Software Obfuscation with Non-Linear Mixed Boolean-Arithmetic Expressions

Binbin Liu[1](✉), Weijie Feng[1], Qilong Zheng[1], Jing Li[1], and Dongpeng Xu[2](✉)

[1] University of Science and Technology of China, Hefei, China
{robbertl,fengwj}@mail.ustc.edu.cn, {QLZheng,lj}@ustc.edu.cn
[2] University of New Hampshire, Durham 03824, USA
dongpeng.xu@unh.edu

Abstract. Mixed Boolean-Arithmetic (MBA) expression mixes bitwise operations (e.g., AND, OR, and NOT) and arithmetic operations (e.g., ADD and IMUL). It enables a semantic-preserving program transformation to convert a simple expression to a difficult-to-understand but equivalent form. MBA expression has been widely adopted as a highly effective and low-cost obfuscation scheme. However, state-of-the-art deobfuscation research proposes substantial challenges to the MBA obfuscation technique. Attacking methods such as bit-blasting, pattern matching, program synthesis, deep learning, and mathematical transformation can successfully simplify specific categories of MBA expressions. Existing MBA obfuscation must be enhanced to overcome these emerging challenges.

In this paper, we first review existing MBA obfuscation methods and reveal that existing MBA obfuscation is based on "linear MBA", a simple subset of MBA transformation. This leaves the more complex "non-linear MBA" in its infancy. Therefore, we propose a new obfuscation method to unleash the power of non-linear MBA. Non-linear MBA expressions are generated from the combination or transformation of linear MBA rules based on a solid theoretical underpinning. Comparing to existing MBA obfuscation, our method can generate significantly more complex MBA expressions. To present the practicability of the non-linear MBA obfuscation scheme, we apply non-linear MBA obfuscation to the Tiny Encryption Algorithm (TEA). We have implemented the method as a prototype tool, named *MBA-Obfuscator*, to produce a large-scale dataset. We run all existing MBA simplification tools on the dataset, and at most 147 out of 1,000 non-linear MBA expressions can be successfully simplified. Our evaluation shows *MBA-Obfuscator* is a practical obfuscation scheme with a solid theoretical cornerstone.

Keywords: Software obfuscation · Mixed Boolean-Arithmetic expression · Expression transformation

1 Introduction

Software obfuscation [26] performs a semantics-preserving transformation to hide the implementation of a program, which leads the program is hard to understand

and analyze. Many obfuscation techniques have been developed in the literature to hide the behavior of code in different ways, *Mixed Boolean-Arithmetic*(MBA) obfuscation is such a prominent example. Zhou et al. [32] define MBA expression that mixes bitwise operations(e.g., $\neg, \wedge, \oplus, \dots,$) and arithmetic operations(e.g., $+, -, \times$). MBA obfuscation transforms a simple expression like $x+y$ to a complex but equivalent expression with mixed bitwise and arithmetic operators. Multiple academic tools and industry projects [6,13,17,21,23] have embedded MBA obfuscation into their products.

The wide application of MBA obfuscation has attracted researchers to explore how to recover the initial form of the MBA expression. Guinet et al. [11] presents a tool, *Arybo*, which normalizes MBA expressions to bit-level symbolic representation with only $\oplus$ and $\wedge$ operations. Eyrolles et al. [9] simplify MBA expressions by a pattern matching method. Blazytko et al. [4] apply program synthesis techniques [12] to learn the underlying semantics of obfuscated code and generate another simpler but equivalent expression. Feng et al. [10] introduce a novel solution based on deep learning, named NeuReduce, to simplify MBA expression. Liu et al. [18] prove a hidden two-way transformation feature and present a novel technique to simplify MBA expression. These methods propose a great challenge to the MBA obfuscation technique. However, we note that existing MBA obfuscation rules are generated from linear MBA expressions, because non-linear MBA research is still at an early stage and related non-linear MBA translation rules are rare.

To improve the resilience of the MBA obfuscation technique, we explore a new research direction: non-linear MBA obfuscation, which is the relative complement of linear MBA expression in the MBA obfuscation area. Firstly, multiple methods are demonstrated to create unlimited non-linear MBA expressions, whose correctness is guaranteed based on the basic math rules. Next, we present a practical application of the non-linear MBA obfuscation technique on the Tiny Encryption Algorithm(TEA), which hides the key and transforms the original operation into another different format. We have implemented the method as an open-source tool named *MBA-Obfuscator*. Given a simple expression as input, *MBA-Obfuscator* generates a related complex non-linear MBA expression. To the best of our knowledge, *MBA-Obfuscator* is the first tool to generate diversified non-linear MBA expressions.

To demonstrate the strength of *MBA-Obfuscator*, we evaluate it on a comprehensive dataset containing $1,000$ diversified linear and related non-linear MBA expressions. The evaluation demonstrates that existing deobfuscation methods cannot effectively simplify non-linear MBA expressions. On the other hand, the overhead to use non-linear MBA expression is low. Our evaluation results show that the non-linear MBA technique is an practical obfuscation scheme.

In summary, we make the following key contributions:

- We propose how to use linear MBA rules to generate non-linear MBA expression and guarantee its correctness.
- We discuss one concrete application of non-linear MBA expression, which obfuscates the Tiny Encryption Algorithm(TEA).

