Evaluating Approaches for Identifying Cryptographic Functions in Binaries

Yongming Fan
Purdue University
fan322@purdue.edu

Priyam Biswas
Intel
priyam.biswas@intel.com

Christina Garman
Purdue University
clg@cs.purdue.edu

Abstract—**Cryptographic functions are instrumental in securing our electronic communications and systems; yet time and time again they are mis-used, mis-implemented, or created in an ad-hoc manner. Additionally, while cryptography plays a fundamental role in securing systems, it is unfortunately also used for malicious purposes, such as hiding payloads in malware. Many such instances occur in closed-source code or binary applications, which inherently present a challenge for independent audit and analysis. Therefore, detecting the presence of cryptographic functions in a binary application can be both an indicator of malicious behavior as well as a point of interest for cryptographic analyses and vulnerability discovery.**

While general purpose binary analysis and function identification techniques are broad and thriving areas that could help address these challenges, a variety of work across industry and academia has focused on a subset of this space: developing techniques and tools specifically tailored to identifying different cryptographic primitives in binary applications. Despite the relative popularity of this work and recent advances in the field, it still lacks consistent means of evaluation and comparison across tools. As such, we focus on exploring the experimental approaches and methodologies in cryptographic function detection. First, we investigate all experiments and performance evaluations conducted by previous cryptographic detection tools to understand the current experimental methodologies, designs, and datasets used for these evaluations. We identify the limitations of previous experiments, and based on our analysis, we conduct comprehensive reproduction and replication studies on all existing work in this space from multiple perspectives. The R+R study not only helps us understand the current state of cryptographic function detection tools but also provides deeper insights into experimental approaches and methodologies. We highlight the experimental methodologies, evaluation, and results with the goal of encouraging improvements in research experiments related to cryptographic function detection in binaries.

I. Introduction

Cryptography has found itself at the core of secure modern technology and both plays a key role in securing communications and is imperative to many modern systems and applications. From data integrity to authentication and online banking to messaging friends, cryptography is everywhere. Well-known cryptographic libraries, such as OpenSSL [1], are widely used not only to generate TLS certificates and validate certificate information, but also to implement cryptography in general purpose applications. Despite their universal benign usage, cryptographic functions are also heavily used in malware and to evade security protocols. In recent ransomware attacks [2], [3], cryptographic functions have been used to encrypt a victim's information and later asked to pay ransom in exchange of recovering their files and information.

On the one hand, the usage of cryptographic functions makes it easier to carry out secure and private operations, and on the other hand, its malicious usage by bad actors makes it harder for cryptography and security experts to perform forensic analyses and reverse engineer code. Hence, the ability to automatically identify or detect the presence of cryptographic functions in binary applications can be a crucial part of the security experts' arsenal. It can assist in binary analysis to give a better depiction of how the functions work. Determining the type of cryptographic function in a binary can help to pinpoint the existence of a malicious payload [4] and also help cryptographic experts identify potentially insecure protocols or implementations (or those that require more manual analysis).

While general purpose binary analysis and function identification techniques are themselves broad and thriving areas that could potentially solve these problems, a variety of work across industry and academia has focused on solving these challenges more specifically: developing techniques and tools that are uniquely tailored to identifying *cryptographic primitives* in binary applications. This allows researchers to often provide better results in different domains, leveraging cryptography-specific features. Cryptographic functions tend to have many mathematical computations, nested loops, and exclusive input-output mappings which are distinct from non-cryptographic functions and can thus be useful as a means of identification. Harvey et al. [5] first utilized magic constants to identify cryptographic primitives in 2001. Later works focused on signature based detection mechanisms. However, due to the limitations of signature based approaches, subsequent work placed a greater emphasis on heuristic based detection, such as identification of basic blocks, instruction patterns, etc. With the popularization of modern machine learning algorithms, newer works utilized deep learning and AI to extract cryptographic features from a binary.

Unfortunately, while there has been a wealth of work in the space, it has generally failed to keep up with modern cryptographic innovations and applications, as well as current software engineering and optimization techniques. This can be difficult to notice as there is no standardized benchmarking or comparison framework, and individual works tend to test on algorithms that they are best at identifying, while neglecting those they cannot. Additionally, as many techniques are tailored for specific cryptographic algorithms, architectures, or settings, which change rapidly over time, reproducing past results can be challenging, making it difficult to provide comparisons across the growing landscape as well.

Given the significance of these tools and techniques for both researchers and practitioners, we aim to reproduce and replicate all existing cryptographic function detection tools in this domain. First, we provide a comprehensive analysis that categorizes and discusses the existing body of work, highlighting its strengths, weaknesses, and employed techniques. To ensure fair and effective experimentation, we examine the evaluation approaches used in prior cryptographic function detection tools and develop a comprehensive, standardized benchmarking framework. This framework facilitates a more consistent comparison of existing and future tools, identifies research gaps, and offers insights into designing effective experiments for cryptographic function detection. We further explore key experimental design questions, analyzing the methodologies used to obtain research results. Finally, we summarize our new findings based on the evaluation results, identifying potential challenges and various factors that may influence performance.

Our contributions. Our primary contributions are:

- We conduct a thorough examination of cryptographic function identification methods, exploring the range of techniques utilized and evaluating their respective strengths and limitations.
- We explore the previous experimental approaches and methodologies used to obtain detection and performance results.
- We create a standardized suite of performance metrics and benchmarks to evaluate the effectiveness of current detection mechanisms and analyze existing tools based on this suite.
- We conduct comprehensive replication and reproduction studies on all existing work.
- We conduct an in-depth analysis and discussion of how to design a better evaluations and experiments to obtain comprehensive results for cryptographic function detection in binaries.

This paper: We present this work as a supplement to our main research contributions in ACSAC 2024. While the structure of this paper is largely similar to that of [6], the content has been added, removed, and reorganized as to be more useful for an experiments-focused reader. Please refer to our ACSAC 2024 paper [6] for additional details on content omitted from this work.

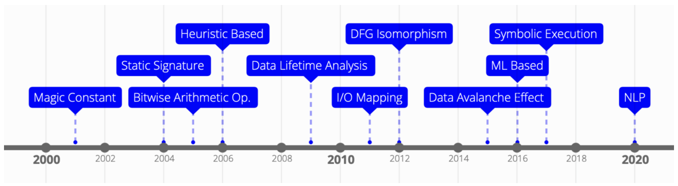

Fig. 1. Evolution of cryptographic function identification techniques

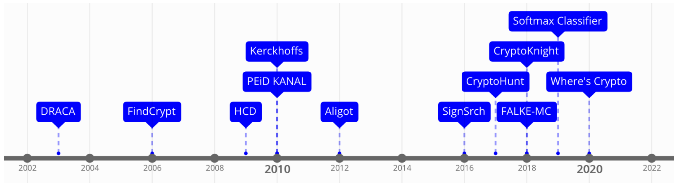

Fig. 2. Timeline of the development of cryptographic algorithm detection tools

II. Background

In this section, we provide background on and a classification of detection approaches that have been employed by existing tools to date. Figure 1 shows the evolution timeline of various techniques and approaches used in cryptographic function identification research whereas Figure 2 shows the development timeline of all studied tools. We also provide an overview of some of the cryptographic features that existing tools focus on, to help better understand how they work and contextualize later discussion.

