Journal of Computer and Communications, 2015, 3, 21-32
Published Online May 2015 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2015.35003

Establishment and Application of Cryptographic Library Model

Tie-Ming Liu, Lie-Hui Jiang, Jing Jing, Yuan-Yuan Zhang

The State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, China
Email: fxliutm@163.com

Received February 2015

Abstract

When doing reverse analysis of program's binary codes, it is often to encounter the function of cryptographic library. In order to reduce workload, a cryptographic library model has been designed by analysts. Models use formalized approach to describe the frame of cryptology and the structure of cryptographic function, complete the mapping from cryptographic function property to its architecture, and accomplish the result presentation of data analysis and mapping at last. The model can solve two problems: the first one is to know the hierarchy of the cryptographic function in the library well; the second one is to know some kinds of information, such as related cryptology algorithm and protocol, etc. These function implements can display the result graphically. The model can find relevant knowledge for the analysts automatically and rapidly, which is helpful to the learning of the overall abstract structure of cryptology.

Keywords

Cryptographic Library Model, Cryptographic Library Architecture, Cryptographic Algorithm, Encryption and Decryption Keys, Encryption and Decryption Data, Information Recovery

1. Introduction

The cryptographic algorithms and protocols are often used in many fields, such as protection of network data transmission, software shelling, code obfuscation and electronic commerce [1] [2]. The difficult point of software reverse engineering is the reverse analysis of cryptographic algorithms in software, and the mains analysis methods are based on the characteristics of cryptographic algorithm, library signature and dynamic tracing, etc.

The encrypting and decrypting algorithm analytic technology [3] is based on the characteristic of cryptographic algorithm only identifies partial characteristics of the algorithm, but fails to do detailed analysis on the information such as data, cryptographic key, pattern, etc. during the process of encrypting and decrypting. The feature analysis can be started with the binary scanning of executable codes, or the analysis can be done with the application of memory access technique [4] or the combination of dynamic debugging. There is also a relatively practical technique of algorithm identification, the IDA's Fast Library Identification and Recognition Technology (FLIRT) [5]. It has a problem that the version of the library and compiler will affect the accuracy of the recognition.

In addition, after the emergence of dynamic tracing tools [6]-[8] such as Pin and Valgrind, the study of dynamic trace is ever-growing. The acquisition of dynamic trace is to record the instructions, functions and the data information in the implementation process of the program with the help of dynamic tracing tools, so as to form record files. The research on dynamic trace has become the hotspot of research [9] [10]. These methods are novel and effective in theory and practice, and their research findings are of reference value. It can restore part of information (data or cryptographic key) of the algorithm, but cannot give the call chain relationship of function in algorithm and the process of matching and corroboration of it is too time-consuming.

The Cryptographic Library in common used (OpenSSL, LibTomCrypt, CryptoAPI, Cryptlib and so forth) accounts for about 70% in the application of software, adopting the feature of developing common cryptographic libraries which is secure and swift to recover the information in cryptographic algorithms or protocols in the libraries of software, which is of great importance. Therefore, this paper puts forward a cryptographic description architecture and a description method of library architecture which are specific to the cryptographic library, so as to recover the relation between algorithmic information and function calling chain when the library function is used in the program.

2. Summary of Functions of the Common Cryptographic Library

To take cryptographic library based on C programming language as an example, the frequently-used ones are OpenSSL, LibTomCrypt, libgcrypt, Cryptlib, CryptoAPI and CNG. It can be seen from **Table 1** that the function focus of every cryptographic library is different. Now the summary is as follows: the function of every cryptographic library contains some symmetries, asymmetries and Hash algorithms. However, regardless of the number of varieties, the requirements of general users can be met on the whole. The support to protocols vary, among which, Cryptlib possesses the most detailed support to all protocols. OpenSSL, CryptoAPI and CNG have the similar support to the SSL protocol, and libgcrypt and LibTomCrypt haven't realized the high-level protocols. In addition, the library which supports the protocols better all contains the analysis and storage of ASN.1 format, X509 certificate management, PEM format and so on. The three additive figures in the column "Algorithm Implementation" of **Table 1** respectively are the varieties of symmetric algorithm, Hash algorithm and asymmetric algorithm, and the statistics are listed in **Table 2**.

The statistical rule in **Table 2** is that consider MD2, MD4, MD5 and SHA0, SHA1, SHA2 as the same kinds, for the reason that a cryptographic library usually includes several pieces of algorithms which are in the same kinds, and have similar algorithms. Compared with them, the differences between RC2, RC4, and RC5 are larger, so we didn't unify them to one kind. And we regard AES series as one kind because they have the similar algorithm principle and the major difference is the size of round and key.

3. Model of Description of Library Cryptography Designed in Terms of Library

The essence of the model of description of library cryptography (hereinafter referred to as the description model) is conducting summary and conclusion of the key attributes of those cryptography-related algorithms and pro-

Table 1. The common functions of the vaults.