– Our large-scale evaluation shows that *MBA-Obfuscator* outperforms existing linear MBA obfuscation in terms of better resilience, potency, and cost. *MBA-Obfuscator*'s source code and the benchmark are available at https://github.com/nhpcc502/MBA-Obfuscator.

The rest of the paper is structured as follows. Section 2 illustrates the background of existing MBA obfuscation and simplification methods. Section 3 introduces our methods to generate infinite non-linear MBA expressions. Next, we present one practical application (Sect. 4) of non-linear MBA obfuscation on the Tiny Encryption Algorithm(TEA). Finally, we give the details on our evaluation results (Sect. 5) and conclude (Sect. 6)

2 Preliminaries

In this section, we provide background on Mixed-Boolean-Arithmetic (MBA) transformation and deobfuscation techniques. Zhou et al. [31,32] formally present how to generate unlimited linear MBA expressions. Since MBA rules are applied as an information hiding technique, it has already generated follow-up deobfuscation research to simplify MBA expressions, including bit-blasting, pattern matching, program synthesis, deep learning, and mathematical transformation approaches.

2.1 MBA-Based Obfuscation

Zhou et al. [31,32] firstly define an MBA expression as the mixture usage of bitwise operations ($\vee$, $\wedge$, $\oplus$, $\neg$) and integer arithmetic operations($+$, $-$,$\times$). The formal definition of polynomial MBA expression is denoted as follows:

Definition 1. *A polynomial MBA expression is:*

$$\sum_{i\in I} a_i * (\prod_{j\in J_i} e_{i,j}(x_1,\ldots,x_t)),$$

where a_i are constants, $e_{i,j}$ are bitwise expressions of variables $x_1,\ldots,x_t$ over B^n, and $I, J_i \subset Z$, $\forall i \in I$. $a_i * (\prod_{j\in J_i} e_{i,j}(x_1,\ldots,x_t))$ is called a term.

Definition 2. *A linear MBA expression is a polynomial MBA expression of the form:*

$$\sum_{i\in I} a_i * e_i(x_1,\ldots,x_t),$$

where a_i are constants, e_i are bitwise expressions of variables $x_1,\ldots,x_t$ over B^n, and $I \subset Z$. $a_i * e_i(x_1,\ldots,x_t)$ is called a term.

Examples of linear and polynomial MBA expression are shown as follows:

$$x + (x \wedge y) + y - 2 * (x \oplus y) + (x \vee y),$$
$$x + y * (x \oplus y) - 2 * (\neg x \vee y) * (\neg x) - 1.$$

In addition, Zhou et al. [32] present an approach, which uses truth tables(linearly dependent column vectors) to generate a linear MBA identity, as shown in Example 1. They propose a formal theoretical foundation to guarantee the correctness of this method to build unlimited linear MBA rules.

Example 1. The 0,1-matrix

$$M = \begin{array}{c} \begin{array}{ccccc} x & y & x \oplus y & \neg(x \vee y) & -1 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} \end{array}, \vec{v} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ -2 \end{pmatrix},$$

$$\Rightarrow M\vec{v} = [0, 0, 0, 0, 0]^T.$$

It produces a linear MBA identity:

$$x + y + (x \oplus y) + 2 * \neg(x \vee y) - 2 * (-1) = 0.$$

This linear MBA identity can be transformed into multiple MBA rules:

$$\begin{aligned} x + y &= -(x \oplus y) - 2 * \neg(x \vee y) - 2, \\ 2 &= -x - y - (x \oplus y) - 2 * \neg(x \vee y). \end{aligned}$$

Example 1 presents how to transform $x + y$ into a much more complex form: $-(x \oplus y) - 2 * \neg(x \vee y) - 2$, another example of the code before and after linear MBA obfuscation is shown in Fig. 1.

Unfortunately, Zhou et al. [32] only propose the formation of polynomial MBA expression, without any further research. Therefore, all existing MBA obfuscation work is mainly based on the linear MBA rules. For example, Cloakware [17], Irdeto [13], and Quarkslab [23] apply MBA transformation in their commercial products. Tigress[1] [5], an academic C source code obfuscation tool, applies MBA rules to encode integer variables and expressions [6,7]. Mougey and Gabriel [21] use MBA rules to do instruction substitution in a Digital Rights Management (DRM) system. Recently, Blazy and Hutin [3] strengthen the CompCert C compiler [16] to generate programs with formally verified MBA obfuscation rules. Xmark applies MBA obfuscation to hide the static signatures of software watermarking [19]. ERCIM News reported in 2016 that the MBA obfuscation technique had been used in malware compilation chains [1].

2.2 MBA Deobfuscation

After the MBA obfuscation technique was proposed, researchers have explored how to simplify MBA expressions. Eyrolles's PhD thesis [8] is the first work to explore this subject at full length. She focuses on simplifying MBA expressions by

[1] https://tigress.wtf/index.html.

```
int fun(int x,int y){
  int res;
  res = x + y;

  return res;
  }
```

(a) Original program.

```
int fun(int x,int y){
  int res;
  res = 5*(x&y)+4*(x&~y)-5*(x|y)+2*x+6*~(x|~y);

  return res;
  }
```

(b) Linear MBA obfuscated program.