Cryptographic primitive vs. cryptographic function. A brief note on notation used throughout this paper. When we refer to a *cryptographic function* or *cryptographic algorithm*, we are generally referring to the entirety of a specific scheme or class of algorithms, such as RSA or hash functions. When we say *cryptographic primitive*, we generally mean a building block for a larger cryptographic scheme, such as a Feistel network.

A. Categorization of detection approaches

Precision, scalability and performance of an identification approach all rely on the underlying techniques used. Detection approaches can be divided into three categories: i) Dynamic, ii) Static, and iii) Machine learning based approaches.

Static Approaches. Static approaches perform various forms of static analysis to detect any static signatures, such as 'magic' constants, instruction sequences, and different code structures such as S-boxes, as a means to identify cryptographic algorithms. This type of approach is based on the assumption that cryptographic functions will perform a large number of arithmetic computations compared to non-cryptographic functions. Harvey et al. [5] first proposed the idea of identifying cryptographic algorithms in binary files. They focused on finding algorithms based on their constant characteristics. By taking advantage of feature matching, subsequent works [7], [8], [9] primarily focused on protocol reverse engineering. Chang et al. [10] created a library of 3000 plus signature characteristics of cryptographic algorithms and also explored recognizing cryptographic algorithms based on Minimum Perfect Hash Function (MPHF). Lestringant et al. [11] used data-flow isomorphism to find symmetric key algorithms.

Off-the-shelf tools such as Draft Crypto Analyzer (DRACA) [12], Kanal [13], Kerckhoffs [14], Hash & Crypto Detector (HCD) [15], Signsrch [16], and Findcrypt [17], utilize static signature patterns of different cryptographic functions. Static based approaches have low performance overhead. However, these detection mechanisms can be easily bypassed using simple obfuscation techniques. For example, the authors of Aligot [18] showed that simple data obfuscation techniques can bypass static analysis based detection mechanisms.

Dynamic Approaches. Dynamic approaches [19], [18], [20] primarily focus on identifying cryptographic primitives from execution traces. Lutz et al. [21] first applied dynamic analysis to identify cryptographic algorithms based on three indicators: i) presence of loops, ii) changes in entropy, and iii) high ratio of bitwise arithmetic instructions. Based on the data avalanche effect, CipherXRay [20] pinpoints the boundary of cryptographic operations and recovers transient cryptographic secrets. However, this approach does not work in the case of stream ciphers, for example, as they do not show any data avalanche effect.

Gröbert et al. [22] proposed heuristic based approaches on both generic characteristics of cryptographic code and on signatures for specific instances of cryptographic algorithms by mapping input-output (I/O) relations. Aligot [18] further extends this idea of I/O mapping. It retrieves I/O parameters in an implementation-independent fashion, and compares them with known cryptographic functions as well as performs an interloop data flow analysis. CryptoHunt [19] used bit-precise symbolic loop mapping to identify cryptographic functions and applies guided fuzzing to make the solution scalable. Park et al. [23] proposed a hardware-assisted tracing technique to detect symmetric-key cryptographic routines in anti-reverse engineered binaries via recording the change of flow instructions from the CPU at run-time. Dynamic approaches in general perform better than static based approaches in obfuscated binaries, but suffer from much greater performance overhead.

Machine Learning Based Approaches. With the growing popularity of machine learning, several recent works have explored utilizing such techniques. Shin et al. [24] discussed how recurrent neural networks can identify functions in binaries with greater accuracy and efficiency. However, their approach was generalized for any function identification. For the purpose of cryptographic function identification, Wright et al. [25] proposed an artificial neural network model to classify functional blocks as being either cryptographic or not by extracting the frequency of different logic instructions from a disassembled program. Benedetti et al. [26] used the ‘grap’ tool to detect cryptographic algorithms by creating patterns for AES and ChaCha20. Falke [27] proposed a neural network based approach by modeling classifiers for arbitrary cryptographic algorithms from sample files and then automatically extracting features to train the neural network. It offered a high detection rate in combination with a low false positive rate. Hill et al. [28] proposed a Dynamic Convolutional Neural Network based learning system (CryptoKnight) which learns from new cryptographic execution patterns to classify unknown software. Jia et al. [29] proposed an NLP-based approach which first extracts the semantic information of assembly instructions and then transfers them into 100-dimensional vectors and later uses K-Max-CNN-Attention to classify cryptographic functions.

B. Cryptographic Features

Cryptographic functions have their own (often quite distinct) set of characteristics, and work thus far has often leveraged these for identification, as they can lead to more tailored or performant techniques than general purpose binary analysis and code identification. These include things like magic constants, loops, entropy changes, and specific instruction sequences. Many of these features only pertain to certain algorithms and cannot be used to identify others. See our ACSAC 2024 paper [6] for a detailed discussion of these features.

C. Challenges of Detecting Cryptographic Function in Binary

Identifying cryptographic functions has its own sets of challenges. Here, we discuss some of the challenges.

1) Obfuscation: One of the major challenges in cryptographic function identification is obfuscation. The initial purpose of obfuscation was to protect software intellectual property [30] from malicious reverse engineering attempts. However, malware authors adopted obfuscation techniques as a way to avoid detection. Various types of obfuscation techniques have been implemented to thwart any form of detection. These obfuscation techniques can be primarily categorized as: control-flow obfuscation, data obfuscation, and layout obfuscation. For a variety of reasons, one might still wish to identify cryptographic code even in the face of such obfuscation techniques, particularly given its prevalence nowadays.

Control-flow obfuscation is the most common type of obfuscation. A control-flow graph (CFG) [31] embodies the graphical representation of the flow of a program. To hinder any form of CFG-based detection, malware authors introduce various techniques such as including bogus control-flow [32], control-flow flattening [33], and opaque predicates [34].

Similar to control flow obfuscation, data obfuscation techniques, like data aggregation and data splitting, attempt to hinder approaches based on input/output relationships.

The final form of obfuscation is layout obfuscation [35], which scrambles the layout of instructions in the program while keeping the original syntax intact. Layouts can be obfuscated by adding meaningless classifiers, stripping redundant symbols, separating related code, as well as adding junk code.