Library	Algorithm Implementation	Message Abstract	Protocol Implementation	Others	Version
OpenSSL	11 + 5 + 6 = 22	HMAC, CMAC, GCM, CCM	SSL/TLS, ocsp, srp	x509, x509v3, ASN.1, pem, pkcs12	1.01c
Lib Tom-Crypt	17 + 5 + 3 = 25	HMAC, CMAC, GCM, CCM	Unrealized, Reserved Interface	PRND, ASN.1, BASE64, BIGNUM	1.17
libgcrypt	15 + 9 + 3 = 27	HMAC, CMAC, GCM, CCM	Unrealized	RNG, MPI, Sexp	1.6.0
cryptlib	10 + 4 + 7 = 21	HMAC, GCM, CCM	SSL/TLS, S/MIME, PGP/OpenPGP, SSH, CMP, OCSP, RTCS, SCEP, TSP	RNG, PKCS11, x509, BASE64	3.4.2
cyptoAPI	4 + 3 + 4 = 11	HMAC	SSL, TLS, PCT1	RNG, PKCS11, x509, BASE64	
CNG	3 + 5 + 5 = 13	HMAC, CMAC, GMAC(GCM)	SSL	RNG, PKCS11, x509, BASE64	

Table 2. Implements of common library algorithm.

Library	Symmetric Algorithm	Hash Algorithm	Asymmetric Algorithm
OpenSSL	Blowfish, CAST, DES, 3DES, idea, RC2, RC4, RC5, seed, camellia, AES (11 kinds)	MD2, MD4, MD5, MDC2, ripemd, SHA series, Whirlpool (7 pieces of 5 kinds)	DH, DSA, ECDH, ECDSA, SRP, J-pake (6 kinds)
LibTomCrypt	Blowfish, X-Tea, RC2, RC5, RC6, SAFER + +, AES (17 kinds)	Whirlpool, SHA-512, SHA-384 (14 pieces of 5 kinds)	RSA, DSA, ECC (3 kinds)
libgcrypt	GCRY_CIPHER_IDEA, GCRY_CIPH, ER_3DES,... (30 pieces of 15 kinds)	GCRY_MD_SHA1, GCRY_MD_RMD160... (20 pieces of 9 kinds)	RSA, DSA, Elgamal (3 kinds)
cryptlib	AES, Blowfish, CAST-128, DES, Triple-DES, IDEA, RC2, RC4, RC5, SkipjackCryptlib(10 kinds)	MD2, MD4, MD5, RIPEMD-160, SHA-1, SHA-2, SHAng: (7 pieces of 4 kinds, md2 and md4 are listed as the outdate)	CRYPT_ALGO_DH, CRYPT_ALGO_RSA... (6 kinds)
CryptoAPI	CALG_3DES, CALG_3DES_112, CALG_AES (12 pieces of 4 kinds)	CALG_HUGHES_MD5 CALG_HASH_REPLACE_OWF (10 pieces of 3 kinds)	CALG_AGREEDKEY, ANY (It is DH) (6 pieces of 4 kinds)
CNG	BCRYPT_3DES_ALGORITHM,... (9 pieces of 3 kinds)	BCRYPT_SHA1_ALGORITHM, ... (10 pieces of 5 kinds)	BCRYPT_DH_ALGORITHM,. .. (10 pieces of 5 kinds)

tocols. Describing Module Construction Method is the way to establish, according to the function and characteristics of the cryptographic library, from the angle of applied cryptography, a module which divides the levels and the modules, has relevant knowledge annotation and has nothing to do with the specific realization.

3.1. The Theories and Factual Basis of Description Model Establishment

Computer security defines five core targets: confidentiality, integrity, availability, authenticity, traceability [2] [3]. In cryptology, the basic function of cryptology algorithm is to realize one of them, and the cryptographic protocol basically and comprehensively realizes all the five functions, such as DES (confidentiality), MAC (integrity and authenticity), RSA signature (authenticity and availability) and undeniable digital signature (non-repudiation), while SSL protocol comprehensively realizes those functions.

Cryptographic algorithm itself has a complete encryption and decryption process. However, the properties of cryptography are not the confidentiality of the processes, but the characteristic that the entity owned secret keys had. Therefore, the abstract of cryptographic algorithm is not aimed at abstracting its encryption and decryption process but attributes, for example, the input/output data and the cryptographic key. See **Figure 1** for algorithm description: 1.

Cryptographic protocol is a combination of complex process and limited function. The complexity of cryptographic protocol lies in the complexity of its consultation and data process, the protocol sets up a procedure to complete the authenticity, security, integrity and data compression. It is proper to use the negotiation processes and results to describe the protocol, among which the most significant one is the negotiation results, because the protocol is using a series of consult results to deal with the data, and determining these results will helps to know the process of dealing data completely due to the consult process of protocol will infect the consult results before the protocol, so, the description of negotiation process is necessary.

As shown in **Figure 2**, the common properties of SSL protocol include version, session ID, cipher Suite, master Secret, cryptographic parameters (client Write MAC Secret, server Write MAC Secret, client Write Key, server Write Key, client IV, server IV), certificate information (cert, peer Cert), and data (send Data, receive Data). Actually, after the completion of the SSL negotiation, during data process, only cryptographic parameters are required. The negotiation process description of SSL includes handshaking and data transferring process, and the most complicated is handshaking process in protocol of SSL, which includes verifying, calculating master cryptography key and generating cryptographic parameters. And the TLS is similar to SSL.