Fig. 1. An example of MBA obfuscation for `x + y`.

applying various analyzing tools. Her experiments show that popular computer algebra software such as Maple [20], SageMath [25], Wolfram Mathematica [29], and Z3 [22] fail to simplify MBA expression, because existing reduction rules only work either on pure bitwise expressions or pure arithmetic expressions. Furthermore, LLVM compiler optimization passes [15] have a very limited effect on MBA simplification. Therefore, Eyrolles et al. [9] propose a pattern matching method to simplify MBA expressions. Multiple MBA rules are hard-coded in the tool, named SSPAM. However, the pattern matching method only detects and simplifies MBA expressions by a range of fixed patterns. Such an approach cannot reduce generic MBA rules.

Guinet et al. [11] create a tool, *Arybo*, which reduces MBA expressions to bit-level symbolic representation with only $\oplus$ and $\wedge$ operations. One of the drawbacks is the performance penalty caused by the bloated size of bit-level expressions. Biondi [2] presents an algebraic approach to simplify MBA expression, reducing the complexity of MBA obfuscation, but the proposed method strongly depends on the specific MBA rules.

Blazytko et al. [4] apply program synthesis [12] guided by Monte Carlo Tree Search (MCTS) to do code deobfuscation. It produces input-output samples from the obfuscated code and then learns the semantics based on those input-output pairs. Then it automatically generates another simpler but equivalent expression. Due to the non-determinism and sampling mechanism of program synthesis, their method cannot guarantee the correctness of the simplification result.

Feng et al. [10] introduce a novel solution, named NeuReduce, to simplify MBA expression. They train NeuReduce using MBA rules based on the sequence to sequence neural network models. The input of NeuReduce is a character string format of complex MBA expression, and the related output is the simplification expression.

Liu et al. [18] investigate the mathematical mechanism of MBA expression and prove a hidden two-way transformation feature in the MBA obfuscation. They transform all bitwise expressions to specific MBA forms and then perform arithmetic reduction to simplify MBA expression.

3 Non-Linear MBA Expression Generation

In this section, we focus on how to generate unlimited non-linear MBA expressions. First, we present how to create polynomial MBA expressions from the linear MBA rules. Next, multiple methods are applied to produce non-linear MBA rules.

3.1 Polynomial MBA Expression

Using Proposition 1, a polynomial MBA expression can be generated based on the existing linear MBA expression.

Proposition 1. *Let E be an expression, n be positive integer,*

- $E \equiv \bar{E}$, *$\bar{E}$ is a linear MBA expression,*
- *E_k is a linear MBA expression,* $E_k = 1$, $k = 1, 2, \ldots, n$,
- $E' = \bar{E} * E_1 * E_2 * \ldots * E_n$,

Then
$E' \equiv E$, *and E' is a polynomial MBA expression.*

By Proposition 1, any simple expression can be transformed into a complex polynomial MBA expression, as shown in Example 2.

Example 2. For a simple expression $x + y$, we have
$x + y = (x \vee y) + (x \wedge y)$, $1 = (x \wedge y) - y - (x \vee \neg y)$,
then

$$\begin{aligned} x + y &= ((x \vee y) + (x \wedge y)) * ((x \wedge y) - y - (x \vee \neg y)) \\ &= (x \vee y) * (x \wedge y) + (x \wedge y) * (x \wedge y) - (x \vee y) * y \\ &\quad - (x \wedge y) * y - (x \vee y) * (x \vee \neg y) - (x \wedge y) * (x \vee \neg y). \end{aligned}$$

However, Proposition 1 exposes a potential drawback: one polynomial MBA expression generated by Proposition 1 has a fixed pattern, which can be used to simplify the polynomial MBA expression through basic algebra laws (i.e., commutation, association, and distribution laws). Firstly, a reverse engineer can carefully recover the original linear MBA expression by factoring. Then, the linear MBA expressions can be simplified with existing linear MBA simplification tools. For instance, the polynomial MBA expression in Example 2 can be simplified as follows:

$$((x \vee y) + (x \wedge y)) * ((x \wedge y) - y - (x \vee \neg y)) = (x + y) * 1 = x + y.$$

This flaw shows that the strength of polynomial MBA expression is the same as linear MBA expression. In order to address this issue, we firstly introduce the concept of *0-equality*, whose formal definition is given in Definition 3. Example 3 presents how to generate a *0-equality*.

Definition 3. *Let E_1 be an expression, E_2 and $\bar{E_2}$ be linear MBA expressions,*

- *the coefficients in E_1 are randomly reversed or not, to construct a new expression $\bar{E_1}$,*
- *a sub-expression of E_2 forms part of $\bar{E_2}$, and $\bar{E_2} \equiv 0$,*

Then $E_1 * E_2$'s 0-equality *is $Z_{E_1 E_2} = \bar{E_1} * \bar{E_2} \equiv 0$.*

Example 3. For expressions
$E_1 = (x \vee y) + (x \wedge y)$, $E_2 = (x \wedge y) - y - (x \vee \neg y)$,we have
$\bar{E_1} = (x \vee y) - (x \wedge y)$, $\bar{E_2} = (x \wedge y) - y + (\neg x \wedge y) = 0$,
$\Rightarrow\ Z_{\bar{E}E_1} = \bar{E_1} * \bar{E_2} = ((x \vee y) - (x \wedge y)) * ((x \wedge y) - y + (\neg x \wedge y)) = 0$.