2) Implementation Variation: Despite having a well-defined specification, one cryptographic algorithm can be implemented in a number of ways while still achieving the same result. In addition, cryptography is not always implemented correctly or exactly to spec. For example, buggy implementations of the

TEA algorithm were found [18] in malware such as Storm Worm and Silent Banker. Hence, even if a detection approach can find an ideal implementation of an algorithm, there is no guarantee it can also detect all the implementation variations of the same algorithm. We believe it is valuable to be able to identify both intended implementation variations, as well as as incorrect (but close) ones.

3) Differences in Cryptographic Functions: There are fundamental differences in different classes of cryptographic algorithms. Cryptographic features in one algorithm may not be present in the other one. For instance, the data avalanche effect [20] which says that an insignificant change in the input parameters can make significant differences in the output, works well for identifying block ciphers. In the case of stream ciphers, there are no such observations like data avalanching. Tools must be designed to account for these differences if they wish to identify broad classes of algorithms.

4) Compilers: The choice of compiler can significantly impact the binary structure of a program in various ways such as format, linking, inline functions, and compatibility with different architectures. The incorporation of anti-analysis techniques further adds to the complexity of cryptographic function detection. Consequently, working with different compilers can introduce numerous challenges. Possessing knowledge about compiler behavior becomes essential for reverse engineers, security analysts, and cryptographic function detection tools to effectively identify cryptographic functions within a binary program.

III. Overview

This paper focuses on the design of cryptographic function detection evaluation benchmarks and experiments. To support this goal, we investigate the evaluation approaches used by previous tools, analyze their gaps and weaknesses, propose a novel analysis framework aimed at facilitating a comprehensive evaluation of the replicability of cryptographic function detection tools, and discuss the experiment experience during the evaluation.

A. Research Experiment Questions

In this paper, we not only discuss the reproduction and replication evaluation results of prior cryptographic function detection tools but also primarily focus on experimental design to ensure that evaluations effectively reflect the actual detection capabilities of these tools. To ensure a comprehensive and thorough study on cryptographic function detection, we take into account several key questions as part of our experimental design to guide our analysis and evaluation.

- R1. Experimental methodologies and design
- R2. Use of specialized hardware or software system
- R3. Software tools used to perform experimentation
- R4. Approaches to experiment validation, monitoring, and data collection
- R5. Datasets used and/or developed to perform experimentation
- R6. Measurements and metrics

To additionally enhance our insights, we will also discuss the following questions.

- Q1. Does the work use experimentation artifacts borrowed from the community?
- Q2. Does the evaluation attempt to replicate or reproduce results of earlier research?
- Q3. What can be learned from the methodology and experience using the methodology?
- Q4. Are there any intermediate results including possible unsuccessful tests or experiments?

B. Cryptographic Algorithm Detection Tools

A number of cryptographic detection tools have been developed based on the different approaches discussed in Section II-A. We perform our reproduction and replication studies on all existing work that we were able to find from 2000 until now:

- Aligot [18]
- CryptoHunt [19]
- CryptoKnight [28]
- DRACA [12]
- FindCrypt2 [17]
- FALKE-MC [27]
- HCD [15]
- Kerckhoff [14]
- PEiD KANAL [13]
- SignSrch [16]
- Softmax Classifier [29]
- Where's Crypto [36]

Our focus is on evaluating tools specifically designed for identifying cryptographic functions and comparing these tools with their stated performance. As such, we exclude other (broader) binary analysis tools, such as Binshape [37], a general-purpose binary analysis tool, or CIS [38], which focuses on testing cryptographic heuristics rather than actual algorithms.

To provide a comprehensive understanding of the overall features and characteristics of each tool, we also employ a systematic approach to categorize, classify, and interrelate the diverse array of knowledge elements that constitute such tools from several directions in Section III-C.

We comprehensively summarize the detection abilities of each cryptographic function detection tool. We did a thorough check of each tool's documents and signatures to see what types of cryptographic algorithms they can find. Figure 3 shows the distribution of various algorithms that existing work has focused on. After that, we gather all the information we found and explain it in more detail in Table VI. This helps us better understand what each tool can do in terms of detecting different cryptographic algorithms.

C. Categorization of Cryptographic Algorithm Detection Tools

We employ a systematic approach to categorize, classify, and interrelate the diverse array of knowledge elements that constitute such cryptographic detection tools from several directions. Table I contains an overview and summary of each tool, including its reproduction and replication status in this work, as well as a variety of information such that one can easily find or organize existing (and future) tools based on the per-column categories.

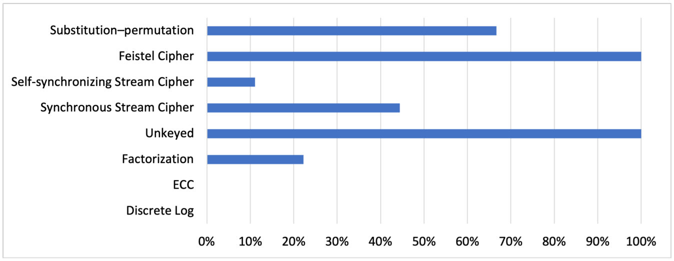

Fig. 3. Types of cryptographic functionality and designs targeted by different tools and approaches, by percentage of tools surveyed

We begin by categorizing the detection approach (broken down by the different methods discussed previously in Section II-A) and the programming language in which the tool's source code is written. We note that some tools are not open source; they provide only an executable file, and as a result, we lack information about the source language. Some tools require specific dependent libraries to run, which can limit or hinder their usability, and as such we list if there are any required dependencies. Some tools are installation-free and do not require compilation, offering an easy, user-friendly experience. And a few tools provide the capability for developers to insert new cryptographic algorithm signatures, thereby expanding their detection capabilities as the field advances.

We categorize these tools based on their design and requirements, as these factors can impact their reproducibility and replication capabilities in our evaluation. Additionally, limited reproducibility may hinder their usability for other researchers or users, making it difficult to run the tools, verify results, and use for further detection.

Takeaway: (R3) The taxonomy table provides an overview of the basic information and requirements for each tool. Based on our summary, some cryptographic function detection tools rely on additional dependent software, such as IDA Pro and Intel Pin, which are essential for their functionality. These dependencies are integral to our reproduction and replication (R+R) study, as they enable us to accurately investigate and evaluate the performance of these tools under the conditions specified in their original experiments.

IV. Previous Evaluation

We first investigate the experimental methods and evaluation approaches used by previous tools to gain a better understanding of the existing research and development in this field. By carefully examining these approaches, we aim to identify common practices, strengths, and potential areas for improvement. In this section, we provide a detailed investigation and discussion of all 12 tools that are listed in Section III-B. Our analysis includes how these tools were tested, the metrics used to evaluate their performance, and any limitations or challenges they faced. This comprehensive review helps us build a clearer picture of the progress made so far and lays the foundation for improving future studies and tools.

