Compositional Verification of Efficient Masking Countermeasures against Side-Channel Attacks*

PENGFEI GAO, ShanghaiTech University, China
YEDI ZHANG, ShanghaiTech University, China
FU SONG†, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China
TAOLUE CHEN, Birkbeck, University of London, UK
FRANCOIS-XAVIER STANDAERT, Université Catholique de Louvain, Belgium

Masking is one of the most effective countermeasures for securely implementing cryptographic algorithms against power side-channel attacks, the design of which however turns out to be intricate and error-prone. While techniques have been proposed to rigorously verify implementations of cryptographic algorithms, currently they are limited in scalability. To address this issue, compositional approaches have been investigated, but insofar they fail to prove the security of recent efficient implementations. To fill this gap, we propose a novel compositional verification approach. In particular, we introduce two new language-level security notions based on which we propose composition strategies and verification algorithms. Our approach is able to prove efficient implementations, which cannot be done by prior compositional approaches. We implement our approach as a tool CONVINCE and conduct extensive experiments to confirm its efficacy. We also use CONVINCE to further explore the design space of the AES Sbox with least refreshing by replacing its implementation for finite-field multiplication with more efficient counterparts. We automatically prove leakage-freeness of these new versions. As a result, we can effectively reduce 1,600 randomness and 3,200 XOR-operations of the state-of-the-art AES implementation.

CCS Concepts: • **Security and privacy → Side-channel analysis and countermeasures**; • **Software and its engineering → Software verification**; **Automated static analysis**.

Additional Key Words and Phrases: Formal verification, power side-channel attacks, cryptographic implementations, countermeasures, compositional reasoning

ACM Reference Format:
Pengfei Gao, Yedi Zhang, Fu Song, Taolue Chen, and Francois-Xavier Standaert. 2023. Compositional Verification of Efficient Masking Countermeasures against Side-Channel Attacks. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 286 (October 2023), 31 pages. https://doi.org/10.1145/3622862

*This work is partly supported by the National Natural Science Foundation of China under Grant No.: 62072309, CAS Project for Young Scientists in Basic Research under Grant No.: YSBR-040, ISCAS New Cultivation Project under Grant No.: ISCAS-PYFX-202201, State Key Laboratory of Novel Software Technology, Nanjing University under Grant No.: KFKT2022A03, and Birkbeck BEI School Project EFFECT.
†Corresponding authors

Authors' addresses: Pengfei Gao, ShanghaiTech University, Shanghai, China, gaopf@shanghaitech.edu.cn; Yedi Zhang, ShanghaiTech University, Shanghai, China, zhangyd1@shanghaitech.edu.cn; Fu Song, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China, songfu@ios.ac.cn; Taolue Chen, Birkbeck, University of London, London, UK, t.chen@bbk.ac.uk; Francois-Xavier Standaert, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, fstandae@uclouvain.be.

2475-1421/2023/10-ART286
https://doi.org/10.1145/3622862

1 INTRODUCTION

Power side-channel attacks, capable of inferring secrets by exploiting power consumption during the execution of cryptographic implementations, have raised severe security concerns [Kocher et al. 1999]. Implementations of almost all major cryptographic algorithms, such as DES [Kocher et al. 1999], AES [Prouff et al. 2009; Wang et al. 2018], RSA [Goubin and Patarin 1999], Elliptic curve cryptography [Coron 1999; Itoh et al. 2002; Luo et al. 2018] and post-quantum cryptography [Kannwischer et al. 2018; Ravi et al. 2019; Schamberger et al. 2020], have been the victims.

A widely used effective countermeasure against power side-channel attacks is *masking* [Ishai et al. 2003]. In a nutshell, given a masking order t, an order-t secret masking scheme splits the secret into $t+1$ shares such that any proper subset of these shares is statistically independent of the secret. For a given cryptographic algorithm f with k as its secret key, a masked version f' should be implemented such that it takes $t+1$ shares of k as inputs and produces $t+1$ shares of $f(k)$ from which the desired output $f(k)$ could be recovered. The security of the masked version f' is usually established using the concept of *d-probing security* [Ishai et al. 2003] for a given security order d, requiring that the joint distribution of each set with at most d observable values is statistically independent of the secret k. The usefulness of increasing the security order d has been justified by [Duc et al. 2015], namely, under reasonable assumptions, the number of physical measurements needed for a successful attack increases exponentially in d, so a higher d would imply considerably more difficulties in mounting an attack.

Unsurprisingly, it is error-prone to implement secure and efficient masked versions for non-linear functions such as finite-field multiplication and Sbox [Ishai et al. 2003], and the program size and number of random bits blow up polynomially in the masking order t. However, it is not uncommon that published implementations that have been proved secure via paper-and-pencil [Carlet et al. 2012; Kim et al. 2011; Rivain and Prouff 2010] were later shown to be vulnerable to power side-channels [Coron et al. 2013]. We also note that efficiency is constantly a major concern for implementations of cryptographic algorithms in, e.g., resource-limited devices [Biryukov et al. 2017]. Overall, the crux of masking countermeasures for cryptographic algorithms is to devise *efficient* and *secure* implementations over the shares.

To address the efficiency, several masked implementations for finite-field multiplication were proposed [Barthe et al. 2020, 2017; Belaïd et al. 2016, 2017; Bordes and Karpman 2021; Groß and Mangard 2018; Karpman and Roche 2018; Wang et al. 2020], which is a key building block to implement most cryptographic algorithms. The need of randomness and/or the number of operations in these implementations have been reduced over the original one proposed by [Ishai et al. 2003]. On the security side, various techniques have been proposed to formally verify probing security. Existing efforts rely upon heuristic rules (e.g., [Barthe et al. 2015; Bayrak et al. 2013]), SAT/SMT solving (e.g., [Bloem et al. 2018; Eldib et al. 2014]) or their combination (e.g., [Gao et al. 2022, 2019a,b; Zhang et al. 2018]). All those approaches are *intra-procedural*, meaning that they only deal with procedures without calling any procedures (called simple gadgets in this work and in related works). Henceforth they are referred to as *non-compositional* approaches. Non-compositional approaches become intractable for (large) composite gadgets (i.e., procedures with calls to some procedures) when procedure inline is applied.

In contrast, compositional reasoning, which verifies simple gadgets in isolation and then checks tailored compositional rules, turns out to be a promising direction for verifying composite gadgets without applying inline. This direction dates back to the seminal work of [Barthe et al. 2016] in which two new stronger security notions, called d-Non-Interference (d−NI) and d-Strong Non-Interference (d−SNI), are proposed to soundly characterize the d-probing security model as a

property in programming languages, based on which masked implementations can be verified compositionally. Along with this direction more verification approaches have been proposed [Barthe et al. 2020, 2021; Belaïd et al. 2020, 2018, 2022; Bordes and Karpman 2021; Coron 2018; Knichel et al. 2020], most of which are designated to precisely and efficiently verify simple gadgets. Recently, an alternative stronger security notion and compositional approach have been proposed [Cassiers et al. 2021; Knichel et al. 2020]. These language-level security notions and tools have been widely adopted for designing and rigorously verifying implementations of cryptographic algorithms, resulting in many verified cryptographic programs. However, though promising, these compositional approaches are achieved by sacrificing the efficiency of masked implementations.

Up to now, it is fair to say no efficient method exists to check the probing security of composition of gadgets, in particular, efficient gadgets. For instance, all existing compositional approaches would fail to prove d-probing security of the efficient implementation of the AES Sbox [Goudarzi and Rivain 2017] when more efficient finite-field multiplications, e.g., those proposed by [Barthe et al. 2017; Belaïd et al. 2016; Groß and Mangard 2018], are used. This is mainly because efficient gadgets use fewer random variables. As a result, they do not satisfy the stronger security notions of [Barthe et al. 2016; Cassiers et al. 2021] which are required to make the previous compositional reasoning approaches work. We remark that efficient gadgets play an increasingly vital role especially in resource constrained devices while compositional verification of efficient gadgets is very challenging and requires novel techniques, e.g., a new technique is proposed to verify efficient gadgets [Belaïd et al. 2020, 2018] on which [Barthe et al. 2016] fails, but [Belaïd et al. 2020, 2018] do not support more efficient gadgets.

Our contributions. We propose a novel compositional verification approach for efficient masking countermeasures. On the theoretical side, we propose two new security notions to characterize the d-probing security model in programming languages and study their compositionality based on which we present composition strategies for verifying composite gadgets without the inlining of gadgets. Our security notions are proper generalizations of the ones proposed in [Barthe et al. 2016], thus, can be used to prove the security of composite gadgets that cannot be achieved using existing compositional approaches. On the practical side, we derive verification algorithms from the new security notions. We propose algorithms for verifying our security notions for both simple and composite gadgets, and algorithms for verifying d-probing security and the security notions of [Barthe et al. 2016].

We have implemented our approach in a tool CONVINCE (**C**ompositi**ON**al **V**erif**I**catio**N** of masking **C**ounterm**E**asures) and conduct extensive evaluations. The benchmarks are derived from existing benchmarks by replacing the implementation for finite-field multiplication with more efficient ones. None of these benchmarks can be proved using the existing compositional approaches, and many of them are actually proved probing secure and/or d-NI for the first time. To demonstrate the usage of our tool, we apply it to explore the design space of the AES Sbox with the least refreshing [Belaïd et al. 2018]. We show that replacing some of them with functional-equivalent but more efficient implementations for finite-field multiplication (e.g., those proposed by [Barthe et al. 2017; Belaïd et al. 2016; Groß and Mangard 2018]) does not compromise security. As a result, we find more efficient implementations of the AES Sbox whose probing security can be automatically proved by our tool, but could not be proved using existing compositional approaches. For instance, our AES Sbox implementations reduced randomness from 92 to 68 for 2nd-order probing security and from 192 to 172 for 3rd-order probing security. We also devise provable secure full AES which effectively reduced 1,600 randomness and 3,200 XOR-operations of the state-of-the-art 3rd-order AES implementation [Belaïd et al. 2018].

$$
\begin{array}{lrcl}
\text{Operation:} & \mathbf{Op} \ni \circ & ::= & \wedge \mid \vee \mid \oplus \mid - \mid + \mid \times \mid \odot \\
\text{Expression:} & e & ::= & c \mid x \mid \neg e \mid e \circ e \\
\text{Statement:} & \texttt{stmt} & ::= & x = e \mid r = \$ \mid \texttt{stmt}; \texttt{stmt} \\
\text{Simple gadget:} & G & ::= & f(\mathbf{a}_1, \cdots, \mathbf{a}_m)\{\texttt{stmt}; \texttt{return } \mathbf{b}_1, \cdots, \mathbf{b}_k;\} \\
\text{Gadget call:} & \texttt{gstmt} & ::= & \mathbf{y}_1, \cdots, \mathbf{y}_k =_\ell f(\mathbf{x}_1, \cdots, \mathbf{x}_m) \mid \texttt{gstmt}; \texttt{gstmt} \\
\text{Composite gadget:} & F & ::= & f(\mathbf{a}_1, \cdots, \mathbf{a}_m)\{\texttt{gstmt}; \texttt{return } \mathbf{b}_1, \cdots, \mathbf{b}_k;\}
\end{array}
$$

Fig. 1. Syntax of the language.

The contributions can be summarized as follows: (1) We propose two novel language-level security notions and study their compositionality properties, which serve as the basis of compositional reasoning of masked implementations of cryptographic algorithms. (2) We provide efficient algorithms for verifying our security notions and existing security notions, all of which are implemented into a tool CONVINCE. (3) We conduct extensive experiments on efficient implementations of cryptographic algorithms and find several more efficient implementations of AES Sbox that are proved leakage-free via CONVINCE.

Outline. Section 2 introduces the basic notions and an illustrating example. Section 3 proposes the new language-level security notions and studies their properties. Section 4 presents algorithms for verifying both simple and composite gadgets. Section 5 reports experimental results. We discuss the related work in Section 6 and conclude this work in Section 7.

2 PRELIMINARIES

We fix the finite-field $\mathbb{F} = \{0, \cdots, 2^n - 1\}$, the masking order $t \geq 1$ and the security order $d \leq t$. We use lowercase letters (e.g., $x, y, a, a_1, \cdots$) to range over scalars on $\mathbb{F}$, and **bold** lowercase letters (e.g., $\mathbf{x}, \mathbf{y}, \mathbf{a}, \mathbf{a}_1, \cdots$) to range over vectors of size $(t+1)$, also known as *sharings*. We normally assume that $\mathbf{x}$ is the sharing of the scalar x. The i-th entry of $\mathbf{x}$, denoted by $\mathbf{x}[i]$, is referred to as a *share* of $\mathbf{x}$. By slight abuse of notation, a vector $\mathbf{x}$ is deemed to be identical to the set of all its entries $\{\mathbf{x}[1], \cdots, \mathbf{x}[t+1]\}$, and $\bigoplus \mathbf{x}$ denotes $\mathbf{x}[1] \oplus \cdots \oplus \mathbf{x}[t+1]$.

2.1 Language

The syntax of the language is given in Figure 1 which can be used to describe both hardware circuits and software programs. The language is designed for implementing cryptographic algorithms (in particular, symmetric ciphers). It does not support tests (e.g., if-then-else), the same as prior work [Barthe et al. 2020, 2016, 2021; Belaïd et al. 2020, 2018; Bloem et al. 2018; Bordes and Karpman 2021; Cassiers et al. 2021; Cassiers and Standaert 2020; Coron 2018; Eldib et al. 2014; Knichel et al. 2020; Zhang et al. 2018], but it supports bounded loops which can be fully unrolled before verification (though we only present the core language without loops).

An expression e is built up from variables and constants using bitwise logical operations: *and* ($\wedge$), *or* ($\vee$), *exclusive-or* ($\oplus$), *negation* ($\neg$); modulo 2^n arithmetic operations: *subtraction* ($-$), *addition* ($+$), *multiplication* ($\times$) for which $\mathbb{F}$ is considered to be $\mathbb{Z}_{2^n}$; and finite-field multiplication ($\odot$).

An assignment of the form $x = e$ is defined as usual and of the form $r = \$$ assigns a uniformly sampled value to the variable r. As a result, r should be read as a random variable.

Definition 1 (Simple gadgets). *A* simple gadget

$$f(\mathbf{a}_1, \cdots, \mathbf{a}_m)\{\texttt{stmt}; \texttt{return } \mathbf{b}_1, \cdots, \mathbf{b}_k;\}$$

is given by the gadget name f, and formal arguments $(\mathbf{a}_1, \cdots, \mathbf{a}_m)$. Its body consists of a sequence of assignments followed by a return statement, where $\mathbf{a}_1, \cdots, \mathbf{a}_m$ and $\mathbf{b}_1, \cdots, \mathbf{b}_k$ are called input sharings

and output sharings *respectively, and members of* $\mathbf{a}_i$ *(resp.* $\mathbf{b}_i$*) are called* input shares *(resp.* output shares*).*

For simple gadgets, we assume that each variable is assigned at most once (i.e., in the static single assignment form). Thus, a simple gadget can be seen as a (randomized) hardware circuit or software procedure where assigned variables have unique names.

A *probe* on a simple gadget refers to a variable x in the gadget whose value can be observed by the adversary via power side-channels, where $\mathcal{E}(x)$ denotes its (symbolic) computation over input shares and random variables, and $\mathrm{Var}(\mathcal{E}(x))$ denotes those shares and random variables. We may further distinguish between *external* probes on the output sharings and the remaining *internal* probes including those on the input shares and other internal variables. Furthermore, variables appearing on the right-hand side of assignments are called *local variables*. An *evaluation* of the simple gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ on the inputs $\mathbf{x}_1, \cdots, \mathbf{x}_m$ under a probe set O, denoted by $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)_O$, refers to the joint distribution of the variables in O when the simple gadget $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ is evaluated. Formally, given the joint distribution μ of the inputs $\mathbf{x}_1, \cdots, \mathbf{x}_m$, $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)_O$ is the joint distribution of the probes in O, where random variables are sampled from the uniform distribution and the inputs $\mathbf{x}_1, \cdots, \mathbf{x}_m$ sampled from the joint distribution μ.

Definition 2 (Composite gadgets). *A* composite gadget

$$f(\mathbf{a}_1, \cdots, \mathbf{a}_m)\{\texttt{gstmt}; \texttt{return}\ \mathbf{b}_1, \cdots, \mathbf{b}_k;\}$$

is defined similar to the simple gadget, except that its body consists of a sequence of gadget calls followed by a return statement, where a gadget call $\mathbf{y}_1, \cdots, \mathbf{y}_k =_\ell g(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ *associated with a unique label* ℓ *passes the inputs* $\mathbf{x}_1, \cdots, \mathbf{x}_m$ *to the formal arguments* $\mathbf{a}_1, \cdots, \mathbf{a}_m$ *of* g *and assigns the output sharings of* g *to* $\mathbf{y}_1, \cdots, \mathbf{y}_k$*.*

In Definition 2, the same gadgets can be called multiple times, which can ease the implementation of cryptographic algorithms, we require a unique label ℓ for each gadget call. A composite gadget can be transformed into an equivalent simple gadget by gadget inline. Inlining a gadget call $\mathbf{y}_1, \cdots, \mathbf{y}_k =_\ell g(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ amounts to replacing it by the body of the gadget g where the formal arguments are replaced by the corresponding inputs, the local variables are appended with $@\ell$ (e.g., x becomes $x@\ell$) to avoid name conflict, and the return statement is replaced by the assignments that mimic the return of the gadget call. We will denote by $f_{\mathtt{in}}(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ the simple gadget counterpart of $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$, obtained by iteratively inlining all the gadget calls.

A *probe* on a composite gadget $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ refers to a probe on its simple gadget counterpart $f_{\mathtt{in}}(\mathbf{x}_1, \cdots, \mathbf{x}_m)$. Similarly, an *evaluation* of a composite gadget $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ on the inputs $\mathbf{x}_1, \cdots, \mathbf{x}_m$ under a probe set O, denoted by $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)_O$, refers to the joint distribution of the probes O when $f_{\mathtt{in}}(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ is evaluated.

