in common libraries, and then classify them, including a total of 10 categories (the categories on the right in Fig. 2) for the subsequent identification of cryptographic algorithms (Sect. 2).
- **Set properties for characteristic data.** 1) We set the **Weight** for the characteristic data to reduce the false negative rate. The **Weight** setting is based on three principles: *Length* - whether the length of the characteristic data is greater than 2 bytes; *Uniqueness* - whether the characteristic data is only used by one cryptographic algorithm; *Activeness* - whether the characteristic data participates in the calculation. 2) The **List** we set we set uses the relationship among different characteristic data as the basis for identification to reduce the false positive rate. The relationship of characteristic data includes two types of positional relationships: a set of characteristic data is in continuous position; a set of characteristic data occurs cyclically at the same interval (Sect. 3.2).
- **Implemented a cryptographic algorithm black-box testing tool.** We provide a method for black-box detection of cryptographic algorithms based on characteristic data, and then we make a tool based on this method. The tool includes a characteristic database (Sect. 3.2), a matching strategy module (Sect. 3.3), a decision module (Sect. 3.4), and a result output module.
- **Evaluation of black-box testing tool.** We evaluate the test tool by testing 150 binary executables of different sizes (1KB-4GB), including 23 popular cryptographic libraries, 8 cryptography-related programs, and 119 cryptography-independent programs. The final false negative rate is 2.10%, and the false positive rate is 1.68%.

The rest of the paper is organized as follows. Section 2 discusses the pre-required knowledge. In Sect. 3, we designed and implemented a black-box testing tool based on characteristic data. In Sect. 4, we evaluated the tool. Section 5 summarizes this paper.

2 Preliminaries

The data characteristics of cryptographic algorithms drive the cryptographic algorithm detection proposed in this paper. Due to the particularity of cryptographic algorithms, many cryptographic algorithms use special constants in the process of data processing. We divided the characteristic data into 10 *categories* **(C1–C10)**.

The characteristic data of these three cryptographic algorithms are described below.

Hash Algorithms map an arbitrary finite-length input to a fixed-length output. The characteristic data of this type of cryptographic algorithms include *initialization vectors* **(C1)**, *round constants* **(C2)** and *logical operation function constants* **(C3)**. When implemented, *initialization vectors* are used to initialize the parameters in the hash calculation process, the *round constants* and *logical operation function constants* participate in the transformation of each round of encryption.

Symmetric Cryptographic Algorithms transform a data-to-be-encrypted into ciphertext under the control of the encryption key. *S-box* **(C4)**, *permutation table* **(C5)** and *lookup table* **(C6)** for fast implementation of encryption are used in the implementation of symmetric cryptographic algorithms. These characteristic data are all used for data substitution, and are generally defined as a static array in software implementation. Symmetric cryptographic algorithms use these data in the implementation to increase the degree of confusion in the encryption process.

Asymmetric Cryptographic Algorithms in real-world applications are mainly divided into two categories: one is based on large integer factorization, such as RSA; The other is based on discrete logarithm problems, such as elliptic curve cryptography (ECC). For asymmetric cryptographic algorithms, we mainly focus on three aspects: 1) Diffie-Hellman key exchange algorithm. Through the study of its principle, we noticed that it uses *preset prime values* **(C7)**, and these prime numbers can be used as its characteristic data. 2) ECC. We found that all the standards and implementations will use the recommended *elliptic curve related parameters* **(C8)**, so the elliptic curve related parameters are used as the characteristic data of the elliptic curve. 3) *RSA*. Since the principle of RSA is based on the factorization of large integers, we found that it will generate many large prime numbers when it is implemented, and the code that generates large prime numbers often contains *a table of small prime numbers* **(C9)**, and small prime numbers will appear in the data segment, we use the small prime number table as the characteristic data.

The pseudo-random number generator is also a class of algorithms that we need to detect, in which the implementation of Dual_EC_DRBG is based on an elliptic curve, and *elliptic curve related parameters.* **(C8)** can be used as characteristic data.

In addition to the above characteristic data, the *OID* (Object Identifier) **(C10)** of the cryptographic algorithm itself can also play an auxiliary role in the identification process of the cryptographic algorithm. We also use OID as a type of characteristic data.