In Table II, we summarize the evaluation benchmarks and metrics selected by each tool in their respective evaluation experiments. We categorize the evaluation experiments of each tool into four main categories:

- Whether the tool was evaluated using at least some binary programs containing basic cryptographic functions.
- Whether the tool was evaluated using real-world applications to assess performance.
- Whether additional metrics, such as optimization flags, were considered in the tool's evaluation.
- Whether the tool's performance was cross-compared with previously developed tools.

Detailed information regarding these experiments will be discussed in the following sections.

A. *Lack of Evaluation Experiments*

Tools developed by industry organizations are often not evaluated by experiments, which limits the understanding of their true performance and reliability. However, these tools frequently come with documentation or instructions provided by their developers, outlining their claimed detection capabilities for single cryptographic algorithms and intended use cases. This information, although not derived from formal testing, can serve as ground truth data for further R+R evaluations. The detailed claims regarding the detection capabilities for each cryptographic function are discussed in our ACSAC '24 paper [6].

B. *Evaluation with Basic Cryptographic Algorithms*

Most tools have been evaluated using a selection of basic cryptographic algorithms to demonstrate their ability to detect cryptographic functions in binaries. These basic algorithms serve as the foundational and critical components for assessing the detection performance of cryptographic function detection tools. However, previous studies have used a variety of different cryptographic algorithms in their evaluations, making it challenging to compare the performance of these tools across each other due to the lack of standardized benchmarks.

Furthermore, the selection criteria of previous evaluations were not clearly explained, leaving questions about why a specific algorithm was chosen over others. This inconsistency has also resulted in recent tools missing many popular and important cryptographic algorithms in their evaluations. Consequently, it remains unclear whether certain cryptographic function detection tools are limited to detecting only the cryptographic functions included in their chosen benchmarks or if their capabilities extend beyond these specific selections. In Table III, we provide a list of the cryptographic algorithms selected by each tool in their evaluation experiments. This compilation highlights the specific algorithms used to assess the performance of the tools and underscores the variations in selection across different studies. The tools listed in Section III-B but not included in Table III did not conduct any evaluation experiments.

Tool	Method	Dependent Library	Language	Requires Compiling	Expandable	R+R	Obf. Binary Design	AI/ML Approach
Aligot	I/O Parameters	Intel Pin	Python	Yes	Yes	Reproducible	No	No
CryptoHunt	Bit-precise Symbolic Loop	Intel Pin	C++	Yes	Yes	Unable	Yes	No
CryptoKnight	Machine Learning Based	Intel Pin, PyTorch	Python	Yes	Yes	Both	No	Yes
DRACA	Unknown	None	Executable	No	No	Replicable	No	No
FindCrypt2	Magic Constants	IDA Pro	C++	Yes	No	Replicable	No	No
FALKE-MC	Neural Network	None	Unknown	Unknown	Unknown	Unable	Yes	Yes
HCD	Unknown	None	Executable	No	No	Replicable	No	No
Kerckhoff	Multiple	Intel Pin	Python	No	Yes	Unable	No	No
PEiD KANAL	Signature Searching	None	Executable	No	Yes	Replicable	No	No
Signsrch	Signature Searching	None	Executable	No	No	Replicable	No	No
Softmax Classifier	Neural Network	Unknown	Python	Unknown	Unknown	Unable	No	Yes
Where's Crypto	DFG Isomorphism	IDA SDK	C++	Yes	Yes	Both	No	No

TABLE I
CATEGORIZATION OF EXISTING TOOLS AND THEIR IMPORTANT PROPERTIES

Evaluation Criteria	Cryptographic Function	Real-world Application	Evaluation Metrics	Cross Comparison
Aligot	✓		✓	✓
CryptoHunt	✓	✓	✓	✓
CryptoKnight	✓			
DRACA				
FindCrypt2				
FALKE-MC	✓	✓	✓	
HCD				
Kerckhoff	✓	✓		
PEiD KANAL				
Signsrch				
Softmax Classifier	✓			
Where's Crypto	✓			

TABLE II
EVALUATION BENCHMARKS AND METRICS SELECTION BY PREVIOUS CRYPTOGRAPHIC FUNCTION DETECTION TOOLS

C. Evaluation with Real World Applications

In addition to basic cryptographic algorithms, some tools have been evaluated using real-world applications to provide a more comprehensive assessment of their performance. For example, CryptoHunt was evaluated using cryptographic algorithms such as AES and MD5, implemented by various libraries, including OpenSSL and Libgcrypt. Similarly, FALKE-MC was tested with cryptographic algorithms developed by different developers. This evaluation design offers a more accurate evaluation of performance across diverse metrics, as it accounts for practical scenarios and variations in implementation settings.

However, similar to the evaluation with basic cryptographic algorithms, previous studies did not specify the selection criteria for the real-world applications used in their experiments. As a result, it remains unclear why certain applications were chosen and how they contribute to a more accurate performance evaluation of cryptographic function detection tools. Additionally, these evaluations appear to lack the inclusion of real-world applications, such as non-cryptographic binary programs that have behavior patterns similar to cryptographic functions or large-scale projects that could introduce runtime complexity challenges. The absence of these scenarios highlights a critical gap in the evaluation process, limiting its ability to assess tool performance under diverse and practical conditions.

D. Measurements and Metrics Design in Previous Experiments

In addition to considering the selection of evaluation benchmarks, other evaluation metrics such as obfuscation and compiler version also play a crucial role in influencing the results of cryptographic function detection. For example, optimization and obfuscation can significantly affect the analysis results for cryptographic function detection [39]. Several tools have specifically addressed these challenges by developing advanced techniques that demonstrate improved performance in detecting cryptographic functions within binary applications compiled with optimizations. For instance, CryptoHunt [19] focuses primarily on detecting cryptographic functions in obfuscated binaries. Similarly, Aligot [18] and FALKE-MC [26] consider optimization flags and include optimized binary programs in their evaluation experiments.

Even though some tools consider certain metrics in their evaluations, their evaluation metrics do not comprehensively address all the challenges associated with cryptographic function detection discussed in Section II-C. This oversight leaves potential areas unexplored, resulting in gaps in the evaluation process and a lack of understanding of the tools' performance under certain conditions. Addressing these unevaluated areas is crucial for advancing the field and ensuring robust assessments of cryptographic function detection tools.

Table IV summarizes the types of optimization flags that have been considered by previous tools in their evaluation studies, providing insights into the approaches taken to address the impact of optimization on detection accuracy.

Takeaway: (Q1) We investigate the experimental evaluations and artifacts from previous tools as a foundational step before proposing a new evaluation benchmarking framework. This investigation allows us to understand the methodologies, approaches, and metrics used in prior studies, enabling a deeper understanding of their limitations and gaps. By identifying these shortcomings, we can address the challenges present in the evaluation experiments conducted by earlier tools, ensuring that our proposed framework is more robust, comprehensive, and capable of facilitating a standardized comparison of cryptographic function detection tools.