We remark that the sharing $\mathbf{x}$ of a variable x comprises t random numbers $\mathbf{x}[1], \cdots, \mathbf{x}[t]$ and the remaining one $\mathbf{x}[t+1]$ is a computation of x and the other shares $\mathbf{x}[1], \cdots, \mathbf{x}[t]$. There are two main secret sharing schemes, i.e., Boolean secret masking scheme where $\mathbf{x}[t+1] = \mathbf{x}[1] \oplus \mathbf{x}[2] \oplus \cdots \oplus \mathbf{x}[t] \oplus x$ and arithmetic secret masking scheme where $\mathbf{x}[t+1] = \mathbf{x}[1] + \mathbf{x}[2] + \cdots + \mathbf{x}[t] + x$. The former is often used to implement cryptographic algorithms that use only logical operations (e.g., $\wedge$ and $\vee$) and Sbox operations (e.g., AES and DES [Coron et al. 2014]), while the latter could be used to implement cryptographic algorithms that use arithmetic operations such as $+$ and $-$ (e.g., IDEA [Lai and Massey 1990]). For algorithms that use both arithmetic and logical operations, conversion algorithms between two secret masking schemes (e.g., [Coron et al. 2015, 2014; Goubin 2001]) are used. Our language supports both logical and arithmetic operations, hence can describe hardware circuits and software programs using both Boolean and arithmetic secret masking schemes.

```
XORMULTI(a, b){
    e =1 Refresh(b);
    c =2 XOR(a, e);     d =3 CM(a, c);
    return d; }
Refresh(a){
    r1 = $;    r2 = $;          m1 = a[1] ⊕ r1;    m2 = a[2] ⊕ r1;    m3 = a[3] ⊕ r2;
    r3 = $;    c[1] = m1 ⊕ r2;  c[2] = m2 ⊕ r3;    c[3] = m3 ⊕ r3;
    return c; }
CM(a, b){ //CompressedMulti
    t1 = a[1] ⊙ b[1];   t2 = a[1] ⊙ b[3];   t3 = a[3] ⊙ b[1];   t4 = a[2] ⊙ b[2];
    t5 = a[1] ⊙ b[2];   t6 = a[2] ⊙ b[1];   t7 = a[3] ⊙ b[3];   t8 = a[2] ⊙ b[3];   t9 = a[3] ⊙ b[2];
    r1 = $;   t10 = t1 ⊕ r1;   t11 = t10 ⊕ t2;   t12 = t11 ⊕ t3;   t13 = t4 ⊕ r2;   t14 = t13 ⊕ t5;
    r2 = $;   t15 = t14 ⊕ t6;   t16 = t7 ⊕ r1;   t17 = t16 ⊕ r2;   t18 = t17 ⊕ t8;   t19 = t18 ⊕ t9;
    return (t12, t15, t19); } // t12 ⊕ t15 ⊕ t19 = (⨁ a) ⊙ (⨁ b)

XOR(a, b){
    for(i = 1; i ≤ 3; i++)
        c[i] = a[i] ⊕ b[i];
    return c; }
```

Fig. 2. Details of XORMULTI, where //... denote comments.

2.2 Probing Security

We use the probing security [Ishai et al. 2003] to establish the security of masked implementations.

Definition 3 (Probing security). *A gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is d-*probing secure *on the inputs $\mathbf{x}_1, \cdots, \mathbf{x}_m$ iff for any set O of at most d probes, the evaluation $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)_O$ is (statistically) independent of the secret.*

Intuitively, d-probing security ensures that the adversary cannot infer any information of the secret when observing the values of any d probes. Note that the inputs $\mathbf{x}_1, \cdots, \mathbf{x}_m$ depend on the secret, otherwise we can directly deduce that $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)_O$ is independent of the secret. When $d \geq 2$, it is often called higher-order otherwise first-order.

Suppose $\mathbb{F} = \{0, 1\}$. Consider $\mathcal{E}(x_1) = k \oplus r$, where r is a random variable and k is the secret. Then, for any value of k, the probability of $x_1 = 1$ is 50%, thus, the distribution of x_1 is independent of k and observing x_1 cannot infer any information of k. However, when $\mathcal{E}(x_2) = k \wedge r$, the probability of $x_2 = 1$ is 50% if $k = 1$ while the probability of $x_2 = 1$ is 0% if $k = 0$, thus, the distribution of x_2 depends upon k and observing the value of x_2 can infer the value of k.

2.3 Illustrating Example

Consider the 2nd-order masked composite gadget XORMULTI (cf. Figure 2) for computing the sharing $\mathbf{d}$ from two sharings $\mathbf{a}$ and $\mathbf{b}$ such that $\bigoplus \mathbf{d} = \bigoplus \mathbf{a} \odot (\bigoplus \mathbf{a} \oplus \bigoplus \mathbf{b})$, where $\mathbf{a}$ and $\mathbf{b}$ are the sharing of two secrets a and b using Boolean secret sharing scheme. XORMULTI computes the sharing $\mathbf{d}$ by invoking the simple gadgets Refresh, XOR and CM. The gadget Refresh is used to re-mask the sharing $\mathbf{b}$ using new random values [Ishai et al. 2003] otherwise XORMULTI has leakage, leading to $\bigoplus \mathbf{e} = \bigoplus \mathbf{b}$. The gadget XOR is a standard sharewise addition [Ishai et al. 2003], which takes the sharings $\mathbf{a}$ and $\mathbf{b}$ as inputs and computes the sharing $\mathbf{c}$ such that $\bigoplus \mathbf{c} = \bigoplus \mathbf{a} \oplus \bigoplus \mathbf{b}$. The gadget CM is a compressed, 2nd-order masked gadget for finite-field multiplication [Belaïd et al. 2016] which takes two sharings $\mathbf{a}$ and $\mathbf{b}$ as inputs and computes the sharing (t_{12}, t_{15}, t_{19}) such that $t_{12} \oplus t_{15} \oplus t_{19} = (\bigoplus \mathbf{a}) \odot (\bigoplus \mathbf{b})$.

While the simple gadgets CM, XOR and Refresh can be proved of 2-probing secure by existing tools (e.g., maskVerif [Barthe et al. 2019]), no existing compositional approaches [Barthe et al. 2016; Belaïd et al. 2016, 2020, 2018; Blot et al. 2017; Cassiers et al. 2021; Gao et al. 2022] can prove that the composite gadget XORMULTI is 2-probing secure.

We point out why the d-NI/d-SNI based compositional approach [Barthe et al. 2016; Belaïd et al. 2016] fails. In general, when reasoning about a composite gadget, their compositional approach first

iteratively collects constraints in the first-order theory of finite sets with cardinality constraints from the bottom gadget call up to the top one using d-NI/d-SNI properties of the invoked gadgets. The cardinalities of shares will be cumulated when they are reused by each non-d-SNI gadget in order to guarantee soundness. Finally, the collected cardinality constraints are used to check if all the probes can be simulated with input shares whose cardinality is bounded by the cardinality of probes. However, such information may not be sufficient for proving some gadgets.

Consider the gadget `XORMULTI`. To prove 2-NI of `XORMULTI`, the total number of probes used to attack is limited to 2, that is $|O_1| + |O_2| + |O_3| \leq 2$, where O_1, O_2 and O_3 are probe sets within three gadgets, respectively. As in [Barthe et al. 2016; Belaïd et al. 2016],

(1) `CM` is 2-NI, thus $|S_3^1| \leq |O_3|$ and $|S_3^2| \leq |O_3|$, where S_3^1 and S_3^2 respectively denote the set of input shares of $\mathbf{c}$ and $\mathbf{a}$ used to simulate the probe set O_3;
(2) `XOR` is 2-NI, thus $|S_2^1| \leq |O_2| + |O_3|$ and $|S_2^2| \leq |O_2| + |O_3|$, where S_2^1 and S_2^2 respectively denote the set of input shares of $\mathbf{e}$ and $\mathbf{a}$ used to simulate the probe set $O_2 \cup O_3$;
(3) `Refresh` is 2-SNI, thus $|S_1^1| \leq |O_1|$, where S_1^1 denotes the set of input shares of $\mathbf{b}$ used to simulate the probe set O_1.

Finally, the probe set $O_1 \cup O_2 \cup O_3$ should be simulated by $S_1^1 \cup S_2^2 \cup S_3^2$, namely, $|S_1^1 \cup S_2^2 \cup S_3^2| \leq |O_1| + |O_2| + |O_3|$ should hold. However, one can only deduce $|S_1^1 \cup S_2^2 \cup S_3^2| \leq |O_1| + |O_2| + 2|O_3|$, thus fails to prove that `XORMULTI` is 2-NI. We will show later that our approach is able to prove that `XORMULTI` indeed is 2-NI.

3 NEW LANGUAGE-LEVEL SECURITY NOTIONS

Before formalizing our new security notions, we first introduce some notations, as well as the concept of simulatability inspired by the one introduced by [Barthe et al. 2016; Belaïd et al. 2016]. It will be used to define security notions as well as verification algorithms for proving security. More specifically, by leveraging the concept of simulatability, we derive a variable set for each probe set in a gadget so that the joint distribution of the probe set can be simulated by only knowing the values of variables in the variable set. Furthermore, if such variable sets are independent of the secret input, all the probe sets are also independent of the secret input. Note that the simulatability defined by [Barthe et al. 2016; Belaïd et al. 2016] only uses input shares to simulate the joint distribution of a probe set. In contrast, ours is more general, where not only input shares but also local variables can be used to simulate the joint distribution of a probe set. This generalization allows our approach to be applicable for verifying efficient masked implementations that cannot be handled before.

Hereafter, we use blackboard-bold UPPERCASE letters, e.g., $\mathbb{I}$, $\mathbb{O}$, $\mathbb{Y}$, etc., to range over families of variable sets. For two families $\mathbb{Y}_1$, $\mathbb{Y}_2$ of variable sets, we denote by $\mathbb{Y}_2 \sqsubseteq \mathbb{Y}_1$ if for every $Y_2 \in \mathbb{Y}_2$, there exists $Y_1 \in \mathbb{Y}_1$ such that $Y_2 \subseteq Y_1$.

Definition 4 (Simulatability). *Fix a gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$, a probe set O on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ can be simulated by a set I of variables, called I-*simulatable*, iff there exists a randomized function $\pi : \mathbb{F}^{|I|} \to \mathbb{F}^{|O|}$ such that for any fixed tuple of values $(v_1, \cdots, v_{|I|}) \in \mathbb{F}^{|I|}$ and any inputs $\mathbf{x}_1, \cdots, \mathbf{x}_m$, the joint distributions $\pi(v_1, \cdots, v_{|I|})$ and $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)_O$ are the same when the values of I in $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ are limited to $(v_1, \cdots, v_{|I|})$.*

Intuitively, a probe set O on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is I-simulatable if there exists a randomized function that simulates the joint distribution $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)_O$ while requiring only the values of the variables in I. Given a set $\mathbb{I}$ of variable sets, a probe set O on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $\mathbb{I}$-*simulatable* if it is $\bigcup \mathbb{I}$-simulatable.

Consider the probe sets $O_1 = \{t_1, t_2\}$ and $O_2 = \{t_1, t_{10}\}$ in the `CM` gadget in Figure 2. Since $t_1 = \mathbf{a}[1] \odot \mathbf{b}[1]$ and $t_2 = \mathbf{a}[1] \odot \mathbf{b}[3]$, we get that O_1 is $\{\mathbf{a}[1], \mathbf{b}[1], \mathbf{b}[3]\}$-simulatable. Since $t_{10} = t_1 \oplus r_1$ and r_1 is a random variable, we get that O_2 is $\{\mathbf{a}[1], \mathbf{b}[1]\}$-simulatable.

3.1 $(\mathbb{Y}, d)$-Non-Interference

We first recall the notion of non-interference [Barthe et al. 2016] which is a sound language-level characterization of the d-probing security for compositional reasoning. Next, we present our new security notion by generalizing non-interference with a family of variable sets.

Definition 5 (Non-Interference). *A gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is* d-Non-Interfering, d-NI *for short, if for any set O of at most d probes, there exists a set of variables I such that I only contains at most $|O|$ shares of each input sharing $\mathbf{a}_i$ and O is I-simulatable on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$.*

Intuitively, knowing the values of the variables in I suffices to simulate the joint distribution of the probe set O to the adversary and I contains at most $|O|$ shares of $\mathbf{a}_i$ for every $1 \leq i \leq m$.

It was shown [Barthe et al. 2016] that a d-NI gadget is d-probing secure if any two input sharings are mutually independent. However, a d-probing secure gadget (provided that any two input sharings are mutually independent) is not necessarily d-NI (cf. [Gao et al. 2023] for a concrete example). To overcome this limitation, we propose the notion of variable non-interference by generalizing d-NI with a family of variable sets $\mathbb{Y}$.

Definition 6 (Variable Non-Interference). *Given a family $\mathbb{Y}$ of variable sets, a gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is* $(\mathbb{Y}, d)$-Non-Interfering, $(\mathbb{Y}, d)$-NI *for short, if for any set O of at most d probes, there exists a subset of variable sets $\mathbb{I} \subseteq \mathbb{Y}$ such that $|\mathbb{I}| \leq |O|$ and O is $\mathbb{I}$-simulatable on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$.*

Intuitively, knowing the values of the variables in $\bigcup \mathbb{I}$ suffices to simulate the joint distribution of the probe set O to the adversary and $\mathbb{I}$ contains at most $|O|$ variable sets of $\mathbb{Y}$, i.e., $|\mathbb{I}| \leq |O|$. Note that in contrast to d-NI, $\mathbb{Y}$ in $(\mathbb{Y}, d)$-NI can contain local variables. Indeed, there exist gadgets where $\mathbb{Y}$ should contain local variables instead of input shares *only*. In general, a local variable whose computation relies upon some random variables but is not perfectly masked by any random variables will be added to the set Y. Hence, they are $(\mathbb{Y}, d)$-NI for proper sets $\mathbb{Y}$ but not d-NI. A concrete example is given in [Gao et al. 2023].

We can observe that if $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-NI, each share $\mathbf{a}_i[j]$ must occur in some set $I \in \mathbb{Y}$, i.e., $\mathbf{a}_i[j] \in \bigcup \mathbb{Y}$. Indeed, if some share $\mathbf{a}_i[j]$ does not occur in any set $I \in \mathbb{Y}$, i.e., $\mathbf{a}_i[j] \notin \bigcup \mathbb{Y}$, we can conclude that $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is not $(\mathbb{Y}, d)$-NI, as the probe $\mathbf{a}_i[j]$ cannot be simulated by any subset $\mathbb{I} \subseteq \mathbb{Y}$. Due to this, hereafter, we assume that $\bigcup_{i=1}^{m} \mathbf{a}_i \subseteq \bigcup \mathbb{Y}$, when we consider $(\mathbb{Y}, d)$-NI.

We exemplify the security notions on various simple gadgets. Consider the simple gadgets `CM`, `XOR` and `Refresh` in Figure 2, for which we define the following three families of variable sets: $\mathbb{Y}_{\mathtt{CM}} = \{\{\mathbf{a}[i], \mathbf{b}[j]\} \mid 1 \leq i, j \leq 3\}$, $\mathbb{Y}_{\mathtt{XOR}} = \{\{\mathbf{a}[i], \mathbf{b}[i]\} \mid 1 \leq i \leq 3\}$, and $\mathbb{Y}_{\mathtt{RF}} = \{\{\mathbf{a}[i]\} \mid 1 \leq i \leq 3\}$. Then, `CM` is $(\mathbb{Y}_{\mathtt{CM}}, 2)$-NI, `XOR` is $(\mathbb{Y}_{\mathtt{XOR}}, 2)$-NI and `Refresh` is $(\mathbb{Y}_{\mathtt{RF}}, 2)$-NI. For instance, the probe set $\{t_6, t_7\}$ on the gadget `CM` can be simulated by the set $\{\{\mathbf{a}[2], \mathbf{b}[1]\}, \{\mathbf{a}[3], \mathbf{b}[3]\}\}$. However, the composite gadget `CM` is not $(\mathbb{Y}_{\mathtt{XOR}}, 2)$-NI, as the probe set $\{t_6, t_7\}$ on the gadget `CM` cannot be simulated by any variable set of $\mathbb{Y}_{\mathtt{XOR}}$ with size 2. By examining $\mathbb{Y}_{\mathtt{CM}}$, $\mathbb{Y}_{\mathtt{XOR}}$ and $\mathbb{Y}_{\mathtt{RF}}$, we can see that the simple gadgets `CM`, `XOR` and `Refresh` are all 2-NI.

Consider the ISW t-order masked implementation `SecMult` for finite-field multiplication in Figure 3. For any security order $d \leq t$, define $\mathbb{Y}_{\mathtt{SecMult}} = \{\{\mathbf{a}[i], \mathbf{b}[j]\} \mid 1 \leq i, j \leq t+1\}$. We can verify that the gadget `SecMult` is both $(\mathbb{Y}_{\mathtt{SecMult}}, d)$-NI and d-NI. For instance, the probe set $\{t_{12}, t'_{12}\}$ can be simulated by the set $\{\{\mathbf{a}[1], \mathbf{b}[2]\}, \{\mathbf{a}[2], \mathbf{b}[1]\}\} \subset \mathbb{Y}_{\mathtt{SecMult}}$. However, it is *not* $(\mathbb{Y}, d)$-NI for any $\mathbb{Y} \subset \mathbb{Y}_{\mathtt{SecMult}}$.