3 Design and Implementation

We aim to achieve a fast, accurate, and comprehensive black-box testing tool for cryptographic algorithms. And the detection is mainly based on the characteristic data of cryptographic algorithms in binary executables. Therefore, it can support cryptographic algorithm detection for closed-source software without running binary executables. This section describes the design goals, outlines the black-box testing tool, and then introduces the design details of the characteristic data set, matching strategy, and determination method.

3.1 Overview

We design the architecture of a black-box testing tool for cryptographic algorithms based on characteristic data, as shown in Fig. 1. The tool follows a modular design, which can reduce the complexity of the tool design and ensure the relative independence between modules, which is convenient for development and modification. The black-box testing tool consists of three modules: I) characteristic data extraction and data set construction module; II) characteristic data matching module; III) cryptographic algorithm determination and output module. Figure 1 outlines the workflow of the tool as follows:

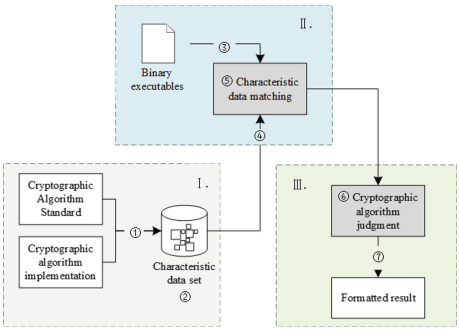

Fig. 1. Cryptographic algorithm black-box testing tool architecture

3.2 Characteristic Data Extraction and Data Set Construction

Characteristic data needs to be extracted and analyzed before using in cryptographic algorithm detection. The process of characteristic data extraction and data set construction is shown in Fig. 2.

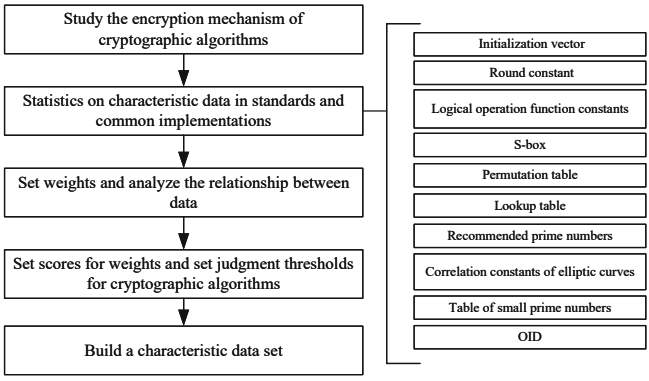

Fig. 2. Characteristic data set construction process

Characteristic Data Extraction The characteristics data of the cryptographic algorithms are the key to black-box testing. The selected characteristic data should be representative and remain unchanged when the environment changes. We researched and analyzed the implementation of cryptographic algorithms in standards and cryptographic libraries. It was discovered that cryptographic algorithms perform special operations on data, such as substitution and obfuscation. When performing special operations on data, the algorithm uses some specific constants. We use the specific constants used in the special data processing as the characteristic data of the cryptographic algorithm.

It's also worth noting that we not only analyze characteristic data from the standards of cryptographic algorithms, but also the popular cryptographic libraries. We have investigated the implementation of cryptographic algorithms in cryptographic algorithm libraries such as MbedTLS [19], OpenSSL [27], Botan [17], Crypto++ [6], libgcrypt [12], etc. We extract the characteristic data of the cryptographic algorithm implementations in the above cryptographic algorithm library.

Characteristic Data Set Construction After collecting the raw characteristic data, we need to process it to constitute the characteristic data set directly used by our tool. We set attributes for each cryptographic algorithm and each characteristic data. The meaning of each attribute is: **Algorithm** refers to the name of the cryptographic algorithm; **DataNum** refers to the total number of characteristic data of the cryptographic algorithm; **DataName** refers to the

name of the characteristic data; **Weight** refers to the artificially set characteristic data weight; **List** is an attribute set for characteristic data with associated relationship. The associated relationship refers to the storage location of characteristic data that occurs next to each other or at cyclic intervals. Characteristic data with associated relationships has the same List value, represented as a group. The List value of characteristic data that is not associated with any characteristic data is 0; **Data** is the value of characteristic data.