Tool	AES	Blowfish	DES	MD5	RC4	RC5	RSA	SHA1	SHA256	TEA
Aligot	✓			✓	✓					✓
CryptoHunt	✓			✓	✓		✓			✓
CryptoKnight	✓	✓		✓	✓		✓			
FALKE-MC	✓						✓			
Kerckhoff	✓		✓	✓	✓		✓			
Softmax Classifier		✓	✓	✓				✓	✓	

TABLE III
CRYPTOGRAPHIC ALGORITHMS SELECTED IN PREVIOUS EVALUATION EXPERIMENTS

Tool	O1	O2	O3	Os	Ofast	AsProtect
Aligot						✓
CryptoHunt	✓	✓				
FALKE-MC	✓	✓	✓	✓	✓	
Kerckhoffs	✓	✓	✓			

TABLE IV
OPTIMIZATION FLAGS SELECTED BY EVALUATION EXPERIMENTS

V. ANALYSIS FRAMEWORK FOR OPEN-SOURCE DETECTION TOOLS

Based on our investigation in Section IV, previous tools have primarily been evaluated using a limited set of basic cryptographic algorithms, often lacking assessments for false positives, real-world applications, and large-scale projects. To address these shortcomings, as the first step in our reproduction and replication (R+R) study, we developed a new benchmarking framework with a comprehensive set of evaluation metrics. This framework builds upon the benchmarks and metrics used in prior evaluations while expanding their scope to establish a more rigorous and unified evaluation standard. By incorporating and enhancing previous methodologies, our R+R study ensures continuity with prior research while providing a broader, more detailed assessment of cryptographic function detection tools.

A. Benchmarking Framework

During our study, we identified a significant gap in the field: the absence of a standardized method for evaluating existing tools and their efficacy. We believe that to do a fair evaluation in any domain requires a standard benchmark. We propose a benchmarking framework that will help us understand the scalability and effectiveness of any given detection approach. We have four categories in our framework: i) basic cryptographic functions, ii) microbenchmarks, iii) libraries, and iv) large projects. Our framework is flexible and can be easily updated as the field of cryptography advances.

Basic cryptographic functions. The basic cryptographic functions category contains cryptographic algorithms from various standard classes within the cryptographic space. A single cryptographic algorithm may have multiple versions, or may use non-standard implementations, so we may select many different implementations or versions for certain algorithms. The cryptographic algorithms we select are:

- AES256
- DES
- ECC
- MD5
- RC4
- RC5
- RSA
- SHA1
- SHA256
- TEA
- XTEA
- XXTEA

We have chosen these algorithms for several reasons. First, we have included those recommended by the National Institute of Standards and Technology (NIST) [40], as they are widely utilized and cover a broad range of real-world applications. Second, we have incorporated algorithms like TEA and MD5, which, although once popular, are now considered unsafe or outdated, but still might have historical significance in applications like malware. Last, we have selected various variants of a single cryptographic algorithm to assess a tool's ability to handle different implementations effectively.

Microbenchmarks. Our microbenchmarks category contains small programs on file manipulation, networking, I/O heavy programs, math heavy programs, matrix and array, and so-called "golden implementations" of well-known cryptographic algorithms. We have chosen small programs such as I/O heavy and math-heavy programs since they have similar instruction sets or behavior patterns as cryptographic functions. This allows us to test for false positives in a detection framework.

Libraries. We have chosen cryptographic libraries as well as encoding and compression libraries that have similar behavior to cryptographic functions including:

- openssl (3.3.1)
- libgcrypt (1.8.11)
- libsodium (1.0.20)
- mbedTLS (3.6.1)
- gnuTLS (3.8.6)
- bzip2 (1.0.8)
- zlib (1.3.1)
- ffmpeg (7.0.2)
- libgsm (1.0.17)
- libjpeg (6b)
- libpng (1.6.43)

This allows us to again test for false positives, but also to test for a variety of different cryptographic algorithms implemented in different ways. We select open source libraries only, so that we are able to compile them based on our evaluation metrics. For each tool, we used the latest version available at the time of evaluation. Our selection includes a diverse range of libraries, from those with frequent recent updates, such as OpenSSL, to those that are a bit older, such as libjpeg.

Large projects. We also select Signal, an application designed for encrypted messaging, to evaluate a tool's ability to both scale for large codebases as well as still detect cryptography within them. We believe that to understand the scalability of a mechanism, it is crucial to determine that the tool performs well irrespective of the code size and its applications. Signal was selected because it both utilizes multiple (modern) cryptographic schemes and is open source. This again means we can recompile Signal with our evaluation metrics, plus establish ground truth for testing. However, as our framework is open-source, one could easily extend it to include any future desired large-scale open-source projects, such as Tor.

Takeaway: (R5) We developed a new evaluation benchmarking framework comprising four distinct categories: basic cryptographic functions, microbenchmarks, libraries, and large-scale projects. Each benchmark within this framework is compiled into a binary program, creating a comprehensive dataset for use in our reproduction and replication (R+R) study. This structured approach ensures that our evaluation captures a wide range of scenarios, enabling a thorough assessment of cryptographic function detection tools across diverse contexts and levels of complexity.

B. *Evaluation Metrics*

Obfuscation mechanisms and optimization flags significantly alter the binary structure, impacting its readability and analyzability. Obfuscation introduces transformations such as instruction substitution, control flow flattening, and bogus control flow, which obscure the logical flow and increase complexity, making reverse engineering and static analysis more challenging. Optimization flags, on the other hand, aim to enhance performance or reduce binary size by reordering instructions, inlining functions, or removing redundant code. While obfuscation adds extraneous complexity, optimization typically simplifies or restructures code, potentially stripping out metadata and debugging symbols. Together, these techniques can drastically modify the binary's layout and behavior, posing challenges for analysis tools.

To ensure a thorough exploration of current tools (and to provide rigorous and fair evaluation metrics for the future), we have meticulously crafted a series of evaluation experiments. These experiments are thoughtfully designed to align with the challenges we discuss in Section II-C, maximizing our ability to uncover the current state of these tools and elucidate their strengths and weaknesses. Our evaluation metrics are as follows:

- **Based on optimization levels:** Evaluate based on different compiler optimizations including -O0, -O1, -O2, -O3, -Os, and -Ofast
- **Based on different combinations of obfuscation mechanisms:**
  - **Control-flow obfuscation:** Adding bogus control-flow, control-flow flattening, substitution of instructions, polymorphism
  - **Data-flow obfuscation:** Data aggregation, data-splitting, variable transformation
  - **Layout Obfuscation:** Address obfuscation, obfuscating debug information, address layout/memory layout randomization
  - **Combination of all prior techniques**
- **Based on different compilers:** Evaluate based on different compilers, including GCC, CLANG, MSVC
- **Based on different operating system:** Evaluate based on operating system, including Windows, MacOS, Linux

We assessed each tool with these benchmarks and metrics to thoroughly explore accuracy and detection capabilities when handling binary programs subjected to specific compiler or optimization/obfuscation techniques.[1] For instance, a tool **A** might successfully detect MD5 when all optimization flags are applied but fail to do so in the presence of obfuscation. Conversely, a tool **B** may be able to detect TEA when compiled with GCC but not when compiled with CLANG. This exploration provides us with a more comprehensive understanding of the current state of cryptographic function detection tools, revealing their weaknesses and indicating areas for future development from various perspectives.