Through the *0-equality*, Proposition 2 transforms a polynomial expression generated by Proposition 1 into another format, which has broken the fixed pattern introduced by Proposition 1. One detailed instance is shown in Example 4.

Proposition 2. *Let E be an expression,*

- *$E = \bar{E}$, $\bar{E}$ is a linear MBA expression,*
- *E_k is a linear MBA expression, $E_k = 1$, $k = 1, 2, \ldots, n$,*
- *$E' = (\ldots((\bar{E} * E_1 + Z_{\bar{E}E_1}) * E_2 + Z_{\bar{E}E_1E_2}) * \ldots) * E_n + Z_{\bar{E}E_1E_2\ldots E_n}$,*

Then
$E' \equiv E$, and E' is a polynomial MBA expression.

Example 4. For an expression $E = x + y$ and Example 3, we have

$$\begin{aligned} x + y &= \bar{E} * E_1 + Z_{\bar{E}E_1} \\ &= ((x \vee y) + (x \wedge y)) * ((x \wedge y) - y - (x \vee \neg y)) \\ &\quad + ((x \vee y) - (x \wedge y)) * ((x \wedge y) - y + (\neg x \wedge y)) \\ &= 2 * (x \vee y) * ((x \wedge y) - y) - (x \vee \neg y) * ((x \vee y) + (x \wedge y)) \\ &\quad + (\neg x \wedge y) * ((x \vee y) - (x \wedge y)). \end{aligned}$$

3.2 MBA-related Rules

So far, a simple expression can be transformed into a complex polynomial MBA expression. Next, we demonstrate how to generate a new non-linear MBA expression based on existing MBA rules.

Recursively Apply MBA Rules. This method firstly transform the simple expression into a linear MBA expression, then recursively apply MBA rules to convert the related linear MBA expression into a complex non-linear MBA format, as seen in Example 5.

Example 5.

$$\begin{aligned} x + y =& y - \neg x - 1 \ \Leftarrow\ x + y = (x \vee y) + (x \wedge y) \\ =&((y - \neg x) \vee -1) + ((y - \neg x) \wedge -1) \ \Leftarrow\ x + y = y + (x \wedge \neg y) + (x \wedge y) \\ =&(((-\neg x) + (y \wedge \neg(-\neg x)) + (y \wedge (-\neg x))) \vee -1) \\ &+ (((-\neg x) + (y \wedge \neg(-\neg x)) + (y \wedge (-\neg x))) \wedge -1). \end{aligned}$$

Replace Sub-expression With MBA Expression. This method replaces part of the original expression with an equivalent non-linear MBA rule, as shown in Example 6.

Example 6.

$$\begin{aligned}
2 =&(-3*(x\vee y)-1-3*(\neg x\wedge \neg y))*(-x-\neg x)\\
&+(3*(x\vee y)+1)*(-x+(x\vee y)+(x\wedge \neg y))\\
=&3*(\neg x\wedge \neg y)*(x+\neg x)+(3*(x\wedge y)+1)*(\neg x+1+3*(x\vee y))\\
\Rightarrow x+y =& -(x\oplus y)-2*\neg(x\vee y)-2\\
=& -(x\oplus y)-2*\neg(x\vee y)-3*(\neg x\wedge \neg y)*(x+\neg x)\\
&-(3*(x\wedge y)+1)*(\neg x+1+3*(x\vee y)).
\end{aligned}$$

Linear Combination of MBA Expression. A new MBA expression can be generated from the linear combination of existing MBA obfuscation rules. More specifically, we add a non-linear MBA expression which is equal to 0 to the original expression, as seen in Example 7.

Example 7.

$$\begin{aligned}
&x=(x\wedge y)+(x\vee y)-y,\\
&0=x*y-(x\wedge y)*(x\vee y)-(x\wedge \neg y)*(\neg x\wedge y),\\
&\Rightarrow\ x=(x\wedge y)+(x\vee y)-y+x*y-(x\wedge y)*(x\vee y)-(x\wedge \neg y)*(\neg x\wedge y).
\end{aligned}$$

Algorithm 1. MBA Expression Generation

```
1: Input: Simple expression $E$, Flag $F$.
2: function MBA-OBFUSCATOR($E$, $F$)
3:     Generate a new linear MBA expression $\bar{E}$ that equals to input $E$.
4:     if $F$.polynomial then
5:         Generate a new polynomial MBA expression $E'$ based on Proposition 2.
6:     else if $F$.recursively then
7:         Generate multiple MBA rules that equal to $x + y$.
8:         Apply the rules to transform $\bar{E}$ into $E'$.
9:     else if $F$.replace then
10:        Generate multiple MBA rules that equal to the sub-expression of $\bar{E}$.
11:        Apply the rules to transform $\bar{E}$ into $E'$.
12:    else if $F$.combination then
13:        Generate a polynomial MBA expression $Z$ that equals to 0.
14:        $E' = \bar{E} + Z$.
15:    end if
16:    return $E'$.
17: end function
```

We integrate all methods described above into Algorithm 1. The algorithm takes a simple expression E and flag F as input and returns its related complex non-linear MBA expression based on the flag F. One random seed is contained in the algorithm, as the one in Definition 3, to generate a randomized obfuscated expression. Note that Algorithm 1 can transform a constant into a complex non-linear MBA expression, as shown in Example 6.