For the characteristic data of the cryptographic algorithms, we set the weight and the association relationship, each weight corresponds to a score, and calculated the threshold of each cryptographic algorithm through the weight value.

Weight. After extracting the characteristic data, we made statistics on the characteristic data in different cryptographic algorithms and the length of the characteristic data. We found that multiple cryptographic algorithms use some characteristic data and some characteristic data is too short to act as a piece of strong evidence. Therefore, we set the individual weight for each characteristic data. We set the weights based on three principles: 1) Length. Whether the length of the characteristic data is greater than 2 bytes; 2) Uniqueness. Whether characteristic data is used by only one cryptographic algorithm; 3) Activeness. Whether the characteristic data is involved in the calculation. The characteristic weight information is initially determined manually and is divided into three levels: *high (H), medium (M),* and *low (L).*

- **High weight:** We specify that the characteristic data used by only one cryptographic algorithm has a high weight, that is, other cryptographic algorithms do not use the characteristic data, and the length of the characteristic data is greater than 2 bytes, such as the S-box of DES.
- **Medium weight:** Our research found that different cryptographic algorithms may use the same characteristic data. We set this shared characteristic data as medium weight. And the length of the characteristic data of the medium weight is greater than 2 bytes. For example, SHA-1 and MD5 have the same initialization vector, this initialization vector is medium-weight characteristic data in both cryptographic algorithms.
- **Low weight:** We experimented on the length of the characteristic data. The empirical result showed that the characteristic data with a length of one or two bytes has a great probability of appearing in the cryptography-irrelevant part of the binary executables, so we set this type as low weight. At the same time, the experiment also found that although the OID of the cryptographic algorithm can uniquely identify the cryptographic algorithm, there is a high possibility that the binary executable contains the data value of OID without the existence of the corresponding cryptographic algorithm, which leads to false positive. OID does not participate in the calculation. Therefore, OID only assists the testing of the cryptographic algorithm, and we set the OID to low weight.

Association Relationship. We noticed that some characteristic data has an association relationship. There are two types of association relationships here: one

is that a group of characteristic data positions appear adjacently, for example, a group of S-boxes in DES are stored in the data segment adjacently; the other is that characteristic data appears cyclically, such as the round constant in the hash algorithm, because it requires multiple rounds of calculations, so it appears regularly in the code segment of the binary executable. We set the attribute *List* for the characteristic data of a cryptographic algorithm. The 0 value of *List* indicating that the characteristic data has no association relationship with other characteristic data, We set the *List* values to the same non-zero value for characteristic data with an associative relationship.

Score and Threshold. To facilitate the subsequent determination of the cryptographic algorithm, we assign scores based on each characteristic data's weight and association relationship and calculate the cryptographic algorithm determination threshold. In different cryptographic algorithms, the scores of each weight are different, we calculate the corresponding scores for the three weights of each cryptographic algorithm according to the following calculation principles:

- The score of high-weight (H) data is greater than that of low-weight data (M, L).
- The total score of middle-weight (M) data in an algorithm must not exceed the value of high-weight (H) data.
- The total score of the low-weight data (L) shall not exceed the score of medium-weight data (M).

The calculation formula of each weight score (S_H, S_M, S_L) is shown in formula (1), where N_H, N_M, N_L are the number of characteristic data with weights H, M, and L in a cryptographic algorithm, and n is the preset score of the H weight.

$$S_H = n; S_M = \lfloor n/(N_M + 1) \rfloor; S_L = \lfloor S_M/(N_L + 1) \rfloor; \tag{1}$$

The weights M and L scores are calculated according to the above regulations and the number of characteristic data contained in M and L in the algorithm. The weight of the characteristic data is assigned a specific score. For each cryptographic algorithm, the threshold is calculated based on the weight of its characteristic data. The weight scores of the characteristic data are sequentially accumulated as the threshold. T is the threshold, and its calculation formula is shown in formula (2).

$$T = S_H * N_H + S_M * N_M + S_L * N_L \tag{2}$$

The two most important parameters are the score of weight and threshold information of each characteristic data. The scores and threshold information are set manually. They are optimized through training to reduce the false negative rate and false positive rate during the testing process.