Seen from the previous section, the cryptographic library can basically perform the following functions: Symmetric algorithm, Asymmetric algorithm, Hash algorithm, MAC (Message Authentication Code), HMAC (Hash-based Message Authentication Code), CMAC (Cipher-based Message Authentication Code), key Derive,

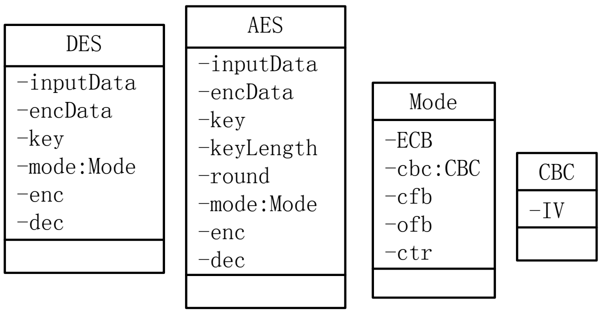

Figure 1. Algorithm description diagram 1.

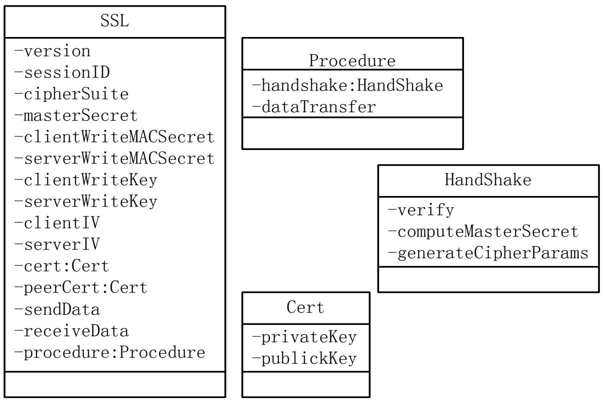

Figure 2. Protocol description diagram 2.

SSL, TLS, PGP, cryptographic key management, certificate, random number, ASN.1, pem, BASE64 and PKCS (The Public-Key Cryptography Standards).

According to elaboration above and based on the following reasons, the establishing of the description model can be successful (accurately describe the algorithm protocols in the cryptographic library), and is extensible (the newly-added and varied cryptographic library functions can answer by modifying the model):

- The cryptology only accomplishes limited capability. No matter how many algorithms or protocols there are, their functions are limited to only five targets.
- The cryptographic algorithm is only classified into limited types. From a perspective focusing only on the algorithm characteristic, the cryptographic algorithm just falls into three types, namely, symmetric algorithm, asymmetric algorithm and Hash algorithm, all of which share some mutual characteristics.
- The library can't realize too many cryptographic algorithms. It is about thirty types according to rough statistics from last section. Under this circumstance, it is more reliable and appropriate to establish a model.
- There aren't too many common protocols realized in library. Generally, they are SSL/TLS, PGP, S/MIME.

3.2. Model Layered

Layering the knowledge of cryptography will contribute to a clear grasp of the hierarchical architecture to which cryptography knowledge belongs. From the functional statistics of different libraries, it can be concluded that the realized function by cryptographic library could be summarized into three layers. The first layer is the support layer, such as ASN.1 encoding, PEM format, X509 certificate, large number calculation, random number generator, etc. The function of this layer supports the cryptographic algorithms or the realization of the protocols. The second layer is cryptographic algorithms layer, which includes various algorithms. The third layer is the protocols layer, which is to complete various more specific tasks like SSL, TLS and so on based on a cryptographic algorithm, not including some simple protocols such as key exchange and the signature protocols.

The corresponding XML description is as follows:

<level grade = "1" name = "support_layer" meaning = "supporting layer, large number operation... "> </level>

<level grade =“2” name =“algorithms_layer” meaning =“cryptographic algorithms layer, the symmetric algorithm...”> </level>

<level grade =“3” name =“protocols_layer” meaning =“the protocols layer, SSL, PGP...”> </level>

As XML is easy to show inclusion relation, but cannot show well about the layer relationship at the same layer in the node tree, it needs to set the properties representing layer relationships between nodes. Set grade to indicate the value of this layer. The higher the value is, the higher the layer is. Set the meaning attribute as the description of this layer, and the modules, algorithm and property in the description model will give meaning attribute to help interpret the relative concepts.

3.3. Layer Modularized

Dividing modules to each layer can help to intensively distinguish the categories contained in the layer and summarize the common property in the same category. With regard to the divided modules, they only need to be conducted in the second and the third layer. The first layer is some scattered knowledge, not having the problem of module partition. According to algorithm characteristic or its function, the second layer can be divided into symmetric algorithms, hash algorithms, asymmetric encryption algorithms, signature algorithms, message authentication algorithms, cryptographic key generation algorithms, and cryptographic key negotiation algorithms. The method to divide modules can adopt multilevel divisions. For example, symmetric algorithms can be divided into block cipher algorithms and flow cipher algorithms. Because the common attributes, such as the encryption mode, of block cipher algorithms are not supported in the same way by each one of the algorithms, these two types of modules are not divided up in this system. The protocols of the third layer can be divided into the application layer protocols, IP layer protocols, transport protocols, cryptographic key management protocols, authentication class protocol.

The principle of modularizing is putting the algorithm protocols that possess common attributes or classification into a module. The difficulty in the algorithm layer’s modularization is that one algorithm has many uses. For example, RSA can be used not only for asymmetric encryption but also for signature, thus the RSA exists not only in the asymmetric encryption module but also in the signature module, which is due to the abstract attributes of these two applications.