4 Case Study

In this section, we present how to apply the MBA obfuscation technique on the Tiny Encryption Algorithm(TEA) to protect the key and algorithm, since MBA obfuscation technique can hide constant or complicate operations.

In cryptography, the Tiny Encryption Algorithm (TEA) is a block cipher designed by David Wheeler and Roger Needham [28]. The encryption routine code in C of TEA is shown in Fig. 2. Given the feature of the symmetric encryption algorithm, the respective decryption routine is similar to the encryption routine. Since its simple structure and low cost, TEA has been widely applied to many scenarios [14,24,27,30].

The whole obfuscation process is shown in Fig. 3. Firstly, the key is transformed into a function that outputs a constant value, as shown in Fig. 3a. Next, the operation in the TEA is replaced with related complex MBA expression, which is shown in Fig. 3b. For every one operation, MBA-Obfuscator generates the related complex and equivalent MBA expression. After that, the original algorithm is replaced with the obfuscation routine, and the entire obfuscation code is seen in Fig. 3c. Considering the randomness in Algorithm 1, every expression generated by MBA-Obfuscator is different from each other.

```
void TEAencrypt(uint32 v[2], uint32 k[4]) {
    uint32 v0=v[0], v1=v[1], sum=0, i;                  /*set up*/
    uint32 delta=0x95136ac5;            /*a key schedule constant*/
    uint32 k0=k[0], k1=k[1], k2=k[2], k3=k[3];             /*key*/

    /*basic cycle*/
    for (i = 0; i < 32; i++) {
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        }

    v[0]=v0;
    v[1]=v1;
    }
```

Fig. 2. The TEA encryption procedure implemented in C programming language.

```
uint32 keyfunction(uint32 x, uint32 y) {
    //let the key be 0x12345678
    uint32 key = MBA-Obfuscator(0x12345678, Flag);

    return key;
    }
```

(a) Hiding the encryption key.

```
uint32 ob_operation(uint32 x, uint32 y, uint32 sum,
                    uint32 (*kf0)(uint32, uint32),
                    uint32 (*kf1)(uint32, uint32)) {
    uint32 k0 = kf0(x, y);
    uint32 k1 = kf1(x, y);
    uint32 y1 = y << 4;
    uint32 z0 = MBA-Obfuscator(y1 + k0, Flag);
    uint32 z1 = MBA-Obfuscator(y + sum, Flag);
    uint32 y2 = y >> 5;
    uint32 z2 =  MBA-Obfuscator(y2 + k1, Flag);
    uint32 res = MBA-Obfuscator(z0 ^ z1 ^ z2, Flag);

    return res + x;
    }
```

(b) Obfuscating the encryption operations.

```
void TEAencrypt_ob(uint32 v[2],uint32 (*kf)(uint32, uint32)[4]) {
    uint32 v0=v[0], v1=v[1], sum=0;                    /*set up*/
    uint32 delta=0x95136ac5;          /*a key schedule constant*/
    uint32 kf0=kf[0], kf1=kf[1];                          /*key*/
    uint32 kf2=kf[2], kf3=kf[3];

    /*basic cycle*/
    for (i = 0; i < 32; i++){
        sum = MBA-Obfuscator(sum + delta, Flag);
        v0 = ob_operation(v0,v1,sum,kf0,kf1);
        v1 = ob_operation(v1,v0,sum,kf2,kf3);
        }

    v[0]=v0;
    v[1]=v1;
    }
```

(c) TEA encryption routine after Obfuscation.

Fig. 3. The steps of running MBA-Obfuscator on TEA encryption.

5 Implementation and Evaluation

We implement Algorithm 1 in a prototype tool called *MBA-Obfuscator*. *MBA-Obfuscator* accepts a simple expression input and generates a random related non-linear MBA expression. The whole prototype is written in 1,900 lines of Python code. We leverage the `NumPy` library for generating a linear MBA expression, and the `SymPy` library for arithmetic operation. *MBA-Obfuscator* also includes utilities for measuring the quantitative metrics for MBA expressions, such as counting MBA alternation and the number of terms.

We conduct a set of experiments to seek for checking the capability of *MBA-Obfuscator*. In particular, we design experiments to answer the following research questions.

1. **RQ1:** Is *MBA-Obfuscator* able to resist related methods simplifying MBA expression? *(resilience)*
2. **RQ2:** How complex is a non-linear MBA expression? *(potency)*
3. **RQ3:** How much overhead does *MBA-Obfuscator* introduce? *(cost)*

As the answer to RQ1, we apply existing deobfuscation tools to simplify non-linear MBA expressions. To address RQ2, we calculate the complexity metrics such as the number of MBA alternation. In response to RQ3, we study *MBA-Obfuscator*'s performance data such as running time and memory usage.