3.3 Characteristic Data Matching Strategy

The characteristic matching strategy is used to query the characteristic of the cryptographic algorithm in the target binary executable. The matching strategy consists of three parts: 1) binary executable information's extraction, including file format, machine word length and byte order, record position offset and size information of data segment, read-only data segment, and code segment; 2) characteristic data conversion, we will convert the byte order according to different data types before matching. In the matching module, different types of feature data are used as input in turn; 3) result record, we take the binary executable to be tested and the characteristic dataset as the input object of the matching strategy, and finally record the matching result.

3.4 Cryptographic Algorithm Determination

This module analyzes the characteristic information that is successfully matched under the characteristic matching logic and determines the existence of the cryptographic algorithm in the tested object by processing the weight and association relationship of the characteristic information. The cryptographic algorithm decision logic is responsible for processing the matched characteristics to verify the existence of specific cryptographic algorithm implementation.

Characteristic Data Relevance Detection. For characteristic data with an association relationship, the successful matching of a single characteristic data is often more likely to cause false positives. By detecting whether all the characteristic data that has an association relationship appears or not, assists in deciding whether the tested binary executable contains a cryptographic algorithm implementation. Suppose each data in a group of data with an association relationship is close to the position in the data segment or periodically appears in the code segment. In that case, it proves that the group of data conforms to the attribute of the association relationship.

Characteristic Data Matching Result Statistics. We design the score calculation to determine the cryptographic algorithm as a reward system. We score each cryptographic algorithm, based on the weight of the successfully matched entry and the existence of the association relationship. Subsequently, we compare the score with the threshold to determine the probability of the existence of the cryptographic algorithm implementation. Finally, we show the cryptographic algorithms supported by the tested file in the form of a list. The output result also includes the matching situation of each characteristic data.

4 Evaluation

To evaluate the effectiveness and efficiency of the proposed method, we employ the tool against various binary executables, which are of different types and sizes.

4.1 Experiments Setup

The evaluation was conducted on a ThinkCentre M920t PC with Intel i7-9700 CPU (3.00 GHz) and 8 GB RAM. The operating system is Ubuntu 16.04 LTS. The binary executable formats tested include PE and ELF. Binary executables come from Intel and ARM processors, and our tool supports detection of binaries in both 32-bit and 64-bit operating systems. Test files include popular crypto libraries, binaries with cryptography, and binaries without cryptography. The source of the test file is shown in the Table 2.

Table 2. File source for false negative testing

Popular crypto libraries	Binaries with cryptography	Binaries w/o cryptography
openssl-0.9.8 openssl-1.0.2u openssl-1.1.0l openssl-1.1.1c openssl-1.1.1k mbedtls-2.1.0 Botan-2.15.0 libgcrypt-20.2.6 crypto++-8.2	7z RAR KeePass ultrasurf	Files in this category are from 58 projects that are easy to obtain executable binary executables labeled by "tool", "release", "exe" and "C++" on GitHub according to the time sequence. See Appendix B for a detailed list

4.2 Effectiveness

We first recorded the cryptographic algorithm support of the test files before the test, then used the black-box testing tools to test the cryptographic algorithms on these test files, and compare and analyze the results with the actual conditions recorded. We evaluate the effectiveness of the tool under two important metrics: false negative rate and false positive rate.

False Negative Rate. Through black-box testing, we detect which cryptographic algorithms are in binaries with cryptography and prevalent cryptographic libraries.

We first count all the cryptographic algorithms in the test file. The total number of cryptographic algorithms is recorded as T. After the test, we add up the number of false negative cryptographic algorithms in each file to be tested. The number is recorded as t. The formula for calculating false negative rate is t/T. In our tests, the false negative rate is 2.10%. The false negatives mainly come from the following aspects:

– *Assembly code invalidates characteristic data.* We discover that if there is assembly code in the implementation of cryptographic algorithms the characteristic data would not work. We found assembly code implementations of Cast, AES and Camellia in OpenSSL.