The main challenge is how to compute a family of variable sets $\mathbb{Y}$ for a given gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$. Obviously, a gadget is $(\mathbb{Y}, d)$-NI if $\mathbb{Y}$ contains all of the probes sets. However, such a set $\mathbb{Y}$ does not make any sense. For instance, to verify d-probing security of the $(\mathbb{Y}, d)$-NI gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ where each set of $\mathbb{Y}$ contains one variable of $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$, we have to check if any set of d probes is independent of the secret. It would be the same as directly verifying d-probing security of the

```
SecMult(a, b){
    for(i = 1; i ≤ t + 1; i++) c[i] = a[i] ⊙ b[i];
    for(i = 1; i ≤ t + 1; i++){
        for(j = i + 1; j ≤ t + 1; j++){
            r_ij = $;      c[i] = c[i] ⊕ r_ij;      t_ij = a[i] ⊙ b[j];      r'_ij = r_ij ⊕ t_ij;
            t'_ij = a[j] ⊙ b[i];      r''_ij = r'_ij ⊕ t'_ij;      c[j] = c[j] ⊕ r''_ij; }
    }
    return c;} // (⊕ c) = (⊕ a) ⊙ (⊕ b)
```

Fig. 3. t-order masked implementation SecMult for finite-field multiplication [Coron 2014].

gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$. To address this challenge, we will propose an algorithm for computing a family $\mathbb{Y}$ of variable sets for each simple gadget so that $\mathbb{Y}$ contains the variable sets as less as possible (cf. Section 4.2).

3.2 $(\mathbb{Y}, d)$-Strong Non-Interference

[Barthe et al. 2016] provided a stronger version of d-NI, called d-Strong Non-Interference, in order to simplify the gadget composition by enforcing that external probes (i.e., output shares) give *no* information of the input shares. We first recall d-Strong Non-Interference and then generalize it to a stronger version of $(\mathbb{Y}, d)$-NI, called $(\mathbb{Y}, d)$-Strong Non-Interference, in order to simplify the gadget composition by enforcing that external probes give *no* information about any set in $\mathbb{Y}$.

Definition 7 (Strong non-interference). *A gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ with output sharings $\mathbf{b}_1, \cdots, \mathbf{b}_k$ is d-*Strong Non-Interfering *(d-SNI for short) if for any set O of internal probes and any sets $O_i \subseteq \mathbf{b}_i$ of external probes for $1 \le i \le k$ such that $|O| + \max_{i=1}^{k} |O_i| \le d$, there exists a set I of input shares with at most $|O|$ shares for each input sharing $\mathbf{a}_i$ such that $O \cup \bigcup_{i=1}^{k} O_i$ on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is I-simulatable.*

Intuitively, knowing the values of the variables in the set I that contains at most $|O|$ input shares of $\mathbf{a}_i$ for every $1 \le i \le m$ suffices to simulate the joint distribution of the probe set $O \cup \bigcup_{i=1}^{k} O_i$ to the adversary. Thus, d-SNI is a stronger security notion than d-NI. A d-NI gadget is d-SNI if the input or output sharings are refreshed by d-SNI refresh gadgets and are not used anywhere else in the gadget.

A composite gadget composed of any d-SNI gadgets is d-SNI as well, which is called "trivial compositionality" by [Cassiers et al. 2021], while a composite gadget composed of any d-NI but not d-SNI gadgets at best can be d-NI or d-probing secure, i.e., cannot be d-SNI and may not be d-NI or d-probing secure. However, to be d-SNI, d-SNI refresh gadgets may be introduced, which makes gadgets less efficient. Thus, we propose the notion of strong variable non-interference by generalizing d-SNI with a family of variable sets $\mathbb{Y}$.

Definition 8 (Strong variable non-interference). *For a family $\mathbb{Y}$ of variable sets of a gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ with the output sharings $\mathbf{b}_1, \cdots, \mathbf{b}_k$, the gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-*Strong Non-Interfering, *$(\mathbb{Y}, d)$-SNI for short, iff for any set O of internal probes and any sets $O_i \subseteq \mathbf{b}_i$ of external probes for $1 \le i \le k$ such that $|O| + \max_{i=1}^{k} |O_i| \le d$, there exists a set $\mathbb{I} \subseteq \mathbb{Y}$ such that $|\mathbb{I}| \le |O|$ and $O \cup \bigcup_{i=1}^{k} O_i$ on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $\mathbb{I}$-simulatable.*

Intuitively, any set comprising at most d_1 internal probes O and at most d_2 output shares O_i per output sharing $\mathbf{b}_i$ with $d_1 + d_2 \le d$ can be simulated to the adversary when knowing the values of variables in $\bigcup \mathbb{I}$ for some set $\mathbb{I} \subseteq \mathbb{Y}$ such that $|\mathbb{I}| \le d_1$.

Consider the gadgets CM, XOR and Refresh in Figure 2. The gadget Refresh is $(\mathbb{Y}_{\mathsf{RF}}, 2)$-SNI and 2-SNI. For instance, the set $\{m_1, \mathbf{c}[1]\}$ consisting of one internal probe m_1 and one external

probe $\mathbf{c}[1]$ in Refresh can be simulated without using any variables, as $\mathcal{E}(m_1) = \mathbf{a}[1] \oplus r_1$ and $\mathcal{E}(\mathbf{c}[1]) = (\mathbf{a}[1] \oplus r_1) \oplus r_2$ are respectively masked by two independent random variables r_1 and r_2. In contrast, CM and XOR are not $(\mathbb{Y}_{\mathtt{CM}}, 2)$-SNI and $(\mathbb{Y}_{\mathtt{XOR}}, 2)$-SNI, respectively. For instance, $\{\mathbf{c}[1], \mathbf{c}[2]\}$ consisting of two external probes in XOR can only be simulated by $\{\{\mathbf{a}[1], \mathbf{b}[1]\}, \{\mathbf{a}[2], \mathbf{b}[2]\}\}$ using input shares. Indeed, there does not exist any set $\mathbb{Y}$ such that XOR or CM is $(\mathbb{Y}, 2)$-SNI.

Consider the gadget SecMult in Figure 3 which is $(\mathbb{Y}_{\mathtt{SecMult}}, d)$-NI. We can show that SecMult is $(\mathbb{Y}_{\mathtt{SecMult}}, d)$-SNI and d-SNI. Assume $d = 3$. Any probe set $\{\mathbf{c}[i], \mathbf{c}[j], \mathbf{c}[k]\}$ in SecMult such that $1 \leq i < j < k \leq 4$ can be simulated without using any variables, as the computations $\mathcal{E}(\mathbf{c}[i])$, $\mathcal{E}(\mathbf{c}[j])$ and $\mathcal{E}(\mathbf{c}[k])$ are respectively masked by three distinct random variables. The probe set $\{\mathbf{c}[1], t_{12}, t'_{12}\}$ can be simulated by the set $\{\{\mathbf{a}[1], \mathbf{b}[2]\}, \{\mathbf{a}[2], \mathbf{b}[1]\}\} \subseteq \mathbb{Y}_{\mathtt{SecMult}}$, as the computation $\mathcal{E}(\mathbf{c}[1])$ is masked by random variables r_{1j} for $1 \leq j \leq 3$.

It is straightforward to observe the following propositions.

Proposition 1. *$(\mathbb{Y}, d)$-SNI entails $(\mathbb{Y}, d)$-NI, but the converse does not hold.* □

Proposition 2. *Consider two families of variable sets $\mathbb{Y}_1$ and $\mathbb{Y}_2$. If $\mathbb{Y}_2 \sqsubseteq \mathbb{Y}_1$, then $(\mathbb{Y}_2, d)$-NI entails $(\mathbb{Y}_1, d)$-NI, and $(\mathbb{Y}_2, d)$-SNI entails $(\mathbb{Y}_1, d)$-SNI.* □

By Proposition 2, $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-NI (resp. $(\mathbb{Y}, d)$-SNI) iff it is $(\mathbb{Y}', d)$-NI (resp.$(\mathbb{Y}', d)$-SNI) for $\mathbb{Y}' = \{Y \in \mathbb{Y} \mid \nexists Y' \in \mathbb{Y}.\ Y \subset Y'\}$. Thus Proposition 2 allows to reduce the size $|\mathbb{Y}|$. Consider $\mathbb{Y}_1 = \{\{a_1, b_1\}, \{a_2, b_2\}\}$ and $\mathbb{Y}_2 = \{\{a_1\}, \{a_2\}, \{b_1\}, \{b_2\}, \{a_1, b_1\}, \{a_2, b_2\}\}$, we prefer to use $\mathbb{Y}_1$, as any probes that are simulated by $\{a_1\}$ and/or $\{b_1\}$ (resp. $\{a_2\}$ and/or $\{b_2\}$) can also be simulated by the set $\{a_1, b_1\}$ (resp. $\{a_2, b_2\}$).

The following proposition allows us to consider less number of probe sets for proving $(\mathbb{Y}, d)$-NI and $(\mathbb{Y}, d)$-SNI.

Proposition 3. *For a family $\mathbb{Y}$ of variable sets of a gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ with output sharings $\mathbf{b}_1, \cdots, \mathbf{b}_k$,*

(1) $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-NI iff for any set O of d probes, there exists a subset $\mathbb{I} \subseteq \mathbb{Y}$ such that $|\mathbb{I}| \leq d$ and O on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $\mathbb{I}$-simulatable.

(2) $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-SNI iff for any set O of at most d internal probes and any sets $O_i \subseteq \mathbf{b}_i$ of $d - |O|$ external probes for $1 \leq i \leq k$, there exists a set $\mathbb{I} \subseteq \mathbb{Y}$ such that $|\mathbb{I}| \leq |O|$ and $O \cup \bigcup_{i=1}^{k} O_i$ on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $\mathbb{I}$-simulatable.

Proof sketch. The direction ($\Rightarrow$) is straightforward. We can prove the direction ($\Leftarrow$) by induction on the size s, where $s = |O|$ for Item (1) and $s = |O| + \max_{i=1}^{k} |O_i|$ for Item (2). The base case $s = d$ is trivial. For the inductive step $0 < s < d$, we can prove the results by contradiction.

Suppose O on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is not $\mathbb{I}$-simulatable for any subset $\mathbb{I} \subseteq \mathbb{Y}$ with $|\mathbb{I}| \leq s$. Then, we can add a new input share into O, resulting in a new set O' such that $|O'| = |O| + 1 = s + 1$. It can be proved that O' on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is not $\mathbb{I}'$-simulatable for any subset $\mathbb{I}' \subseteq \mathbb{Y}$ with $|\mathbb{I}'| \leq s + 1$, otherwise O on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $\mathbb{I}$-simulatable for some subset $\mathbb{I} \subseteq \mathbb{Y}$ with $|\mathbb{I}| \leq s$. It contradicts the induction hypothesis on O'. Thus, Item (1) holds. Item (2) can be proved in a similar way. Full proof is rather involved and thus is given [Gao et al. 2023]. □

We remark that $(\mathbb{Y}, d)$-NI and $(\mathbb{Y}, d)$-SNI are not directly defined like in Proposition 3, in order to follow the same style (i.e., the size of probe sets) with the definitions of d-probing security, d-NI and d-SNI. For instance, Proposition 3(1) states that it suffices to consider the sets O comprising exactly d probes, instead of all the sets of at most d probes in Definition 6.

3.3 Relating to d-Probing Security and d-NI/d-SNI

We relate $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI to d-probing security and d-NI/d-SNI. These relations reveal that our new security notions could be used as an intermediate step for proving d-probing security, d-NI and d-SNI.

According to the definitions of $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI, it is straightforward to see the following proposition which relates $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI to d-probing security and provides a sound approach to prove d-probing security.

Proposition 4. *For any family $\mathbb{Y}$ of variable sets of a gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$, if $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-NI and the evaluation $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)_{\bigcup \mathbb{I}}$ is independent of the secret for every set $\mathbb{I} \subseteq \mathbb{Y}$ such that $|\mathbb{I}| = d$, then $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ is d-probing secure.* □

$(\mathbb{Y}, d)$-NI and $(\mathbb{Y}, d)$-SNI are very flexible security notions as the parameter $\mathbb{Y}$ can vary. The relation between $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI and d-NI/d-SNI is characterized by the following proposition, which provides a new way to prove d-NI/d-SNI via our new security notions.

Proposition 5. *Fix a gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ with output sharings $\mathbf{b}_1, \cdots, \mathbf{b}_k$. Let $\mathbb{Y}$ be a family of variable sets. If $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-NI (resp. $(\mathbb{Y}, d)$-SNI) and for any set $\mathbb{I}$ of d variable sets of $\mathbb{Y}$, there exists a set I with at most $|\mathbb{I}|$ input shares of each input sharing $\mathbf{a}_i$ such that $\bigcup \mathbb{I}$ on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is I-simulatable, then $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is d-NI (resp. d-SNI).*

Proof. Suppose $f(\mathbf{a}_1, ..., \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-NI (resp. $(\mathbb{Y}, d)$-SNI) and for any set $\mathbb{I} \subseteq \mathbb{Y}$ such that $|\mathbb{I}| = d$, there exists a set I containing at most $|\mathbb{I}|$ shares for each input sharing $\mathbf{a}_i$ such that $\bigcup \mathbb{I}$ on $f(\mathbf{a}_1, ..., \mathbf{a}_m)$ is I-simulatable. Then, for any set $\mathbb{I} \subseteq \mathbb{Y}$ such that $|\mathbb{I}| \leq d$, there exists a set I containing at most $|\mathbb{I}|$ shares for each input sharing $\mathbf{a}_i$ such that $\bigcup \mathbb{I}$ on $f(\mathbf{a}_1, ..., \mathbf{a}_m)$ is I-simulatable (this can be proved following the proof of Proposition 3).

Let us consider a set consisting of at most d internal probes O' and external probes O_i of the output sharing $\mathbf{b}_i$ for $1 \leq i \leq k$ such that $|O'| + \sum_{i=1}^{k} |O|_i \leq d$ (resp. $|O'| + \max_{i=1}^{k} |O|_i \leq d$). $f(\mathbf{a}_1, ..., \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-NI (resp. $(\mathbb{Y}, d)$-SNI), thus, there exists a set $\mathbb{I} \subseteq \mathbb{Y}$ such that $O' \cup \bigcup_{i=1}^{k} O_i$ on $f(\mathbf{a}_1, ..., \mathbf{a}_m)$ is $\mathbb{I}$-simulatable and $|\mathbb{I}| \leq |O'| + \sum_{i=1}^{k} |O|_i$ (resp. $|\mathbb{I}| \leq |O'|$).

Since $\mathbb{I} \subseteq \mathbb{Y}$ and $|\mathbb{I}| \leq d$, there exists a set I containing at most $|\mathbb{I}|$ shares for each input sharing $\mathbf{a}_i$ such that $\bigcup \mathbb{I}$ on $f(\mathbf{a}_1, ..., \mathbf{a}_m)$ is I-simulatable. Recall that $O' \cup \bigcup_{i=1}^{k} O_i$ on $f(\mathbf{a}_1, ..., \mathbf{a}_m)$ is $\mathbb{I}$-simulatable. Thus, $O' \cup \bigcup_{i=1}^{k} O_i$ on $f(\mathbf{a}_1, ..., \mathbf{a}_m)$ is I-simulatable. The result follows from the fact that $|I| \leq |\mathbb{I}|$ and $|\mathbb{I}| \leq |O'| + \sum_{i=1}^{k} |O|_i$ (resp. $|\mathbb{I}| \leq |O'|$). □

Consider the simple gadgets `CM`, `XOR` and `Refresh` in Figure 2 are $(\mathbb{Y}_{\mathtt{CM}}, 2)$-NI, $(\mathbb{Y}_{\mathtt{XOR}}, 2)$-NI and $(\mathbb{Y}_{\mathtt{RF}}, 2)$-SNI, respectively. By applying Proposition 5 and checking the sets $\mathbb{Y}_{\mathtt{CM}}$, $\mathbb{Y}_{\mathtt{XOR}}$ and $\mathbb{Y}_{\mathtt{RF}}$, we can deduce that the gadgets `CM` and `XOR` are 2-NI and the gadget `Refresh` is 2-SNI. Furthermore, when the input sharings are mutually independent, we can deduce that the gadgets `CM`, `XOR` and `Refresh` are 2-probing secure.

3.4 Compositionality of $(\mathbb{Y}, d)$-NI and $(\mathbb{Y}, d)$-SNI Gadgets

In this subsection, we study the compositionality of $(\mathbb{Y}, d)$-NI and $(\mathbb{Y}, d)$-SNI gadgets which will be leveraged to compute a set $\mathbb{Y}$ for each composite gadget.

We start by defining some notations. Given a gadget call $\mathbf{y} =_\ell f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ to the gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$, for every set $\mathbb{Y}$, let $\mathbb{Y}[\mathbf{x}_1/\mathbf{a}_1, \cdots, \mathbf{x}_m/\mathbf{a}_m]@\ell$ be the set $\mathbb{Y}$ where

- $\mathbf{a}_i[j]$ for each $1 \leq i \leq m$, $1 \leq j \leq t+1$ is replaced by $\mathbf{x}_i[j]$,
- and each local variable x is replaced by $x@\ell$.

Intuitively, $\mathbb{Y}[\mathbf{x}_1/\mathbf{a}_1, \cdots, \mathbf{x}_m/\mathbf{a}_m]@\ell$ lifts the set $\mathbb{Y}$ from the gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ to the gadget call $\mathbf{y} = f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$, where formal arguments $\mathbf{a}_1, \cdots, \mathbf{a}_m$ are replaced by their corresponding inputs $\mathbf{x}_1, \cdots, \mathbf{x}_m$ and local variables x are appended with the label ℓ to avoid name conflict.