The modules of the second layer have some common attributes. For instance, symmetrical algorithms’ common attribute are the input data, output data, symmetrical cryptographic keys, and cryptographic identities; Hash algorithms’ common attribute are the input data and hash value; The common attributes of asymmetric encryption algorithms are public keys, private keys and cryptographic identities; Signature algorithms’ common attribute are the input data, signature value, public key, private key and signature identification. The common attributes of cryptographic key negotiation algorithms are the negotiating results. The common attributes of message authentication algorithms are authentication codes and cryptographic key. The common attributes of the key generation algorithms are the generated keys, as shown in **Table 3**.

Due to the diversity and complexity of protocol layer, the protocols inside the module do not have common attributes, but classification contributes to understanding the protocols, as shown in **Table 4**.

The descriptions of modularization:

<level grade=“2” name =“algorithm _layer” meaning=“ cryptographic algorithm layer, symmetric algorithm…”>

<module name =“symmetric_algorithm” meaning=“symmetric algorithm make decryption calculation using the same key …”>

```
<attribute>
  <input _data meaning = “input data”>
  </input _data>
  <enc_data meaning =“encryption data”>
  </enc_data>
  <key meaning = “key”></key>
  <original_key meaning =“ key of the original string “></original_key>
<encFlag meaning =“encryption flag”>
  <enc meaning = “encryption”>
  <dec meaning = “ decryption”>
```

Table 3. Algorithms of the modules in the second layer 2.

Modules	Algorithms
Symmetric Algorithm	{Block cipher algorithm: AES, Blowfish, DES, Triple DES, Serpent, Twofish, Camellia, CAST-128, IDEA, RC2, RC5, SEED, Skipjack TEA XTEA} {Stream cipher algorithm: RC4}
Hash Algorithm	GOST, HAVAL, MD2, MD4, MD5, PANAMA, RadioGatún, RIPEMD, RIPEMD-160, RIPEMD-128, RIPEMD-320, SHA-0, SHA-1, SHA-2, SHA-3, Tiger, WHIRLPOOL
Asymmetric Encryption Algorithm	RSA, ElGamal, Rabin, ECC
Signing Algorithm	RSA, ElGamal, DSA, ECDSA, GMR, Schnorr
Key Negotiation Algorithm	DH, SRP, EKE, J-PAKE
Message Authentication Algorithm	HMAC, CMAC, GMAC, DAA
Key Generation Algorithms	PBKDF2, KDF

Table 4. Protocols of every modules in the third layer 3.

Modules	Protocols
IP Security	IPSEC
Transport Layer Protocol	SSL SSH
Application Layer Protocol	HTTPS PGP/OPENPGP DKIM S/MIME
Certificate and Key Management	CMP OCSP SCEP RTCS
Authentication Protocol	Kerberos

```
  </encFlag>
    </attribute>
  </module>
  <module name = “hash_algorithm”… >
…
</level>
```

Put the common attributes possessed by the algorithms inside the modules into the attribute node.

3.4. Attribute Descriptions of Algorithm and Protocol

The core of the model is algorithm and protocol, part of whose attributes are embedded in the attributes of the module it belongs to. When this algorithm or protocol has certain attribute, the attribute is included in the algorithm protocol node. For example, the block cipher algorithm generally has such encryption modes as ECB, CBC, CFB, OFB, CTR and so on, but the stream cipher algorithm doesn’t. Thus some algorithms, like DES, contain the mode attributes, while the RC4 algorithm doesn’t. Because the protocols are greatly different, the related attributes are all listed in the protocol.

Algorithm attributes are described as follows:

```
< module name=“symmetric_algorithm...>
  < alg_protocol name= “DES” meaning = “...”>
    <block_size default_value = “64” meaning = “the size of the plaintext block digits” ></block_size>
    <key_size default_value=“56” meaning = “the size of the key block digits”></key_size>
  <mode>
    <ECB meaning=“ECB mode will divide the plaintext blocks, and each block will be encrypted with the same key...” ></ECB>
    <CBC...> <IV...></IV> </CBC>
    <CFB...> <IV...></IV> </CFB>
    <OFB...>...</OFB>
    <CTR...>    <counter...></counter>    </CTR>
    </mode>
   </alg_protocol>
   <alg_protocol name = "RC4" meaning = "...">...
   </alg_protocol>
  </module>
```

Put algorithms, protocols and relevant cipher knowledge on the supporting layer into a unified node "alg_ protocol", and name attribute refers to the name of algorithms or protocols. We need to set independent nodes for the unique attributes of the algorithms or protocols under the node of "alg_protocol", such as the encryption flag of DES (encFlag), the size of the block digit (block_size), the size of the key digit (key_size). And if the algorithms or protocols contain fairly complex attributes, we should newly establish the node describing tree for this attribute, such as DES algorithm mode, SSL protocol related certificate.

3.5. Describing the Overall Architecture of the Model

In the previous descriptions, algorithm and their attributes were all mapped to independent nodes, and the nodes were given a describe attribute as brief description. In fact, the common attributes of modules and the unique attributes of algorithm can be all extracted uniformly as the algorithm attribute. Then, the model description looks like **Figure 3**.