5.1 Experimental Setup

Dataset. Note that the use of non-linear MBA expression is by simply substituting where the linear MBA rule is used. Therefore we check the capacity of linear MBA expression and the related non-linear MBA expression generated by *MBA-Obfuscator*. We collect 1,000 linear MBA expressions and related ground truth (correctly simplified form) from existing works [4,5,8,18,31,32]. Next, we use *MBA-Obfuscator* to generate a non-linear MBA expression that is equivalent to the ground truth. Therefore, we get the Dataset including 1,000 MBA expressions. Every sample in the dataset is a 3-tuple: (G, L, NL). G is a simple expression, named ground truth. L is the related complex linear MBA expression. NL is the related complex non-linear MBA expression. One example is shown in Fig. 4.

```
Linear MBA rule:
  x + y = 3*(x&y)+8*(x&~y)-1*y-7*~(x&y)+7*~(x|y)+9*~(x|~y)
Non-Linear MBA rule:
  x + y = 2*(x|y)*((x&y)-y)-(x+y)*(x|~y)+(~x&y)*(x^y)
```

Fig. 4. The linear and non-linear MBA expressions for the ground truth expression `x + y`.

MBA Complexity Metrics. We use the following metrics to measure MBA complexity. For these complexity metrics, a larger value indicates a more complex MBA expression.

1. **MBA Alternation.** MBA alternation is to count the number of operations that connect arithmetic operation and bitwise operation. For example, in $(x \wedge y) + 2 * (x \vee y)$, the $+$ represents an MBA alternation operation, because its left operator is a bitwise operator, and its right operator is an arithmetic operator.
2. **Number of Terms.** How many terms are included in an MBA expression.
3. **MBA Length.** The string length of an MBA expression is measured as the MBA length.

MBA Deobfuscation Tools. We collect and check existing, state-of-the-art MBA deobfuscation tools, as described in Sect. 2.2. Arybo[2], MBA-Blast[3], NeuReduce[4], SSPAM[5], and Syntia[6] focus on simplifying MBA expression by bit-blasting, mathematical transformation, machine learning-based, pattern matching, and program synthesis. Arybo is a tool for transforming MBA expression to a symbolic representation at the bit-level written in Python. MBA-Blast is a novel technique to simplify MBA expressions to a normal simple form by arithmetical reduction. SSPAM (Symbolic Simplification with Pattern Matching) is a Python tool for simplifying MBA expression. Syntia is a program synthesis framework for synthesizing the semantic of obfuscated code. It produces input-output pairs from the obfuscated rules and then generates a new simple expression based on these pairs. NeuReduce is a string to string method based on neural networks to learn and reduce complex MBA expressions automatically. We download and apply those tools to simplify MBA expressions for checking the strength of the related obfuscation technique.
Machine Configuration. Our experiments were performed on an Intel Xeon W-2123 4-Core 3.60GHz CPU, with 64GB of RAM, running Ubuntu 18.04.

5.2 Deobfuscation on Non-Linear MBA Expression

In this evaluation, we check the *resilience* of *MBA-Obfuscator*, which refers to the robustness of an obfuscation tool for an automatic deobfuscator. Eyrolles's PhD thesis [8] states that reverse engineer focuses on recovering the initial form of the expression, meaning simplifying the MBA expression. Her experiments show that existing symbolic software cannot simplify MBA expression because math reduction rules only work on pure Boolean expressions(e.g., normalization and constraint solving), or on pure arithmetic expressions(e.g., the algebra laws). So far, no publicly known methods, including both static and dynamic methods, can effectively simplify MBA expressions. This fact attracts researchers to

[2] https://github.com/quarkslab/arybo.
[3] https://github.com/softsec-unh/MBA-Blast.
[4] https://github.com/nhpcc502/NeuReduce.
[5] https://github.com/quarkslab/sspam.
[6] https://github.com/RUB-SysSec/syntia.

develop multiple tools to simplify MBA expression, such as Arybo, MBA-Blast, NeuReduce, SSPAM, and Syntia. Therefore, we test those deobfuscation tools on linear and related non-linear MBA expression. After simplification, we use the Z3 solver [22] library[7] to check whether every simplified result is equivalent to the original expression.

Table 1. Deobfuscation results. ✓ means equivalent(Z3 returns a UNSAT solution), ✗ means not equivalent(Z3 returns an SAT solution), – means time out(deobfuscation tools cannot return a result in 1 h), "Ratio" indicates the ratio of outputs passing equivalence checking, "Time" reports the average processing time (seconds) that each tool takes to process an MBA sample in the ✓ column.

Method	Linear MBA					Non-Linear MBA				
	✓	✗	– –	Ratio(%)	Time	✓	✗	– –	Ratio(%)	Time
Arybo	569	0	431	56.9	1936.5	84	0	916	8.4	2304.9
MBA-Blast	1,000	0	0	100.0	0.06	147	853	0	14.7	0.09
NeuReduce	756	244	0	75.6	0.06	1	999	0	0.0	0.06
SSPAM	386	356	258	38.6	1465.7	103	192	705	10.3	2132.1
Syntia	97	903	0	9.7	29.7	98	902	0	9.8	29.8

Table 2. The complexity distribution of the MBA expressions in the Dataset.

Metrics	Linear MBA			Non-Linear MBA		
	Min	Max	Average	Min	Max	Average
MBA Alternation	2	99	30.6	5	92	30.3
Number of Terms	3	99	31.4	6	79	27.4
MBA Length	17	1498	438.0	47	1140	391.0

Table 1 shows the deobfuscation result on the Dataset. All existing deobfuscation tools can simplify part or all of the linear MBA expressions. However, up to 14.7% of non-linear MBA expressions can be simplified by these tools in 1 h. Arybo only simplifies 84 out of 1,000 samples, because it suffers from severe performance penalties when it normalizes all operators to algebraic normal form. We observe that MBA-Blast performs well on 2-variable polynomial MBA expressions. However, it has a limited simplification effect on other non-linear MBA expressions. Thus, MBA-Blast can simplify 147 non-linear MBA expressions. Since NeuReduce is trained on the dataset of linear MBA expressions, it cannot simplify non-linear MBA expressions. SSPAM can simplify 103 out of 1,000 non-linear MBA expression. Limited by the nature of program synthesis, Syntia only returns 98 correct results for non-linear MBA expressions.

[7] https://github.com/Z3Prover/z3.

5.3 Complexity of Non-Linear MBA Expression

In the second evaluation, we seek to check the capability of *potency* after obfuscation by MBA expressions. *Potency* represents how *complex* or *unreadable* the obfuscated rules are to a reverse engineer. Table 2 shows the complexity results on the Dataset. From Definition 2, the complexity of a linear MBA expression mainly depends on the number of terms in the expression. However, a non-linear MBA expression's complexity depends on multiple factors, such as the number of terms and MBA alternation. Overall, the non-linear MBA expression is slightly simpler than the linear MBA expression from the "Average" column.

Table 3. *MBA-Obfuscator*'s performance for generating a non-linear MBA expression with different complexity. "Time" reports the time (seconds) that *MBA-Obfuscator* generates a non-linear MBA expression 10,000 times.

# Of Terms	Time(Second)	Memory(MB)
10	103.5	0.01
50	259.4	0.01
100	381.1	0.01
200	534.2	0.02

Table 4. Run-time overhead on non-linear MBA expressions with different complexity. The timing result is the time to run the MBA expression 10,000 times repeatedly.

# of Terms	Time (Second)		
	Linear MBA	Non-Linear MBA	Ratio(%)
10	0.5	1.5	300.0
50	3.6	8.2	227.8
100	8.1	22.4	276.5
200	18.3	57.9	316.4

5.4 Cost of Non-Linear MBA Expression

As the last experiment, we study the *cost* of *MBA-Obfuscator*: performance overhead representing the cost for generating an MBA expression, and run-time overhead that refers to the cost when the obfuscated code is running. Table 3 presents the time and memory cost when *MBA-Obfuscator* generates a non-linear MBA expression with different complexity measured by the number of terms. *MBA-Obfuscator* is effective because it relies on the existing linear MBA rules.

For run-time overhead, we integrate MBA expressions into a C program, and then use `gcc` to compile it with the -O2 option. Table 4 shows that run-time overhead introduced by non-linear MBA rules is about 3X as long as the related linear MBA rules. However, the average run-time overhead for a non-linear MBA rule is low, which is less than 0.01 s (0.0058 s).

6 Conclusion

Mixed Boolean-Arithmetic (MBA) expression, which mixes both bitwise and arithmetic operations, can be applied to complicate a simple expression. Our work is the first research to propose a non-linear MBA obfuscation challenge. We investigate the class of MBA expression, and demonstrate how to generate infinite non-linear MBA expressions based on existing linear MBA expressions. Furthermore, we present a practical application of the non-linear MBA obfuscation technique on obfuscating the Tiny Encryption Algorithm(TEA). We develop a prototype tool *MBA-Obfuscator*, a novel non-linear MBA obfuscation technique. Our large-scale experiment demonstrates that *MBA-Obfuscator* is effective and efficient—existing deobfuscation tools may be available to simplify polynomial MBA expressions but hardly for other non-linear MBA expressions, and the cost of applying non-linear MBA obfuscation is low. Developing *MBA-Obfuscator* not only advances the application of the MBA obfuscation technique, but also develops a dataset for future research on MBA deobfuscation direction.

Acknowledgments. We would like to thank our shepherd Roland Yap and anonymous paper reviewers for their helpful feedback. We also thank team members from Anhui Province Key Laboratory of High Performance Computing (USTC) and the UNH SoftSec group for their valuable suggestions.