Takeaway: (R2, R6) We developed some new comprehensive evaluation metrics in addition to our benchmarking approach. These new evaluation metrics account for various factors impacting cryptographic function detection performance, including software level considerations such as optimization and obfuscation, as well as system level factors like different operating systems and compilers. By including these metrics, our framework enables a more detailed understanding of the tools' performance and limitations, providing insights into their abilities across various environments and configurations.

VI. EVALUATION

After proposing a new evaluation benchmark for cryptographic function detection tools, we conducted a reproduction and replication (R+R) study to assess the performance and current state of these tools. The R+R study not only enhances our understanding of the fundamental functionalities of existing tools but also provides deeper insights into the field, uncovering issues, identifying gaps, and highlighting directions for future research. Meanwhile, the R+R study, incorporating our newly developed evaluation benchmarking framework, will offer valuable insights into the experimental design of cryptographic function detection tools. The gaps and future research directions we identify based on the evaluation results will highlight the limitations of previous experiments and guide future works toward more effective evaluation methodologies.

A. *R+R Experiment Design*

We start by assessing the reproducibility of each tool by employing consistent experimental setups, procedures, and operating conditions to replicate their purported capabilities. In our reproducibility evaluation we follow the ACM guidelines on reproducibility [41], using the same measurement procedure, system setting, and operating conditions as the tool's original test cases. This will help us to understand the basic fundamental reliability for each tool. We conduct the reproducibility experiment as the follow steps:

[1]All benchmarks and metrics we evaluate are available at https://github.com/BARC-Purdue/CryptoBinary.

1) Obtain access to each tool; compile the tool if required, following the provided instructions.
2) Set up the environment to replicate the tool's original configuration (follows the same operating conditions).
3) Obtain the original test cases for each tool. If unavailable, the reproduction evaluation will be skipped (follows the same measuring system).
4) Execute each tool with the provided test cases to reproduce the results (follows the same measurement procedure).

Meanwhile, beside the reproducibility experiment, we also conduct the replicability evaluation of existing work across different categories of cryptographic algorithms with our newly proposed analysis framework, in order to understand their consistency, robustness, and effectiveness. Our replication evaluation follows the ACM guidelines on replicability [41]. We conduct the reproducibility experiment as the follow steps:

1) Obtain access to each tool; recompile the tool on our system if the source code is available.
2) Build and run each tool on Windows 10, Ubuntu 18.04, and macOS Sonoma (follows the different measuring system).
3) Compile our custom benchmarks with the evaluation metrics into binaries for testing.
4) Execute each tool with our benchmarks to assess replication capability (follows the different location on multiple trials).

Takeaway: (R1) We evaluated each tool using the same standardized evaluation approach. Our newly developed benchmarking framework and evaluation metrics served as the test cases, allowing us to assess the latest performance of each cryptographic function detection tool across various factors. This approach ensures consistency in evaluation and provides a comprehensive understanding of the tools' capabilities under diverse conditions.

B. Unsuccessful Tests or Experiments

During our evaluation process, we found that certain tools, as well as the results reported for them in the original studies, could no longer be reproduced successfully.

First, we observed differences in the results during our evaluation of each tool using the basic cryptographic function section in our benchmarking framework. Specifically, we identified instances of both false positives and false negatives that did not align with the original evaluation results reported in the corresponding studies. These mismatches indicate potential limitations in the ability of the tools to perform consistently when applied to our updated benchmark, suggesting that the performance reported in the original experiments may not fully capture their behavior in different environments or with revised datasets.

Secondly, we encountered challenges in running some of the tools entirely. Certain tools required dependent libraries such as IDA and Pin, which, in some cases, required specific versions of Pin that are no longer available. Despite our efforts to locate the closest available versions of the dependent libraries and make modifications to the tools to accommodate the updated library versions, we were still unable to successfully execute a few of the tools. This underscores the importance of maintaining compatibility and documentation to ensure the longevity and reproducibility of research tools.

Note that we compiled these results based on information provided by the developers of each tool. Through our comprehensive analysis, we found that commercial tools like DRACA [12], FindCrypt2 [17], and Signsrch [16] can detect a diverse range of cryptographic algorithms. However, their performance falter when encountering obfuscation or atypical situations. Certain academic-developed tools such as Aligot [18], CryptoHunt [19], and Where's Crypto [36] offer superior analysis results in specific scenarios, such as detecting cryptographic algorithms from obfuscated binaries. Many of them can only detect a few cryptographic algorithm, but they can be easily expanded.

Takeaway: (Q4) During our evaluation, we encountered several unsuccessful tests and experiments, including unexecutable cryptographic function detection tools and unreproducible experimental results. These unsuccessful tests highlight issues within the field that need attention from researchers in future work.

C. Additional Evaluation Experiments

Cryptographic function detection can be a useful component in identifying malware. However, malware often employs intentional evasion tactics, such as code obfuscation, packing, and control flow manipulation, to conceal its true behavior and bypass detection. These techniques distort the binary structure, making it more difficult for analysis tools to accurately identify cryptographic functions and understand the program's functionality. To understand the impact of intentional evasion tactics when detecting cryptographic functions in binary programs, we also selected multiple additional ransomware binaries across different versions, including:

- CryptoLocker_10Sep13
- CryptoLocker_20Nov13
- CryptoLocker_22Jan14
- CryptoWall
- Locky
- TeslaCrypt_v1
- TeslaCrypt_v2
- TeslaCrypt_v3
- Win32Dircrypt

This selection helps assess whether the tools can reliably identify cryptographic functions despite attempts to conceal them. However, since the source code for these malware samples is unavailable, we cannot recompile them using the evaluation metrics introduced in Section V-B. Instead, we use malware binary detection as a supplementary evaluation, complementing our primary benchmarking framework assessment.