Model is divided into three layers, architecture of which is a tree and the algorithms or protocols that look like leaf nodes include lot of attributes. These attributes which are carefully defined describe an algorithm or protocol. Because they are all the attributes describing algorithms or protocols in the models, and do not involve the concrete implements, the algorithm attribute can be considered as a finite set P and therefore, an algorithm or protocol can be defined as:

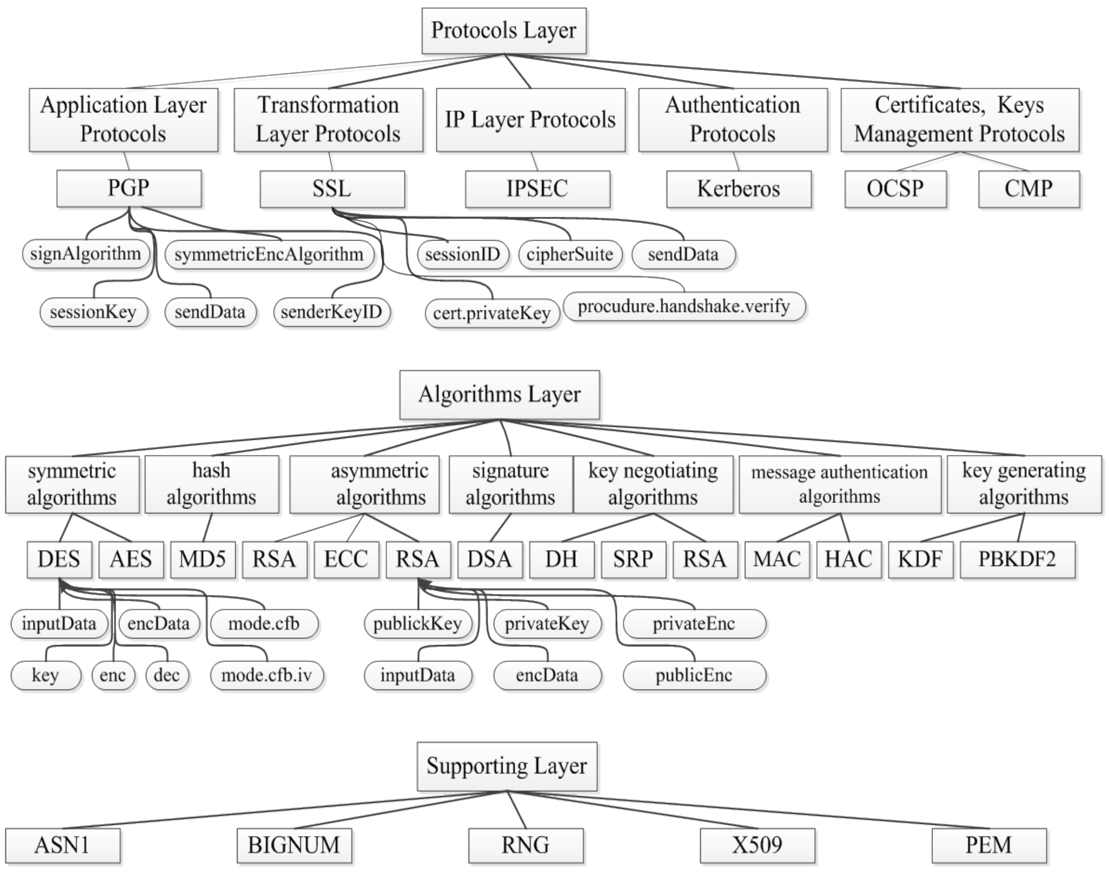

Figure 3. Description of the model's overall library architecture 3.

$$A = \{ p \mid p \in P \}$$

where p could only be the related attributes of the algorithm or protocol.

A contains a limited number of attributes, and A is the proper subset of P. Therefore, the definition of the model is the whole of algorithms and protocols which has been analyzed:

$M = \left\{ All\ the\ analyzed\ algorithms\ and\ protocols \right\}$

4. The Mapping Method of the Library Function Attributes to the Cryptographic Model and the Library Architecture Models

4.1. The Mapping of Single Function Attributes

Whether using the dynamic library calls or static links, when using the cryptographic library, what the program will first consider is to directly use its library function, which contains all the cryptology information. Considering the function prototype:

type CallingConvension funcName(*type param*1,...)

Cryptography information of code library functions is included in the function names, parameters and the return values of functions. Therefore, cryptography information can be extracted from these function attributes. Consider one function of OpenSSL:

int EVP_EncryptUpdate(*EVP_CIPHER_CTX*ctx*, *unsigned char *out*, *int *outl*, *const unsigned char *in*, *int inl*)

This function completes the core action of encryption and is bound to be used if encrypted by EVP mode. Its parameter ctx includes the information such as key, algorithm and so on which are needed for encryption. Parameter out is the result of encryption; Parameter in is encryption data; inl and outl are the corresponding data length. Design a method to map the parameter value of cryptographic library function or the value combination onto the description model algorithm nodes and algorithm attribute nodes, which is shown as the following mapping function:

EVP_EncryptUpdate{(*ctx->cipher.nid*): (*SYSTEM.NID*); (*SYSTEM.output out*): (*algorithm_layer. symmetric_algorithm.attribute.enc_data*); (*SYSTEM.input in*):
(*algorithm_layer.symmetric_algorithm.attribute.input_data*)}

SYSTEM.NID(*nid*){(*nid*==31): (*algorithm_layer.symmetric_algorithm.
DES*, *algorithm_layer.symmetric_algorithm.DES.CBC*);
(*nid*==30): (*algorithm_layer.symmetric_algorithm.
DES*, *algorithm_layer.symmetric_algorithm.DES.CFB*);…}

Give the definition of mapping function:

Mapping function: The approach of extracting the mapping relation with the description model nodes for a given series of value is called a mapping function. Mapping function has parameters and function body. Behind the name of mapping function are the parentheses and parameter list, which are followed by the braces with mapping function body in it, and in the mapping function body are map items separated by semicolons.

