# Automatic Detection of Cryptographic Algorithms in Executable Binary Files using Advanced Code Chain

Han Seong Lee
Div. of Computer Engineering, Hanshin Univ.,
137, Yangsan-dong, Osan, Gyeong-gi, Rep. of Korea.
jkhanseong@naver.com

Hyung-Woo Lee
Div. of Computer Engineering, Hanshin Univ.,
137, Yangsan-dong, Osan, Gyeong-gi, Rep. of Korea.
hwlee@hs.ac.kr

*Abstract*: **Executable binary files can be developed using cryptographic modules using open libraries such as OpenSSL and Crypto++ in Windows environments. To determine the embedded encryption algorithms and detect cryptographic modules used in binary files, a high degree of knowledge on internal structure is required in de-assembling and analyzing. And the reverse engineering process on executable binary file is very difficult. Therefore, we developed an automatic detection tool that can automatically detect the cryptographic algorithm to efficiently analyze cryptographic algorithms as a form of IDA plug-in module. This tool can be used to detect and track cryptographic algorithms used in arbitrary executables on Windows OS system.**

**Keyword: Cryptographic Algorithms, Executable Binary File, Reverse Engineering, Automatic Detection, Identification, IDA.**

## I. INTRODUCTION

Programs such as executable binary files are being developed using cryptographic modules in Windows environments. For example, open source libraries such as OpenSSL[1] and Crypto++ are used to develop cryptographic executable file(.exe). However, in order to analyze the executable file using the IDA Debugger[2] for reversing and to determine a cryptographic library or algorithm included in the executable program, an efficiency auto-detection algorithm and several feature extraction know-hows about the cryptographic library must be acquired in advance. Therefore, it is very difficult to automatically detect cryptographic algorithm or modules embedded in arbitrary executable binary files on its reverse engineering process using IDA.

Therefore, in order to analyze the executable file using the common encryption algorithm and the executable file using the encryption module in the Windows environment, it is necessary to analyze the generally used encryption library and the encryption / decryption code such as OpenSSL and Crypto ++. If reverse analysis of the binary file is performed based on the detailed analysis of the encryption algorithm provided by OpenSSL Architecture and OpenSSL, analysis and automatic detection of the encryption algorithm included in the executable file can be improved.

In this paper, we analyze the internal structure of OpenSSL which is an encryption / decryption library that can be used in Windows environment and derive the fast detection and discrimination mechanism of encryption algorithm applied in executable file through inverse analysis of developed executable file. Then, we design and implement a plug-in that can automate the encryption algorithm when using IDA Pro debugger.

## II. EXISTING CRYPTOGRAPHIC ALGORITHM DETECTION TECHNOLOGY

As a detection algorithm of existing encryption algorithm, the most representative technique is a *constant-based analysis method*. In this method, the information such as constants used in the implementation of the encryption algorithm is set as the detection feature value, and a constant value included in the executable file is determined to determine whether the encryption algorithm is used. For example, Rijndael S-box and Inverse S-Box are used for AES. Therefore, if there is a constant value defined in the AES algorithm in the executable file, it is deduced that AES algorithm is used on executable binary file. The SHA-1 algorithm initializes the five constants 0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476, and 0xC3D2E1F0[3]. Therefore, if there are five constant values defined by SHA-1 in the program, it can be determined that the SHA-1 algorithm is used based on this constant information.

| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0a | 0b | 0c | 0d | 0e | 0f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe | d7 | ab | 76 |
| 10 | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| 20 | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| 30 | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| 40 | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| 50 | 53 | d1 | 00 | ed | 20 | fc | b1 | 5b | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| 60 | d0 | ef | aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| 70 | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| 80 | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| 90 | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | b8 | 14 | de | 5e | 0b | db |
| a0 | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| b0 | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 | 7a | ae | 08 |
| c0 | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b | bd | 8b | 8a |
| d0 | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 | c1 | 1d | 9e |
| e0 | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce | 55 | 28 | df |
| f0 | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | b0 | 54 | bb | 16 |

| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0a | 0b | 0c | 0d | 0e | 0f |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 52 | 09 | 6a | d5 | 30 | 36 | a5 | 38 | bf | 40 | a3 | 9e | 81 | f3 | d7 | fb |
| 10 | 7c | e3 | 39 | 82 | 9b | 2f | ff | 87 | 34 | 8e | 43 | 44 | c4 | de | e9 | cb |
| 20 | 54 | 7b | 94 | 32 | a6 | c2 | 23 | 3d | ee | 4c | 95 | 0b | 42 | fa | c3 | 4e |
| 30 | 08 | 2e | a1 | 66 | 28 | d9 | 24 | b2 | 76 | 5b | a2 | 49 | 6d | 8b | d1 | 25 |
| 40 | 72 | f8 | f6 | 64 | 86 | 68 | 98 | 16 | d4 | a4 | 5c | cc | 5d | 65 | b6 | 92 |
| 50 | 6c | 70 | 48 | 50 | fd | ed | b9 | da | 5e | 15 | 46 | 57 | a7 | 8d | 9d | 84 |
| 60 | 90 | d8 | ab | 00 | 8c | bc | d3 | 0a | f7 | e4 | 58 | 05 | b8 | b3 | 45 | 06 |
| 70 | d0 | 2c | 1e | 8f | ca | 3f | 0f | 02 | c1 | af | bd | 03 | 01 | 13 | 8a | 6b |
| 80 | 3a | 91 | 11 | 41 | 4f | 67 | dc | ea | 97 | f2 | cf | ce | f0 | b4 | e6 | 73 |
| 90 | 96 | ac | 74 | 22 | e7 | ad | 35 | 85 | e2 | f9 | 37 | e8 | 1c | 75 | df | 6e |
| a0 | 47 | f1 | 1a | 71 | 1d | 29 | c5 | 89 | 6f | b7 | 62 | 0e | aa | 18 | be | 1b |
| b0 | fc | 56 | 3e | 4b | c6 | d2 | 79 | 20 | 9a | db | c0 | fe | 78 | cd | 5a | f4 |
| c0 | 1f | dd | a8 | 33 | 88 | 07 | c7 | 31 | b1 | 12 | 10 | 59 | 27 | 80 | ec | 5f |
| d0 | 60 | 51 | 7f | a9 | 19 | b5 | 4a | 0d | 2d | e5 | 7a | 9f | 93 | c9 | 9c | ef |
| e0 | a0 | e0 | 3b | 4d | ae | 2a | f5 | b0 | c8 | eb | bb | 3c | 83 | 53 | 99 | 61 |
| f0 | 17 | 2b | 04 | 7e | ba | 77 | d6 | 26 | e1 | 69 | 14 | 63 | 55 | 21 | 0c | 7d |

Fig. 1 AES S-Box and Inverse S-Box[4]

Another method of cryptographic algorithm analysis is *symbol-based analysis*. It is a method of identifying the cryptographic algorithm API existing in the library by using the *Import and Export Table symbols* generated when using the dynamic link library. The existing cryptographic algorithm detection methods described above have advantages and disadvantages in terms of effectiveness and reliability of analysis, and there are various advantages and disadvantages in each analysis technique. First, a constant-based detection method, which is a cryptographic algorithm analysis detection method, has already implemented tools and open source, and can quickly search and detect if the analyzed source code or program includes constants. In addition, if you are developing

978-1-7281-0627-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ICGHIT.2019.00031



Authorized licensed use limited to: Stanford University. Downloaded on March 21,2021 at 05:22:19 UTC from IEEE Xplore. Restrictions apply.

your own analysis tools, the implementation is simple. However, the reliability of the analysis result cannot be guaranteed and it is impossible to judge whether or not the function is called. And it is impossible to detect and analyze using cryptographic algorithms without constant. The symbol-based analysis method can track whether a function is called in contrast to constant-based detection. This can lead to more reliable analysis results. But, the API information of the library used in the program needs to be known in advance, and in the case of the static library, detection is impossible. As a result, there is a need for a new detection technique that exploits the advantages of constant-based detection and library functions and improves overcoming disadvantages.

## III. Improved Cryptographic Algorithm Detection Mechanism

We have developed *Code Chain* based detection method to solve the existing problems. The proposed scheme uses the mathematical operation of the cryptographic algorithm and finds the same operation code pattern in terms of the code. In other words, the binary pattern of the cryptographic algorithm is analyzed and the function used in the analysis target is detected. Then, the code chain is generated, compared and verified in the same manner as the following Fig. 2.