- *Characteristic data generated at runtime cannot be detected.* The characteristic data will be generated only when the cryptographic algorithm is used. The detection of RC4 failed for this reason.
- *No characteristic data in the implementation of cryptographic algorithms.* Implementing some cryptographic algorithms does not require characteristic data in the software code form. The hardware implementation of AES in Botan does not require characteristic data and thus cannot be detected using our method.
- *Characteristic data detection capabilities are limited.* For RSA, we select a small prime table (used for key generation and prime detection) and OID as characteristic data. Still, OID can only play a supporting role, and other algorithms also use the small prime number table.

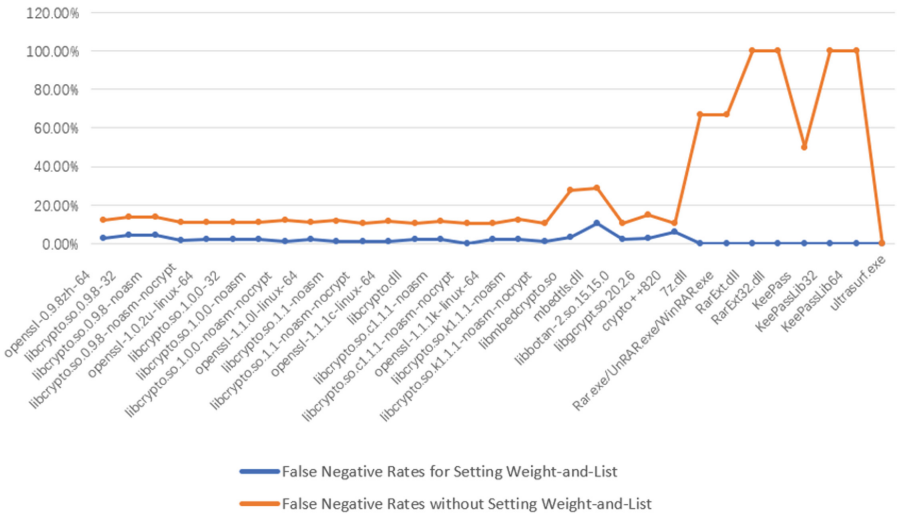

Fig. 3. Influence of weight-and-list on false negative rate

We set the *Weight-and-List* attribute for the characteristic data, Fig. 3 shows the comparison of false negative rates for setting *Weight-and-List* or not. We conclude that *Weight-and-List* is indeed effective in reducing the false negative rate. Multiple cryptographic algorithms use the characteristic data and the length of the characteristic data is different. After setting the *Weight-and-List* attribute for the data, the contribution of the characteristic data to the detection and the correlation between the characteristic data are taken into account, and the alarm leakage rate decreases.

False Positive Rate. We conduct black-box testing on binaries without cryptography and calculate the false positive rate based on whether a cryptographic algorithm is detected.

We use files that do not contain cryptographic algorithms for testing, the number of files to be tested is F, after the test, the number of files with cryptographic algorithms detected is f, and the formula for calculating the false alarm rate is f/F. In our tests, the false positive rate was 1.68%. False positive algorithms include RC4 and XTEA, and the reasons are as follows: 1) The characteristic data of RC4 is a numeric array from 0 to 255, which is commonly used and thus, easily collided with other regular valuables; 2) XTEA uses a constant δ as characteristic data, which is defined as $\delta = (\sqrt{5} - 1)2^{31}$ (0x9E3779B9). In addition to being used in XTEA, δ is often used in mathematical calculations, so it is easy to cause false positives.

4.3 Performance

The black-box testing tool only matches the characteristic data of the binary file data segment, read-only data segment, and code segment during detection, and performs page-by-page detection during implementation. By narrowing the matching range of characteristic data, the detection time is reduced and improved performance. Table 3 illustrates the time-consuming comparison of segment matching and full matching for binary executable files of different sizes. For larger files to be tested, as the size of the test file increases, the test time is shortened while ensuring the accuracy of the test results.

Table 3. Single file detection performance

File size	Segment match time	Full match time	STR[a]
3 KB	0.71 s	0.70 s	0.99
10 KB	0.72 s	0.71 s	0.99
100 KB	1.33 s	1.31 s	0.98
1 MB	7.61 s	7.82 s	1.03
10 MB	64.71 s	67.41 s	1.04
80 MB	451.79 s	490.83 s	1.08

[a]STR refers to the time-consuming ratio of detection under different methods for a single file, that is, the ratio of the time consumed by full matching to the time consumed by segment matching.