We present the composition rules using the gadget,

$$\begin{aligned} g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)\{ \quad & \mathbf{x}_{m+1}, \cdots, \mathbf{x}_{m+k} = \ell_f f(\mathbf{x}_1, \cdots, \mathbf{x}_m); \\ & \mathbf{z}_1, \cdots, \mathbf{z}_h = \ell_g g(\mathbf{x}_1, \cdots, \mathbf{x}_{m+k}); \\ & \text{return } \mathbf{z}_1, \cdots, \mathbf{z}_h; \quad \} \end{aligned}$$

which invokes the gadgets $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ and $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$. For any set $\mathbb{Y}_f$ (resp. $\mathbb{Y}_g$) of variable sets of $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ (resp. $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$), let $\phi_{g\circ f}(\mathbb{Y}_f, \mathbb{Y}_g)$ be the set

$$\mathbb{Y}'_f \cup (\mathbb{Y}'_g \setminus \{\{\mathbf{x}_j[i]\} \mid 1 \le i \le t+1, m+1 \le j \le m+k\},$$

where $\mathbb{Y}'_f$ denotes $\mathbb{Y}_f[\mathbf{x}_1/\mathbf{a}_1, \cdots, \mathbf{x}_m/\mathbf{a}_m]@\ell_f$ and $\mathbb{Y}'_g$ denotes $\mathbb{Y}_g[\mathbf{x}_1/\mathbf{b}_1, \cdots, \mathbf{x}_{m+k}/\mathbf{b}_{m+k}]@\ell_g$. Intuitively, $\phi_{g\circ f}(\mathbb{Y}_f, \mathbb{Y}_g)$ is the union of $\mathbb{Y}_f$ and $\mathbb{Y}_g$ where formal arguments of $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ and $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$ are substituted by their corresponding inputs and local variables are appended with the corresponding labels ℓ_f and ℓ_g to avoid name conflict. The substitution ensures that $\phi_{g\circ f}(\mathbb{Y}_f, \mathbb{Y}_g)$ only contains variables of $(g \circ f)_{\mathtt{in}}(\mathbf{x}_1, \cdots, \mathbf{x}_m)$. Furthermore, the sets $\{\mathbf{x}_j[i]\}$ for $1 \le i \le t+1, m+1 \le j \le m+k$ are removed from $\mathbb{Y}'_g$, as they are the same as the external probes of $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$.

Composition rule R1. We present the first rule for composing a $(\mathbb{Y}, d)$-NI gadget with another $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI gadget.

Lemma 1. *If $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}_f, d)$-NI and for any nonempty probe set O on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ and nonempty set $\mathbb{O} \subseteq \mathbb{Y}_g$ such that $|O| + |\mathbb{O}| = d$, the probe set $O[\mathbf{x}_1/\mathbf{a}_1, \cdots, \mathbf{x}_m/\mathbf{a}_m]@\ell_f \cup \bigcup \mathbb{O}[\mathbf{x}_1/\mathbf{b}_1, \cdots, \mathbf{x}_{m+k}/\mathbf{b}_{m+k}]@\ell_g$ on $g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ is $\mathbb{I}$-simulatable for some set $\mathbb{I} \subseteq \phi_{g\circ f}(\mathbb{Y}_f, \mathbb{Y}_g)$ with $|\mathbb{I}| \le d$, then the following statements hold:*

(1) If $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$ is $(\mathbb{Y}_g, d)$-NI, then $g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ is $(\phi_{g\circ f}(\mathbb{Y}_f, \mathbb{Y}_g), d)$-NI.
(2) If $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$ is $(\mathbb{Y}_g, d)$-SNI, then $g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ is $(\phi_{g\circ f}(\mathbb{Y}_f, \mathbb{Y}_g), d)$-SNI.

Proof sketch. For any set consisting of at most d internal probes O and $d - |O|$ external probes O_i for each output sharing $\mathbf{z}_i$ on $g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$:

- If $O \cup \bigcup_{i=1}^{h} O_i$ contains only probes from $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$, then since $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}_f, d)$-NI, we get that $O \cup \bigcup_{i=1}^{h} O_i$ can be simulated with at most d variable sets from $\mathbb{Y}_f[\mathbf{x}_1/\mathbf{a}_1, \cdots, \mathbf{x}_m/\mathbf{a}_m]@\ell_f$, hence can be simulated with at most d variable sets from $\phi_{g\circ f}(\mathbb{Y}_f, \mathbb{Y}_g)$.
- If $O \cup \bigcup_{i=1}^{h} O_i$ contains only probes from $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$, then since $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$ is $(\mathbb{Y}_g, d)$-NI (resp. $(\mathbb{Y}_g, d)$-SNI), we get that $O \cup \bigcup_{i=1}^{h} O_i$ can be simulated by at most d (resp. $|O|$) variable sets from $\mathbb{Y}_g[\mathbf{x}_1/\mathbf{b}_1, \cdots, \mathbf{x}_{m+k}/\mathbf{b}_{m+k}]@\ell_g$, hence can be simulated by at most d (resp. $|O|$) variable sets from $\phi_{g\circ f}(\mathbb{Y}_f, \mathbb{Y}_g)$.
- If $O \cup \bigcup_{i=1}^{h} O_i$ contains probes from both $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ and $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$, we can conclude the proof following from Proposition 3 and the fact that for any nonempty probe set O'' on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ and any nonempty set $\mathbb{O} \subseteq \mathbb{Y}_g$ with $|O''| + |\mathbb{O}| = d$, $O''[\mathbf{x}_1/\mathbf{a}_1, \cdots, \mathbf{x}_m/\mathbf{a}_m]@\ell_f \cup \bigcup \mathbb{O}[\mathbf{x}_1/\mathbf{b}_1, \cdots, \mathbf{x}_{m+k}/\mathbf{b}_{m+k}]@\ell_g$ on the gadget $g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ is $\mathbb{I}$-simulatable for some set $\mathbb{I} \subseteq \phi_{g\circ f}(\mathbb{Y}_f, \mathbb{Y}_g)$ such that $|\mathbb{I}| \le d$. Indeed, since $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$ is $(\mathbb{Y}_g, d)$-NI (resp. $(\mathbb{Y}_g, d)$-SNI), the variable subset O' of $O \cup \bigcup_{i=1}^{h} O_i$ that come from $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$ can be first simulated by some set $\mathbb{O}[\mathbf{x}_1/\mathbf{b}_1, \cdots, \mathbf{x}_{m+k}/\mathbf{b}_{m+k}]@\ell_g \subseteq \mathbb{Y}_g[\mathbf{x}_1/\mathbf{b}_1, \cdots, \mathbf{x}_{m+k}/\mathbf{b}_{m+k}]@\ell_g$ with the desired size, together with the remaining variables of $O \cup \bigcup_{i=1}^{h} O_i$ (i.e., $O''[\mathbf{x}_1/\mathbf{a}_1, \cdots, \mathbf{x}_m/\mathbf{a}_m]@\ell_f$), can be simulated by at most d (resp. $|O|$) variable sets from $\mathbb{I} \subseteq \phi_{g\circ f}(\mathbb{Y}_f, \mathbb{Y}_g)$.

This concludes the proof. □

Lemma 1 provides a sound approach for computing a set $\mathbb{Y}$ from the sets $\mathbb{Y}_f$ and $\mathbb{Y}_g$ and checking if the composite gadget $g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ is $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI without inlining the called gadgets. This reduces the number of probe sets to be checked when computing the set $\mathbb{Y}$ and verifying $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI for the composite gadget $g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$.

Composition rule R2. One may notice that in the first composition rule R1, probe sets across the gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ and the set $\mathbb{Y}_g$ should be checked. We present the second composition rule to further reduce the number of probe sets to be checked, and hence improve the efficiency. This composition rule requires the preceding gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ to be $(\mathbb{Y}_f, d)$-SNI instead of its weak counterpart $(\mathbb{Y}_f, d)$-NI.

Lemma 2. *If $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}_f, d)$-SNI and for any nonemptyset set $\mathbb{O} \subseteq \mathbb{Y}_g \setminus \{\{\mathbf{b}_j[i]\} \mid m+1 \leq j \leq m+k, 1 \leq i \leq t+1\}$ such that $|\mathbb{O}| \leq d$, the set $\bigcup \mathbb{O}$ on the gadget $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$ is $\mathbb{I}$-simulatable for some $\mathbb{I}$ that only contains at most $|\mathbb{O}|$ input shares of each input sharing $\mathbf{b}_j$ for $m+1 \leq j \leq m+k$, then the following statements hold:*

(1) If $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$ is $(\mathbb{Y}_g, d)$-NI, then $g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ is $(\phi_{g \circ f}(\mathbb{Y}_f, \mathbb{Y}_g), d)$-NI.
(2) If $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$ is $(\mathbb{Y}_g, d)$-SNI, then $g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ is $(\phi_{g \circ f}(\mathbb{Y}_f, \mathbb{Y}_g), d)$-SNI.

Proof. The proof of Lemma 2 follows that of Lemma 1 except that we leverage $(\mathbb{Y}_f, d)$-SNI of $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$, where the external shares of $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ when passed to $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$ as inputs, could be used in the gadget $g(\mathbf{b}_1, \cdots, \mathbf{b}_{m+k})$ without requiring any sets of $\phi_{g \circ f}(\mathbb{Y}_f, \mathbb{Y}_g)$ for simulation. □

Lemma 2 provides an efficient approach for computing a set $\mathbb{Y}$ and checking if the composite gadget $g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$ is $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI, without inlining the called gadgets. Recall that the rule R1 (cf. Lemma 1) requires to check $O[\mathbf{x}_1/\mathbf{a}_1, \cdots, \mathbf{x}_m/\mathbf{a}_m]@\ell_f \cup \bigcup \mathbb{O}[\mathbf{x}_1/\mathbf{b}_1, \cdots, \mathbf{x}_{m+k}/\mathbf{b}_{m+k}]@\ell_g$ for any nonempty set probe O on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ and nonempty set $\mathbb{O} \subseteq \mathbb{Y}_g$ such that $|O| + |\mathbb{O}| = d$. By Lemma 2, we only need to check the sets from $\mathbb{O} \subseteq \mathbb{Y}_g \setminus \{\{\mathbf{b}_j[i]\} \mid m+1 \leq j \leq m+k, 1 \leq i \leq t+1\}$. Therefore, this further reduces the number of sets to be checked when computing the set $\mathbb{Y}$ and checking the composite gadget $g \circ f(\mathbf{x}_1, \cdots, \mathbf{x}_m)$.

In general, a composite gadget may contain more than two gadget calls. For such composite gadgets, we will present a bottom-up algorithm in Section 4 which iteratively composes the head and tail of the sequence of gadget calls by leveraging Lemma 1 and Lemma 2, where the head is the first gadget call, and the tail is the remaining sequence of gadget calls.

As a warm-up, we exemplify our overall approach using the illustrating example shown in Figure 2.

Example 1. *Recall that the simple gadgets* CM, XOR *and* Refresh *are $(\mathbb{Y}_{\mathsf{CM}}, d)$-NI, $(\mathbb{Y}_{\mathsf{XOR}}, d)$-NI and $(\mathbb{Y}_{\mathsf{RF}}, d)$-SNI, respectively, where $\mathbb{Y}_{\mathsf{CM}} = \{\{\mathbf{a}[i], \mathbf{b}[j]\} \mid 1 \leq i, j \leq 3\}$, $\mathbb{Y}_{\mathsf{XOR}} = \{\{\mathbf{a}[i], \mathbf{b}[i]\} \mid 1 \leq i \leq 3\}$, and $\mathbb{Y}_{\mathsf{RF}} = \{\{\mathbf{a}[i]\} \mid 1 \leq i \leq 3\}$. We build the proof from the bottom gadget call to the top one.*

(1) For the gadget call $\mathbf{d} = \mathsf{CM}(\mathbf{a}, \mathbf{c})$: we compute the set

$$\mathbb{Y}'_{\mathsf{CM}} = \mathbb{Y}_{\mathsf{CM}}[\mathbf{a}/\mathbf{a}, \mathbf{c}/\mathbf{b}]@3 = \{\{\mathbf{a}[i], \mathbf{c}[j]\} \mid 1 \leq i, j \leq 3\},$$

i.e., passing the actual parameters $\mathbf{a}$ and $\mathbf{c}$ to the formal arguments $\mathbf{a}$ and $\mathbf{b}$ and appending @3 to the local variables of CM.

(2) For the gadget calls $\mathbf{c} = \mathsf{XOR}(\mathbf{a}, \mathbf{e})$; $\mathbf{d} = \mathsf{CM}(\mathbf{a}, \mathbf{c})$: similar to the first step, we first compute

$$\mathbb{Y}'_{\mathsf{XOR}} = \mathbb{Y}_{\mathsf{XOR}}[\mathbf{a}/\mathbf{a}, \mathbf{e}/\mathbf{b}]@2 = \{\{\mathbf{a}[i], \mathbf{e}[i]\} \mid 1 \leq i \leq 3\}.$$

By Lemma 1, we compute the set

$$\begin{aligned} \mathbb{Y}_{\mathsf{XORM2}} &= Y y'_{\mathsf{XOR}} \cup (\mathbb{Y}'_{\mathsf{CM}} \setminus \{\{\mathbf{c}[i]\} \mid 1 \leq i \leq 3\}) \\ &= \{\{\mathbf{a}[i], \mathbf{e}[i]\}, \{\mathbf{a}[i], \mathbf{c}[j]\} \mid 1 \leq i, j \leq 3\} \end{aligned}$$

and check if for any nonempty probe set O on XOR *and nonempty set* $\mathbb{O} \subseteq \mathbb{Y}_{\mathtt{CM}}$ *such that* $|O| + |\mathbb{O}| = d$*, the probe set* $O[\mathbf{a}/\mathbf{a}, \mathbf{e}/\mathbf{b}]@2 \cup \mathbb{O}[\mathbf{a}/\mathbf{a}, \mathbf{c}/\mathbf{b}]@3$ *can be simulated by two variable sets of* $\mathbb{Y}_{\mathtt{XORM2}}$*. It is indeed the case.*

(3) For the gadget calls $\mathbf{e} = \mathtt{Refresh}(\mathbf{b});\ \mathbf{c} = \mathtt{XOR}(\mathbf{a}, \mathbf{e});\ \mathbf{d} = \mathtt{CM}(\mathbf{a}, \mathbf{c})$*: we first compute the set*

$$Yy'_{\mathsf{RF}} = \mathbb{Y}_{\mathsf{RF}}[\mathbf{b}/\mathbf{a}]@1 = \{\{\mathbf{b}[i]\} \mid 1 \le i \le 3\}.$$

Next, by Lemma 2, we compute

$$\begin{aligned} \mathbb{Y}_{\mathtt{XORMULTI}} &= \mathbb{Y}'_{\mathsf{RF}} \cup (\mathbb{Y}_{\mathtt{XORM2}} \setminus \{\{\mathbf{e}[i]\} \mid 1 \le i \le 3\}) \\ &= \{\{\mathbf{b}[i]\}, \{\mathbf{a}[i], \mathbf{e}[i]\}, \{\mathbf{a}[i], \mathbf{c}[j]\} \mid 1 \le i, j \le 3\} \end{aligned}$$

and check if for any $\mathbb{O} \subseteq \mathbb{Y}_{\mathtt{XORM2}} \setminus \{\{\mathbf{e}[i]\} \mid i = 1, 2, 3\}$ *such that* $|\mathbb{O}| \le d$*,* $\bigcup \mathbb{O}$ *can be simulated by a set* I *comprising shares of* $\mathbf{a}$ *and at most* $|\mathbb{O}|$ *shares of* $\mathbf{e}$*. It is indeed the case.*

Finally, we get that XORMULTI *is* $(\mathbb{Y}_{\mathtt{XORMULTI}}, 2)$-NI.

By checking that any two variable sets of $\mathbb{Y}_{\mathtt{XORMULTI}}$ *can be simulated by a set* I *that only contains at most two shares of each input sharing, we can prove that* XORMULTI *is* 2-NI*. For instance,* $\{\{\mathbf{b}[i]\}, \{\mathbf{a}[3], \mathbf{e}[3]\}\}$ *for* $1 \le i \le 3$ *can be simulated by the set* $\{\mathbf{b}[i], \mathbf{a}[3]\}$ *as* $\mathcal{E}(\mathbf{e}[3]) = \mathbf{b}[3] \oplus r_2@1 \oplus r_3@1$ *is masked by the random variables* $r_2@1$ *and* $r_3@1$*.*

4 ALGORITHMIC VERIFICATION

We first present a proof system for proving simulatability. We then propose algorithms for computing $\mathbb{Y}$ for each gadget so that the gadget is $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI. Finally, we present algorithms for verifying d-NI/d-SNI and d-probing security through the new notions $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI.

4.1 A Proof System for Simulatability

Inspired by the notion of dominant random variable [Gao et al. 2019a], we introduce the notion of perfect masking for computations based on which we devise a proof system for proving simulatability.

A computation e is *perfectly masked* by a random variable r if r (syntactically) occurs in e exactly once, and each operator $\circ$ along the path from r to the root in the abstract syntax tree of e must satisfy one of the following conditions:

- $\circ \in \{\oplus, +, -, \neg\}$;
- $\circ$ is $\odot$ and one of its children is a non-zero constant.

For instance, $k \oplus r$ is perfectly masked by r while $(k \oplus r) \wedge r$ is *not* perfectly masked by r.

Fix a gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ and consider a probe x on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$. If $\mathcal{E}(x)$ is perfectly masked by a random variable r, then the evaluation $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)_x$ is uniform for any inputs $\mathbf{x}_1, \cdots, \mathbf{x}_m$. Remark that the above two conditions are designated to obtain the uniform evaluation $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)_x$. In general, $\circ$ can be any univariate bijective function.

For a given computation e, let $\mathsf{sub}(e)$ denote the set of all the non-trivial sub-computations in e (i.e., including e but excluding variables). We lift $\mathsf{sub}(\cdot)$ to a set E of computations, namely, $\mathsf{sub}(E) = \bigcup_{e \in E} \mathsf{sub}(e)$. For each computation $e \in \mathsf{sub}(E)$ and random variable r, we denote by $E[r/e]$ the set E of computations where all occurrences of e are substituted by r. This substitution is called (e, r)-*simplification* of E.

Given a variable set O, we denote by $\mathcal{E}(O)$ the set of computations $\{\mathcal{E}(x) \mid x \in O\}$. For two variable sets O_1 and O_2, we write $O_2 \xrightarrow{(e,r)} O_1$, if the following three conditions hold: (i) $e \in \mathsf{sub}(\mathcal{E}(O_2))$ is perfectly masked by the random variable r; (ii) r only occurs in e and does not occur anywhere else in $\mathcal{E}(O_2)$; and (iii) $\mathcal{E}(O_2)[r/e] = \mathcal{E}(O_1)$. Intuitively, $O_2 \xrightarrow{(e,r)} O_1$ if the computations $\mathcal{E}(O_2)$ are equivalent to the computations $\mathcal{E}(O_1)$ after applying an (e, r)-simplification on $\mathcal{E}(O_2)$.