D. Results and Insights from R+R Evaluation

Based on our reproduction and replication experiments with our new developed evaluation benchmarking framework, we identified several gaps in this field and proposed future

Tool	False Positive	False Negative	Overall
CryptoKnight	12.43%	7.14%	19.57%
DRACA	21.21%	0.00%	21.21%
FindCrypt2	12.07%	12.07%	12.07%
Signsrch	7.14%	14.28%	21.42%
Where's Crypto	0%	4.93%	4.93%

TABLE V
FALSE POSITIVES AND FALSE NEGATIVES RATE FOR EACH CRYPTOGRAPHIC FUNCTION DETECTION TOOL

research directions. We provide only a brief summary of these insights here, and for a detailed discussion, please refer to our ACSAC '24 paper [6]. In summary, our evaluation results revealed that:

1) Several tools are no longer maintained or available for use, making them incompatible with modern environments and difficult to integrate into current workflows.
2) The detection capabilities of several tools cannot be reproduced as reported, with discrepancies observed in terms of false positives and false negatives, raising concerns about their reliability. Table V shows the false positive/negative rate for each tool.
3) Some tools are limited in their support for specific operating systems or binary architectures, restricting their applicability across diverse testing scenarios.
4) Variations in compilers, optimization levels, obfuscation techniques, and algorithm versions significantly affect detection results, highlighting the sensitivity of these tools to changes in the binary generation process.

Additionally, we identified further issues in cryptographic function detection that warrant further investigation:

1) The performance of AI/ML-based approaches remains suboptimal, especially when handling large or complex binaries.
2) There is a lack of support for modern cryptographic algorithms, limiting the tools' effectiveness in detecting newer implementations used in contemporary software.
3) Certain tools are challenging to use due to limitations in user experience, which can hinder their adoption and reduce their practical utility in real-world scenarios.

Takeaway: (Q3) Based on our evaluation experiences and results, we have proposed several future research directions, discussions, and recommendations for advancing this field. In our ACSAC '24 paper [6], we provide a detailed account of these proposed research directions, along with an in-depth discussion of the user experiments and challenges we encountered while running previous tools. These insights not only serve as valuable suggestions for future research but also offer lessons learned to guide the development of more effective tools in cryptographic function detection.

VII. DISCUSSION

A. Why Reproduction and Replication

A Systematization of Knowledge paper provides valuable insights that help researchers quickly understand the current developments in a field. However, a simple summarizing of prior works may not accurately capture the state of the field, as it relies solely on reported findings without validating their accuracy through experimentation, making them less effective at identifying errors or inconsistencies in previous research. For instance, many cryptographic function detection tools were developed years, or even decades, ago and may no longer be compatible with modern programming languages, compilers, or hardware architectures. Performance reports from the original evaluations may not reflect the current capabilities of these tools, and experiments conducted years ago may no longer align with current design goals and research objectives. This highlights the importance of our reproduction and replication study, which not only addresses these significant gaps as the field evolves but also provides broader insights and identifies promising directions for future research.

We also provide a detailed discussion of each cryptographic function detection tool's advantages and disadvantages based on our experiences during the evaluation process. This information serves as a practical guideline to assist future users and researchers in selecting the most suitable tool for detecting cryptographic functions in binaries based on their specific needs and available resources. For further details, including comprehensive insights, please refer to the appendix in our ACSAC '24 paper [6].

Takeaway: (Q2) Replication and reproduction are central components of this paper and our ACSAC '24 paper [6]. These processes are essential for understanding the current performance of each tool, uncovering inconsistencies with their original evaluations, and identifying potential limitations. This is particularly important for tools developed a significant time ago, as their performance may have degraded or diverged from initial claims due to evolving technologies and methodologies.

B. User Experience

Considering the experience and insights we have garnered from our evaluation process, we believe that discussing the developer experience is also an intriguing topic, in terms of how developers interact with these tools and their ease of use.

Older cryptographic detection tools, such as DRACA, PEiD KANAL, and SignSrch, do not require developers to compile and build the tools from source code. Developers can readily utilize these tools with supported operating systems. However, when compared to some better performing tools, these user-friendly alternatives exhibit lower performance and utility. For example, obfuscation can disturb the binary analysis process of Signsrch, leading to false negatives in the detection of certain cryptographic algorithms. Furthermore, DRACA struggles to accurately analyze packed executable programs, and consequently, it can only provide a rudimentary analysis outcome, despite its ease of use.

In addition to the aforementioned challenges, some tools pose difficulties for users due to technological complexity or data-related issues. Take, for instance, IDA Scope, Findcrypt2,

Tool	Aligot	Crypto-Hunt	Crypto-Knight	DRACA	FindCrypt2	FALKE-MC	HCD	Kerckhoff	PEiD KANAL	SignSrch	Softmax Classifier	Where's Crypto
ADLER32											✓	
AES (Rijndael)	✓	✓	✓	✓	✓	✓		✓		✓		✓
BASE64										✓	✓	
Blowfish			✓	✓	✓					✓	✓	
Camellia					✓							
CAST					✓					✓		
CAST-256				✓	✓					✓		
CRC32				✓	✓					✓	✓	
DES				✓	✓			✓		✓	✓	✓
EC										✓		
GOST					✓					✓		
HAVAL					✓					✓	✓	
MARS				✓	✓					✓		
MD2					✓					✓		
MD4					✓					✓		
MD5	✓	✓	✓	✓	✓			✓		✓	✓	✓
RC2				✓	✓					✓		
RC4	✓	✓	✓			✓	*Not Available*	✓	*Not Available*			
RC5				✓	✓					✓	✓	
RC6				✓	✓					✓	✓	
Ripemd-160				✓						✓		
RSA	✓	✓	✓					✓				
SAFER				✓	✓					✓		
SHA-1				✓	✓					✓	✓	✓
SHA-256					✓					✓	✓	✓
SHA-512					✓					✓		✓
SHARK					✓					✓		
Skipjack				✓	✓					✓		
Square					✓					✓		
TEA	✓	✓		✓						✓		
Tiger				✓	✓					✓		
Twofish				✓	✓					✓		
WAKE					✓					✓		
Whirlpool					✓					✓		
XTEA												✓

TABLE VI
CRYPTOGRAPHIC ALGORITHM DETECTION SUPPORT AS STATED IN THE PAPER OR DOCUMENTATION FOR EACH TOOL

and Where's Crypto, which all rely on IDA Pro. IDA Pro is not a free, open-source software, and these tools cannot be utilized by developers without it. Conversely, tools such as CryptoKnight and other machine learning-based techniques require users to provide substantial amounts of data, which further compounds the difficulty of using these tools.

However, these tools generally exhibit superior performance when compared to user-friendly alternatives. For instance, CryptoHunt consistently yields accurate analysis results for identification within obfuscated programs. Where's Crypto, one of the most recent cryptographic detection tools, broadens the scope of analysis to encompass unfamiliar and proprietary cryptographic primitives, without relying on heuristics to select code fragments.

C. Lessons Learned

We successfully reproduced the original evaluation of certain tools using their provided testing binary programs and the evaluation results they claimed, as shown in Table VI. However, when evaluating these tools with our newly developed benchmarking framework, we observed that their detection capabilities were inconsistent. As discussed in Section IV, most prior tools were primarily evaluated on basic cryptographic functions without the support of well-defined metrics. Consequently, factors such as the operating system, compiler, and binary architecture significantly influenced and reduced detection accuracy.