References

1. Biondi, F., Josse, S., Legay, A.: Bypassing Malware Obfuscation with Dynamic Synthesis. https://ercim-news.ercim.eu/en106/special/bypassing-malware-obfuscation-with-dynamic-synthesis (July 2016)
2. Biondi, F., Josse, S., Legay, A., Sirvent, T.: Effectiveness of synthesis in concolic deobfuscation. Comput. Secur. **70**, 500–515 (2017)
3. Blazy, S., Hutin, R.: Formal verification of a program obfuscation based on mixed boolean-arithmetic expressions. In: Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2019) (2019)
4. Blazytko, T., Contag, M., Aschermann, C., Holz, T.: Syntia: synthesizing the semantics of obfuscated code. In: Proceedings of the 26th USENIX Security Symposium (USENIX Security 2017) (2017)
5. Collberg, C., Martin, S., Myers, J., Nagra, J.: Distributed application tamper detection via continuous software updates. In: Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC 2012) (2012)
6. Collberg, C., Martin, S., Myers, J., Zimmerman, B.: Documentation for Arithmetic Encodings in Tigress. http://tigress.cs.arizona.edu/transformPage/docs/encodeArithmetic

7. Collberg, C., Martin, S., Myers, J., Zimmerman, B.: Documentation for Data Encodings in Tigress. http://tigress.cs.arizona.edu/transformPage/docs/encodeData
8. Eyrolles, N.: Obfuscation with Mixed Boolean-Arithmetic Expressions: Reconstruction, Analysis and Simplification Tools. Ph.D. thesis, Université Paris-Saclay (2017)
9. Eyrolles, N., Goubin, L., Videau, M.: Defeating MBA-based obfuscation. In: Proceedings of the 2016 ACM Workshop on Software PROtection (SPRO 2016) (2016)
10. Feng, W., Liu, B., Xu, D., Zheng, Q., Xu, Y.: Neureduce: Reducing mixed boolean-arithmetic expressions by recurrent neural network. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, pp. 635–644 (2020)
11. Guinet, A., Eyrolles, N., Videau, M.: Arybo: Manipulation, canonicalization and identification of mixed boolean-arithmetic symbolic expressions. In: Proceedings of GreHack 2016 (2016)
12. Gulwani, S., Polozov, O., Singh, R.: Program Synthesis. Found. Trends in Program. Lang. **4**(1–2), 1–119 (2017)
13. Irdeto: Irdeto Cloaked CA: a secure, flexible and cost-effective conditional access system. www.irdeto.com (2017)
14. Israsena, P.: Securing ubiquitous and low-cost rfid using tiny encryption algorithm. In: 2006 1st International Symposium on Wireless Pervasive Computing, pp. 4-pp. IEEE (2006)
15. Lattner, C., Adve, V.: LLVM: a compilation framework for lifelong program analysis & transformation. In: Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization (CGO 2004) (2004)
16. Leroy, X.: Formal verification of a realistic compiler. Commun. ACM **52**, 107–115 (2009)
17. Liem, C., Gu, Y.X., Johnson, H.: A compiler-based infrastructure for software-protection. In: Proceedings of the 3rd ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS 2008) (2008)
18. Liu, B., Shen, J., Ming, J., Zheng, Q., Li, J., Xu, D.: Mba-blast: unveiling and simplifying mixed boolean-arithmetic obfuscation. In: 30th USENIX Security Symposium (USENIX Security 2021) (2021)
19. Ma, H., Jia, C., Li, S., Zheng, W., Wu, D.: Xmark: dynamic software watermarking using collatz conjecture. IEEE Trans. Inf. Forensics Secur. **14**, 2859–2874 (2019)
20. MapleSoft: The Essential Tool for Mathematics. https://www.maplesoft.com/products/maple/ (2020)
21. Mougey, C., Gabriel, F.: DRM obfuscation versus auxiliary attacks. In: REcon Conference (2014)
22. Moura, L.D., Bjørner, N.: Z3: an efficient SMT solver. In: Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2008)
23. Quarkslab: Epona Application Protection v1.5. https://epona.quarkslab.com (July 2019)
24. Rahim, R., et al.: Tiny encryption algorithm and pixel value differencing for enhancement security message. Int. J. Eng. Technol. **7**(2.9), 82–85 (2018)
25. Sagemath: SageMath. http://www.sagemath.org/ (2020)
26. Schrittwieser, S., Katzenbeisser, S., Kinder, J., Merzdovnik, G., Weippl, E.: Protecting software through obfuscation: Can it keep pace with progress in code analysis? ACM Comput. Surv. (CSUR) **49**(1), 1–37 (2016)

27. Suwartadi, E., Gunawan, C., Setijadi, A., Machbub, C.: First step toward internet based embedded control system. In: IEEE 5th Asian Control Conference, vol. 2, pp. 1226–1231 (2004)
28. Wheeler, D.J., Needham, R.M.: TEA, a tiny encryption algorithm. In: Proceedings of the 2nd International Workshop on Fast Software Encryption (1994)
29. WOLFRAM: WOLFRAM MATHEMATICA. http://www.wolfram.com/mathematica/ (2020)
30. Zafar, F., Olano, M., Curtis, A.: Gpu random numbers via the tiny encryption algorithm. In: Proceedings of the Conference on High Performance Graphics, pp. 133–141 (2010)
31. Zhou, Y., Main, A.: Diversity via Code Transformations: A Solution for NGNA Renewable Security. The National Cable and Telecommunications Association Show (2006)
32. Zhou, Yongxin, Main, Alec, Gu, Yuan X., Johnson, Harold: Information hiding in software with mixed boolean-arithmetic transforms. In: Kim, Sehun, Yung, Moti, Lee, Hyung-Woo (eds.) WISA 2007. LNCS, vol. 4867, pp. 61–75. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77535-5_5