Since the random variable r only occurs in e and does not occur anywhere else in $\mathcal{E}(O_2)$, e can be seen as a random variable in the computations $\mathcal{E}(O_2)$. This implies that for any inputs $\mathbf{x}_1, \cdots, \mathbf{x}_m$, the evaluation $f(\mathbf{x}_1, \cdots, \mathbf{x}_m)_{O_2}$ remains the same when e is replaced by r.

Proposition 6. *If $O_2 \xrightarrow{(e,r)} O_1$, then for any variable set I, O_1 is I-simulatable iff O_2 is I-simulatable.*

Proof. Suppose $O_2 \xrightarrow{(e,r)} O_1$. Then, the computation e is perfectly masked by the random variable r. Since r only occurs in e and does not occur anywhere else in $\mathcal{E}(O_2)$. Thus, we can deduce that for any fixed inputs $\mathbf{x}_1, \cdots, \mathbf{x}_m$, the evaluation of $\mathcal{E}(O_2)$ will not change after replacing all the occurrences of e in $\mathcal{E}(O_2)$ by r, which implies that O_2 is I-simulatable on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ iff I suffices to simulate O_2 after replacing all the occurrences of e in $\mathcal{E}(O_2)$ by r. □

The judgement of our proof system is of the form $\vdash I \rightsquigarrow O$, where I and O are two sets of variables in $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ such that I does not contain any random variable. A judgement $\vdash I \rightsquigarrow O$ is valid iff the probe set O is I-simulatable.

$$\frac{O = O_1 \uplus O_2 \qquad O_2 \subseteq I \qquad \bigcup_{x \in O_1} \mathsf{Var}(\mathcal{E}(x)) \setminus \{\text{random variables}\} \subseteq I}{\vdash I \rightsquigarrow O}\ (\textsc{Comp}) \qquad \frac{O_2 \xrightarrow{(e,r)} O_1 \qquad \vdash I \rightsquigarrow O_1}{\vdash I \rightsquigarrow O_2}\ (\textsc{Dom})$$

Fig. 4. Proof rules.

The proof rules for deriving valid judgments are given in Figure 4. Rule Comp states that the probe set O can be partitioned into two subsets O_1 and O_2 such that O_2 is a subset of I and each variable $y \in \mathsf{Var}(\mathcal{E}(x))$ involved in the computation $\mathcal{E}(x)$ for $x \in O_1$ is either a random variable or a member of I. Rule Dom leverages Proposition 6 via (e, r)-simplification. Intuitively, rule Comp uses only the syntactic information of computations, while rule Dom is semantic where random variables and operations are exploited. It is easy to see that the proof system is sound.

Theorem 1. *If $\vdash I \rightsquigarrow O$ is valid, then the probe set O on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is I-simulatable.*

Proof. Suppose the judgement $\vdash I \rightsquigarrow O$ is valid, we prove that the probe set O on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is I-simulatable by induction on the number m of derivation steps of the judgement $\vdash I \rightsquigarrow O$.

- **Base case** $m = 1$. The judgement $\vdash I \rightsquigarrow O$ must be derived by applying the proof rule Comp. Suppose $O = O_1 \uplus O_2$ such that $\bigcup_{x \in O_1} \mathsf{Var}(\mathcal{E}(x)) \setminus \{\text{random variables}\} \subseteq I$ and $O_2 \subseteq I$. From $\bigcup_{x \in O_1} \mathsf{Var}(\mathcal{E}(x)) \setminus \{\text{random variables}\} \subseteq I$, we get that O_1 is I-simulatable. From $O_2 \subseteq I$, we get that O_2 is I-simulatable. Thus, $O_1 \uplus O_2$ is I-simulatable.
- **Inductive step** $m > 1$. The last derivation step of the judgement $\vdash I \rightsquigarrow O$ must be the proof rule Dom. Suppose the premises of this step are $O \xrightarrow{(e,r)} O_1$ and $\vdash I \rightsquigarrow O_1$. Then the judgement $\vdash I \rightsquigarrow O_1$ can be derived in $m - 1$ steps. By the induction hypothesis on the judgement $\vdash I \rightsquigarrow O_1$, we get that O_1 is I-simulatable. By Proposition 6, we get that O is I-simulatable.

This concludes the proof. □

4.2 Computing the Sets $\mathbb{Y}$ for Simple Gadgets

Given a security order d, a security type $\tau \in \{\mathsf{NI}, \mathsf{SNI}\}$ and a simple gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$, procedure SGadget in Algorithm 1 computes a set $\mathbb{Y}$ ensuring $(\mathbb{Y}, d)$-τ of $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$. SGadget first computes a candidate $\mathbb{Y}$ at Line 3 each of which is required to simulate a probe x on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ whose computation $\mathcal{E}(x)$ only depends on input shares. The set $\mathbb{Y}$ is reduced at Line 4 according to Proposition 2. We then verify if the gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI.

Algorithm 1 Computing $\mathbb{Y}$ for a simple gadget

1: **Proc** SGadget($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d, \tau$)
2: Let $(\mathbf{b}_1, \cdots, \mathbf{b}_k)$, $\mathcal{P}$ and $\mathcal{P}_{\mathtt{int}}$ be output sharings, set of all the probes and set of all internal probes of $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$, respectively
3: $\mathbb{Y} = \{\mathsf{Var}(\mathcal{E}(x)) \subseteq \bigcup_{i=1}^{m} \mathbf{a}_i \mid x \text{ is a probe on } f(\mathbf{a}_1, \cdots, \mathbf{a}_m)\}$
4: $\mathbb{Y} = \{Y \in \mathbb{Y} \mid \nexists Y' \in \mathbb{Y}. Y \subset Y'\}$
5: **if** τ == NI **then** ▷ Check $(\mathbb{Y}, d)$-NI
6: **return** (Explore^d($\{(d, \mathcal{P})\}, \mathbb{Y}, \tau$), τ)
7: **for** $(i = 0; i \leq d; i++)$ **do**
8: $\mathbb{Y}$=Explore^i($\{(i, \mathcal{P}_{\mathtt{int}}), (d-i, \mathbf{b}_1), \cdots, (d-i, \mathbf{b}_k)\}, \mathbb{Y}, \tau$)
9: **return** $(\mathbb{Y}, \tau)$
10: **Proc** Explore^d($\{(d_j, X_j)\}_j, \mathbb{Y}, \tau$)
11: **if** $\exists j, d_j > |X_j|$ **then return** $\mathbb{Y}$
12: $\{O_j\}_j$ =Choose($\{(d_j, X_j)\}_j$)
13: $(res, \mathbb{Y}')$ = Check^d($\bigcup_j O_j, \mathbb{Y}$)
14: **if** res == $\top$ **then** ▷ $\bigcup_j O_j$ is $\mathbb{Y}'$-simulatable if res is $\top$
15: $\{O_j\}_j$ = Extend^d($\{(O_j, X_j \backslash O_j)\}_j, \mathbb{Y}'$) ▷ Extend the sets $(O_j)_j$
16: $\mathbb{Y}' = \mathbb{Y}$ ▷ Assign the assumption $\mathbb{Y}$ to $\mathbb{Y}'$
17: **else if** τ == SNI **then** ▷ Fail to prove that $\bigcup_j O_j$ is $\mathbb{Y}'$-simulatable
18: Abort and emit the set $\bigcup_j O_j$ ▷ $\bigcup_j O_j$ is a potential leak
19: **for** $j; 0 \leq i_j \leq d_j$ s.t. $\sum_j i_j \neq 0$ **do** ▷ Explore other possible probe sets
20: $\mathbb{Y}$ =Explore^d($\{(d_j - i_j, O_j), (i_j, X_j \backslash O_j)\}_j, \mathbb{Y}', \tau$)
21: **return** $\mathbb{Y}$
22: **Proc** Check^d($O, \mathbb{Y}$) ▷ Check if O is $\mathbb{I}$ simulatable for $\mathbb{I} \subseteq \mathbb{Y}$ with $|\mathbb{I}| = d$
23: **if** $\exists \mathbb{I} \subseteq \mathbb{Y}$ such that $|\mathbb{I}| = d$ and $\vdash \bigcup \mathbb{I} \rightsquigarrow O$ is valid **then**
24: **return** $(\top, \mathbb{I})$ ▷ O is $\mathbb{I}$-simulatable
25: $\mathbb{Y} = \mathbb{Y} \cup \{\{x\} \mid x \in O \wedge x \notin \bigcup \mathbb{Y}\}$ ▷ Fail to prove and extend $\mathbb{Y}$
26: **return** $(\bot, \mathbb{Y})$ ▷ O is $\mathbb{I}'$-simulatable for $\mathbb{I}' \subseteq \mathbb{Y}$ with $|\mathbb{I}'| = d$
27: **Proc** Extend^d($\{(O_j, X_j)\}_j, \mathbb{Y}$) ▷ Extend $(O_j)_j$ using the witness $\mathbb{Y}$
28: **for all** $j, x \in X_j$ **do**
29: $O = \bigcup_j O_j \cup \{x\}$
30: $(res, \mathbb{Y}')$ =Check^d($O, \mathbb{Y}$)
31: **if** res == $\top$ **then**
32: $O_j = O_j \cup \{x\}$ ▷ $\bigcup_j O_j$ is still $\mathbb{Y}$-simulatable after adding x
33: **return** $\{O_j\}_j$

If τ is NI, by Proposition 3, it checks if any set of d probes can be simulated by at most d variable sets of $\mathbb{Y}$. This is achieved via invoking the procedure Explore^d (Line 6), which returns $\mathbb{Y}$ such that $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-NI. Remark that $\mathbb{Y}$ initialized at Lines 3–4 may not suffice to ensure $(\mathbb{Y}, d)$-NI, thus $\mathbb{Y}$ should be extended by Explore^d (Line 25).

If τ is SNI, it checks for each $0 \leq i \leq d$, if any set comprising i internal and $d - i$ external probes per output sharing $\mathbf{b}_j$ can be simulated by at most i variable sets of $\mathbb{Y}$. This is done by invoking Explore^i($\{(i, \mathcal{P}_{\mathtt{int}}), (d-i, \mathbf{b}_1), \cdots, (d-i, \mathbf{b}_k)\}, \mathbb{Y}, \tau$) (Line 8), where $\mathcal{P}_{\mathtt{int}}$ is the set of all internal probes on $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$. We note if τ is SNI, the set $\mathbb{Y}$ initialized at Lines 3–4 will not be extended by Explore^i, because it is useless for proving d-SNI/d-probing security, nor improving the verification efficiency. If no abort occurs, $\mathbb{Y}$ is returned such that $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-SNI,

otherwise a probe set O is emitted (Line 18) which is a potential leak. The emitted probe sets are recorded for further analysis, e.g., manual inspection or other computational-expensive techniques to remove false positives when the extended Explore (i.e., Algorithm 2) is utilized for checking d-probing security (cf. Section 4.4).

Explore^d checks if each probe set O in the list of pairs $\{(d_j, X_j)\}_j$ can be simulated by at most d variable sets of $\mathbb{Y}$, where (d_j, X_j) denotes taking d_j probes from the set X_j. (Note that $\sum_j d_j$ is the desired security order d.) Explore^d first chooses d_j probes from X_j for each pair (d_j, X_j) (Line 12), resulting in subsets $\{O_j\}_j$. Then it invokes Check^d (Line 13) to check if the set $\bigcup_j O_j$ can be simulated by a set $\mathbb{I}$ of d variable sets of $\mathbb{Y}$. The procedure Check^d returns a pair $(res, \mathbb{Y}')$ comprising a flag res and a family $\mathbb{Y}'$ of variable sets. If there exists $\mathbb{I} \subseteq \mathbb{Y}$ such that $|\mathbb{I}| = d$ and $\vdash \bigcup \mathbb{I} \rightsquigarrow \bigcup_j O_j$ is valid, Check^d returns $(\top, \mathbb{I})$, where $\mathbb{I}$ is the witness of the proof (Line 24). Otherwise, Check^d returns $(\bot, \mathbb{Y})$ where $\mathbb{Y}$ is extended with $\bigcup_j O_j$ so that $\bigcup_j O_j$ can be simulated by d variable sets after updating $\mathbb{Y}$ (Lines 25 and 26).

- If $res = \top$, Explore^d invokes the procedure Extend^d (Line 15) which for each pair $(O_j, X_j \setminus O_j)$, extends O_j with probes $x \in X_j \setminus O_j$ if the proof witness $\mathbb{Y}'$ is still able to prove the extended probe set (Lines 30–32).
- If $res = \bot$ and $\tau = \mathsf{SNI}$, Explore^d aborts and emits the probe set $\bigcup_j O_j$ on which we failed to prove (Line 18).

After that, to cover all the desired probe sets, the pairs $\{(d_j, X_j)\}_j$ are partitioned into the pairs $\{(d_j - i_j, O_j), (i_j, X_j \setminus O_j)\}_j$ for all the combinations $\{i_j\}_j$ such that $0 \le i_j \le d_j$ and $\sum_j i_j \neq 0$, where $\sum_j i_j \neq 0$ ensures that the exploring probe sets always contain probes from some $X_j \setminus O_j$, because otherwise they would have been proved. For each combination $\{i_j\}_j$, Explore^d recursively invokes Explore^d to check the worklist $\{(d_j - i_j, O_j), (i_j, X_j \setminus O_j)\}_j$ with the up-to-date set $\mathbb{Y}'$ in the loop at Lines 7–8. Note that $\mathbb{Y}'$ is an extension of $\mathbb{Y}$ if $res = \bot$. We remark that the loop at Lines 7–8 exits early *only* when a probe set O cannot be proved (i.e., an abort occurs at Line 18) for computing $\mathbb{Y}$ of a simple gadget to be $(\mathbb{Y}, d)$-SNI. Otherwise, the for-loop will not exit early, because we have to make sure that none of possible probe sets result in an abort for the initial $\mathbb{Y}$.

Theorem 2. *The procedure* SGadget$(f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d, \tau)$ *always terminates and if it returns* $(\mathbb{Y}, \tau)$ *for* $\tau \in \{\mathsf{NI}, \mathsf{SNI}\}$*, then the simple gadget* $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ *is* $(\mathbb{Y}, d)$*-*τ*.*

The termination of SGadget$(f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d, \tau)$ follows from Lemmas 3 and 4. We first prove two invariant properties of all the invocations of the procedure Explore^d by induction on the number of invocations of the procedure Explore^d.

Lemma 3. *In Algorithm 1, each invocation* Explore$^d(\{(d_j, X_j)\}_j, \mathbb{Y}, \tau)$ *has the following two properties:*

(1) $\sum_j d_j = d$ *if* $\tau = \mathsf{NI}$ *and* $\sum_j d_j \le kd$ *if* $\tau = \mathsf{SNI}$*, where* $k \ge 1$ *is the number of output sharings;*
(2) and $\bigcup_j X_j = \mathcal{P}$*, where* $\mathcal{P}$ *denotes the set of all the probes on* $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$*.*

Now, we can show that Explore^d in Algorithm 1, hence Algorithm 1, always terminates.

Lemma 4. *Each invocation of the procedure* Explore^d *in Algorithm 1 always terminates*

Proof. Observe that at each step of the recursive invocation of the procedure Explore^d, all the sets $\{X_j\}$ are partitioned into two subsets $\{O_j, X_j \setminus O_j\}_j$ such that $\bigcup_j O_j$ is non-empty and at least one index i_j is non-zero. By Lemma 3, we have: $\sum_j d_j = d$ if $\tau = \mathsf{NI}$ and $\sum_j d_j \le kd$ if $\tau = \mathsf{SNI}$, and $\bigcup_j X_j = \mathcal{P}$. Therefore, along with the recursion, the number of d_j's decreases and the cardinality of each X_j decreases, until there exists j such that $d_j > |X_j|$, Hence, the recursive invocation always terminates. □

Algorithm 2 Extended EXPLORE

1: **Proc** EXTEXPLORE$^d(\{(d_j, \mathbb{X}_j)\}_j, \mathbb{Y}, \tau)$
2: **if** $\exists j, d_j > |\mathbb{X}_j|$ **then return** $\mathbb{Y}$
3: $\{\mathbb{O}_j\}_j$ =EXTCHOOSE$(\{(d_j, \mathbb{X}_j)\}_j)$
4: $(res, \mathbb{Y}')$ = EXTCHECK$^d(\bigcup_j \mathbb{O}_j, \mathbb{Y})$
5: **if** res == $\top$ **then**
6: $\{\mathbb{O}_j\}_j$ = EXTEXTEND$^d(\{(\mathbb{O}_j, \mathbb{X}_j \backslash \mathbb{O}_j)\}_j, \mathbb{Y}')$
7: $\mathbb{Y}' = \mathbb{Y}$
8: **else if** τ == SNI **then** ABORT and EMIT the set $\bigcup_j \mathbb{O}_j$
9: **for** $j; 0 \leq i_j \leq d_j$ s.t. $\sum_j i_j \neq 0$ **do** $\mathbb{Y}$ =EXTEXPLORE$^d(\{(d_j - i_j, \mathbb{O}_j), (i_j, \mathbb{X}_j \backslash \mathbb{O}_j)\}_j, \mathbb{Y}', \tau)$
10: **return** $\mathbb{Y}$
11: **Proc** EXTCHECK$^d(\mathbb{O}, \mathbb{Y})$
12: **if** $\exists \mathbb{I} \subseteq \mathbb{Y}.\ |\mathbb{I}| = d \wedge \vdash \bigcup \mathbb{I} \rightsquigarrow \bigcup \mathbb{O}$ **then return** $(\top, \mathbb{I})$
13: $\mathbb{Y} = \mathbb{Y} \cup \{Y \in \mathbb{O} \mid \nexists Y' \in \mathbb{Y}. Y \subseteq Y'\}$
14: **return** $(\bot, \mathbb{Y})$
15: **Proc** EXTEXTEND$^d(\{(\mathbb{O}_j, \mathbb{X}_j)\}_j, \mathbb{Y})$
16: **for all** $j, X \in \mathbb{X}_j$ **do**
17: $\mathbb{O} = \bigcup_j \mathbb{O}_j \cup \{X\}$
18: $(res, \mathbb{Y}')$ =EXTCHECK$^d(\mathbb{O}, \mathbb{Y})$
19: **if** res == $\top$ **then** $\mathbb{O}_j = \mathbb{O}_j \cup \{X\}$
20: **return** $\{\mathbb{O}_j\}_j$

The following lemma ensures the correctness of Theorem 2.