At the same time, the method of narrowing the matching range to the segment also has a significant performance improvement in batch detection. Table 4 shows the time-consuming comparison between segment and full matching for batch testing of different sizes.

Table 4. Batch file detection performance

Number	Test folder size	Number of files	Time for matching segment[a]	Time for matching total file	BTR[b]
1	10 MB	91	64.55 s	64.68 s	1.002
2	100 MB	109	394.89 s	400.03 s	1.013
3	200 MB	472	1093.93 s	1117.59 s	1.022
4	500 MB	630	3420.35 s	3550.83 s	1.038
5	1 GB	1734	6632.59 s	6902.62 s	1.041
6	2 GB	3468	13166.64 s	14206.78 s	1.079

a. Segments here refer to data segments, read-only data segments, and code segments in binary executables.
b. BTR refers to the time-consuming ratio of detection under different methods for batch file, that is, the ratio of the time consumed by full matching to the time consumed by segment matching.

4.4 Limitations

Although there are already many cryptographic algorithms that can be detected by our tool, there are also limitations. First, our tool does not support the accurate identification of the following cryptographic algorithms.

- The characteristic data is not applied in some implementations. For example, in the AES-NI implementation and the white-box implementation of AES [4] and SM4 [3], S-box no longer exists.
- Other algorithms completely share the characteristic data, for example, characteristic data of XTEA is fully shared by SEED, characteristic data of MD4 is fully shared by SHA-1.
- The characteristic data is generated at runtime, such as RC4.

In addition, for some algorithms, the tool is temporarily unable to distinguish its key length and working mode based on characteristic data. For example, we cannot distinguish between RSA-1024 and RSA-2048; we cannot recognize AES's CBC and CTR working modes. At last, if a shell processes the tested object, the characteristic data of the cryptographic algorithm will not be found. The location of the read-only data segment of the data segment code segment cannot be determined, as a result, our tool cannot be used.

5 Summary

In this paper, we propose and implement a black-box testing tool for cryptographic algorithms based on characteristic data. Characteristic data is extracted from the standard and general implementation of cryptographic algorithms. In particular, we set Weight and List for characteristic data and set a threshold for

cryptographic algorithm determination to reduce the false positive and false negative rates. The tool can detect 15 symmetric cryptographic algorithms, 13 hash algorithms, RSA, 8 Diffie-Hellman key exchange algorithm prime numbers, 85 elliptic curves, and 1 pseudo-random number generator. The evaluation results show that the tool can accurately, comprehensively, and quickly detect cryptographic algorithms in binaries and supports the detection of non-open source software. In the future, we will train the Weights and thresholds to obtain a lower false positive rate and false negative rate Table 5.

A Cryptographic Algorithm Supports

The cryptographic algorithms supported by the cryptographic algorithm black-box testing tool include: 13 cryptographic hash algorithms, 15 symmetric cryptographic algorithms, 1 asymmetric cryptographic algorithm, 85 elliptic curves, 8 prime domains of Diffie-Hellman key exchange algorithm, three types of double elliptic curve pseudo-random number generators based on elliptic curves.

Table 5. Cryptographic Algorithm Detection Support

Category	Name			
Cryptographic hash algorithm	SHA-1	MD5	SHA-384/512	RIPEMD-160
	MD4	SHA-224/256	SHA-3	Blake2
	SM3	MD2	Siphash	
Symmetric cryptographic algorithm	AES	RC4	XTEA	Camellia
	RC2	ChaCha20	Cast	SEED
	ARIA	Blowfish	SM4	Gost
	DES	RC5	Seal	
RSA	RSA			
ECC	brainpoolP-(160/192/224/256/320/384/512)-(r/t)1			
	c2tnb-(163v1/163v2/163v3/176v1/191v1/191v2/191v3/208w1)			
	c2tnb-(239v1/239v2/239v3/272w1/304w1/359v1/368w1/431r1)			
	sect-(113r1/113r2/131r1/131r2/163k1/163r1/163r2)			
	sect-(233k1/233r1/239k1/283k1/238r1/409k1/409r1/571k1/571r1)			
	secp-(112r1/112r2/128r1/128r2/160k1/160r1/160r2)			
	secp192k1/192r1/224r1/224k1/256k1/256r1/384r1/521r1			
	wap-wsg-idm-ecid-wtls-(8/9/12)			
	Oakley-EC2N-(3/4)	prime192v-(1/2/3)	wapip192v1	M-(221/383/511)
	sm2p256v1	prime239v-(1/2/3)	gost-(256/512)	Curve-(448/25519)
	sm9bn256v1	prime256v1	frp256v1	
Diffie-Hellman key exchange algorithm	DH-dh-(1024_160/2048_224/2048_224)			
	DH-ffdhe-(2048/3072/4096/6144/8192)			
Dual_EC_DRBG	Dual_EC_DRBG-x9_62_prime_256v1		Dual_EC_DRBG-nist_prime_-(384/521)	