This highlights the importance of conducting comprehensive evaluation experiments to uncover the true capabilities and limitations of a given tool. Without careful consideration of critical factors, such as experimental criteria that directly affect detection, the results may fail to provide an accurate representation of a tool's performance.

Thus, this paper aims not only to evaluate existing tools and investigate their performance, gaps, and future research directions, but also serves as a key reminder that future experiments which should incorporate broader considerations to ensure robust evaluation. In the domain of cryptographic function detection in binaries, our benchmarking framework establishes a standardized experimental approach to enable accurate experimental design and reliable evaluation results.

D. Evaluation Design for Future Work

Based on our investigation of previous evaluations, we found that flawed experimental designs fail to accurately reflect a tool's actual performance and capabilities. This can also lead researchers to omit critical factors that should be the focus to improve the detection result and performance. Therefore, with our newly developed evaluation benchmarking framework and evaluation metrics, future research will not only benefit from a standardized evaluation framework that facilitates direct comparisons with previous tools but also gain a more comprehensive approach to identifying potential gaps and limitations in a tool's design based on evaluation results.

Acknowledgment

This work was supported by NSF grant CNS-2047991.

References

[1] "Openssl," https://www.openssl.org/.

[2] "Wannacry ransomware," https://en.wikipedia.org/wiki/WannaCry_ransomware_attack.

[3] "Petya ransomware," https://www.proofpoint.com/us/glossary/petya.

[4] W. Yan, Z. Zhang, and N. Ansari, "Revealing packed malware," *IEEE S&P*, 2008.

[5] I. Harvey, "Cipher hunting: How to find cryptographic algorithms in large binaries," *NCipher Corporation Ltd*, 2001.

[6] Y. Fan, P. Biswas, and C. Garman, "R+r: A systematic study of cryptographic function identification approaches in binaries," in *Annual Computer Security Applications Conference*, 2024.

[7] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic protocol format reverse engineering through context-aware monitored execution." in *NDSS*, 2008.

[8] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: Automatic extraction of protocol message format using dynamic binary analysis," in *ACM CCS*, 2007.

[9] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace, "Reformat: Automatic reverse engineering of encrypted messages," in *ESORICS*, 2009.

[10] R. Chang, L. Jiang, H. Shu, and H. He, "Cryptographic algorithms analysis technology research based on functions signature recognition," in *CIS*, 2014.

[11] P. Lestringant, F. Guihéry, and P.-A. Fouque, "Automated identification of cryptographic primitives in binary code with data flow graph isomorphism," in *ASIACCS*, 2015.

[12] "Draft crypto analyzer," http://www.literatecode.com/draca.

[13] "Kanal - krypto analyzer for peid," http://www.dcs.fmph.uniba.sk/zri/6.prednaska/tools/PEiD/plugins/kanal.htm.

[14] "Kerckhoffs," https://github.com/felixgr/kerckhoffs.

[15] "Hash & crypto detector (hcd)," https://webscene.ir/tools/show/Hash-and-Crypto-Detector-1.4.

[16] "Signsrch," http://aluigi.altervista.org/mytoolz/signsrch.zip.

[17] "Findcrypt2," http://www.hexblog.com/?p=28.

[18] J. Calvet, J. M. Fernandez, and J.-Y. Marion, "Aligot: cryptographic function identification in obfuscated binary programs," in *ACM CCS*, 2012.

[19] D. Xu, J. Ming, and D. Wu, "Cryptographic function detection in obfuscated binaries via bit-precise symbolic loop mapping," in *IEEE S&P*, 2017.

[20] X. Li, X. Wang, and W. Chang, "Cipherxray: Exposing cryptographic operations and transient secrets from monitored binary execution," *IEEE TDSC*, 2012.

[21] N. Lutz, "Towards revealing attacker's intent by automatically decrypting network traffic," *Mémoire de maıtrise, ETH Zürich, Switzerland*, 2008.

[22] F. Gröbert, C. Willems, and T. Holz, "Automated identification of cryptographic primitives in binary programs," in *RAID*, 2011.

[23] J. Park and Y. Park, "Symmetric-key cryptographic routine detection in anti-reverse engineered binaries using hardware tracing," *Electronics*, 2020.

[24] E. C. R. Shin, D. Song, and R. Moazzezi, "Recognizing functions in binaries with neural networks," in *USENIX Security*, 2015.

[25] J. L. Wright and M. Manic, "Neural network approach to locating cryptography in object code," in *ETFA*, 2009.

[26] L. Benedetti, A. Thierry, and J. Francq, "Detection of cryptographic algorithms with grap." *IACR Cryptology ePrint Archive*, 2017.

[27] A. Aigner, "Falke-mc: A neural network based approach to locate cryptographic functions in machine code," in *ARES*, 2018.

[28] G. Hill and X. Bellekens, "Cryptoknight: Generating and modelling compiled cryptographic primitives," *Information*, 2018.

[29] L. Jia, A. Zhou, P. Jia, L. Liu, Y. Wang, and L. Liu, "A neural network-based approach for cryptographic function detection in malware," *IEEE Access*, 2020.

[30] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," Department of Computer Science, The University of Auckland, New Zealand, Tech. Rep., 1997.

[31] "Control flow graph," https://en.wikipedia.org/wiki/Control-flow_graph.

[32] Y. Li, Z. Sha, X. Xiong, and Y. Zhao, "Code obfuscation based on inline split of control flow graph," in *ICAICA*, 2021.

[33] "Control-flow flattening," https://github.com/obfuscator-llvm/obfuscator/wiki/Control-Flow-Flattening.

[34] D. Xu, "Opaque predicate: Attack and defense in obfuscated binary code," 2018.

[35] H. Xu, Y. Zhou, J. Ming, and M. Lyu, "Layered obfuscation: a taxonomy of software obfuscation techniques for layered security," *Cybersecurity*, 2020.

[36] C. Meijer, V. Moonsamy, and J. Wetzels, "Where's crypto?: Automated identification and classification of proprietary cryptographic primitives in binary code," in *USENIX Security*, 2021.

[37] P. Shirani, L. Wang, and M. Debbabi, "Binshape: Scalable and robust binary library function identification using function shape," in *DIMVA*, 2017.

[38] F. Matenaar, A. Wichmann, F. Leder, and E. Gerhards-Padilla, "Cis: The crypto intelligence system for automatic detection and localization of cryptographic functions in current malware," in *MALWARE*, 2012.

[39] X. Ren, M. Ho, J. Ming, Y. Lei, and L. Li, "Unleashing the hidden power of compiler optimization on binary code difference: an empirical study," in *PLDI*, 2021.

[40] "Cryptographic standards and guidelines," https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines.

[41] Association for Computing Machinery, "Artifact Review and Badging Policy," 2024, accessed: 2024-09-16. [Online]. Available: https://www.acm.org/publications/policies/artifact-review-and-badging-current