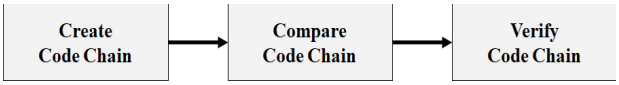


Fig. 2 Code Chain Process

Since the code chain only compares opcodes except for operands, it has a strong advantage that it is simple to detect because it is not influenced by the operands, and detection speed is fast because of the simple search. However, if an operand has a constant, it cannot detect initial variables that can be a signature. In addition, there is a problem that it cannot be discriminated in the case of a code chain having the same operation code but different operands. This is a major issue in our research and suggests a new method, the *Advanced Code Chain (ACC)*, to improve this problem.

Advanced Code Chain performs the following process by sequentially searching all the functions in the program and extracting the binary of each function. The function code chain is generated through the wild card processing for the operands, the data is created, and then the comparison process is performed. Finally, it is determined whether or not the function code discrimination information is useful. And the function code chain is compared based on this. The verification and identification process of the cryptographic algorithm is performed by the function code chain verification and identification process in the end.

The procedure for extracting Advanced Code Chain of all functions in the program is shown in Fig. 3 shows. The functions in the program are searched sequentially. Functions are extracted as binary. And an advanced code chain is generated from that binaries. Then the count variable is initialized to zero. The advanced code chain is searched for existence in the program, and if the same advanced code chain is present in the program, the count variable is incremented by 1. If not, go to the next function. The count variable is a means for checking whether the advanced code chain has uniqueness.

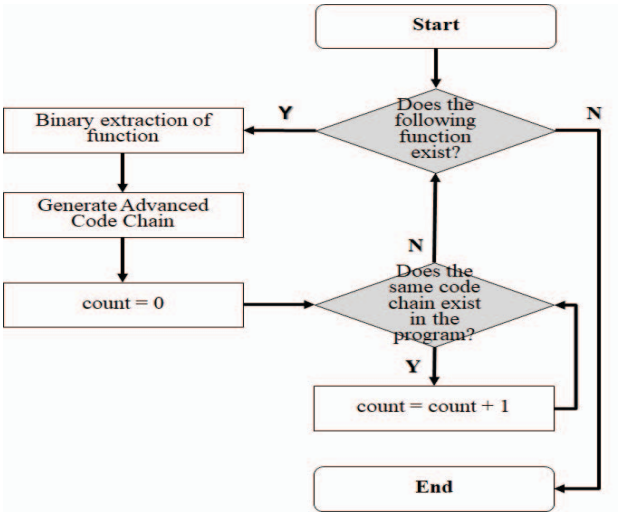


Fig. 3 Advanced Code Chain Generation Overall Process

We implemented the IDA Pro plug-in based Advanced Code Chain modules. OpenSSL SHA1 sample program is loaded from IDA Pro and the result of executing the plug-in is shown in Fig. 4. All functions are generated with the advanced code chain. The header of the result is address, function name, count, and advanced code chain. If "Count" is 2 or more, the background of the row is displayed in red. The row is not suitable for use with the advanced code chain. Therefore, the advanced code chain is not recommended. Several same advanced code chains are found because the length is too short. So, using it can be a reason for generating false detection results.

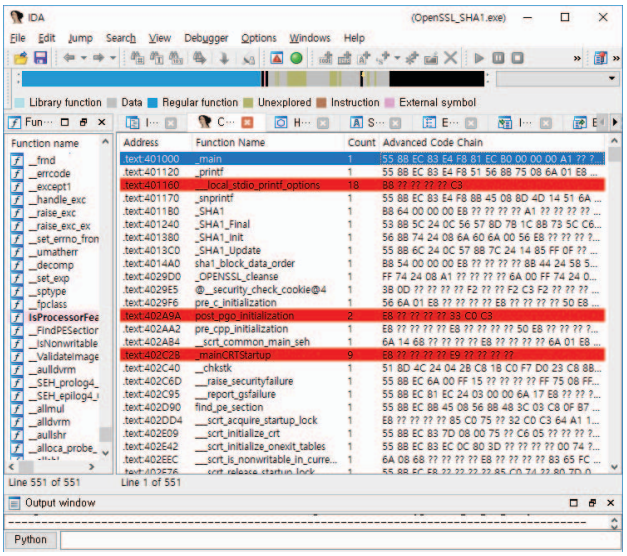


Fig. 4 The result of loading Advanced Code Chain Modules was implemented in IDA Pro plug-in in OpenSSL SHA1 program



Authorized licensed use limited to: Stanford University. Downloaded on March 21,2021 at 05:22:19 UTC from IEEE Xplore. Restrictions apply.