Mapping items: The internal mapping item is divided into the left and right sides by colon, with the left side the pre-defined relation of values, such as equal, range, and the right side the mapping relation from the iterative value to description model. If the right side value depends on the left one, then, put the right side into the braces. Inside the braces is still the dependency item, and the rightmost of it are the nodes of the description model. It shows the cryptographic algorithm knowledge of the description model which corresponds to this dependency chain, and divides multiple nodes in the description model with commas.

Every common cryptographic library function corresponds with a homonymous mapping function, and this kind of mapping function is called **library mapping function**. When the library mapping function with numbers of common mapping relations that can be extracted to be independent mapping function so that the mapping frequency can be reduced for easy understanding, then create a new mapping function, and this kind of mapping function is named the **self-created mapping function**.

The library mapping functions starts with its name and for the reason that its parameters are fixed, there is no parameter list exists. But because the input value of parameter and the output value when the function ends may have discrepancies in their meaning, it is needed to explain whether the parameter is input or output value, respectively marked with *SYSTEM. input* and *SYSTEM. output*, with *SYSTEM. input* as the default if neither.

In the situations where parameter values have only one single meaning, direct mapping can be applied to the nodes of the description model, forming a simple mapping item. When parameter values have different meanings or mutually dependent relationship exists among them, mapping cannot be applied directly unless by establishing multiple mapping items or iterative mapping. In the above examples, different values of the parameter *ctx. cipher. nid* have different meanings. Its mapping relation is relatively complex and will be applied in several places repeatedly. Therefore, it is extracted to form self-created mapping function. Set the keyword *SYSTEM* as the start of self-created mapping function, and then establish self-created mapping function named *nid*. The corresponding situation of partial *nid* in the library and cryptography knowledge is shown in **Table 5**.

The corresponding relation between the possible values of parameters in the mapping function and nodes in the corresponding description model is a one-to-many relationship, firstly because in the design of cryptographic library, a set of given values only have determined meanings, and secondly because the meanings of nodes in the cipher model are cryptography knowledge which has been decomposed into atomic types.

In relation to the dependencies between values, the values of some certain parameters determine the types or value meanings of some other parameters, and it can be seen in many libraries. For example, in the two calls of *crypt Set Attribute String* function in cryptlib, set the *salt* when the key was generated at first time and set the original string when the key was generated for the second time. As you see that the meaning of the third parameter depends on the value of the second parameter:

Prototype:

C_RET cryptSetAttributeString(*C_IN CRYPT_HANDLE cryptHandle*,*C_IN CRYPT_ATTRIBUTE_TYPE attributeType*,*C_IN void C_PTR value*, *C_IN int valueLength*)

cryptSetAttributeString(*cryptContext*, *CRYPT_CTXINFO_KEYING_SALT*, *salt*, *saltLength*);

cryptSetAttributeString(*cryptContext*, *CRYPT_CTXINFO_KEYING_VALUE*,*passPhrase*, *passPhraseLength*);

The mapping function:

cryptSetAttributeString{(*attributeType*, *value*):(*SYSTEM.ATTRSTRING*)}

SYSTEM.ATTRSTRING(*type*,*value*){

(*type*==1011){(*TYPE value*): (*algorithm_layer.* *symmetric_algorithm.attribute.original_key*)};(*type*==1010){…}}

Among the parameter values there exist dependencies. Put the parameter value that is relied on the left of the mapping item and the relying one in the braces behind. If the type of the dependent item also depends on the item depended on, the type should be given. For example, the capitalized TYPE in the above mapping item will appoint the type of the parameter value of value. This is because this situation exists in some functions.

If mapping items exist in the name and return value of the function, give it a specific name first. For example, the return value can be named *SYSTEM. Return Value*, and the specific function name can be named *SYSTEM. Function Name*. Then create normal mapping items for them. This mapping method can be used to map the function to the corresponding cryptographic library architecture as well. Merely there is only the mapping from the function name to the library architecture here, which is relatively simple.

4.2. Links of Relations between Multiple Functions

If there exists a relationship where certain algorithm is completed together among several functions, then in this way, the algorithmic information will be distributed across different functions and fail to be integrated. There are two kinds of cases where the function establish correlation with other functions. One is that a function correlates with the parameter of other functions by only one parameter. For example, in **Figure 4(a)**, the *funcA* correlates with *funcB* merely by parameter *M*. The other one is that a function correlates with others by multiple

Table 5. Meaning of id in the library 4.