B Test File Information

When we tested the false positive rate, we used"tool", "release" and "exe" as search keywords on Github, sorted the search results by year, and selected 58 projects that are not related to cryptographic algorithms for compilation to get the test file. Github project information is shown in the Table 6.

Table 6. Github project for testing the false positive rate

No	GitHub link	No	GitHub link
1	GH:/TonyChen56/HackerTools.git	30	GH:/v-star0719/MFC_ImageSlider.git
2	GH:/christopher5106/FastAnnotationTool.git	31	GH:/JoshCodesJava/Platform-Game.git
3	GH:/vedderb/bldc-tool.git	32	GH:/bjk12/LittleBird_TypeExercise.git
4	GH:/yasserhcn/MinCPT.git	33	GH:/zwang452/MGAFeedAssayProcessor.git
5	GH:/jptr218/bgp_hijack.git	34	GH:/L3cr0f/DccwBypassUAC.git
6	GH:/yoursunny/ndn6-tools.git	35	GH:/yanncam/exe2powershell.git
7	GH:/zouxiaofei1/TopDomianTools.git	36	GH:/aaaddress1/RunPE-In-Memory.git
8	GH:/MichaelKCortez/CrackMapExecWin.git	37	GH:/BeNhNp/CallOCRDLLDemo.git
9	GH:/blundar/analyze.exe.git	38	GH:/DiegoRicardo26/20-programas.git
10	GH:/QQProtectUdpInspector.git	39	GH:/loveemu/bin2psf1.git
11	GH:/ZisBoom/MsiInv.exe.git	40	GH:/crea7or/getwindowsversion.git
12	GH:/ps1337/getuser.git	41	GH:/matthiasg/zeromq-node-windows.git
13	GH:/PhilJollans/RegSpy.git	42	GH:/lordmulder/TimedExec.git
14	GH:/LGinC/ImageLadle.git	43	GH:/Valorant-AntiCheat-Disabler.git
15	GH:/yangjiechina/GB28181_Stress_Tools.git	44	GH:/areve/node2exe.git
16	GH:/chorushe/princekin.git	45	GH:/dolarsecurity/vurbana.git
17	GH:/Jiangxiaogang/FontMaker.git	46	GH:/codepongo/zshellext.git
18	GH:/aaaddress1/RunPE-In-Memory.git	47	GH:/hufuman/sym_size.git
19	GH:/liangfu/bet2.git	48	GH:/Arma-Remote-Code-Executor.git
20	GH:/PLohrmannAMD/update_check_api.git	49	GH:/Paulo-D2000/PNexe.git
21	GH:/TinkerEdgeR/release_tools.git	50	GH:/ethereum/aleth.git
22	GH:/district10/boost-tools.git	51	GH:/jptr218/bgp_hijack.git
23	GH:/A1kmm/usbrelease.git	52	GH:/HoI4-Map-Normalizer-Tool.git
24	GH:/vczh-libraries/Release.git	53	GH:/SergioMartin86/jaffar2.git
25	GH:/Win-LocalPriv-Escalation-polarbear.git	54	GH:/rnlf/rad2exe.git
26	GH:/kristiankoskimaki/vidupe.git	55	GH:/floooh/sokol-tools.git
27	GH:/z3r0d4y5/Simple-PE32-Packer.git	56	GH:/abreheret/PixelAnnotationTool.git
28	GH:/frankgorhamengard/SparkyBrain.git	57	GH:/kliment-olechnovic/voronota.git
29	GH:/alcan2jc/FlappyBird.git	58	GH:/arggscomputerecke/cmd_start.git