The cryptographic algorithm consists of a combination of multiple functions. Therefore, it is difficult to judge that a specific cryptographic algorithm is used by comparing the advanced code chain of the function. To solve this problem, YARA[5] based pattern matching technique is applied. For example, if there are 4 functions in the SHA-1 algorithm, you can create a rule as shown in Fig. 5.

YARA

```
rule SHA1 {
    strings:
        $_SHA1_Init = { Binary Pattern }
        $_SHA1_Update = { Binary Pattern }
        $_SHA1_Final = { Binary Pattern }
        $sha1_block_data_order = { Binary Pattern }
    condition:
        $_SHA1_Init and $_SHA1_Update and $_SHA1_Final
and $sha1_block_data_order
}
```

Fig. 5 Example of SHA1 detection rules using YARA

There are three steps to create a YARA rule for cryptographic algorithms. We implement the program using a cryptographic library. We reverse engineer the implemented program to analyze the function call structure that is the core of the cryptographic algorithm. The call structure of the function can be analyzed by a tool such as IDA Pro Digraph. And core functions are extracted into Advanced Code Chain. Finally, we create a detection rule based on the results of the analysis and extraction.

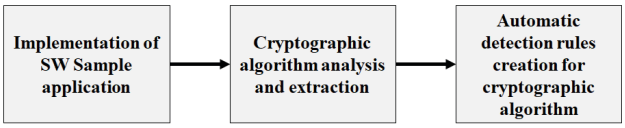


Fig. 6 YARA rules creation process for automatic detection of cryptographic algorithm

## IV. Implementation of Crypto Auto Detector Plugin and Detection of Cryptographic Algorithm

The architecture of the Crypto Auto Detector Plugin consists of two core modules and two other modules. "Advanced Code Chain Modules" generates advanced code chains, which is the signature of a function in a program. Its modules provide to check whether generated the advanced code chain have uniqueness. "YARA Modules" is a module that performs detection of cryptographic algorithms. Its modules retrieve the pre-written rules and return the detection results by combining the metadata analyzed by IDA Pro's API. In addition, it has functions to track calls of detected functions and function name change function. "Database Modules" stores analysis results and detection results in the form of metadata. Its module can load the results of the earlier analysis. "GUI Modules" is a module for displaying results to IDA Pro. Its module includes related tools for comparing and analyzing the detection code or function's advanced code chain.

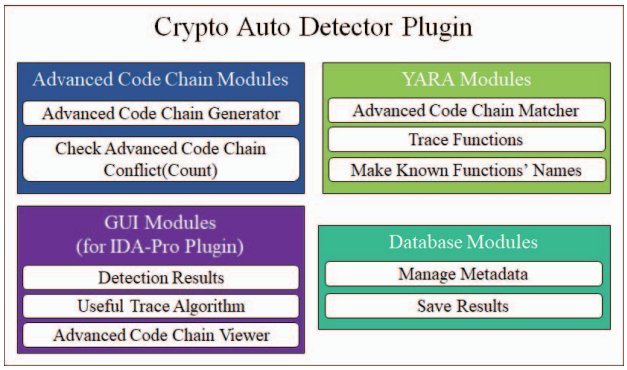


Fig. 7 Architecture of Crypto Auto Detector Plugin

We designed the database schema to manage cryptographic algorithm detection results. We have configured the three tables functions, tags, and xref_functions in the database schema. The "tags" table has the names of the cryptographic algorithms (e.g. SHA512, AES, RSA). The "functions" table has the cryptographic algorithm functions detected in a program. And the "xref_functions" table has the call information of the detected cryptographic algorithm function.

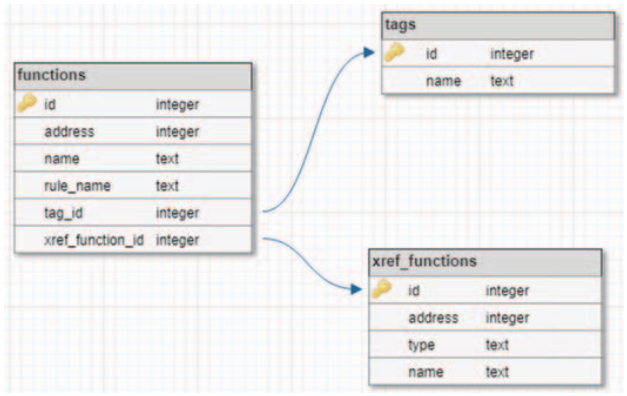


Fig. 8 DB Schema of Crypto Auto Detector Plugin

The system of Crypto Auto Detector Plugin receives the rule file created by the advanced code chain when the executable comes in. Before analyzing, it is checked whether there is a previous analysis result, and if it is reanalyzed, the previous result can be load without reanalysis. Otherwise, when performing analysis, YARA Match is executed with execution file and rule file. YARA Match performs the cryptographic algorithm pattern matching defined in the rule file. By using the returned results, we can find the function using the API provided by IDA Pro, so that we can detect the cryptographic algorithm not only for detection but also to change the name of the detected cryptographic function to a known name and to trace back the function call. The results are displayed in IDA Pro as a graphical user interface and saved as a database.



Authorized licensed use limited to: Stanford University. Downloaded on March 21,2021 at 05:22:19 UTC from IEEE Xplore. Restrictions apply.

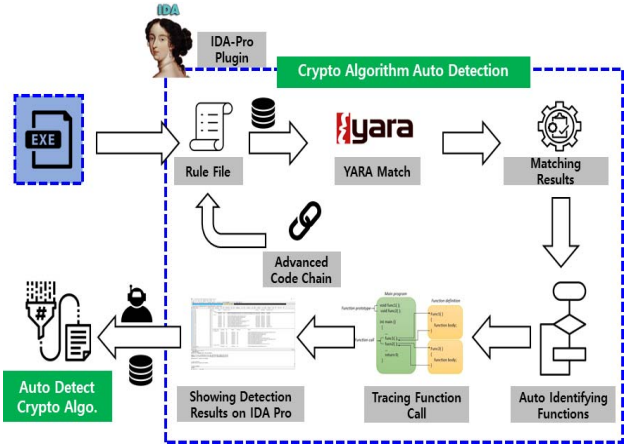


Fig. 9 Advanced Code Chain based automatic cryptographic algorithm detection system

This is the result of using the *Advanced Code Chain-based Cryptographic Algorithm Auto-detection Plug-in* developed in this study and applying it to the executable file applied to the OpenSSL library. We experimented with the YARA rule for executables with the SHA256 algorithm in the OpenSSL library. The Crypto Auto Detection Plug-in automatically detects SHA256_Init(), SHA256_Update(), SHA256_Final(), and wrapper function SHA256() functions that are core functions of OpenSSL's SHA256 algorithm implemented in the program.

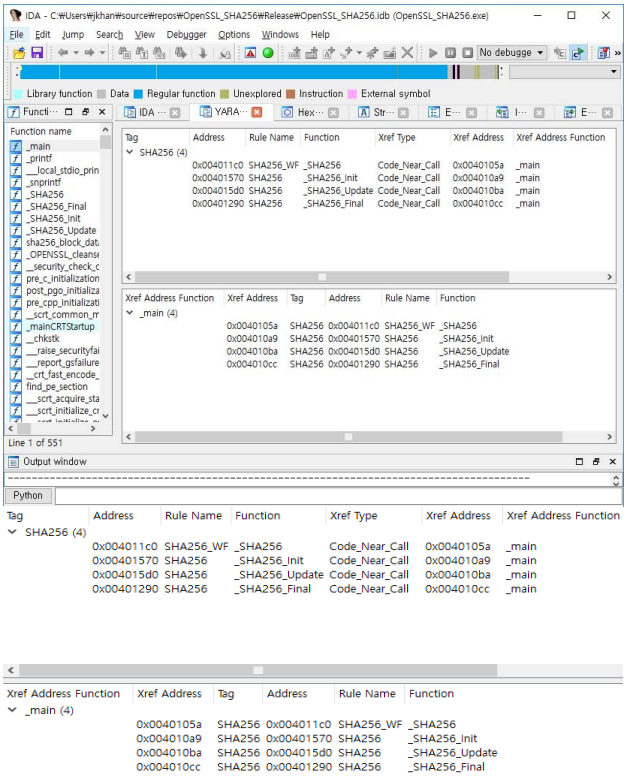

Fig. 10 SHA256 algorithm detection results of OpenSSL library implemented in the program using Crypto Auto Detector Plugin

This is the result of using the OpenSSL library for an executable file using various cryptographic algorithms. We used YARA rules to detect cryptographic algorithms in TABLE I. Crypto Auto Detector Plugin not only automatically detects hash algorithm but also AES, RSA, and HMAC implemented in OpenSSL. And it changed the name of the function to the function name used in OpenSSL. This result shows that the Crypto Auto Detector Plugin can detect various cryptographic algorithms.

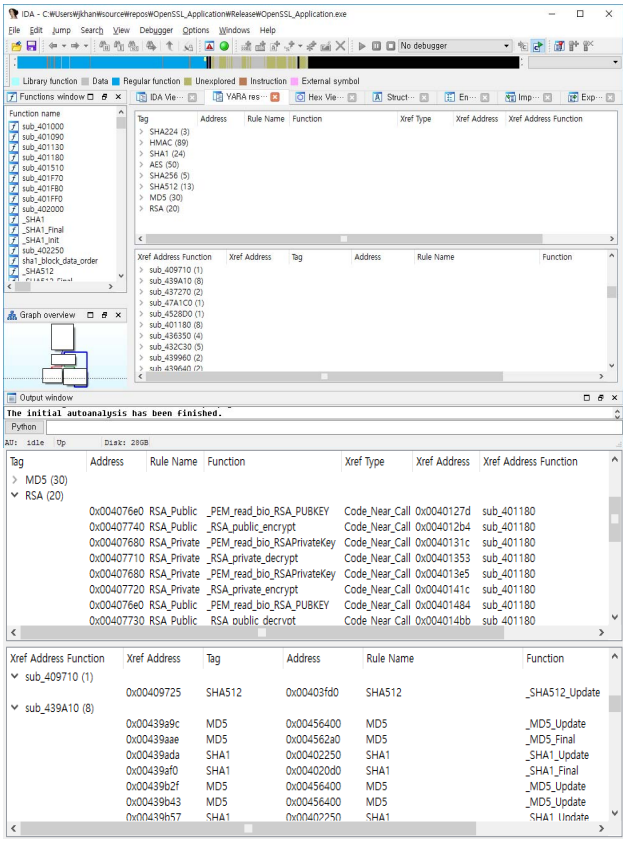

Fig. 11 Various cryptographic algorithms detection results of OpenSSL library implemented in the program using Crypto Auto Detector Plugin

## V. Crypto Auto Detector Plugin and Existing Cryptographic Algorithm Detection Programs Comparison

The following TABLE I is the detection performance of the existing cryptographic algorithms detection programs and the crypto algorithms of the OpenSSL library in the Crypto Auto Detector Plugin. The existing cryptographic algorithm detection program cannot detect HMAC and RSA. Because it is a constant-based detection, it cannot detect algorithms that do not have a constant. The Hash & Crypto Detector and DRACA[6] do not detect the AES algorithm in the OpenSSL library. However, FindCrypt plug-in[7] and KANAL plugin for PEiD[8] detect the AES algorithm but do not detect the cipher mode.



Authorized licensed use limited to: Stanford University. Downloaded on March 21,2021 at 05:22:19 UTC from IEEE Xplore. Restrictions apply.