Given a list of pairs $\{(d_j, X_j)\}_j$, let $C(\{(d_j, X_j)\}_j)$ denote the set of all the possible subsets $O \subseteq \mathcal{P}$ such that O contains d_j elements of X_j for each pair (d_j, X_j), where $\mathcal{P}$ denotes the set of all the probes.

LEMMA 5. *Suppose no abort occurs during all the invocations of* EXPLORE^d *and* $(\mathbb{Y}, \tau)$ *is the return of Algorithm 1. For every invocation of* EXPLORE^d *with a list of pairs* $\{(d_j, X_j)\}_j$ *and every probe set* $O \in C(\{(d_j, X_j)\}_j)$, O *is covered, i.e., there exist a probe set* O' *and a set of variable sets* $\mathbb{I} \subseteq \mathbb{Y}$ *such that* $O \subseteq O'$, $|\mathbb{I}| = d$ *and the judgement* $\vdash \bigcup \mathbb{I} \rightsquigarrow O'$ *is valid.*

PROOF. By induction on the number h of invocations of the procedure EXPLORE^d, where the base case is the largest h. Note that the largest h exists by Lemma 4.

- **Base case.** Since no abort occurs during all the invocations of EXPLORE^d, then the h-th invocation of the procedure EXPLORE^d must return at Algorithm 1 Line 11. Hence, there exists a pair (d_j, X_j) such that $d_j > |X_j|$. This implies that $C(\{(d_j, X_j)\}_j) = \emptyset$, thus, the result follows.
- **Inductive step.** Let $\{O_j\}_j$ be the sets before calling EXPLORE^d at Algorithm 1 Line 20. For every set $O \in C(\{(d_j^h, X_j^h)\}_j)$, either $O \subseteq \bigcup_j O_j$ or $O \nsubseteq \bigcup_j O_j$. If $O \subseteq \bigcup_j O_j$, the result follows from the fact that CHECK$^d(\bigcup_j O_j)$ succeeds. If $O \nsubseteq \bigcup_j O_j$, then there must exist a combination of values $i_j : 0 \leq i_j \leq d_j$ such that $\sum_j i_j \neq 0$ and $O \in C(\{(d_j - i_j, O_j), (i_j, X_j \backslash O_j)\}_j)$. By the induction hypothesis, O is covered.

This concludes the proof. □

Algorithm 3 Computing $\mathbb{Y}$ for a composite gadget

1: **Proc** CGadget($g(\mathbf{a}'_1, \cdots, \mathbf{a}'_k), d$)
2: Let gstmt be the sequence of gadget calls of $g(\mathbf{a}'_1, \cdots, \mathbf{a}'_k)$
3: **return** GadgetCalls(gstmt, d) ▷ Check gstmt
4: **Proc** Gadget($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$)
5: **if** f is a simple gadget **then**
6: $(\mathbb{Y}, \tau)$ =SGadget($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$, SNI) ▷ To prove $(\mathbb{Y}, d)$-SNI
7: **if** Aborted **then** ▷ Fail to prove $(\mathbb{Y}, d)$-SNI
8: $(\mathbb{Y}, \tau)$ =SGadget($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$, NI) ▷ To prove $(\mathbb{Y}, d)$-NI
9: **else** $(\mathbb{Y}, \tau)$ =CGadget($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$)
10: **return** $(\mathbb{Y}, \tau)$
11: **Proc** GadgetCalls(gstmt, d) ▷ Recursively check gstmt
12: $\mathbf{x}_1, \cdots, \mathbf{x}_h =_\ell f(\mathbf{y}_1, \cdots, \mathbf{y}_m)$ is Head(gstmt) ▷ Get the head of gstmt
13: $(\mathbb{Y}, \tau)$ =Gadget($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$)
14: $\mathbb{Y} = \mathbb{Y}[\mathbf{y}_1/\mathbf{a}_1, \cdots, \mathbf{y}_m/\mathbf{a}_m]@\ell$ ▷ Rename the variables of $\mathbb{Y}$
15: **if** Tail(gstmt)==empty **then return** $(\mathbb{Y}, \tau)$
16: $(\mathbb{Y}_1, \tau_1)$ =GadgetCalls(Tail(gstmt), d)
17: $\mathbb{Y}'_1 = \mathbb{Y}_1 \setminus \{\{\mathbf{x}_j[i]\} \mid 1 \le i \le t+1, 1 \le j \le h\}$
18: $\mathbb{Y} = \mathbb{Y} \cup \mathbb{Y}'_1$
19: **if** τ == SNI **then** ▷ Check the condition of Lemma 2
20: **for** $(i = 1; i \le d; i++)$ **do**
21: $\mathbb{Y}_2 = \{I \mid I$ comprises i shares of $\mathbf{x}_j$ for each j and all the input shares of Tail(gstmt)$\}$
22: ExtExplore$^i(\{(i, \mathbb{Y}'_1)\}, \mathbb{Y}_2$, SNI)
23: **if** Aborted or τ == NI **then** ▷ Check the condition of Lemma 1
24: Let $\mathcal{P}$ be the set of all the probes of $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$
25: $X = \mathcal{P}[\mathbf{y}_1/\mathbf{a}_1, \cdots, \mathbf{y}_m/\mathbf{a}_m]@\ell$
26: **for** $(i = 1; i < d; i++)$ **do**
27: $\mathbb{Y}$ =ExtExplore$^d(\{(i, \{\{x\}\}_{x \in X}), (d-i, \mathbb{Y}_1)\}, \mathbb{Y}$, NI)
28: **return** $(\mathbb{Y}, \tau')$

4.3 Computing the Sets $\mathbb{Y}$ for Composite Gadgets

To compute the sets $\mathbb{Y}$ for composite gadgets, as shown in Algorithm 2, we first extend the procedure Explore^d such that variable sets X_j in the pairs $\{(d_j, X_j)\}_j$ are sets $\mathbb{X}_j$ of variable sets for which d_j variable sets of $\mathbb{X}_j$ are added to O_j while other operations are generalized accordingly.

Following Lemmas 3 and 4, we can get that

Lemma 6. ExtExplore$^d(\{(d_j, \mathbb{X}_j)\}_j, \mathbb{Y}, \tau)$ *has following properties:*

(1) $\sum_j d_j = d$ *if* $\tau =$ NI *and* $\sum_j d_j \le kd$ *if* $\tau =$ SNI, *where* $k \ge 1$ *is the number of output sharings;*
(2) each invocation of ExtExplore^d *in Algorithm 2 always terminates;*
(3) each set $\mathbb{O} \in C(\{(d_j, \mathbb{X}_j)\}_j)$ *is covered if no abort occurs, where* $C(\{(d_j, X_j)\}_j)$ *now denotes the set of all the sets* $\mathbb{O}$ *comprising of* d_j *sets from* $\mathbb{X}_j$ *for each pair* $(d_j, \mathbb{X}_j)$.

According to Lemmas 1 and 2, we present the procedure CGadget in Algorithm 3. Given a composite gadget $g(\mathbf{a}'_1, \cdots, \mathbf{a}'_k)$ and a security order d, it computes a set $\mathbb{Y}$ such that the gadget $g(\mathbf{a}'_1, \cdots, \mathbf{a}'_k)$ is $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI by invoking the procedure GadgetCalls (Line 3) with the sequence gstmt of gadget calls in the body of $g(\mathbf{a}'_1, \cdots, \mathbf{a}'_k)$.

Algorithm 4 Checking d-probing security

1: **Proc** PROBING($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$)
2: $(\mathbb{Y}, \tau)$ =GADGET($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$)
3: EXTEXPLORE$^d(\{(d, \mathbb{Y})\}, \emptyset, \mathsf{SNI})$
4: **for all** probe sets O emitted by Algorithm 2 at Line 8 **do**
5: r =CHECKBYGPUENUM(O)
6: **if** r == No **then return** O ▷ O is a potential flaw
7: **return** Yes ▷ $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is d-probing secure

GADGETCALLS(gstmt, d) first computes a set $\mathbb{Y}$ for the first gadget call $\mathbf{x}_1, \cdots, \mathbf{x}_h =_\ell f(\mathbf{y}_1, \cdots, \mathbf{y}_m)$ of gstmt by invoking GADGET (Line 13), which returns a pair $(\mathbb{Y}, \tau)$ such that $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-τ with $\tau \in \{\mathsf{NI}, \mathsf{SNI}\}$, where $(\mathbb{Y}, d)$-NI is used only if an abort occurs when proving $(\mathbb{Y}, d)$-SNI. The set $\mathbb{Y}$ is revised accordingly using the actual parameters $\{\mathbf{y}_j\}_{1 \le j \le m}$ and the label ℓ (Line 14). The pair $(\mathbb{Y}, \tau)$ is returned (Line 15) if the tail TAIL(gstmt) is empty. Otherwise, it invokes GADGETCALLS to recursively compute a pair $(\mathbb{Y}_1, \tau_1)$ for the tail TAIL(gstmt) such that TAIL(gstmt) is $(\mathbb{Y}_1, d)$-τ_1, where TAIL(gstmt) is regarded as a gadget.

After proving that first gadget call is $(\mathbb{Y}, d)$-τ and the remaining gadget calls is $(\mathbb{Y}_1, d)$-τ_1, we compute the set $\mathbb{Y}$ of gstmt from $\mathbb{Y}$ and $\mathbb{Y}_1$ (Line 18), analogous to $\phi_{g \circ f}(\mathbb{Y}_f, \mathbb{Y}_g)$ in Lemmas 1 and 2. We check the composition condition according to the security type τ of the gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$.

- If $\tau = \mathsf{SNI}$, the condition of the composition rule R2 (i.e., Lemma 2) is checked by invoking EXTEXPLORE, i.e., the extension of the procedure EXPLORE.
- If abort occurs during the loop at Lines 20–22 or τ is NI, the condition of the composition rule R1 (i.e., Lemma 1) is checked by invoking EXTEXPLORE.

THEOREM 3. *The procedure* GADGET($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) *always terminates and if it returns* $(\mathbb{Y}, \tau)$ *for* $\tau \in \{\mathsf{NI}, \mathsf{SNI}\}$, *then the gadget* $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ *is* $(\mathbb{Y}, d)$-τ.

Termination of GADGET($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) follows from Lemma 6 and the correctness of Theorem 3 follows from Lemma 1, Lemma 2 and Theorem 2.

4.4 Verifying d-Probing Security

We present the procedure PROBING shown in Algorithm 4, to verify if a gadget is d-probing secure by first proving $(\mathbb{Y}, d)$-NI or $(\mathbb{Y}, d)$-SNI. PROBING($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) checks if $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is d-probing secure when the computations of input sharings $\mathbf{a}_j[t+1] = ms(a_j, \mathbf{a}_j[1], \cdots, \mathbf{a}_j[t])$ for $1 \le j \le m$ and $ms \in \{+, \oplus\}$ are given. It first invokes GADGET($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) to compute a pair $(\mathbb{Y}, \tau)$ so that $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $(\mathbb{Y}, d)$-τ. Next, by Proposition 4, it checks if any d variable sets of $\mathbb{Y}$ can be simulated without using any secrets $a_1, \cdots, a_m$ by invoking EXTEXPLORE$^d(\{(d, \mathbb{Y})\}, \emptyset, \mathsf{SNI})$. The parameter τ is set to SNI when invoking EXTEXPLORE^d, to disallow the update of $\mathbb{Y}$ during checking. All the emitted probe sets O by Algorithm 2 at Line 8 are recorded during EXTEXPLORE$^d(\{(d, \mathbb{Y})\}, \emptyset, \mathsf{SNI})$. These sets are potential leaks. To remove as many false positives as possible, for each potential leak O emitted by Algorithm 2 at Line 8 on which the proof system fails, we invoke the procedure CHECKBYGPUENUM(O) which computes the evaluation (i.e., distribution) $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)_O$ by iteratively enumerating all the possible valuations of the secrets and random variables from $\mathbb{F}$, similar to maskVerif and HOME [Gao et al. 2021]. In our implementation, we adopt the GPU-based brute-force algorithm in HOME. The gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is d-probing secure, if either no probe set O is emitted at Line 8 of Algorithm 2 or all of them are proved by invoking CHECKBYGPUENUM. Otherwise, O is a potential flaw.

THEOREM 4. $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ *is* d-*probing secure if* PROBING($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) *returns* Yes.

Algorithm 5 Checking d-NI and d-SNI

Proc CheckNI($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) ▷ Check if $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is d-NI
  $(\mathbb{Y}, \tau)$ =Gadget($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) ▷ τ is SNI or NI
  $\mathbb{X} = \{\{\mathbf{a}_j[i_j]\}_{1 \le j \le m} \mid 1 \le i_j \le t+1\}$
  ExtExplore$^d(\{(d, \mathbb{Y})\}, \mathbb{X}, \mathsf{SNI})$ ▷ Check the condition of Proposition 5
  return Yes
Proc CheckSNI($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) ▷ Check if $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is d-SNI
  $(\mathbb{Y}, \tau)$ =Gadget($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) ▷ τ is SNI or NI
  if $\tau \neq$ SNI **then return** No ▷ Fail to prove that $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $\mathbb{Y}$-SNI
  $\mathbb{X} = \{\{\mathbf{a}_j[i_j]\}_{1 \le j \le m} \mid 1 \le i_j \le t+1\}$
  ExtExplore$^d(\{(d, \mathbb{Y})\}, \mathbb{X}, \mathsf{SNI})$ ▷ Check the condition of Proposition 5
  return Yes ▷ $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is $\mathbb{Y}$-SNI

The correctness of Theorem 4 follows from Proposition 4 and Theorem 3.

4.5 Verifying d-NI and d-SNI

We present the procedures CheckNI and CheckSNI shown in Algorithm 5, to verify if a gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is d-NI or d-SNI by first proving $(\mathbb{Y}, d)$-NI or $(\mathbb{Y}, d)$-SNI.

Procedure CheckNI($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) checks whether $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is d-NI. It computes a set $\mathbb{Y}$ so that $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is either $(\mathbb{Y}, d)$-NI or $(\mathbb{Y}, d)$-SNI. It then computes the set $\mathbb{X} = \{\{\mathbf{a}_j[i_j]\}_{1 \le j \le m} \mid 1 \le i_j \le t+1\}$, where each set $\{\mathbf{a}_j[i_j]\}_{1 \le j \le m}$ contains one input share of each input sharing $\mathbf{a}_j$. Next, following Proposition 5, it invokes ExtExplore$^d(\{(d, \mathbb{Y})\}, \mathbb{X}, \mathsf{SNI})$ which checks if any d variable sets of $\mathbb{Y}$ can be simulated by a set comprising at most d input shares of each input sharing $\mathbf{a}_j$. Recall that τ is set to SNI when invoking ExtExplore^d, to disallow the update of $\mathbb{X}$ during checking. If CheckNI($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) returns Yes, $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is d-NI.

Procedure CheckSNI($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) checks whether the gadget $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ is d-SNI, similar to CheckNI($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$), except that τ should be SNI.

Theorem 5. *If the procedure* CheckNI($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$) *(resp.* CheckSNI($f(\mathbf{a}_1, \cdots, \mathbf{a}_m), d$)*) returns* Yes*, then the gadget* $f(\mathbf{a}_1, \cdots, \mathbf{a}_m)$ *is* d-NI *(resp.* d-SNI*).*

The correctness of Theorem 5 follows from Proposition 5 and Theorem 3.

We remark that in theory, the (worse case) time-complexity of the verification (as well as verifying the conditions in Lemmas 1 and 2) is exponential in the number of variables after inlining no matter whether Lemmas 1 and 2 are applied. The composition of gadgets (i.e., Lemmas 1 and 2) is polynomial in the number of variables when the conditions hold. In our experiments, Lemmas 1 and 2 are very effective in improving efficiency, cf. Table 3 for comparison between our tool and the state-of-the-art tool maskVerif [Barthe et al. 2019].

5 EVALUATION

We implement our algorithms as a verification tool CONVINCE. We first evaluate the scalability of CONVINCE on various higher-order composite gadgets based on efficient masked implementations of finite-field multiplication, the key non-linear building block of symmetric ciphers. We then show how to utilize CONVINCE to explore the design space of efficient higher-order masked implementations based on the AES Sbox, and report more efficient variants. All those efficient higher-order composite gadgets cannot be verified by all the existing compositional verification approaches.

Table 1. Statistics of four masked finite-field multiplication

d	SecMult				Para				Comp				UMA			
	#r	#⊕	#∧	Type	#r	#⊕	#∧	Type	#r	#⊕	#∧	Type	#r	#⊕	#∧	Type
2	3	12	9	2-SNI	3	12	9	2-SNI	2	10	9	2-NI	2	10	9	2-NI
3	6	24	16	3-SNI	4	20	16	3-NI	4	20	16	3-NI	4	20	16	3-NI
4	10	40	25	4-SNI	5	30	25	4-NI	5	30	25	4-NI	5	30	25	4-NI
5	15	60	36	5-SNI	12	48	36	5-NI	11	45	36	5-NI	9	48	36	5-NI

All experiments were conducted on a server with CentOS 7.6, Intel(R) Xeon(R) CPU E5-2690 v4@2.60GHz and 256GB RAM (only one core is used in our experiments).

In summary, the experimental results confirm that CONVINCE is more effective than the state-of-the-art compositional approaches for proving efficient masked implementations of cryptographic algorithms, and can be utilized by cryptographers to devise efficient and provable secure cryptographic software and hardware.

5.1 Scalability of CONVINCE

To evaluate the effectiveness and efficiency of our approach for checking d-probing security, d-NI and d-SNI of composite gadgets, we built benchmarks from four base composite gadgets, called B1, B2, B3 and B4, where B1 is XORMULTI, the last three are taken from [Belaïd et al. 2018] (cf. Figure 1, 10 and 15 of [Belaïd et al. 2018]), all of which rely on finite-field multiplication. When SecMult is used for finite-field multiplication in [Belaïd et al. 2018], B2 with order-t secret masking is d-probing secure for any orders t and $d \leq t$, B3 with order-t secret masking is *not* t-probing secure for some (unknown) order t, and B4 with order-t secret masking is a fixed version of B3 by refreshing one of the input sharings of SecMult using a t-SNI Refresh gadget, and is d-probing secure for any orders t and $d \leq t$.

We use three efficient masked implementations for finite-field multiplication, i.e., Para [Barthe et al. 2017], Comp [Belaïd et al. 2016] and UMA [Groß and Mangard 2018], whose numbers of randomness bits (#r), XOR operations (#⊕), AND operations (#∧), and security types are shown Table 1, compared over the finite-field multiplication gadget SecMult [Ishai et al. 2003] originally used in all four base composite gadgets. The corresponding security properties of those simple gadgets are the best ones and can be proved by many existing tools such as maskVerif [Barthe et al. 2019] and our tool CONVINCE. We did not compare CONVINCE with others on these simple gadgets, because it is not the focus of this work. Note that Comp and UMA (resp. Para and UMA) are the same implementations when the masking order $t = 2$ (resp. $t = 3, 4$) and all those gadgets for finite-field multiplications with the same masking order are functionally equivalent.

Our benchmarks are obtained by equivalently replacing the gadget SecMult in a base gadget "Base" with one efficient finite-field multiplication gadget "Multi", yielding the benchmark "Base" + "Multi". For instance, B1 + Para is the benchmark where the gadget Para is used for finite-field multiplication in the base gadget B1. These give rise to a total of 12 combinations, none of which with order-d secret masking can be proved of d-probing secure or d-NI by the existing compositional approaches. We remark that none of B1, B2, B3 and B4 with the finite-field multiplication gadget SecMult can be proved by the compositional approach [Barthe et al. 2016] as claimed in [Belaïd et al. 2018]. When SecMult in B1, B2, B3 and B4 are replaced with more efficient ones, the compositional approach [Barthe et al. 2016] still fails while the resulting gadgets further go beyond the verification capability of [Belaïd et al. 2018].

When verifying d-probing security, to ensure that any two input sharings are mutually independent, presharing operations are added for each benchmark using Boolean secret masking, which first computes sharings of input secrets and then invokes the benchmark with those sharings. As

Table 2. Verification results, where PS=Probing Security

Name	2nd-order		3rd-order		4th-order		5th-order	
	Result	Time(s)	Result	Time(s)	Result	Time(s)	Result	Time(s)
B1+Para	2-NI	0.01	3-NI	0.19	4-NI	5	5-NI	255
B1+Comp	2-NI	0.01	3-NI	0.27	4-NI	6	5-NI	244
B1+UMA	2-NI	0.01	3-NI	0.19	4-NI	5	5-NI	288
B2+Para	2-PS	0.01	3-PS	0.17	4-PS	5	5-PS	374
B2+Comp	2-PS	0.01	3-PS	0.25	4-PS	6	5-PS	240
B2+UMA	2-PS	0.01	3-PS	0.17	4-PS	5	5-PS	279
B3+Para	2-PS	0.01	3-PS	0.42	✗ (30)	21	✗ (287)	738
B3+Comp	2-PS	0.01	3-PS	0.63	✗ (30)	22	✗ (287)	730
B3+UMA	2-PS	0.01	3-PS	0.42	✗ (30)	21	✗ (287)	770
B4+Para	2-PS	0.02	3-PS	0.72	4-PS	41	5-PS	2,849
B4+Comp	2-PS	0.02	3-PS	0.79	4-PS	41	5-PS	2,809
B4+UMA	2-PS	0.02	3-PS	0.72	4-PS	41	5-PS	2,854

mentioned in [Belaïd et al. 2016], from a practical point of view, cases $d \leq 4$ are actually used in current real-life implementations. Thus, we consider secret masking orders d such that $2 \leq d \leq 5$. We do not consider the case $d = 1$ because it is too trivial.

The verification results are reported in Table 2, where the security order d is set to the masking order t. Column (Name) shows the benchmark name. Column (Result) shows the strongest security type proved via CONVINCE, where ✗ denotes that there are some subsets of $\mathbb{Y}$ (the number of such subsets is provided in the parentheses) that cannot be proved secure while all the other cases are proved without invoking CheckByGPUEnum. Column (Time) shows the execution time in seconds. We can observe that the security of all the benchmarks can be quickly proved in a few seconds when the security order is no more than 3. At the 4th-order and 5th-order, CONVINCE can still prove the security of 9 out of 12 benchmarks. To better understand the effectiveness of our approach on these benchmarks, we manually check them to identify the strongest security properties they may have. We find that (1) gadgets that are proved d-probing secure (PS) are indeed not d-NI, (2) gadgets that are proved d-NI are not d-SNI, and (3) gadgets that cannot be proved d-probing secure remain unknown (6 cases) because they are too complicated to be determined manually or proved automatically using existing tools including ours.

In detail, `XORMULTI` with the gadgets `Para`, `Comp` and `UMA` are proved of d-NI for all $d \in \{2, 3, 4, 5\}$. `Example1`, `Example2` and `Example2Corr` with the gadgets `Para`, `Comp` and `UMA` are proved of d-probing secure for $d \in \{2, 3, 4, 5\}$, except for `Example2` at 4th-order and 5th-order. Interestingly, we prove that `Example2` is actually d-probing secure for any $d \in \{2, 3\}$ when `SecMult` is replaced by any of the `Para`, `Comp` and `UMA`. We found that `Example2` has 30 subsets at 4th-order and 287 subsets at 5th-order that cannot be proved secure. We analyzed them manually and confirmed that `Example2` is indeed not d-probing security for any $d \in \{4, 5\}$ and any of the gadgets `Para`, `Comp` and `UMA`. (Note that `Example2` with masking order d was *not* d-probing secure for some order d [Belaïd et al. 2018]; it remains an open question to identify such an order explicitly.)

Recall that our composition rules, as well as Algorithm 3, may not produce the smallest set $\mathbb{Y}$ to ensure $(\mathbb{Y}, d)$-NI/$(\mathbb{Y}, d)$-SNI. Nevertheless, our results show that Algorithm 3 is able to compute a sufficiently satisfactory set $\mathbb{Y}$ for each gadget by which d-probing security or d-NI can be proved.

5.2 Application to the AES Sbox

We show how to utilize CONVINCE to explore the design space of efficient masked implementations based on the AES Sbox.

[Goudarzi and Rivain 2017] proposed a secure and efficient bitsliced masked implementation of the AES Sbox (`GR-Sbox`) using order-d Boolean secret masking, which contains 32 gadget calls

Table 3. Statistics of Sboxes and verification time, where Timeout (T.O.) is set to be 10 hours

Benchmark	#r	Number of operators				CONVINCE	maskVerif [Barthe et al. 2015]	SILVER [Knichel et al. 2020]
		XOR	AND	NOT	Total			
2nd-Order								
GR-Sbox	192	825	288	4	1,117	19s	270s	T.O.
BGR-Sbox	96	633	288	4	925	39s	252s	T.O.
P-Sbox-v1	96	633	288	4	925	38s	563s	T.O.
C-Sbox-v1	68	577	288	4	869	52s	466s	T.O.
P-Sbox-v2	96	633	288	4	925	39s	449s	T.O.
C-Sbox-v2	86	613	288	4	905	34s	665s	T.O.
3rd-Order								
GR-Sbox	384	1,484	512	4	2,000	1,837s	T.O.	T.O.
BGR-Sbox	192	1,100	512	4	1,616	22,463s	T.O.	T.O.
P-Sbox-v1	172	1,060	512	4	1,576	26,295s	T.O.	T.O.
C-Sbox-v1	172	1,060	512	4	1,576	25,912s	T.O.	T.O.
P-Sbox-v2	182	1,080	512	4	1,596	17,557s	T.O.	T.O.
C-Sbox-v2	182	1,080	512	4	1,596	17,186s	T.O.	T.O.

to the d-SNI finite-field multiplication SecMult. In this implementation, for each SecMult, one of two operands is refreshed via a d-SNI refresh gadget, resulting in 32 d-SNI refresh gadget calls. To obtain verified efficient implementations of the AES Sbox which is the key building block of AES, [Belaïd et al. 2018] developed tightPROVE and proved that GR-Sbox remains d-probing secure at any masking order d with *no* d-SNI refresh gadget call prior to each SecMult, named BGR-Sbox. This yields the AES Sbox with less refreshing. However, prior to this work it was an open problem whether BGR-Sbox could be improved further. We show that CONVINCE can be used to explore the design space of efficient gadgets using BGR-Sbox and report more efficient variants of BGR-Sbox.

To explore the design space of efficient gadgets using BGR-Sbox, we use two efficient masked implementations for finite-field multiplication, i.e., Para [Barthe et al. 2017] and Comp [Belaïd et al. 2016] whose numbers of randomness bits (#r), XOR operations (#⊕), AND operations (#∧), and security types are shown Table 1, compared over the finite-field multiplication gadget SecMult [Ishai et al. 2003]. All those gadgets for finite-field multiplications with the same masking order are functionally equivalent.

We iteratively substitute one of 32 gadget calls to SecMult in BGR-Sbox with the more efficient one Para and verify if the implementation remains probing secure using CONVINCE. The substitution is kept if it is secure, and this process is repeated until no more SecMult can be substituted, yielding a new implementation, called P-Sbox-v1. Based on P-Sbox-v1, we apply similar substitutions to SecMult and Para using Comp, yielding a new gadget, called C-Sbox-v1, as Comp is more efficient than Para at the 2nd-order. Table 3 reports the amount of randomness and numbers of operations of P-Sbox-v1 and C-Sbox-v1, compared to GR-Sbox and BGR-Sbox. We can observe that in general P-Sbox-v1 and C-Sbox-v1 are more efficient than the state-of-the-art Sbox BGR-Sbox. To satisfy the composition rules proposed by [Belaïd et al. 2018] so that one can build provable secure masked implementations of full AES using more efficient implementations of Sbox, some gadget calls to Para/Comp gadgets in P-Sbox-v1 and C-Sbox-v1 are replaced by SecMult, resulting in P-Sbox-v2 and C-Sbox-v2. While they may use more randomness and operations than P-Sbox-v1 and C-Sbox-v1, they are still more efficient than the state-of-the-art Sbox BGR-Sbox except for 2nd-order P-Sbox-v2.

As no existing compositional verifier can be used to verify BGR-Sbox after some substitutions, we compare CONVINCE with two state-of-the-art open-sourced d-probing security verifiers maskVerif [Barthe et al. 2019] and SILVER [Knichel et al. 2020] for which all the gadget calls have to be inlined manually. We do not compare with HOME [Gao et al. 2021], QMVerif [Gao et al. 2022], SCInfer [Zhang et al. 2018] and SC Sniffer [Eldib et al. 2014], REBECCA [Bloem et al.

Table 4. The number of randomness and operations in provable secure masking of full AES

Benchmark	#r	#XOR	#AND	#NOT	#Operation
2nd-Order					
GR-AES	30,720	151,344	46,080	640	198,064
BGR-AES	15,360	120,624	46,080	640	167,344
Para-AES	15,360	120,624	46,080	640	167,344
Comp-AES	13,760	117,424	46,080	640	164,144
3rd-Order					
GR-AES	61,440	263,232	81,920	640	345,792
BGR-AES	30,720	201,792	81,920	640	284,352
Para-AES	29,120	198,592	81,920	640	281,152
Comp-AES	29,120	198,592	81,920	640	281,152

2018] (and its variant CocoAlma [Gigerl et al. 2021; Hadzic and Bloem 2021]), because HOME and REBECCA are significantly less efficient than maskVerif for verifying d-probing security of the masked implementations of AES Sbox (cf. [Barthe et al. 2019; Gao et al. 2021]), and QMVerif, SCInfer and SC Sniffer are limited to first-order security only. The verification results for d-probing security are shown in the last three columns of Table 3, where the security order d is set to be the masking order t. maskVerif fails on the 3rd-order Sbox implementations, and for the 2rd-order Sbox implementations, its verification time is at least 6.4 times more than that of CONVINCE. SILVER does not terminate in 10 hours. CONVINCE can prove d-probing security of them without invoking CheckByGPUEnum and SILVER is included here as it has never been compared with other verifiers. However, we do not know if they are d-NI or d-SNI because they are too complicated to be determined manually or proved automatically using existing tools including ours.

To demonstrate the efficiency gains in provable secure full AES using more efficient Sbox implementations P-Sbox-v2 and C-Sbox-v2, Table 4 shows the randomness and number of operations of full AES Para-AES and Comp-AES using P-Sbox-v2 and C-Sbox-v2. We can observe that for the 3rd-order full AES, 1,600 random variables and 3,200 XOR operations are reduced over the state-of-the-art BGR-AES that uses BGR-Sbox. Although we can deduce that the d-order benchmarks for $d = 2, 3$ in Table 4 are d-probing secure according to the proved d-order probing security properties of C-Sbox-v2 and P-Sbox-v2 and the composition rules of [Belaïd et al. 2018, Proposition 14] (indeed, they are *only* d-probing secure,) no existing tool is able to directly prove them right now.

6 RELATED WORK

In this section, we discuss the related work on automated formal verification of masked implementations and synthesis of leakage resilient programs against power side-channel attacks.

Non-compositional verifications. The pioneering work is done by [Moss et al. 2012], which proposed a type system for checking masked implementations, limited to certain operations (i.e., $\oplus$). [Bayrak et al. 2013] proposed heuristic rules based on *don't care* random variables, however it is neither sound nor complete. [Eldib et al. 2014] proposed a model-counting based approach which reduces to a series of satisfiability problems which are checked by existing SMT solvers. While this approach is sound and complete in theory, the reduction is exponential in the number of random variables and thus limited to Boolean programs *only*. To mitigate the scalability issue of [Eldib et al. 2014], inference and symbolic approaches have been proposed [Meunier et al. 2020; Ouahma et al. 2017]. In general, the inference and symbolic approaches are efficient and sound but incomplete while the model-counting based approaches are complete but not scalable, which motivated hybrid approaches [Gao et al. 2019a,b; Zhang et al. 2018], bringing the best of two worlds. All the above works focus on first-order probing security *only* and are non-compositional. In contrast to those

works, this work addresses higher-order probing security and our compositional approach is sound for verifying higher-order probing security.

To verify higher-order probing security, [Barthe et al. 2015] proposed a language-level characterization of d-probing security, called d-NI, based on which a sound verification approach was presented. Following this line, [Bloem et al. 2018] proposed a Fourier analysis based approach to verify higher-order probing security with glitch. Both of them are limited to Boolean programs while arithmetic programs cannot be tackled. [Coron 2018] proposed an approach to support arithmetic programs by leveraging elementary transformations. The Fourier analysis based approach [Bloem et al. 2018] has been extended to verify RISC-V assembly implementations [Gigerl et al. 2021] and further improved for verifying hardware circuits [Hadzic and Bloem 2021]. However, this approach usually requires execution traces which have to be obtained from highly tedious simulations and suffers from limited scalability (e.g., [Gigerl et al. 2021] took 18 minutes to verify a *first-order* masked RISC-V implementation of AES Sbox, and [Hadzic and Bloem 2021] could only verify a *first-order* masked hardware circuit of the first round of ten AES rounds). The hybrid approach [Gao et al. 2019a] has been extended to verify higher-order probing security in [Gao et al. 2021], aided with a GPU based model-counting approach. SILVER [Knichel et al. 2020] proposed to encode masked gadgets as a binary decision diagram on which the security is verified. All those approaches are non-compositional and thus become intractable for large composite gadgets via gadget inline. Compared with them, our approach can verify both Boolean and arithmetic programs in a compositional way, thus is more efficient (cf. Section 5).

Compositional verification. Compositional approaches have been proposed to verify composite gadgets without gadget inline. [Barthe et al. 2016] extended the non-compositional d-NI notion [Barthe et al. 2015] to compositional ones d-SNI and d-NI and proposed the first compositional reasoning approach for verifying d-SNI and d-NI. This idea has been extended to verify d-probing security, d-SNI and d-NI with glitches and transitions of *simple* gadgets [Barthe et al. 2019] while does not support compositional reasoning. Though promising, this compositional approach fails to verify efficient gadgets, hence may incur more random variables and operations. To overcome this limitation, [Belaïd et al. 2018] proposed a new compositional verification approach, named tightPROVE, for verifying d-probing security via matrix analysis. However, their approach can only verify composite gadgets built up from specific gadgets, thus, tightPROVE+ [Belaïd et al. 2020] is proposed to support more fixed implementations of operations. This line of work in general only works for gadgets satisfying d-NI/d-SNI which are stronger than d-probing security and rule out many efficient masked implementations widely used in practice.

[Blot et al. 2017] lifted the SMT based approach of [Eldib et al. 2014] to higher-order probing security with a new compositionality property that requires inserting unnecessary input encoders for sequential composition, similar to inserting d-SNI `Refresh` gadgets. [Cassiers and Standaert 2020] proposed another security notion PINI achieving the same "trivial compositionality" of d-SNI, and algorithms for verifying d-PINI notion have been proposed [Cassiers et al. 2021; Knichel et al. 2020]. Finally, [Gao et al. 2022] extended the non-compositional type system of [Gao et al. 2019a] to a compositional type system. However, it requires user-defined annotations and is limited to first-order security *only*.

Compared with existing compositional approaches, the current work targets d-probing security of any given composition of gadgets, in particular, efficient masked implementations. To the best of our best knowledge, it is a first-of-its-kind compositional approach for verifying higher-order probing security of composite gadgets. We finally remark that this work is not intended to compete with the existing compositional approaches on composite gadgets that can already be proved, but is aimed to prove efficient composite gadgets that cannot be proved by the existing compositional approaches, which is vital for resource limited devices, e.g., Internet of things devices.

Synthesis. Automatic synthesis of leakage resilient programs has been studied. Early work [Agosta et al. 2012; Bayrak et al. 2011] relied on compiler-like pattern matching, and they do not use formal verification to provide guarantees. With their model-counting based verification approaches, [Eldib and Wang 2014] introduced a constraint-based synthesis approach which can generate verified first-order leakage resilient programs, and [Blot et al. 2017] proposed a compositional synthesis approach which is able to generate higher-order leakage resilient programs. [Barthe et al. 2016] uses a proof system to check if a program is d-NI or d-SNI. In case it fails, some refresh gadgets can be inserted to pass the check. This idea has been adopted in [Belaïd et al. 2020] by leveraging the verifier tightPROVE+. [Wang et al. 2019] proposed an approach to eliminate compiler-induced leakages by leveraging the type inference of [Zhang et al. 2018] to detect potential leakages.

Our focus is on verification, but could also be extended to synthesize leakage resilient programs by inserting refresh gadgets, similar to [Barthe et al. 2016; Belaïd et al. 2020].

7 CONCLUSION AND FUTURE WORK

We have introduced new language-level security notions and novel compositional approaches, as well as their implementation, for rigorously proving efficient masked implementations of cryptographic algorithms. This work paves a new way for this task by explicitly tracking variables for simulating probes.

This work fills the gap between compositional verification and efficient implementations of gadgets. However, similar to prior work [Barthe et al. 2015, 2016; Belaïd et al. 2020, 2018; Blot et al. 2017; Cassiers and Standaert 2020; Eldib et al. 2014; Gao et al. 2021; Meunier et al. 2020; Ouahma et al. 2017; Zhang et al. 2018], we did not consider physical defaults (such as glitches and transitions) [Barthe et al. 2019] which essentially allows the adversary to observe one or more internal variables by one probe. Our approach could readily be adapted to verify security with glitches or transitions (similar to [Barthe et al. 2019]), as our approach proves security by finding a variable set to simulate each observable set while the size of the observable set does not matter. We leave the extensions of transitions and glitches as future work.

Our approach only guarantees soundness, thus may introduce false positives. There exist non-compositional and compositional verification approaches that are both sound and complete in theory, but the former requires model-counting (e.g. [Barthe et al. 2019; Eldib et al. 2014; Gao et al. 2019a,b; Zhang et al. 2018]) that has limited scalability; the latter (e.g., [Belaïd et al. 2020, 2018]) only supports composite gadgets built up from some fixed implementations. Developing sound and complete—and scalable—compositional verification approaches with full support of higher-order efficient masked implementations would be challenging future work as well.

Our verification approach could potentially be applied for verifying Shamir's secret sharing scheme [Shamir 1979] which has been widely used in secure multiparty computation (MPC) [Cramer et al. 2015]. Given a computation $P(s)$, Shamir's (d, t)-threshold secret sharing scheme for $1 \leq d < t$ splits the secret s into t shares $(f(x_1), \cdots, f(x_t))$ for a random polynomial $f(x) = s + a_1x + a_2x^2 + \cdots + a_{d-1}x^{d-1} \mod p$, a random prime $p > t$ and t random values $x_1, \cdots, x_t$. Note that x_i's, d, t, and q are public. Moreover, the secret $s = f(0)$ and the coefficients $a_1, \cdots, a_{d-1}$ can be recovered by any d shares of $(f(x_1), \cdots, f(x_t))$ via Lagrange's Interpolation Theorem, but cannot for any $d' < d$ shares of $(f(x_1), \cdots, f(x_t))$. The computation P should be implemented by t programs $P_1, \cdots, P_t$ in such a way that $P(s)$ can be recovered by any d shares of $(P_1(f(x_1)), \cdots, P_t(f(x_t)))$ whereas any $d' < d$ parties of the programs $P_1, \cdots, P_t$ cannot infer any information of the secret s. The latter can be reduced to checking whether the (intermediate) computation results of any $d' < d$ parties of the programs $P_1, \cdots, P_t$ are statistically independent of the secret s where our verification approach could be adapted. We leave this as interesting future work.

8 DATA-AVAILABILITY STATEMENT

The tool and verification benchmarks (excluding full AES implementations) are available at [Gao et al. 2023].

REFERENCES

Giovanni Agosta, Alessandro Barenghi, and Gerardo Pelosi. 2012. A code morphing methodology to automate power analysis countermeasures. In *Proceedings of the 49th Annual Design Automation Conference*. 77–82. https://doi.org/10.1145/2228360.2228376

Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. 2020. Improved parallel mask refreshing algorithms: generic solutions with parametrized non-interference and automated optimizations. *J. Cryptogr. Eng.* 10, 1 (2020), 17–26. https://doi.org/10.1007/s13389-018-00202-2

Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. 2015. Verified Proofs of Higher-Order Masking. In *Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 457–485. https://doi.org/10.1007/978-3-662-46800-5_18

Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. 2016. Strong Non-Interference and Type-Directed Higher-Order Masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 116–129. https://doi.org/10.1145/2976749.2978427

Gilles Barthe, Sonia Belaïd, Pierre-Alain Fouque, and Benjamin Grégoire. 2019. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *Proceedings of the 24th European Symposium on Research in Computer Security*. 300–318. https://doi.org/10.1007/978-3-030-29959-0_15

Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. 2017. Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. In *Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 535–566. https://doi.org/10.1007/978-3-319-56620-7_19

Gilles Barthe, Marc Olivier Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, and Lars Porth. 2021. Masking in fine-grained leakage models: Construction, implementation and verification. *IACR transactions on cryptographic hardware and embedded systems* (2021). https://doi.org/10.46586/tches.v2021.i2.189-228

Ali Galip Bayrak, Francesco Regazzoni, Philip Brisk, François-Xavier Standaert, and Paolo Ienne. 2011. A first step towards automatic application of power analysis countermeasures. In *Proceedings of the 48th Design Automation Conference*. 230–235. https://doi.org/10.1145/2024724.2024778

Ali Galip Bayrak, Francesco Regazzoni, David Novo, and Paolo Ienne. 2013. Sleuth: Automated Verification of Software Power Analysis Countermeasures. In *Proceedings of the 15th International Workshop on Cryptographic Hardware and Embedded Systems*. 293–310. https://doi.org/10.1007/978-3-642-40349-1_17

Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. 2016. Randomness Complexity of Private Circuits for Multiplication. In *Proceedings of the 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 616–648. https://doi.org/10.1007/978-3-662-49896-5_22

Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. 2017. Private Multiplication over Finite Fields. In *Proceedings of the 37th Annual International Cryptology Conference*. 397–426. https://doi.org/10.1007/978-3-319-63697-9_14

Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. 2020. Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations. In *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 311–341. https://doi.org/10.1007/978-3-030-45727-3_11

Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. 2018. Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In *Proceedings of the 24th International Conference on the Theory and Application of Cryptology and Information Security*. 343–372. https://doi.org/10.1007/978-3-030-03329-3_12

Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. 2022. IronMask: Versatile Verification of Masking Security. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy*. IEEE, 142–160. https://doi.org/10.1109/SP46214.2022.9833600

Alex Biryukov, Daniel Dinu, Yann Le Corre, and Aleksei Udovenko. 2017. Optimal first-order boolean masking for embedded IoT devices. In *Proceedings of the International Conference on Smart Card Research and Advanced Applications*. 22–41. https://doi.org/10.1007/978-3-319-75208-2_2

Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. 2018. Formal Verification of Masked Hardware Implementations in the Presence of Glitches. In *Proceedings of the 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 321–353. https://doi.org/10.1007/978-

3-319-78375-8_11

Arthur Blot, Masaki Yamamoto, and Tachio Terauchi. 2017. Compositional Synthesis of Leakage Resilient Programs. In *Proceedings of the 6th International Conference on Principles of Security and Trust*. 277–297. https://doi.org/10.1007/978-3-662-54455-6_13

Nicolas Bordes and Pierre Karpman. 2021. Fast verification of masking schemes in characteristic two. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 283–312. https://doi.org/10.1007/978-3-030-77886-6_10

Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. 2012. Higher-Order Masking Schemes for S-Boxes. In *Proceedings of the 19th International Workshop Fast Software Encryption*. 366–384. https://doi.org/10.1007/978-3-642-34047-5_21

Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. 2021. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Computers* 70, 10 (2021), 1677–1690. https://doi.org/10.1109/TC.2020.3022979

Gaëtan Cassiers and François-Xavier Standaert. 2020. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Inf. Forensics Secur.* (2020), 2542–2555. https://doi.org/10.1109/TIFS.2020.2971153

Jean-Sébastien Coron. 1999. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*. 292–302. https://doi.org/10.1007/3-540-48059-5_25

Jean-Sébastien Coron. 2014. Higher Order Masking of Look-Up Tables. In *Proceedings of the 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 441–458. https://doi.org/10.1007/978-3-642-55220-5_25

Jean-Sébastien Coron. 2018. Formal Verification of Side-Channel Countermeasures via Elementary Circuit Transformations. In *Proceedings of the 16th International Conference on Applied Cryptography and Network Security*. 65–82. https://doi.org/10.1007/978-3-319-93387-0_4

Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. 2015. Conversion from Arithmetic to Boolean Masking with Logarithmic Complexity. In *Proceedings of the 22nd International Workshop on Fast Software Encryption*. 130–149. https://doi.org/10.1007/978-3-662-48116-5_7

Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. 2014. Secure Conversion between Boolean and Arithmetic Masking of Any Order. In *Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems*. 188–205. https://doi.org/10.1007/978-3-662-44709-3_11

Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. 2013. Higher-Order Side Channel Security and Mask Refreshing. In *Proceedings of the 20th International Workshop on Fast Software Encryption*. 410–424. https://doi.org/10.1007/978-3-662-43933-3_21

Ronald Cramer, Ivan Bjerre Damgård, et al. 2015. *Secure multiparty computation*. Cambridge University Press.

Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. 2015. Making masking security proofs concrete. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 401–429. https://doi.org/10.1007/978-3-662-46800-5_16

Hassan Eldib and Chao Wang. 2014. Synthesis of Masking Countermeasures against Side Channel Attacks. In *Proceedings of the 26th International Conference on Computer Aided Verification*. 114–130. https://doi.org/10.1007/978-3-319-08867-9_8

Hassan Eldib, Chao Wang, and Patrick Schaumont. 2014. Formal Verification Of Software Countermeasures against Side-Channel Attacks. *ACM Transactions on Software Engineering and Methodology* 24, 2 (2014), 11. https://doi.org/10.1145/2685616

Pengfei Gao, Hongyi Xie, Fu Song, and Taolue Chen. 2021. A Hybrid Approach to Formal Verification of Higher-Order Masked Arithmetic Programs. *ACM Trans. Softw. Eng. Methodol.* 30, 3 (2021), 26:1–26:42. https://doi.org/10.1145/3428015

Pengfei Gao, Hongyi Xie, Pu Sun, Jun Zhang, Fu Song, and Taolue Chen. 2022. Formal Verification of Masking Countermeasures for Arithmetic Programs. *IEEE Trans. Software Eng.* 48, 3 (2022), 973–1000. https://doi.org/10.1109/TSE.2020.3008852

Pengfei Gao, Hongyi Xie, Jun Zhang, Fu Song, and Taolue Chen. 2019a. Quantitative Verification of Masked Arithmetic Programs Against Side-Channel Attacks. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 155–173. https://doi.org/10.1007/978-3-030-17462-0_9

Pengfei Gao, Jun Zhang, Fu Song, and Chao Wang. 2019b. Verifying and Quantifying Side-channel Resistance of Masked Software Implementations. *ACM Trans. Softw. Eng. Methodol.* 28, 3 (2019), 16:1–16:32. https://doi.org/10.1145/3330392

Pengfei Gao, Yedi Zhang, Fu Song, Taolue Chen, and Francois-Xavier Standaert. 2023. CONVINCE. https://doi.org/10.5281/zenodo.8416208

Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. 2021. Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs. In *Proceedings of the 30th USENIX Security Symposium*. 1469–1468.

Louis Goubin. 2001. A Sound Method for Switching between Boolean and Arithmetic Masking. In *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems*. 3–15. https://doi.org/10.1007/3-540-44709-1_2

Louis Goubin and Jacques Patarin. 1999. DES and Differential Power Analysis (The "Duplication" Method). In *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. 158–172. https://doi.org/10.1007/3-540-48059-5_15

Dahmun Goudarzi and Matthieu Rivain. 2017. How Fast Can Higher-Order Masking Be in Software?. In *Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 567–597. https://doi.org/10.1007/978-3-319-56620-7_20

Hannes Groß and Stefan Mangard. 2018. A unified masking approach. *J. Cryptogr. Eng.* 8, 2 (2018), 109–124. https://doi.org/10.1007/s13389-018-0184-y

Vedad Hadzic and Roderick Bloem. 2021. COCOALMA: A Versatile Masking Verifier. In *Proceedings of the Formal Methods in Computer Aided Design*. 1–10. https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_9

Yuval Ishai, Amit Sahai, and David A. Wagner. 2003. Private Circuits: Securing Hardware against Probing Attacks. In *Proceedings of the 23rd Annual International Cryptology Conference*. 463–481. https://doi.org/10.1007/978-3-540-45146-4_27

Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. 2002. Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. In *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems, Revised Papers*. 129–143. https://doi.org/10.1007/3-540-36400-5_11

Matthias J. Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. 2018. Differential Power Analysis of XMSS and SPHINCS. In *Proceedings of the 9th International Workshop on Constructive Side-Channel Analysis and Secure Design*. 168–188. https://doi.org/10.1007/978-3-319-89641-0_10

Pierre Karpman and Daniel S Roche. 2018. New instantiations of the CRYPTO 2017 masking schemes. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*. 285–314. https://doi.org/10.1007/978-3-030-03329-3_10

HeeSeok Kim, Seokhie Hong, and Jongin Lim. 2011. A Fast and Provably Secure Higher-Order Masking of AES S-Box. In *Proceedings of the 13th International Workshop on Cryptographic Hardware and Embedded Systems*. 95–107. https://doi.org/10.1007/978-3-642-23951-9_7

David Knichel, Pascal Sasdrich, and Amir Moradi. 2020. SILVER - Statistical Independence and Leakage Verification. In *Proceedings of the 26th International Conference on the Theory and Application of Cryptology and Information Security*. 787–816. https://doi.org/10.1007/978-3-030-64837-4_26

Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *Proceedings of the 19th Annual International Cryptology Conference*. 388–397. https://doi.org/10.1007/3-540-48405-1_25

Xuejia Lai and James L. Massey. 1990. A Proposal for a New Block Encryption Standard. In *Proceedings of the Workshop on the Theory and Application of of Cryptographic Techniques*. 389–404. https://doi.org/10.1007/3-540-46877-3_35

Chao Luo, Yunsi Fei, and David R. Kaeli. 2018. Effective simple-power analysis attacks of elliptic curve cryptography on embedded systems. In *Proceedings of the International Conference on Computer-Aided Design*. 115. https://doi.org/10.1145/3240765.3240802

Quentin L. Meunier, Inès Ben El Ouahma, and Karine Heydemann. 2020. SELA: a Symbolic Expression Leakage Analyzer. In *Proceedings of the International Workshop on Security Proofs for Embedded Systems*.

Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. 2012. Compiler Assisted Masking. In *Proceedings of the 14th International Workshop on Cryptographic Hardware and Embedded Systems*. 58–75. https://doi.org/10.1007/978-3-642-33027-8_4

Inès Ben El Ouahma, Quentin Meunier, Karine Heydemann, and Emmanuelle Encrenaz. 2017. Symbolic Approach for Side-Channel Resistance Analysis of Masked Assembly Codes. In *Proceedings of the 6th International Workshop on Security Proofs for Embedded Systems*, Vol. 49. 17–32. https://doi.org/10.29007/hhnf

Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. 2009. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Computers* 58, 6 (2009), 799–811. https://doi.org/10.1109/TC.2009.15

Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. 2019. Generic Side-channel attacks on CCA-secure lattice-based PKE and KEM schemes. *IACR Cryptol. ePrint Arch.* 2019 (2019), 948.

Matthieu Rivain and Emmanuel Prouff. 2010. Provably Secure Higher-Order Masking of AES. In *Proceedings of the 12th International Workshop on Cryptographic Hardware and Embedded Systems*. 413–427. https://doi.org/10.1007/978-3-642-15031-9_28

Thomas Schamberger, Julian Renner, Georg Sigl, and Antonia Wachter-Zeh. 2020. A Power Side-Channel Attack on the CCA2-Secure HQC KEM. In *Proceedings of the 19th International Conference on Smart Card Research and Advanced Applications*. 119–134. https://doi.org/10.1007/978-3-030-68487-7_8

Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (1979), 612–613. https://doi.org/10.1145/359168.359176

Jingbo Wang, Chungha Sung, and Chao Wang. 2019. Mitigating power side channels during compilation. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 590–601. https://doi.org/10.1145/3338906.3338913

Weijia Wang, Chun Guo, François-Xavier Standaert, Yu Yu, and Gaëtan Cassiers. 2020. Packed Multiplication: How to Amortize the Cost of Side-Channel Masking?. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*. 851–880. https://doi.org/10.1007/978-3-030-64837-4_28

Weijia Wang, Yu Yu, François-Xavier Standaert, Junrong Liu, Zheng Guo, and Dawu Gu. 2018. Ridge-Based DPA: Improvement of Differential Power Analysis For Nanoscale Chips. *IEEE Trans. Information Forensics and Security* 13, 5 (2018), 1301–1316. https://doi.org/10.1109/TIFS.2017.2787985

Jun Zhang, Pengfei Gao, Fu Song, and Chao Wang. 2018. SCInfer: Refinement-Based Verification of Software Countermeasures Against Side-Channel Attacks. In *Proceedings of the 30th International Conference on Computer Aided Verification*. 157–177. https://doi.org/10.1007/978-3-319-96142-2_12

Received 2023-04-14; accepted 2023-08-27