ID Value	Meaning	Value	Time
31	“DES-CBC”	656	“DES-CFB1”
643	“DES-CDMF”	657	“DES-CFB8”
30	“DES-CFB”	29	“DES-ECB”

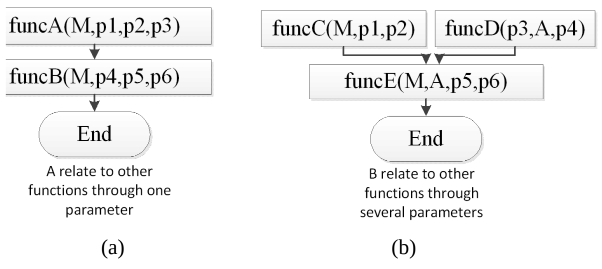
funcA(M,p1,p2,p3)
funcB(M,p4,p5,p6)
End
A relate to other functions through one parameter
(a)
funcC(M,p1,p2)
funcD(p3,A,p4)
funcE(M,A,p5,p6)
End
B relate to other functions through several parameters
(b)

Figure 4. Relation generated by different functions via parameters.

parameters. For example, in **Figure 4(b)**, the *funcE* correlates with *funcC* by parameter *M* and with *funcD* by parameter *A*. In the first case, all the algorithmic information is on this function chain. In the second case, algorithmic information maybe exist on two chains, or only on one of the two. Therefore, two types of interrelated parameters need to be distinguished. One is the main chain parameter, and the other is the assisted chain parameter.

Main Chain Parameters: The cryptographic algorithm or protocol information which is included in current function exists in the parameter or the function derived from the parameter, which is marked as *SYSTEM. mainLink*.

Assisted Chain Parameter: The cryptographic algorithmic information which is included in the current function is irrelevant to the parameter. The parameter may be part of other algorithms, which is marked as *SYSTEM. assistLink*.

Besides, in judging the same parameter, one method is needed to be established. For example, handle parameter should go according to handle value; pointer parameter according to address value or content value of pointer and second-level pointer according to value of its own secondary address or premier address value or the content value. Therefore, for this kind of parameters, classification still needs to be established, such as *SYSTEM.handl*e, *SYSTEM.pointer*1, *SYSTEM.content* and *SYSTEM.pointer*2. In general cases, it's okay to accord to the pointer value (handle value), but it may need to determine whether parameters are equal according to equality of pointed content in the case of existence of transferring secondary pointers.

If the parameters are emptied or reused in the process of making the chain according to equality in pointer parameter addresses, or if the structure copying exists to speed up the structure establishment in the process of making the chain according to equality in contents, then it needs to be clear whether the parameters have been used up in an application process, and an end marker needs to be given to this kind of parameters, such as *SYSTEM.endParam*, and a start marker is used to determine the initiation and termination of the chain, such as *SYSTEM.startParam*.

Therefore, in the mapping items, the variety mapping of parameters needs to be added, such as the above function *cryptSetAttributeString*, to which mapping items should be added:

cryptSetAttributeString{...;*cryptContext*: (*SYSTEM.mainLink*,*SYSTEM.pointer*1)}

4.3. The Theoretical Basis of the Mapping Relation

The mapping from the function attribute to the description model is an one-to-many multi-mapping relation. In the situation where the description model has already completely described the cryptographic algorithm or protocol, as for the algorithm or protocol *A* in the description model, if algorithm or protocol A_f is obtained by mapping in the cryptographic library function, then A_f is the subset of *A*. Give A_f the following definition:

A_f ={*p*|*p is the attribute of the map from function* (*or function chains*) *to the corresponding algorithms in description model*}

$A_f \subseteq A$

5. API Design and Application Examples

API provides analysis of the binary parameter data information and syntactical mapping, makes the analysis results the mapping between description model and library architecture model, and conducts the graphic information display of the mapping results by Graphviz. Taking the encryption algorithm of highlvl.c in the library cryptlib as an example, it uses DES algorithm to complete the calculation of encrypt data. Details are as follows:

```
Source Code:
    CRYPT_CONTEXT cryptContext;
    cryptCreateContext( &cryptContext,
            cryptUser,CRYPT_ALGO_DES );
    cryptEncrypt( cryptContext, data, dataLength );
    cryptDestroyContext( cryptContext );
Function Prototype:
    C_RET cryptCreateContext( C_OUT CRYPT_CONTEXT C_PTR cryptContext, C_IN CRYPT_USER cryptUser, C_IN
CRYPT_ALGO_TYPE cryptAlgo );
    C_RET cryptEncrypt( C_IN CRYPT_CONTEXT cryptContext, C_INOUT void C_PTR buffer, C_IN int length );
    C_RET cryptDestroyContext( C_IN CRYPT_CONTEXT cryptContext );
The mapping function of each function is as follows:
    cryptCreateContext
     {(SYSTEM.output cryptContext): (SYSTEM.startParam,
      SYSTEM.mainLink, SYSTEM.pointer1);
     (cryptAlgo): ( SYSTEM.cryptAlgo); }
    SYSTEM.cryptAlgo(id)
    {(id==1): (algorithm_layer.symmetric_algorithm.DES);
    (id==2): (algorithm_layer.symmetric_algorithm.3DES); …}
    cryptEncrypt
    {( cryptContext): (SYSTEM.mainLink, SYSTEM.pointer1);
    (SYSTEM.input data):
        (algorithm_layer.symmetric_algorithm.attribute.input _data);
    (SYSTEM.output data):
        (algorithm_layer.symmetric_algorithm.attribute.enc_data);
    (SYSTEM.functionName):
       (algorithm_layer.symmetric_algorithm.attribute.encFlag.enc); }
    cryptDestroyContext
    { cryptContext:
(SYSTEM.endParam, SYSTEM.mainLink, SYSTEM.pointer1); }
```

Combining the data recording technology on the dynamic binary platform, application model and mapping method, recovery results are shown in **Figure 5**, which contains all kinds of information of DES algorithm, data values of function parameters that have been recovered and the module situation where the function is categorized into the teammate architectural model.