TABLE I: DETECTION PERFORMANCE COMPARISON

| OpenSSL Algorithm | Crypto Auto Detector Plugin | FindCrypt Plugin [7] | Hash & Crypto Detector | KANAL plugin for PEiD [8] | DRACA [6] |
|---|---|---|---|---|---|
| MD5 | ○ | ○ | ○ | ○ | ○ |
| SHA1 | ○ | ○ | ○ | ○ | ○ |
| SHA224 | ○ | ○ | × | ○ | × |
| SHA256 | ○ | ○ | ○ | ○ | × |
| SHA512 | ○ | ○ | ○ | ○ | × |
| HMAC | ○ | × | × | × | × |
| AES | ○ | △ | × | △ | × |
| RSA | ○ | × | × | × | × |

It has a disadvantage that it only knows the position of constants in the detection result of the existing cryptographic algorithms detection programs and cannot detect the functions of the cryptographic algorithms or track the function calls (Fig. 12). On the other hand, Crypto Auto Detector Plugin provides detailed information about cryptographic algorithms detection results and traced functions and function calls tracing (Fig. 13). In addition, it provides the following features. It is displayed in yellow for function calls detected in IDA View. You can move to the location of the detected function or function call or add and remove breakpoints. These features will be helpful in reversing engineering.

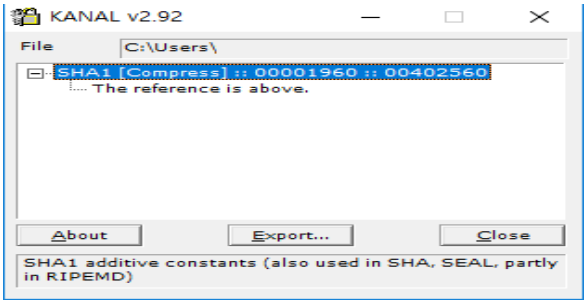


Fig. 12 KANAL plugin for PEiD[7]

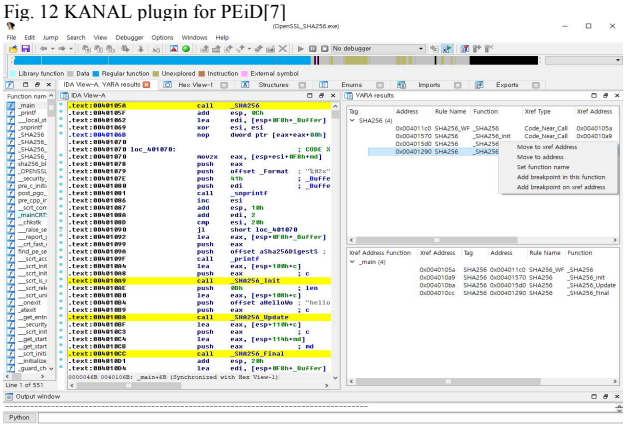

Fig. 13 Crypto Auto Detector Plugin

## VI. CONCLUSIONS

To detect the cryptographic algorithm automatically, we could derive the code chain technique. The code chain for each cryptographic algorithm was generated based on the assembly code information extracted from the IDA analysis and applied to the process of identifying the cryptographic algorithm based on this code chain. We also developed Advanced Code Chain method that solved the problems of code chain technique. In order to perform automated verification and identification, YARA based pattern matching technique was applied. As a result of the analysis, it was confirmed that the Crypto Auto Detector Plugin implemented in this study correctly analyzed and detected hash functions, AES and RSA cryptographic algorithms included in OpenSSL library. The results of this study were as follows. This is developed in the form of software that can be run in Windows system. It minimizes the additional software needed to use the tool, simplifies the installation process, and provides quick analysis function for arbitrary executable files. In addition, it is implemented as a plugin type in IDA Pro and can be used universally. It is designed and implemented as a software structure considering scalability. It provides efficient and fast binary analysis function for cryptographic library when using this tool.

## ACKNOWLEDGEMENT

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (NRF-2017R1D1B03035040)

## REFERENCES

[1] OpenSSL, https://www.openssl.org
[2] IDA Debugger, https://www.hex-rays.com/products/ida/index.shtml
[3] James H. Burrows, "SECURE HASH STANDARD", Federal Information Processing Standards Publications (FIPS PUBS), 1995.
[4] Rijndael S-box, WIKIPEDIA, https://en.wikipedia.org/wiki/Rijndael_S-box
[5] YARA, GITHUB, https://github.com/VirusTotal/yara
[6] Draft Crypto Analyzer (DRACA), http://www.literatecode.com/draca
[7] Findcrypt, GITHUB, https://github.com/you0708/ida/tree/master/idapython_tools/fin
[8] PEiD, https://www.aldeid.com/wiki/PEiD



Authorized licensed use limited to: Stanford University. Downloaded on March 21,2021 at 05:22:19 UTC from IEEE Xplore. Restrictions apply.