Note: "GH:" stands for "https://github.com"

References

1. A quarter of major CMSs use outdated MD5 as the default password hashing scheme. www.zdnet.com/article/a-quarter-of-major-cmss-use-outdated-md5-as-the-default-password-hashing-scheme/ (2020)

2. AlFardan, N., Bernstein, D.J., Paterson, K.G., Poettering, B., Schuldt, J.C.: On the security of RC4 in TLS. In: 22nd USENIX Security Symposium (USENIX Security 13), pp. 305–320 (2013)
3. Bai, K., Wu, C.: A secure white-box SM4 implementation. Secur. Commun. Netw. **9**(10), 996–1006 (2016)
4. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a white box AES implementation. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 227–240. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30564-4_16
5. Boneh, D., Joux, A., Nguyen, P.Q.: Why textbook ElGamal and RSA encryption are insecure. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 30–43. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_3
6. Dai, W.: Crypto++ library 5.1-a free c++ class library of cryptographic schemes. https://www.cryptopp.com/ (2004)
7. Daniel, J., Bernstein, T.L.: Safecurves: choosing safe curves for elliptic-curve cryptography. https://safecurves.cr.yp.to/
8. Egele, M., Brumley, D., Fratantonio, Y., Kruegel, C.: An empirical study of cryptographic misuse in android applications. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (2013)
9. Greenwood, D.S.J.S.G., Khan, Z.L.L.: SMV-HUNTER: large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in android apps. In: Network and Distributed System Security Symposium (NDSS). Internet Society, San Diego, CA, pp. 1–14. Citeseer (2014)
10. Harvey, I.: Cipher hunting: how to find cryptographic algorithms in large binaries. NCipher Corporation Ltd., pp. 46–51 (2001)
11. Kessler, G.C.: An overview of cryptography (2003)
12. Koch, W., Schulte, M.: The libgcrypt reference manual. Free Software Foundation Inc, pp. 1–47 (2005)
13. Krüger, S., Späth, J., Ali, K., Bodden, E., Mezini, M.: CrySL: an extensible approach to validating the correct usage of cryptographic APIs. IEEE Trans. Softw. Eng. **47**(11), 2382–2400 (2019)
14. Lazar, D., Chen, H., Wang, X., Zeldovich, N.: Why does cryptographic software fail?: a case study and open problems. In: APSys (2014)
15. Li, J., Lin, Z., Caballero, J., Zhang, Y., Gu, D.: K-hunt: pinpointing insecure cryptographic keys from execution traces. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 412–425 (2018)
16. Lindström, P., Pap, O.: Mapping the current state of SSL/TLS (2017)
17. Lloyd, J.: Botan: crypto and TLS for modern C++. https://botan.randombit.net/ (2018)
18. Mouha, N., Dworkin, M., et al.: Review of the advanced encryption standard (2021)
19. Paul Bakker, A.: mbedTLS. tls. mbed. org (2019)
20. Rahaman, S., et al.: Cryptoguard: high precision detection of cryptographic vulnerabilities in massive-sized java projects. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 2455–2472 (2019)
21. Rogers, M., Eden, G.: The Snowden disclosures, technical standards and the making of surveillance infrastructures. Int. J. Commun. **11**, 802–823 (2017)
22. Sindhu, S., Sindhu, D.: Cryptographic algorithms: applications in network security. Int. J. New Innovations Eng. Technol. (2017). ISSN 2319-6319
23. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol.

10401, pp. 570–596. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_19
24. Turner, S., Chen, L.: MD2 to Historic Status. Technical report, RFC 6149, March (2011)
25. Turner, S., Chen, L.: RFC 6151: updated security considerations for the MD5 message-digest and the HMAC-MD5 algorithms. Internet Eng. Task Force (2011)
26. Wang, Z., Jiang, X., Cui, W., Wang, X., Grace, M.: ReFormat: automatic reverse engineering of encrypted messages. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 200–215. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04444-1_13
27. Young, E.A., Hudson, T.J., Engelschall, R.S.: OpenSSL. World Wide Web. https://www.openssl.org/. Accessed September 2001 (2001)