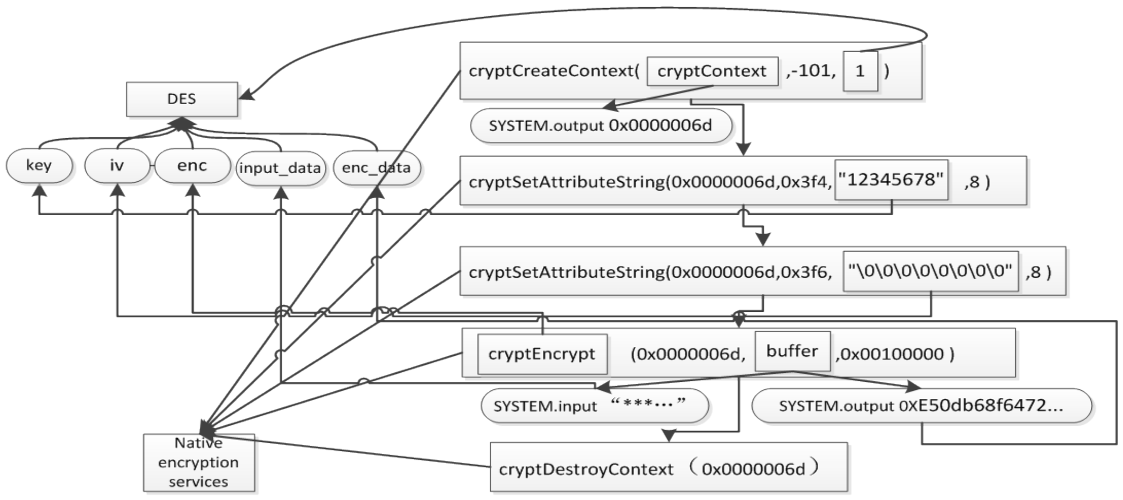
DES
key
iv
enc
input_data
enc_data
cryptCreateContext(cryptContext ,-101, 1)
SYSTEM.output 0x0000006d
cryptSetAttributeString(0x0000006d,0x3f4, "12345678" ,8)
cryptSetAttributeString(0x0000006d,0x3f6, "\0\0\0\0\0\0\0\0" ,8)
cryptEncrypt (0x0000006d, buffer ,0x00100000)
SYSTEM.input "***…"
SYSTEM.output 0XE50db68f6472…
Native encryption services
cryptDestroyContext (0x0000006d)

Figure 5. Cryptlib results of information recovery.

6. Conclusions

If the model and mapping methods could use the cryptographic library in application, algorithmic information and library architecture information in the library function shall be extracted and exhibited at the same time for the comprehension convenience of relevant personnel. It can be used for analysis of the relevant codes of some virus software.

The model interface could be applied to the function information query after IDA signature identification and those queries, displays of some dynamic debugging tools like Ollydbg and Windbg.

References

[1] Brooks, R.R. (2013) Introduction to Computer and Network Security. CRC Press.

[2] Dwivedi, A., Dwivedi, A., Kumar, S., *et al.* (2013) A Cryptographic Algorithm Analysis for Security Threats of Semantic E-Commerce Web (SECW) for Electronic Payment Transaction System. *Advances in Computing and Information Technology*, Springer Berlin Heidelberg, 367-379. http://dx.doi.org/10.1007/978-3-642-31600-5_36

[3] Li, J.-Z. and Shu, H. (2012) The Research of Crypto Algorithm Recognition Technology. *The Security of Information Network*, **11**, 46-49.

[4] Halderman, J.A., Schoen, S.D., Heninger, N., *et al.* (2009) Lest We Remember: Cold-Boot Attacks on Encryption Keys. *Communications of the ACM*, **52**, 91-98. http://dx.doi.org/10.1145/1506409.1506429

[5] Guilfanov, I. (2015) FLIRT: Fast Library Acquisition for Identification and Recognition. https://www.hex-rays.com/products/ida/tech/flirt/in_depth.shtml

[6] Nethercote, N. (2004) Dynamic Binary Analysis and Instrumentation. *Technical Report*, University of Cambridge, UK.

[7] Nethercote, N. and Seward, J. (2007) Valgrind: A Architecture for Heavyweight Dynamic Binary Instrumentation. *ACM SIGPLAN Notices*, **42**, 89-100. http://dx.doi.org/10.1145/1273442.1250746

[8] Gröbert, F., Willems, C. and Holz, T. (2011) Automated Identification of Cryptographic Primitives in Binary Programs: Recent Advances in Intrusion Detection. Berlin: Springer Berlin Heidelberg, 41-60. http://dx.doi.org/10.1007/978-3-642-23644-0_3

[9] Calvet, J., Fernandez, J.M. and Marion, J.Y. (2012) Aligot: Cryptographic Function Identification in Obfuscated Binary Programs. *Proceedings of the* 2012 *ACM conference on Computer and Communications Security*, 19*th ACM Conference on Computer and Communications Security*, 2012, 169-182. http://dx.doi.org/10.1145/2382196.2382217

[10] Allen, R.J. (1997) A Formal Approach to Software Architecture. Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh.