eVer: Universal and Automated Verification of Side-Channel Security for Additive, Inner Product, Polynomial and General Code-Based Masking

Marc Gourjon
marc.gourjon@mpi-sp.org
Max Planck Institute for Security and Privacy (MPI-SP)
Bochum, North Rhine-Westphalia, Germany

Maximilian Orlt
maximilian.orlt@uclouvain.be
Université catholique de Louvain
Louvain-la-Neuve, Walloon Brabant, Belgium

Pajam Pauls
p.pauls@uni-luebeck.de
University of Luebeck
Lübeck, Schleswig-Holstein, Germany

Alexander Treff
a.treff@uni-luebeck.de
University of Luebeck
Lübeck, Schleswig-Holstein, Germany

Abstract

Automated verification of side-channel security is essential as countermeasures and protected schemes grow in complexity, and as developers increasingly rely on LLMs to generate security-critical code. In these settings, independent verification serves as a dependable security check, assuring the absence of security vulnerabilities.

However, existing verification tools support only a narrow class of masking countermeasures due to fundamental technical limitations. For example, current tools fail to verify many instances of masking countermeasures, as well as entire classes of polynomial, inner-product, or code-based masking schemes in general. These practical countermeasures and schemes are particularly interesting as many of them offer stronger resistance to practical side-channel attacks, yet they remain out of reach of formal verification.

We close this gap with a sound, field-agnostic verification approach that, for the first time, automatically checks standard side-channel security notions for *all* common masking schemes, including polynomial, inner-product, and all variants of code-based masking. In addition to addressing the inherently distinct algebraic structure of these schemes, our method relies on a new proof rule within a general proof system for establishing the joint probabilistic independence of expressions. We implement our approach in the tool eVer and evaluate it on masked algorithms previously beyond the reach of automated verification, including BGW, LaOla, inner-product, and general code-based-masked multiplication algorithms.

Keywords

formal verification, language-based security, side-channel, masking, probabilistic equivalence, threshold probing security

1 Introduction

The strong security of cryptographic schemes can be broken in practice by launching physical side-channel attacks on concrete implementations. Power side-channel attacks (SCAs) conduct measurements of the power consumption or electromagnetic radiation of devices to infer information on intermediate computation results. But these intermediates are unknown to adversaries in many cases and knowledge about them is not considered in security proofs of cryptographic schemes. Thereby, power-based SCAs enable practical attacks reconstructing cryptographic keys to eavesdrop or tamper with otherwise secure connections in real-world settings.

The widely-used *masking* countermeasure provides strong security against SCAs by computing on randomized encodings of secrets instead of plain secret values. Commonly, algorithms securely computing on these encodings are called *gadgets*. Designing or implementing secure gadgets is an error-prone task with many pitfalls both at the algorithmic and the implementation level [4, 19, 28]. These pitfalls hinder the intense research on devising and improving gadgets, such as in [2, 12, 15, 21].

Automated verification tools allow to verify the security of gadgets w.r.t. established formal security notions such as t-Probing Security (t-PS) [25] and the more recent t-(Strong) Non-Interference (t-(S)NI) [7, 8]. Besides confirming (or rejecting) pen-and-paper proofs, verification also facilitates secure implementations and permits significant optimizations or more fine-grained security in practice [10]. The automation enables rapid design iterations, instead of iteratively adapting gadgets and their pen-and-paper proofs, or relying on practical security evaluations [15].

Verification of the mentioned probabilistic security properties is achieved by dedicated tools based on *language-based rewriting*, model counting, or satisfiability modulo theories (SMT). The latter two approaches are inherently limited in scalability w.r.t. gadget size and security orders. Language-based rewriting scales significantly better but suffers from limited applicability to gadgets as the technique is based on proof systems with incomplete *deduction* and *term rewrite rules*. These allow rewriting a term, i.e., an algebraic expression, into equivalently distributed terms [9]. Consider, for example, the expression $s + r$, where s is a secret variable, and r is an independently and uniformly distributed (i.u.d.) random variable. The widely-used one-time pad (OTP) rule allows deducing that this expression is identically distributed to r, which is independent of the secret value s. Since both expressions are identically distributed, it follows that $s + r$ is independent of the secret value s too. This reasoning is agnostic to the term's underlying field, which is in contrast to approaches relying on model counting or SMT and explains the superior scalability of language-based approaches.

This is the long version with additional detail and re-ordered content.

1.1 Problem Statement

The state-of-the-art rules for determining equivalence of distributions in existing tools are insufficient to analyze the security of inner product masking (IPM) [4, 5], polynomial masking (PM) [2, 3, 12, 21, 23], and general code-based masking (GCM) [16, 17, 27, 34]. This makes all language-based tools fail in verifying these schemes. But these schemes enjoy a better practical security compared to additive masking, which is due to an encoding of secrets with higher algebraic complexity [5, 18, 27, 35]. Interestingly, the more complex encoding, which is causal for this security amplification, is also the reason for verification to fail.

IPM, PM, and the GCM variants, have a higher algebraic complexity than additive masking [25, 31] because of two differences. First, they are parametric in some variables, meaning that encodings and gadgets involve, besides secret, public, and random variables, another type of variables which we call parameters. During implementation of gadgets, the parameter variables are substituted by concrete values; however, the crucial difference is that not all choices of parameter values are valid. In particular, the schemes involve increasingly complex constraints, ranging from non-zeroness in IPM and distinctness in PM to maximum distance separable matrix conditions in GCM. Moreover, masked algorithms leave parameters abstract and provide symbolic proofs for security claims that universally hold for all valid choices of parameters.

Second, in contrast to additive masking, the encodings and gadgets of these schemes involve multiplicative inverses, operators which are not supported by existing tools because they possess the property that t^{-1} is a bijection in term t if and only if (iff) t has no roots. This requires an additional step of reasoning that is not supported and nontrivial when parameter variables involve constraints.

Consider as example term $e = s + \alpha r_1 + \beta r_2$ in some field with α, β parameters and r_i i.u.d. randoms. Term e is not a bijection in either r_i since parameters α, β may become zero. However, under constraint $\alpha \neq \beta$ it is evident that e is always bijective in at least one r_i. Consequently, e can be rewritten for any valid choice of parameters and is independent of secret s for any valid choice of parameters. Existing tools do not support this reasoning.

The inherent scientific goal is to prove security exhaustively for all valid parameter choices. This requires accounting for scheme-specific parameter constraints, since gadgets with higher algebraic complexity can be insecure for invalid parameter choices. Moreover, existing verification approaches are already costly for a single gadget without symbolic parameters. Brute-forcing re-verification across up to q^n parameter choices, with q the size of the underlying field, is exponentially expensive and quickly intractable. Consequently, systematic methods that verify these gadgets' security for all parameter choices without case distinctions are lacking.

1.2 Contributions

We propose a universal and sound verification approach and a tool for verifying the security of a large variety of gadgets in inner product (IPM), polynomial (PM), and general code-based masking (GCM), for which all existing tools fail. Our main contributions are:

Novel proof system. We define a general proof system for establishing joint probabilistic independence of systems of expressions that, for the first time, supports expressions involving multiplicative inverses. We define a novel derivation rule Perm that generalizes the OTP rule by substituting multiple terms by random variables simultaneously. Moreover, the rule addresses issues arising from the intricate constraints on the parameters used in IPM, PM, and GCM, without mandating case distinction. This is central to enable verification of security notions exhaustively for all valid parameters, which is the inherent intention when designing gadgets. The proof system generalizes to Boolean and arithmetic masking in arbitrary rings, without bit-blasting.

Universal tool. We present the first automated verification tool that is capable of automatically checking both security and functional correctness universally for all masking schemes. Notably, it exhaustively verifies security notions for all valid choices of parameter variables without case distinctions, and has a runtime that is agnostic of the field size. We develop term rewrites that check the conditions for the derivation rules using Gröbner bases if needed.

Case study. We show that our tool correctly verifies a broad set of masked algorithms previously out of reach, including the BGW, the LaOla, two GCM, an IPM multiplication, addition, refresh and AES S-box gadgets. Further, we evaluate and discuss the scalability and performance of our approach.

Impact. We emphasize that our work answers the questions of which proof rules are needed, respectively missing, to formally verify all masking countermeasures: we provide the missing Perm rule, justify its necessity, and validate that our proof system is sufficient. The open question and missing rule have so far blocked verification of concrete implementations and more advanced security notions like random probing security for IPM, PM, and GCM [10, 11]. By enabling both research directions, our work also enables formally verifying their better practical security.

Using our tool eVer, we provide the first independent verification of the security claims of IPM, PM, and GCM gadgets, which were so far backed by handwritten proofs only. We provide an adapted LaOla multiplication that is generalized to arbitrary parameters, whereas the original algorithm is defined and proven only for even security parameters n, d. In addition, we optimize out a superfluous sequence of refresh applications to improve performance and randomness consumption and re-verify security and correctness with eVer. Finally, we use eVer to optimize the costly refresh gadgets for PM and IPM with $d = 2$, thereby saving significant computation and randomness cost.

Open Source. We publish eVer and re-usable implementations of all 37 gadgets[1].

2 A Generic Approach for Proving Probabilistic Equivalence and Independence

In this section, we introduce our proof system and prove its soundness. Proof systems are fundamental in the automated verification of gadgets. They consist of a set of formal derivation rules which enable to make formal judgments on the probability distribution and independence of terms. Our proof system permits two judgments: the judgment $\vdash \vec{e} \equiv \vec{e}'$ states that the vectors of terms $\vec{e}, \vec{e}'$ have point-wise equivalent joint probability distributions, whereas

[1] https://github.com/leakage-resilience/eVer/

$\vdash \vec{e} \perp \mathcal{Z}$ states that the joint probability distribution of the terms $\vec{e}$ is independent of a set of variables $\mathcal{Z}$. These statements enable to formally check whether side-channel observations made by adversaries are independent of secrets by representing these observations as terms and chaining the derivation rules of the proof system until independence is achieved or no more rules can be applied. Such independence judgments are at the heart of all common security notions in the various threshold and random probing models [11]. Hence, our automated verification tool implements an automated decision procedure for various security notions based on the derivation rules of our proof system. Section 3 details the implementation and several steps, such as generating the set of leakages of a gadget.

We start by introducing the formal structure for representing leakage of the quite diverse masking schemes in Section 2.1, which is fundamental for the proof system. Proofs of soundness for selected derivation rules appear in Section 2.2; all rules are stated in Figure 1.

2.1 Formal Representation of Leakage

We introduce a domain specific language (DSL) for representing computation steps and leakage of gadgets. The DSL is associated with two formal structures to enable practical verification: a *term algebra* $T(\Sigma, \mathcal{V})$ and a polynomial ring $\mathbb{P}$ with a special structure.

Our representation follows a language-based approach, whereas cryptographers model gadgets as circuits where nodes represent operations and variables, and wires represent intermediate values. Our equivalent formal representation is based on *term algebras*, where terms, say mathematical expressions, are built from variables and operations. In all settings, side-channel leakage is modeled by functions taking one or more (sub)terms, respectively, circuit edges as arguments. W.l.o.g. we focus on single-sorted signatures and the widely-used threshold probing model where adversaries may observe a bounded number of intermediate values, i.e., terms computed within a gadget.

2.1.1 Term Algebra. We start by defining the term algebra and defer the polynomial representation $\mathbb{P}$ to Section 2.1.3. It is parametric in an underlying nonzero commutative structure $\mathbb{K}$ that is either a finite field if multiplicative inverses are used, or a finite ring otherwise. The set of terms $T(\Sigma, \mathcal{V})$ is defined inductively by the following DSL in Backus-Naur form (BNF) grammar:

$$t ::= c \mid v \mid o(t_1, \ldots, t_{n_o})$$

where $c \in \mathbb{K}$ represents constants from $\mathbb{K}$, $v \in \mathcal{V}$ represents variables from the variables $\mathcal{V}$ defined as the union of the disjoint sets of secret variables $\mathcal{V}_H$, e.g., shares and encoded secrets, i.u.d. random variables $\mathcal{V}_\$$, public variables $\mathcal{V}_L$, and parameters $\mathcal{V}_C$, and $o \in \Sigma$ represents operations with $o : T(\Sigma, \mathcal{V})^{n_o} \to T(\Sigma, \mathcal{V})$ where n_o is the arity of o.

First, we define a subset $\Sigma^\times$ of the full signature Σ: We require that signature $\Sigma^\times$ contains exactly: addition $+ : T(\Sigma, \mathcal{V})^2 \to T(\Sigma, \mathcal{V})$, multiplication $\cdot : T(\Sigma, \mathcal{V})^2 \to T(\Sigma, \mathcal{V})$, additive inverse $- : T(\Sigma, \mathcal{V}) \to T(\Sigma, \mathcal{V})$, and any operations derivable from these, e.g., subtraction $t_1 - t_2 := t_1 + (-t_2)$. This defines the term algebra $T(\Sigma^\times, \mathcal{V})$ that contains only terms without multiplicative inverses.

The full signature Σ contains in addition to $\Sigma^\times$ the multiplicative inverse $\cdot^{-1} : T(\Sigma^\times, \mathcal{V}_C) \to T(\Sigma, \mathcal{V}_C)$ on nonzero terms that contain no multiplicative inverses and parameter variables only. Clearly, $T(\Sigma^\times, \mathcal{V}) \subset T(\Sigma, \mathcal{V})$. We denote the least set of variables of term $t \in T(\Sigma, \mathcal{V})$ by $\mathsf{vars}(t) := \bigcap_{\mathcal{V}' \subseteq \mathcal{V}} \{\mathcal{V}' \mid t \in T(\Sigma, \mathcal{V}')\}$.

The term algebra $T(\Sigma, \mathcal{V})$ covers a vast range of masked algorithms, i.e., all those using the operations of a field and operations that can be represented as such. In contrast, $T(\Sigma^\times, \mathcal{V})$ covers only masked algorithms without multiplicative inverses, but allows underlying structures $\mathbb{K}$ corresponding to rings. We emphasize that our representation therefore includes besides IPM, PM, GCM and additive masking in fields also Boolean masking in $\mathbb{Z}_{2^k}$ as well as arithmetic masking in $\mathbb{Z}_{2^k}$ and $\mathbb{Z}_p$ for prime p. Remark, only masking schemes with higher algebraic complexity such as IPM, PM, and GCM involve multiplicative inverses. All others fall in the subset $T(\Sigma^\times, \mathcal{V})$, which is easier to handle.

2.1.2 Congruence Relation and Canonical Representation. We equip $T(\Sigma, \mathcal{V})$ with a congruence relation $=_\mathbb{K}$ for multiple reasons: The proof system relies on the congruence relation to permit semantic rewrites that are supported by the axioms of the underlying ring $\mathbb{K}$ and constant folding. Further, the relation is essential to define a unique canonical form, which is heavily used in the verification algorithms. Last, it enables proving an isomorphism between term algebra and polynomial ring representation, thereby enabling Gröbner approaches.

We define $=_\mathbb{K}$ as the smallest congruence relation satisfying for all terms $s, t, u \in T(\Sigma, \mathcal{V})$ and constants $a, b \in \mathbb{K}$:

(1) the axioms for addition $+$:
   - $(s + t) + u =_\mathbb{K} s + (t + u)$ (associativity)
   - $s + t =_\mathbb{K} t + s$ (commutativity)
   - $s + 0 =_\mathbb{K} s$ where $0 \in \mathbb{K}$ is the additive identity
   - $s + (-s) =_\mathbb{K} 0$ (additive inverse)
(2) the axioms for multiplication $\cdot$:
   - $(s \cdot t) \cdot u =_\mathbb{K} s \cdot (t \cdot u)$ (associativity)
   - $s \cdot t =_\mathbb{K} t \cdot s$ (commutativity)
   - $s \cdot 1 =_\mathbb{K} s$ where $1 \in \mathbb{K}$ is the multiplicative identity
   - $s \cdot s^{-1} =_\mathbb{K} 1$ for nonzero $s \in T(\Sigma, \mathcal{V}_C)$
(3) distributivity: $s \cdot (t + u) =_\mathbb{K} (s \cdot t) + (s \cdot u)$
(4) constant folding:
   - $a + b =_\mathbb{K} c$ where $c = a +_\mathbb{K} b$
   - $a \cdot b =_\mathbb{K} d$ where $d = a \cdot_\mathbb{K} b$
   - $-a =_\mathbb{K} e$ where $e = -_\mathbb{K} a$
   - $a^{-1} =_\mathbb{K} e$ where $e = a^{-1_\mathbb{K}}$

Notably, it excludes an axiom $s =_\mathbb{K} s^q$ for finite $\mathbb{K}$ of cardinality q. This is necessary for the isomorphism to the polynomial ring, where s and s^q are distinct elements. The quotient algebra $T(\Sigma, \mathcal{V})/=_\mathbb{K}$ forms a commutative ring with operations induced by:

- $[s] +_{/=_\mathbb{K}} [t] = [s + t]$
- $-_{/=_\mathbb{K}} [s] = [-s]$
- $[s] \cdot_{/=_\mathbb{K}} [t] = [s \cdot t]$
- $[s]^{-1_{/=_\mathbb{K}}} = [s^{-1}]$ for nonzero $s \in T(\Sigma, \mathcal{V}_C)$

where $[s] := \{t \in T(\Sigma, \mathcal{V}) \mid s =_\mathbb{K} t\}$ denoting the equivalence class of term s.

Now, we introduce the unique canonical forms for terms. The congruence relation is key to enabling the proofs of uniqueness.

Definition 2.1 (Canonical Form for $T(\Sigma^\times, \mathcal{V})$)**.** Given a fixed ordering of variables, a term $t \in T(\Sigma^\times, \mathcal{V})$ is in canonical form iff

$$t = \sum_i c_i \prod_{v \in \mathcal{V}} v^{e_{i,v}} \quad \text{with} \quad c_i \in \mathbb{K}.$$

Theorem 2.1 (Unique Canonical Representative for $T(\Sigma^\times, \mathcal{V})$)**.** *For all $t \in T(\Sigma^\times, \mathcal{V})$ there exists exactly one congruent term $t' \in [t]$ in canonical form, denoted as* canonical representative *of* $[t]$.

Proof of Theorem 2.1. The proof of existence is synonym to Definition 2.2. For uniqueness, we show that any pair of congruent canonical representatives $t, t' \in T(\Sigma^\times, \mathcal{V})$ is syntactical identical and perform case analysis on the set of variables $\mathcal{V}$. In the case where $\mathcal{V} = \emptyset$ terms have no variables and therefore $t = c$, respectively $t' = c'$, for $c, c' \in \mathbb{K}$. Since congruence is preserved under field operations we have that $t =_\mathbb{K} t' \Leftrightarrow t - t' =_\mathbb{K} 0 =_\mathbb{K} (c - c') \cdot 1$. By the field axioms of $\mathbb{K}$ additive and multiplicative identities are distinct, i.e., $0 \neq 1$, and there are no zero divisors, hence factor $c - c'$ must be zero which is the case iff $c = c' = t = t'$. For terms with variables, we have canonical representatives $t, t' \in T(\Sigma^\times, \mathcal{V})$, i.e., $t = \sum_i c_i \prod_{v \in \mathcal{V}} v^{e_{i,v}}$ and $t' = \sum_i c'_i \prod_{v \in \mathcal{V}} v^{e_{i,v}}$ and need to show that $t = t'$ if $t =_\mathbb{K} t'$. These terms consist of monomials with unique exponents $e_{i,v}$ and hence congruence $t =_\mathbb{K} t'$ implies for each k^{th} monomial that the coefficients are congruent $\forall k : c_k =_\mathbb{K} c'_k$. As in the former case, this implies $\forall k : (c_k - c'_k) \cdot 1 =_\mathbb{K} 0$ and for the same arguments this implies that $\forall k : c_k = c'_k$. Hence, the canonical representation of terms in $T(\Sigma^\times, \mathcal{V})$ exists and is unique. □

Definition 2.2 (Canonical Form for $T(\Sigma, \mathcal{V})$)**.** Given a fixed ordering of variables and a notion of a leading coefficient lc induced by this ordering, a term $t \in T(\Sigma, \mathcal{V})$ is in canonical form iff

$$t = \sum_i \frac{n_i}{d_i} \prod_{v \in \mathcal{V} \setminus \mathcal{V}_C} v^{e_{i,v}} \quad \text{with} \quad n_i, d_i \in T(\Sigma^\times, \mathcal{V}_C)$$

where $\forall i$ terms n_i, d_i are in canonical form and $\gcd(n_i, d_i) = 1$ and $\mathsf{lc}(d_i) = 1$.

Theorem 2.2 (Unique Canonical Representative for $T(\Sigma, \mathcal{V})$)**.** *For all $t \in T(\Sigma, \mathcal{V})$ there exists exactly one congruent term $t' \in [t]$ in canonical form, denoted as* canonical representative *of* $[t]$.

Proof of Theorem 2.2. Existence: We show by structural induction on term t that it can be rewritten to canonical form by the axioms defining congruence $=_\mathbb{K}$. The base cases for constants and variables are already in canonical form. The case for $t = -t_1$ follows by distributivity of -1 on the induction hypothesis that a canonical representative exists for t_1. For $t = (t_1)^{-1}$ we have $0 \neq t_1 \in T(\Sigma^\times, \mathcal{V}_C)$ and by induction hypothesis t_1 has a canonical representative t'_1. Since $t_1 \neq 0$ the leading coefficient $\mathsf{lc}(t'_1)$ is nonzero and has a unique inverse. Hence, the canonical representative is the normalized inverse $\frac{(\mathsf{lc}(t'_1))^{-1}}{t'_1 \cdot (\mathsf{lc}(t'_1))^{-1}}$ after applying distributivity. For $t = t_1 + t_2$ we have by induction hypothesis that the canonical representative of t_1 is t'_1, respectively t'_2 for t_2. Then, the canonical representative of t is the reordered sum $t'_1 + t'_2$ using associativity and constant folding. For $t = t_1 \cdot t_2$ we have by induction hypothesis that the canonical representative of t_1 is t'_1, respectively t'_2 for t_2. Then, the canonical representative of t is the reordered product $t'_1 \cdot t'_2$ using distributivity, associativity and constant folding.

Uniqueness: We show $t =_\mathbb{K} t' \Rightarrow t = t'$ for any pair of canonical representatives $t, t' \in T(\Sigma, \mathcal{V})$. Since monomials have unique exponents $t =_\mathbb{K} t'$ implies that the coefficients of each monomial are congruent, i.e., $\forall k : \frac{n_i}{d_i} =_\mathbb{K} \frac{n'_i}{d'_i}$. We show that $\frac{n_i}{d_i} =_\mathbb{K} \frac{n'_i}{d'_i} \Rightarrow n_i = n'_i \wedge d_i = d'_i$ for any i which implies the desired conclusion $t = t'$. The terms n_i, n'_i, d_i, d'_i are canonical terms in $T(\Sigma^\times, \mathcal{V}_C)$ and since t, t' are canonical we have $\gcd(n_i, d_i) = 1 = \gcd(n'_i, d'_i)$ and $\mathsf{lc}(d_i) = 1 = \mathsf{lc}(d'_i)$. Assume that d_i, d'_i have unique factorizations. Then, from $\frac{n_i}{d_i} = \frac{n'_i}{d'_i}$ follows $n_i \cdot d'_i = n'_i \cdot d_i$ and because $\gcd(n_i, d_i) = 1$ every irreducible factor of d_i is coprime to n_i, likewise for d'_i, n'_i. Hence, d_i and d'_i are equal up to a factor and since the leading coefficient is 1 they are identical. The assumption that d_i, d'_i have unique factorizations follows from the isomorphism between $T(\Sigma^\times, \mathcal{V})$ and the unique factorization domain $\mathbb{K}[\mathcal{V}]$, proven in Theorem 2.3. Then, $n_i = n'_i$ follows from the fact that they are in canonical form and this form is unique for $T(\Sigma^\times, \mathcal{V})$. Hence, the canonical representation of terms in $T(\Sigma, \mathcal{V})$ exists and is unique. □

2.1.3 Isomorphic Polynomial Representation. We introduce an equivalent polynomial representation $\mathbb{P}$. This representation is essential for our verification approach, which relies on Gröbner bases, which are not defined for $T(\Sigma, \mathcal{V})$. Hence, we switch between the representations whenever needed.

Such a representation is standard for additive, Boolean and arithmetic masking where the multivariate polynomial ring $\mathbb{K}[\mathcal{V}]$ in the variables $\mathcal{V}$ fully represents $T(\Sigma^\times, \mathcal{V})$ as the used operations directly correspond to operations of the underlying ring $\mathbb{K}$ and no multiplicative inverses occur. However, a more involved structure is necessary for representing terms with multiplicative inverses, as occurring in IPM, PM, and GCM. In these cases, our construction of $\mathbb{P}$ consists of a multivariate polynomial ring on variables $\mathcal{V} \setminus \mathcal{V}_C$ where the coefficients are fractions of polynomials, i.e., rational functions, in the parameters $\mathcal{V}_C$. Before going into the details we prove an isomorphism between $T(\Sigma^\times, \mathcal{V})$ and $\mathbb{K}[\mathcal{V}]$.

Theorem 2.3 (Polynomial Representation for $T(\Sigma^\times, \mathcal{V})$)**.** *For a nonzero commutative ring $\mathbb{K}$ there exists a ring isomorphism:*

$$\hat{\phi} : T(\Sigma^\times, \mathcal{V})/=_\mathbb{K} \xrightarrow{\cong} \mathbb{K}[\mathcal{V}]$$

with $\hat{\phi}([t]) := \phi(t)$ and $\phi(t)$ substituting operations in the signature Σ by ring operations and variables by generators.

Proof. Observe that the canonical representation of terms syntactically coincides with the unique monomial expansion in polynomial representation, and hence applying ϕ on a canonical term results in a polynomial in monomial form. Then, well-definedness of ϕ follows from the fact that every term has a unique canonical representation. Surjectivity and injectivity follow immediately since ϕ is a bijective substitution. □

Observe that $\mathbb{K}[\mathcal{V}_C]$ is an integral and unique factorization domain if the underlying structure $\mathbb{K}$ is a field, as is the case for IPM, PM, and GCM. Then, $\mathbb{K}(\mathcal{V}_C)$ denotes the field of fractions of $\mathbb{K}[\mathcal{V}_C]$, i.e., the rational functions $p = \frac{n}{d}$ for polynomials $n, d \in \mathbb{K}[\mathcal{V}_C]$ and nonzero d. Whenever multiplicative inverses occur, we define $\mathbb{P} := \mathbb{K}(\mathcal{V}_C)[\mathcal{V} \setminus \mathcal{V}_C]$, which denotes the multivariate polynomial ring over all variables which are not parameters. The coefficients are then rational functions in $\mathbb{K}(\mathcal{V}_C)$, involving no other variables

than parameters. Remark, this representation is analogous to a circuit where nodes representing multiplicative inverses take as input sub-circuits that involve only parameter-variables in $\mathcal{V}_C$.

The monomial expansion in $\mathbb{K}(\mathcal{V}_C)[\mathcal{V} \setminus \mathcal{V}_C]$ is a unique canonical normal form for a fixed order on the monomials and $\mathcal{V}$, It is defined as $p = \sum_i c_i \prod_{v \in \mathcal{V} \setminus \mathcal{V}_C} v^{e_{i,v}}$ with coefficients c_i exhibiting likewise a canonical normal form, i.e., for $c_i = \frac{n}{d} \in \mathbb{K}(\mathcal{V}_C)$ for $n, d \in \mathbb{K}[\mathcal{V}_C]$, $\gcd(n, d) = 1$ and monic d. n and d are unique since $\mathbb{K}[\mathcal{V}_C]$ is a unique factorization domain.

Why not consider the rational functions in all variables $\mathbb{K}(\mathcal{V})$ instead of our more complicated two-layered structure? The reason is that this would on the one hand be too general since no existing gadget involves terms with multiplicative inverses in variables other than parameters. On the other hand, it would needlessly render prior art on the analysis in polynomial representation, especially Gröbner bases, inapplicable. In this sense, our term algebra $T(\Sigma, \mathcal{V})$ precisely captures the whole variety of gadgets, and its equivalence to our two-layered polynomial structure $\mathbb{P}$ enables to re-use existing reasoning techniques by adapting them to work with coefficients that are rational functions.

There remains a striking problem: Boolean and arithmetic masking operate in ring structures that are not necessarily integral domains and therefore do not meet the requirements to establish a fraction field. However, then there are no multiplicative inverses, allowing to define $\mathbb{P} := \mathbb{K}[\mathcal{V} \setminus \mathcal{V}_C]$ instead[2]. Note, this definition is not a subring of the former since $\mathbb{K}$ is then not an integral domain.

We prove an isomorphism between the term algebra $T(\Sigma, \mathcal{V})$ and our towered polynomial ring construction $\mathbb{P} := \mathbb{K}(\mathcal{V}_C)[\mathcal{V} \setminus \mathcal{V}_C]$. The isomorphism justifies that verification can be performed equally on both structures. Specifically, two terms are equivalently distributed iff their polynomial representations are equivalently distributed. We emphasize that the term algebra remains necessary since the Destr rule is implemented syntax oriented, i.e., based on the order of operations in a gadget, which would be lost in polynomial representation. Also, our structural assumption, specifically the fraction field in parameter variables only, is not intuitively reflected as (polynomial) circuit.

Theorem 2.4 (Polynomial Representation for $T(\Sigma, \mathcal{V})$)**.** *For a field $\mathbb{K}$ there exists a ring isomorphism:*

$$\hat{\phi} : T(\Sigma, \mathcal{V})/=_{\mathbb{K}} \xrightarrow{\cong} \mathbb{K}(\mathcal{V}_C)[\mathcal{V} \setminus \mathcal{V}_C]$$

with $\hat{\phi}([t]) := \phi(t)$ and $\phi(t)$ substituting operations in the signature Σ by field operations and variables by generators.

The proof is synonymous with Theorem 2.3.

2.2 Proof System

We are now ready to introduce our proof system, which operates on terms $T(\Sigma, \mathcal{V})$. We must convey that all derivations based on its rules are sound, i.e., the judgments are indeed valid. Another point we want to clarify is the need for our new rule Perm, which we explain and exemplify. Figure 1 lists the derivation rules. Substitution $\vec{e}\,[\vec{e}' \mapsto \bot]$ denotes substitution of $\vec{e}'_i \in \vec{e}'$ by unique symbols $\bot_i$ that do not occur in $\vec{e}$ or $\vec{e}'$. We define the evaluation of terms,

$$\frac{}{\vdash \vec{e} \equiv \vec{e}}\ \textsc{Refl} \qquad \frac{\vdash \vec{e} \equiv \vec{e}'}{\vdash \vec{e}' \equiv \vec{e}}\ \textsc{Sym}$$

$$\frac{\vdash \vec{e} \equiv \vec{e}' \qquad \vdash \vec{e}' \equiv \vec{e}''}{\vdash \vec{e} \equiv \vec{e}''}\ \textsc{Trans} \qquad \frac{\vec{e} =_{\mathbb{K}} \vec{e}'}{\vdash \vec{e} \equiv \vec{e}'}\ \textsc{Con}$$

$$\frac{\mathcal{Z} \subseteq \mathcal{V}_H \qquad \mathsf{vars}(\vec{e}) \cap \mathcal{Z} = \emptyset}{\vdash \vec{e} \perp \mathcal{Z}}\ \textsc{Indep}$$

$$\frac{\vdash \vec{e} \equiv \vec{e}' \qquad \vdash \vec{e}' \perp \mathcal{Z}}{\vdash \vec{e} \perp \mathcal{Z}}\ \textsc{iTrans}$$

$$\frac{\mathcal{Z} \subseteq \mathcal{V}_H \qquad n = |\vec{e}| \qquad i \in [n] \qquad \vec{e}_i =_{\mathbb{K}} o\left(e_0, \ldots, e_{n_o}\right) \qquad \vdash (\vec{e}_0, \ldots, \vec{e}_{i-1}, \vec{e}_{i+1}, \vec{e}_{n-1}, e_0, \ldots, e_{n_o}) \perp \mathcal{Z}}{\vdash \vec{e} \perp \mathcal{Z}}\ \textsc{Destr}$$

$$\frac{\mathcal{Z} \subseteq \mathcal{V}_H \qquad n = |\vec{e}| \qquad 0 \le i < j < n \qquad (\vec{e}_i =_{\mathbb{K}} \vec{e}_j) \vee (\mathsf{vars}(\vec{e}_j) \setminus (\mathcal{V}_P \cup \mathcal{V}_C) = \emptyset) \qquad \vdash (\vec{e}_0, \ldots, \vec{e}_{j-1}, \vec{e}_{j+1}, \ldots, \vec{e}_{n-1}) \perp \mathcal{Z}}{\vdash \vec{e} \perp \mathcal{Z}}\ \textsc{Dedup}$$

$$\frac{r \in \mathcal{V}_\$ \qquad e' \in T(\Sigma, \mathcal{V}) \qquad \mathsf{perm}(e', \{r\})}{\vdash \vec{e} \equiv \vec{e}\,[r \mapsto e']}\ \textsc{eOTP}$$

$$\frac{\vec{e}' \in (T(\Sigma, \mathcal{V}))^n \qquad \vec{r} \in (\mathcal{V}_\$)^m \qquad m = |\mathsf{vars}(\vec{r})| \qquad m \ge n \qquad \mathsf{perm}(\vec{e}', \mathsf{vars}(\vec{r})) \qquad \forall i \in [n] : \vec{r}_i \notin \mathsf{vars}(\vec{e}\,[\vec{e}' \mapsto \bot])}{\vdash \vec{e} \equiv \vec{e}[\vec{e}' \mapsto (\vec{r}_i)_{i \in [n]}]}\ \textsc{Perm}$$

$$\frac{\vec{e} \in (T(\Sigma, \mathcal{V}))^n \qquad \mathcal{R} \subseteq \mathcal{V} \setminus \mathcal{V}_C \qquad |\mathcal{R}| \ge n \qquad \forall (\sigma_C : \mathcal{V}_C \to \mathbb{K}) : \exists \mathcal{R}' \subseteq \mathcal{R}, |\mathcal{R}'| = n : \forall \vec{y} \in \mathbb{K}^n, (\sigma : \mathcal{V} \setminus (\mathcal{V}_C \cup \mathcal{R}') \to \mathbb{K}) : \exists (\sigma_{\mathcal{R}'} : \mathcal{R}' \to \mathbb{K}) : \vec{y} =_{\mathbb{K}} [\![\vec{e}]\!]_{\sigma, \sigma_C, \sigma_{\mathcal{R}'}}}{\mathsf{perm}(\vec{e}, \mathcal{R})}$$

Figure 1: Proof system with perm **for finite fields $\mathbb{K}$.**

denoted $[\![\vec{e}]\!]_{\sigma_1, \sigma_1, \ldots}$ under substitutions $\sigma_i : \mathcal{V}_i \to \mathbb{K}$, which map disjoint sets of variables $\mathcal{V}_i \subseteq \mathcal{V}$ to values, as substitution followed by application of the axioms for congruence $\cdot =_{\mathbb{K}} \cdot$.

Main drivers of our approach are rules for syntactic independence Indep, and the rules for probabilistic term rewriting OTP, eOTP, and Perm. We omit description and proofs for standard rules such as reflexivity Refl, symmetry Sym, transitivityTrans, iTrans and congruence Con. Besides these, there are two more rules: Dedup allows to derive independence of a vector of terms $\vec{t}$ whenever the vector without duplicate components or without a term depending on parameter or public variables only, is independent. In contrast, Destr is essential in verifying masking as it allows to destructure an operator and verify independence for the vector augmented by the operator's operands instead. The rule's soundness is clear: if the distribution of terms is independent of

[2]For this reason we distinguish parameters from public variables and require that variable $v \in \mathcal{V}_C$ iff there exists a term t^{-1} s.t. v occurs in t.

some variables, then the result of any function applied to these terms remains independent. We continue with the probabilistic rewrites eOTP and Perm.

2.2.1 Expanding OTP. The expanding OTP rule is an existing generalization of the renowned OTP proven by Shannon. The core concept of the OTP rules is that any function $f(\vec{e}')$ that is bijective in some argument $\vec{e}'_i$ simply permutes the possible values of the argument. If argument $\vec{e}'_i$ is bound to a random variable $r_j \in \mathcal{V}_\$$ and r_j does not occur in other arguments, then the argument's inputs are uniformly distributed and the corresponding outputs, by bijectivity, also occur with equal probabilities, i.e., are uniformly distributed. In the original OTP rule, the term representing function $f(\vec{e}')$ thus can be replaced by the random variable r_j if the variable does not occur elsewhere in the terms. The eOTP drops the requirement that random variable r_j must not occur elsewhere by instead rewriting a random variable globally by an expression e' that must be a bijection in the random variable, as formally stated in Figure 1 using the more general $\text{perm}(e', \{r\})$ property detailed in Section 2.2.2. The expanding OTP rule appears in variants in the tool MaskVerif [6], a generalization named Bij, is presented in [9].

We demonstrate the strength of eOTP over OTP. Consider the terms $\vec{e} = (r_0 + r_1, s + r_0 + 2r_1)$ where $r_i \in \mathcal{V}_\$$ are random variables and $s \in \mathcal{V}_H$ is secret. Here, the standard OTP rule does not apply, as each component depends on both random variables. However, eOTP can be applied sequentially by first rewriting $r_0 \mapsto r_0 - r_1$, which is a bijection, yielding $(r_0, s + r_0 + r_1)$. Subsequently, rewriting $r_1 \mapsto r_1 - r_0 - s$, which is a bijection, yields $\vec{e} \equiv (r_0, r_1)$ and proves that $\vec{e}$ is independent of secrets and uniformly distributed. While the eOTP rule is used in our tool, it remains insufficient for verifying GCM, and we introduce rule Perm.

2.2.2 Permute rule. We put forward a new rule Perm to enable a different and broader class of probabilistic rewrites. It generalizes the OTP rule in a different direction than eOTP. The motivation for this generalization arises from the following observation. Consider term $e = s + \alpha \cdot r_0 + \beta \cdot r_1$, where α and β are parameters in $\mathcal{V}_C$ and constraint $\alpha \neq \beta$. At least one coefficient, α or β, must be nonzero due to the constraint. Hence, either r_i is multiplied by a nonzero factor, yielding a bijection, since $T(\Sigma, \mathcal{V})$ is an integral domain because of the isomorphism in Theorem 2.4. It follows immediately that e is uniformly distributed if both r_i are independent and uniformly random. However, the constraints prevent an application of the expanding OTP rule since e is only in specific cases a bijection. To apply eOTP, each case would need to be handled distinctly. This observation becomes particularly relevant for IPM, PM, and GCM, since these schemes are initialized with generic constants and many constraints on parameters. These constants are either fixed across all executions, being initialized prior to the first execution, or are even uniformly randomized for each execution.

The Perm rule in Figure 1 captures these, and many more cases, of probabilistic rewrites. It allows to rewrite n terms $\vec{e}'$ simultaneously by random variables $\vec{r}$, conditioned that for any choice of parameters there exists a subset of n random variables $\vec{r}'$ s.t. $\vec{e}'$ is a permutation, i.e., invertible function, in $\vec{r}'$. The rewrite also requires that the random variables do not occur outside $\vec{e}'$ in the terms $\vec{e}$ where the rewrite is performed. Note, the Perm comes with a smaller search space for verification compared to eOTP and Bij: in Perm, candidate terms $\vec{e}'$ are subexpressions in $\vec{e}$, whereas for the others the entire set of term $T(\Sigma, \mathcal{V})$ are candidates while only few terms yield desirable progress for checking independence. Remark, the initial OTP rule (but not eOTP) is entirely captured by Perm.

Theorem 2.5. *The derivation rule Perm is sound.*

Proof. Let $\vec{X}$ be the vector containing the variables occurring in $\vec{e}$ and $\vec{e}'$ exactly once, excluding the random variables $\vec{r} = (r_i)_{i\in[m]}$, and let $l := |\vec{X}|$. Consequently, there exists a function g such that $\vec{e}' = g\big((r_i)_{i\in[m]}, \vec{X}\big)$. Note that $\vec{X}$ may contain variables that do not actually appear in $\text{vars}(\vec{e}')$; in such cases, the function g can simply ignore those inputs. Due to $\vec{r}_i \notin \text{vars}(\vec{e}\,[\vec{e}' \mapsto \bot])$, there exists a function f such that $\vec{e} = f\big(g((r_i)_{i\in[m]}, \vec{X}), \vec{X}\big)$. We make two preliminary observations: (1) The function f does not take $(r_i)_{i\in[m]}$ as direct inputs, but only through the term $g((r_i)_{i\in[m]}, \vec{X})$. This holds because we assume that $\forall i \in [m] : \vec{r}_i \notin \text{vars}\big(\vec{e}[\vec{e}' \mapsto \bot]\big)$. (2) It is possible that f receives more inputs than necessary; any arguments from $\vec{e}' = g((r_i)_{i\in[m]}, \vec{X})$ or $\vec{X}$ that are not used by f can be ignored again. Using this notation, we prove Perm.

Step 1: Permutation property of g. From the condition $\text{perm}\big(\vec{e}', \text{vars}((r_i)_{i\in[m]})\big)$ we know that for all $\vec{x} \in \mathbb{K}^l$ there exists an index set $I_{\vec{x}}$ with $|I_{\vec{x}}| = n$ such that $g'_{\vec{x},(r_i)_{i\in[m]\setminus I_{\vec{x}}}}\big((r_i)_{i\in I_{\vec{x}}}\big) := g\big((r_i)_{i\in[m]}, \vec{x}\big)$ is a permutation for any fixed $(r_i)_{i\in[m]\setminus I_{\vec{x}}}$. Hence, $\forall \vec{y} \in \mathbb{K}^n : g'_{\vec{x},(r_i)_{i\in[m]\setminus I_{\vec{x}}}}\big((r_i)_{i\in I_{\vec{x}}}\big) = \vec{y} \iff (r_i)_{i\in I_{\vec{x}}} = g'^{-1}_{\vec{x},(r_i)_{i\in[m]\setminus I_{\vec{x}}}}(\vec{y})$. Since $(r_i)_{i\in I_{\vec{x}}}$ is uniform random it follows $\Pr\left[(r_i)_{i\in I_{\vec{x}}} = g'^{-1}_{\vec{x},(r_i)_{i\in[m]\setminus I_{\vec{x}}}}(\vec{y})\right] = \Pr\left[(r_i)_{i\in I_{\vec{x}}} = y)\right]$, or alternatively

$$\Pr\left[g'_{\vec{x},(r_i)_{i\in[m]\setminus I_{\vec{x}}}}\big((r_i)_{i\in I_{\vec{x}}}\big) = \vec{y}\right] = \Pr\left[(r_i)_{i\in I_{\vec{x}}} = \vec{y}\right].$$

Step 2: Uniform distribution of g.

$$\begin{aligned}
&\forall \vec{y} \in \mathbb{K}^n : \Pr\left[g((r_i)_{i\in[m]}, \vec{X}) = \vec{y}\right] \\
&= \sum\nolimits_{\vec{x}} \Pr\left[g'_{\vec{x},(r_i)_{i\in[m]\setminus I_{\vec{x}}}}\big((r_i)_{i\in I_{\vec{x}}}\big) = \vec{y},\, \vec{X} = \vec{x}\right] \\
&= \sum\nolimits_{\vec{x}} \Pr\left[g'_{\vec{x},(r_i)_{i\in[m]\setminus[n]}}\big((r_i)_{i\in[n]}\big) = \vec{y},\, \vec{X} = \vec{x}\right] \\
&= \sum\nolimits_{\vec{x},\vec{z}} \Pr\left[g'_{\vec{x},\vec{z}}\big((r_i)_{i\in[n]}\big) = \vec{y},\, \vec{X} = \vec{x},\, (r_i)_{i\in[m]\setminus[n]} = \vec{z}\right] \\
&= \sum\nolimits_{\vec{x},\vec{z}} \Pr\left[(r_i)_{i\in[n]} = \vec{y},\, \vec{X} = \vec{x},\, (r_i)_{i\in[m]\setminus[n]} = \vec{z}\right] \\
&= \sum\nolimits_{\vec{x},\vec{z}} \Pr\left[(r_i)_{i\in[n]} = \vec{y}\right] \Pr\left[\vec{X} = \vec{x}\right] \Pr\left[(r_i)_{i\in[m]\setminus[n]} = \vec{z}\right] \\
&= \Pr\left[(r_i)_{i\in[n]} = \vec{y}\right] \sum_{\vec{x},\vec{z}} \Pr\left[\vec{X} = \vec{x}\right] \Pr\left[(r_i)_{i\in[m]\setminus[n]} = \vec{z}\right] \\
&= \Pr\left[(r_i)_{i\in[n]} = \vec{y}\right].
\end{aligned}$$

Step 3: Independence of g and $\vec{X}$.

$$\begin{aligned}
&\forall \vec{x} \in \mathbb{K}^l, \vec{y} \in \mathbb{K}^n : \Pr\left[g((r_i)_{i\in[m]}, \vec{X}) = \vec{y}, \vec{X} = \vec{x}\right] \\
&= \Pr\left[g'_{\vec{x},(r_i)_{i\in[m]\setminus I_{\vec{x}}}}((r_i)_{i\in I_{\vec{x}}}) = \vec{y}, \vec{X} = \vec{x}\right] \\
&= \Pr\left[g'_{\vec{x},(r_i)_{i\in[m]\setminus[n]}}((r_i)_{i\in[n]}) = \vec{y}, \vec{X} = \vec{x}\right] \\
&= \sum\nolimits_{\vec{z}} \Pr\left[g'_{\vec{x},\vec{z}}((r_i)_{i\in[n]}) = \vec{y}, \vec{X} = \vec{x}, (r_i)_{i\in[m]\setminus[n]} = \vec{z}\right] \\
&= \sum\nolimits_{\vec{z}} \Pr\left[(r_i)_{i\in[n]} = \vec{y}, \vec{X} = \vec{x}, (r_i)_{i\in[m]\setminus[n]} = \vec{z}\right] \\
&= \sum\nolimits_{\vec{z}} \Pr\left[(r_i)_{i\in[n]} = \vec{y}\right] \Pr\left[\vec{X} = \vec{x}\right] \Pr\left[(r_i)_{i\in[m]\setminus[n]} = \vec{z}\right] \\
&= \Pr\left[(r_i)_{i\in[n]} = \vec{y}\right] \Pr\left[\vec{X} = \vec{x}\right] \sum_{\vec{z}} \Pr\left[(r_i)_{i\in[m]\setminus[n]} = \vec{z}\right] \\
&= \Pr\left[(r_i)_{i\in[n]} = \vec{y}\right] \Pr\left[\vec{X} = \vec{x}\right].
\end{aligned}$$

Step 4: Equivalence $\vec{e} \equiv \vec{e}\left[\vec{e}' \mapsto (\vec{r}_i)_{i\in[n]}\right]$. It remains to prove $f\big(g((r_i)_{i\in[m]}, \vec{X}), \vec{X}\big) \equiv f\big((r_i)_{i\in[n]}, \vec{X}\big)$:

$$\begin{aligned}
&\Pr\left[f(g((r_i)_{i\in[m]}, \vec{X}), \vec{X}) = \vec{z}\right] \\
&= \sum\nolimits_{\vec{x},\vec{y}} \Pr\left[f(\vec{y}, \vec{x}) = \vec{z},\ g((r_i)_{i\in[m]}, \vec{X}) = \vec{y},\ \vec{X} = \vec{x}\right] \\
&\overset{S3}{=} \sum\nolimits_{\vec{x},\vec{y}} \Pr\left[f(\vec{y}, \vec{x}) = \vec{z}\right] \Pr\left[g((r_i)_{i\in[m]}, \vec{X}) = \vec{y}\right] \Pr\left[\vec{X} = \vec{x}\right] \\
&\overset{S2}{=} \sum\nolimits_{\vec{x},\vec{y}} \Pr\left[f(\vec{y}, \vec{x}) = \vec{z}\right] \Pr\left[(r_i)_{i\in[n]} = \vec{y}\right] \Pr\left[\vec{X} = \vec{x}\right] \\
&= \Pr\left[f((r_i)_{i\in[n]}, \vec{X}) = \vec{z}\right].
\end{aligned}$$

□

3 Verification

The proof system introduced in Section 2 enables to decide whether gadgets fulfill the defined security notions. This section explains how we verify the notions in an automated manner. We achieve this by introducing security predicates based on judgments of the proof systems and algorithms for automated rule application and decision making. In Section 4, the approach is implemented and evaluated on a large set of gadgets.

3.1 Outline of the Verification

We outline the steps of the verification procedure. Users specify gadgets manually in a general-purpose programming language or a DSL. In our case, we implement the quite complex gadgets in Python to make use of existing libraries and ease debugging. However, approaches like scVerif permit to interface verification with concrete Assembler, executable, or low-level implementations in a purposely designed and easy-to-use DSL [10].

Users initiate the security verification by providing the gadget specification, a security notion, and security order t to verify, as well as concrete values for the security parameters k, d, e. Then, an automated symbolic analysis computes the set of observable leakages $\mathcal{L}$ and outputs O. Our analysis implements the widely used *threshold probing model* [25], which produces for each operation $o \in \Sigma$ in the gadget a leak that contains the result of the operation. While this can be easily extended to cover additional models such as the robust probing model [20], we believe that interfacing existing tools that permit user-defined or provably complete leakage models [13], like scVerif, is more general, but left to future work. In contrast to other approaches, our analysis keeps the parameter variables $\mathcal{V}_C$ symbolic to allow our verification to cover all possible assignments of parameters at once. It generates an additional set of constraints C, detailed in Section 3.3.5, that ensure that parameters $\mathcal{V}_C$ can only be assigned valid combinations of values and encodings of inputs are valid. Examples are that weights in IPM must be nonzero, or support points for PM need to be distinct.

The set of leakages $\mathcal{L}$, outputs O is then analyzed using our main verification routine for the desired security notion and parameter $t \le d$ under constraints C, which we present in Section 3.2. Informally, this requires repeatedly applying derivation rules to show that all subsets $\vec{e}$ of cardinality t are independent of certain secret variables or depend on at most t shares per input. Details of the rule implementation are given in Section 3.3.

Finally, the tool displays a success message or the terms $\vec{e}$ violating the security property with additional information to aid debugging. However, the main difficulty besides designing a sound tool that does not return false positives is to avoid false negatives, i.e., reporting insecurities when gadgets are actually secure.

We detail the main steps in the following. Our functional verification, i.e., whether a gadget correctly computes a desired function on the encoded inputs, is presented in Section 3.4.

3.2 Verification of Security Notions

Several security notions allow proving information-theoretical security of gadgets computing on shares. These notions require exhaustively checking a predicate for d-tuples (vectors) of leakages, in- and outputs. We denote this Boolean predicate by $\varphi_{id} : (T(\Sigma, \mathcal{V}))^* \to \{\top, \bot\}$ where identifier id distinguishes the security notion. This predicate is used as a continuation predicate in our main verification algorithm, which repeatedly applies term rewrites until the predicate is satisfied or no more rules can be applied. We define the predicate for three most common notions and introduce them.

Definition 3.1 (t-(S)NI [7, 8])**.** A gadget with masked inputs $\mathcal{I}$, masked outputs O and leakages $\mathcal{L}$ is t-NI if for any choice of parameters any subset $\mathcal{L}' \subseteq \mathcal{L}$ of leakages and any subset $O' \subseteq O$ of outputs with $|\mathcal{L}'| + |O'| \le t$, there exists a subset of input shares $\mathcal{J}$ of at most $|\mathcal{L}'| + |O'|$ shares per encoded input, s.t. the joint distribution of $\mathcal{L}', O'$ is perfectly simulatable from $\mathcal{J}$. Additionally, it is t-SNI if each said subset of input shares $\mathcal{J}$ contains at most $|\mathcal{L}'|$ shares per encoded input.

Definition 3.1 ensures that a t-NI gadget does not leak any information about secrets through combinations of t or fewer leakages, assuming randomized input shares. t-SNI implies a stricter security notion than t-NI, as it distinguishes leakages and output values. The stronger property is particularly important for ensuring composability when protecting larger cryptographic operations: any gadget composed of single-output t-SNI gadgets remains secure and is immediately t-SNI.

The notions require simulatability of terms $\vec{e}$ from secret shares $\mathcal{J} \subseteq \mathcal{V}_H$, public and parameter variables. Assuming that inputs are d-private, this requirement is equivalent to requiring that $\vec{e}$ is independent of $\mathcal{V}' := \mathcal{V}_H \setminus \mathcal{J}$, which corresponds to the judgment $\vdash \vec{e} \bot \mathcal{V}'$. d-privacy requires that every subset of d out of all shares encoding one input of a gadget are independent of the encoded

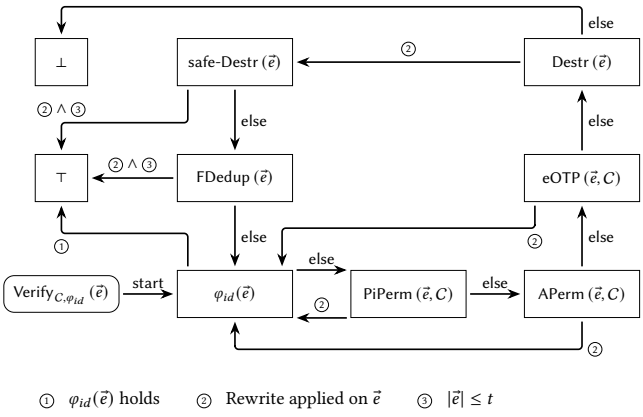


Figure 2: Flowchart for the order in which rewrites are applied in $\mathsf{Verify}_{C,\varphi_{id}}(\vec{e})$ on terms $\vec{e}$ under constraints C for a security predicate φ_{id}.

secrets, i.e., d-wise independence of secrets. Then, the continuation predicate for t-(S)NI is

$$\varphi_{d\text{-NI}}(\vec{e}) := \exists \mathcal{V}' \subseteq \mathcal{V}_H, \wedge_{i\in[k]} |\mathcal{V}' \cap \mathcal{X}_i| \leq d \;:\; \vec{e} \perp \mathcal{V}_H \setminus \mathcal{V}', \quad (1)$$

where $\mathcal{X}_i$ denotes the shares belonging to the i^{th} encoded input of a gadget and d is instantiated for $\vec{e}$ accordingly to Definition 3.1.

The seminal t-PS security notion does not guarantee composability but remains in use. To verify t-PS, it is necessary to consider the unmasked secrets as symbols in $\mathcal{V}_H$ and the shares as terms, whereas for t-(S)NI shares are symbols in $\mathcal{V}_H$. The security predicate conforming to Definition 3.2 is $\varphi_{t\text{-PS}}(\vec{e}) := \vec{e} \perp \mathcal{V}_H$.

Definition 3.2 (t-PS [25])**.** A gadget with masked inputs $\mathcal{I}$, masked outputs $\mathcal{O}$ and leakages $\mathcal{L}$ is *t-probing secure* t-PS if for any choice of parameters each subset $\mathcal{P} \subseteq \mathcal{O} \cup \mathcal{L}$ s.t. $|\mathcal{P}| \leq t$, is independently distributed of the unmasked, secret inputs encoded in $\mathcal{I}$.

Our overall verification routine exhaustively enumerates all tuples (vectors $\vec{e}$) and invokes $\mathsf{Verify}_{C,\varphi_{id}}(\vec{e})$ with the tuple and continuation predicate set accordingly to the security notion to be verified. Initially, we apply FDedup and safe-Destr on the set of all leakages to reduce the enumeration. The rule ordering of the Verify algorithm is visualized in Figure 2. It starts by checking the predicate, iteratively tries to apply one of the rewrite rules, and recursively invokes more rules. The rules are detailed next, and their ordering is crucial for performance and avoiding false negatives.

3.3 Implementation of the Proof System

We describe the algorithms for the rewrites used in Figure 2, which are based on the proof system in Figure 1.

3.3.1 Deduplication. For FDedup, we compute the dependency set of the canonical representation of $\vec{e}_i$ for increasing i. If there is some $\vec{e}_j$ for $j < i$ with the same dependency set, then we first check if $\vec{e}_i =_{\mathbb{K}} \vec{e}_j$ by syntactic comparison of their canonical representations. If this succeeds, we drop $\vec{e}_i$ from $\vec{e}$. Otherwise, we check if there exists a factor $f \in T(\Sigma, \mathcal{V}_C)$ of parameter variables only s.t. $\vec{e}_i =_{\mathbb{K}} f \cdot \vec{e}_j$. This is achieved by computing a candidate as $f = \frac{\text{coeff}_0(\vec{e}_i)}{\text{coeff}_0(\vec{e}_j)}$ where coeff_k yields the coefficient of the k^{th} monomial. If the candidate satisfies $\text{coeff}_k(\vec{e}_i) = f \cdot \text{coeff}_k(\vec{e}_j)$ for all monomials, we drop $\vec{e}_i$ from $\vec{e}$. After completing the procedure for all components, we end verification of the tuple if a component of $\vec{e}$ was dropped.

Clearly, this rewrite relies on the existence and uniqueness of canonical forms proven in Theorem 2.2 and its soundness is backed by Conv and Dedup.

3.3.2 Linear Randomness and Destruct. We adhere to a specific representation to check the perm property as required for the rewrites based on the probabilistic rules eOTP and Perm. Specifically, we consider terms in canonical form, i.e., monomial expansion, and reorder the monomials and coefficients s.t.

$$\vec{e} =_{\mathbb{K}} \vec{s} + \mathbf{A} \cdot \vec{r}$$

where $\vec{r} \in (\mathcal{V}_\$)^{|\vec{e}|}$ consists of distinct random variables and $\text{vars}(\vec{r}) \cap \text{vars}(\vec{s}) = \emptyset$. We refer to $\vec{s} \in (T(\Sigma, \mathcal{V}))^n$ as residue, $\vec{r}$ as the linear randomness consisting of m distinct random variables in $\mathcal{V}_\$$ and $\mathbf{A} \in (T(\Sigma, \mathcal{V}_C))^{n \times m}$ as coefficient matrix.

However, this approach is problematic: checking bijectivity or permute properties on $\mathbf{A}$ restricts analysis to cases where random variables $r \in \vec{r}$ appear only linearly. But a term might be bijective, yet not have linear randoms in this representation, as for example in $(a + r_1) \cdot (br_3 + r_2)$ with secrets a, b and randoms r_1, r_2, r_3. Its canonical form $abr_3 + ar_2 + br_1r_3 + r_1r_2$ has no linear randomness, hence there is no $\mathbf{A}$ with $\vec{e} =_{\mathbb{K}} \vec{s} + \mathbf{A} \cdot \vec{r}$, which prevents application of eOTP or Perm.

In practice we remedy such situations with the Destr and safe-Destr rewrites, which replace some $\vec{e}_i$ in $\vec{e}$ by two or more terms. While the rewrites are based on the general Destr rule, they are more specific: Namely, for these rewrites we consider the original term $\vec{e}_i$ that strictly adheres to the order of computation steps in the gadget, i.e., whenever $\vec{e}_i = o(e_0, \ldots, e_{n_o})$ then, o is the last operation applied to the term and no reordering axioms have been applied. The restriction turns out sufficient in practice, and moreover, it avoids nondeterminism from the general case that considers any operations o and terms $e_0, \ldots, e_{n_o}$ s.t. $\vec{e}_i =_{\mathbb{K}} o(e_0, \ldots, e_{n_o})$. In above example, the rewrites destructures $\vec{e} = ((a + r_1) \cdot (br_3 + r_2))$ as $\vec{e}' = ((a + r_1), (br_3 + r_2))$ with

$$\vec{e}' = \begin{pmatrix} a \\ br_3 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}.$$

But the rewrite may introduce false negatives, as for example the term $a+r_1$ is independent of a, but this can no longer be derived after splitting it, leading to false negatives. Hence, we treat the rewrite as a fall-back and apply it only if all other rewrites fail to apply. However, there is a safe variant: destructuring $\vec{e}_i = o(e_0, \ldots, e_{n_o})$ cannot introduce false negatives when for all but at most one i the operands e_i already appear as other terms $\vec{e}_j \in \vec{e}$. The safe-Destr hence progresses the verification safely, and its results are simplified using Dedup. Note that when safe-Destr or FDedup applications produce a tuple $|\vec{e}'| \leq t$, verification can be deferred since our enumeration-based approach guarantees that the tuple $\vec{e}'$ is checked in another invocation.

We experimented with a rewrite that replaces a term e by terms e_i with disjoint random variables s.t. $e =_{\mathbb{K}} e_0 + \prod_{i>0} e_i$. However,

this required significant computation due to involved factorization and provided no benefit over Destr in all tested gadgets.

3.3.3 Parameter Independent Permute. We now turn to the PiPerm rewrite, which is one of two rules implementing the Perm rule and relies on the linear randomness representation $\vec{e} =_{\mathbb{K}} \vec{s} + \mathbf{A} \cdot \vec{r}$. There are two fundamental properties of this representation: First, whenever $\mathbf{A}$ is invertible, then this implies the permutation property $\mathsf{perm}(\mathbf{A}\vec{r}, \mathsf{vars}(\vec{r}))$ that is required for the probabilistic rewrites. Second, $\mathsf{perm}(\mathbf{A}\vec{r}, \mathsf{vars}(\vec{r}))$ implies $\mathsf{perm}(\vec{e}, \mathsf{vars}(\vec{r}))$ due to the representation's linearity and disjointness of randomness, i.e., $\mathsf{vars}(\vec{r}) \cap \mathsf{vars}(\vec{s}) = \emptyset$.

Hence, we check invertibility of the coefficient matrix $\mathbf{A}$. However, in many cases, there are terms without linear randomness, then $\mathbf{A}$ has linear dependent rows and is not invertible, or $\mathbf{A}$ has more random variables than terms, making it non-square. We solve this problem by computing an unreduced row echelon form $\mathbf{A}'$ of $\mathbf{A}$ s.t. $\vec{e}' = \vec{s}' + \mathbf{A}' \cdot \vec{r}'$ and $\vec{e}', \vec{s}', \vec{r}'$ are permutations of their counterparts (which is due to row swapping). During the computation, we carefully pay attention to choose pivots which have no roots. This is especially important since $\mathbf{A}$ consists, depending on the masking scheme, of terms involving parameter variables which may become zero. In Section 3.3.4 we introduce a predicate $\mathsf{roots}(p, C)$ that checks if a polynomial $p \in \mathbb{K}[\mathcal{V}_C]$ can become zero under constraints C. Note, that due to the isomorphism between $T(\Sigma^{\times}, \mathcal{V})$ and $\mathbb{K}[\mathcal{V}_C]$ in Theorem 2.3 we can represent coefficient-terms as polynomials instead. Equipped with the roots predicate, we soundly compute the echelon form. Then, the upper $n' \leq n$ rows of $\mathbf{A}'$ have nonzero, root-free pivots in columns $\mathcal{M} \subseteq [m]$ and we deduce $\mathsf{perm}(\vec{e}'_{i\in[n']}, \{\vec{r}'_{i\in\mathcal{M}}\})$. However, the Perm rule requires that the random variables do not occur outside of $\vec{e}'_{i\in[n']}$ in $\vec{e}$, which is not apparent in the reduced $\mathbf{A}'$ since linearly dependent rows have been eliminated. To satisfy this, we search for a subset of root-free pivots with rows $\mathcal{N}' \subseteq [n']$ and columns $\mathcal{M}' \subseteq [m]$ s.t. $\mathsf{vars}(\vec{e}_{i\notin\mathcal{N}}) \cap \mathsf{vars}(\vec{r}'_{i\in\mathcal{M}'})$ where $\mathcal{N}$ contains the rows in $\mathbf{A}$ corresponding to the rows of $\mathbf{A}'$ in $\mathcal{N}'$.

The Perm rewrite is a limited implementation of the Perm rule: It rewrites entire expressions in $\vec{e}$, instead of considering subterms, and considers only linear randomness (up to Destr rewrites). This choice is deliberate to avoid the computational complexity of searching and checking candidate subterms in the large space of congruent terms. But it opens the possibility of false negatives since necessary permutes cannot be found. One such instance is discussed in Section 3.3.6 and mitigated by the APerm rewrite. As it relies significantly on constraints, we present the related techniques first.

3.3.4 Checking Absence of Roots under Constraints. The main problem we aim to solve in this subsection is to determine if a polynomial $p \in \mathbb{K}[\mathcal{V}_C]$ has roots, i.e., if there exists an assignment of its variables $\mathcal{V}_C$ that causes it to vanish. Since this depends on the constraints imposed by the respective masking scheme, this problem can be phrased as solving a system of polynomial equations. Fortunately, Gröbner bases, a fundamental tool in computational algebra, are perfectly suited to solving such systems. We denote the computation of a Gröbner basis as the function $\mathrm{G} : \mathbb{K}[\mathcal{V}_C]^* \to \mathbb{K}[\mathcal{V}_C]^*$. Additionally, an important property that we make use of is that if $1 \in \mathrm{G}(\mathcal{P})$, where $\mathcal{P} \subset \mathbb{K}[\mathcal{V}_C]$, then the system of equations that $\mathcal{P}$ represents is inconsistent. Using this, root checking can be formulated as: Is the system of equations requiring p to become zero and the constraints C to hold inconsistent? If this is the case, then we know that p is root free. This judgment extends to elements in our term algebra $T(\Sigma^{\times}, \mathcal{V}_C)$ due to the isomorphism in Theorem 2.3. We check if a polynomial p has roots under constraints C using $\mathrm{roots} : \mathbb{K}[\mathcal{V}_C] \times (\mathbb{K}[\mathcal{V}_C])^* \to \mathbb{B}$

$$\mathrm{roots}(p, C) := 1 \in \mathrm{G}(C \cup \{p\})$$

We minimize the calculations of Gröbner basis by making use of persistent memoization and establishing ground truths for the given masking scheme. Next, we explain how to represent the constraints imposed by masking schemes, s.t. they can be used in the computation of a Gröbner Basis.

3.3.5 Constraints of Masking Schemes. PM, IPM, and GCM impose different constraints on their respective parameter variables $\mathcal{V}_C$. Therefore, analyzing the security of these schemes requires capturing these constraints for sound verification.

All PM variants, require that an encoding has *distinct* support points $(\alpha_i)_{i\in[n+k]}$ s.t. $\forall i \neq j, \alpha_i \neq \alpha_j$, e.g., packed secret sharing [21]. The shamir-secret-sharing-based PM variant [12, 29, 32] introduces an additional constraint $\forall i, \alpha_i \neq 0$. Since these constraints are phrased as inequalities, we apply the Rabinowitsch trick [30] to convert them into polynomial equalities. Given an inequality, e.g., $\alpha \neq 0$, the Rabinowitsch trick introduces an auxiliary variable β to form the equation $\beta\alpha - 1 = 0$. Since $\mathbb{K}[\mathcal{V}]$ is an integral domain, both β and α must be non-zero for the equation to be satisfiable, i.e., zero. Likewise, inequalities can be enforced using Fermat's little theorem in conjunction with Lagrange's theorem, expressed as $\alpha^{q-1} - 1 = 0$, for finite fields with cardinality $q = |\mathbb{K}|$. This approach eliminates the need for auxiliary variables, but results in high-degree polynomials, which proved significantly more computationally expensive in our experiments. For PM the resulting constraints are C_{PM} for packed PM, respectively $C_{s\text{-}PM}$ for the Shamir variant,

$$\begin{aligned} C_{PM} &= \{\beta_i(\alpha_i - \alpha_j) - 1 \mid i, j \in [n] \wedge i \neq j\} \\ C_{s\text{-}PM} &= C_{PM} \cup \{\beta'_i\alpha_i - 1 \mid i \in [n]\}. \end{aligned}$$

For IPM [4, 5] constraints C_{IPM} ensure weights $\vec{\omega}$ are nonzero

$$C_{IPM} = \{\beta_i\vec{\omega}_i - 1 \mid i \in [n]\}.$$

GCM is a more special case. Here, an encoding is defined as $\vec{y} = \vec{x}\mathbf{G} + \vec{r}\mathbf{H}$, where $\vec{x}, \vec{r}$ are vectors of secret and random variables. This is often written as $\vec{y} = [\vec{x}\,\vec{r}]\,\mathbf{A}$, where $\mathbf{A} = [\mathbf{G}; \mathbf{H}]$ is the vertical concatenation of $\mathbf{G}, \mathbf{H}$. The first requirement for GCM is that of a $d-$private encoder, which informally states that any linear combination of at most d shares of an encoding $\vec{y}$ is independent of $\vec{x}$. Let $\mathrm{HW}(\vec{w})$ denote the Hamming-weight, then this is defined as

$$\forall \vec{w} \in \mathbb{F}_q^n, \mathrm{HW}(\vec{w}) \leq d : \mathbf{H}\vec{w}^{\top} = \vec{0} \implies \mathbf{G}\vec{w}^{\top} = \vec{0}.$$

By a series of equivalences, we develop a formula that is amenable to Gröbner-bases. For the first step, observe that the property describes null spaces, e.g., $\mathbf{H}\vec{w}^{\top} = \vec{0}$, but the weight of $\vec{w}$ is bounded. Hence, it is necessary to repeatedly verify that $\mathsf{null}(\mathbf{H}_S) \subseteq \mathsf{null}(\mathbf{G}_S)$ holds, for any selection S of at most d columns such that $\mathbf{H}_S$ and $\mathbf{G}_S$ consist of d columns of $\mathbf{H}$ and $\mathbf{G}$ selected by S.

Second, we use that null and row spaces are orthogonal complements, i.e., $(\text{null}(\mathbf{H}_S) \subseteq \text{null}(\mathbf{G}_S)) \iff (\text{row}(\mathbf{G}_S) \subseteq \text{row}(\mathbf{H}_S))$, where $\text{row}(\mathbf{H}_S)$ denotes all linear combinations of its rows. Hence, we check that each row in $\mathbf{G}_S$ is a linear combination of rows in $\mathbf{H}_S$: $\forall g_{S,i} \in \text{rows}(\mathbf{G}_S) : g_{S,i} = \sum_{h_{S,j} \in \text{rows}(\mathbf{H}_S)} t_{i,j} h_{S,j}$, where $t_{i,j} \in \mathbb{K}$ form a matrix $\mathbf{T}^{(S)}$, boiling down to $\mathbf{G}_S = \mathbf{T}^{(S)}\mathbf{H}_S$.

Third, it suffices to verify for any $|S| = d$ instead of $|S| \leq d$. Observe that by the definition of matrix multiplication $\mathbf{G}_S = \mathbf{T}^{(S)}\mathbf{H}_S \iff \forall i \in [d] : \mathbf{G}_{S,i} = \mathbf{T}^{(S)}\mathbf{H}_{S,i}$, where $\mathbf{G}_{S,i}, \mathbf{H}_{S,i}$ denote the respective i^{th} column. Clearly, $\mathbf{G}_S = \mathbf{T}^{(S)}\mathbf{H}_S \implies \mathbf{G}_{S'} = \mathbf{T}^{(S)}\mathbf{H}_{S'}$ for any $S' \subset S$. Finally, d-privacy is encoded by the polynomial equations

$$C_{d\text{-}priv} = \bigcup_{S \subseteq [n], |S|=d} \{\mathbf{G}_S - \mathbf{T}^{(S)}\mathbf{H}_S = \mathbf{0}\}.$$

Another requirement is that the encoding matrix $\mathbf{A}$ of GCM in [34] has full rank. Note that the less general work [36] structurally ensures this property in their encoder and does not require this constraint. To encode the notion of rank, we use the property that any matrix $\mathbf{A}$ with full rank must have a right inverse $\mathbf{B}$ s.t. $\mathbf{AB} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Then, the constraint for full rank is

$$C_{rank} = \{c_{i,j} = 0 \mid [c_{i,j}]_{i,j} = \mathbf{AB} - \mathbf{I}\}.$$

3.3.6 Abstract Permute. We turn to the most important rule for verifying GCM. The APerm rewrite also implements the Perm rule but covers some cases not handled by PiPerm. The key observation is that the random variables that are used for a permute may differ between choices of parameters $\mathcal{V}_C$.

We start with a simple example, followed by a realistic example highlighting the need for this rewrite and the Perm rule. Consider the term $e = s + \alpha_0 r_0 + \alpha_1 r_1$ with secret s, randoms r_i and subject to constraint $\alpha_0 \neq \alpha_1$. Clearly, α_0, α_1 have poles which violates $\text{perm}(e, \{r_i\})$ for each random. Hence, the eOTP rule cannot be applied. Likewise, roots would prevent the PiPerm rewrite. However, this term is secret independent, since for any valid choice of α_i there is one r_i s.t. $\text{perm}(e, \{r_i\})$. But this requires a layer of abstraction, i.e., checking for any valid choice of parameters.

We justify the need again with a realistic use-case, visualized in Figure 3. GCM encompasses all other masking schemes in its general encoding [18]. This includes IPM, with its characteristic structure in Figure 3a, but also many variants, including the one in Figure 3b. Consequently, verifying GCM for arbitrary valid parameters requires treating both IPM variants at once. For our example, let the shares be computed as $\vec{y} = \vec{x}\mathbf{G} + \vec{r}\mathbf{H}$ where $\vec{x}, \vec{r}$ are secret, respectively random, variables. Verifying that the share belonging to the first column in Figure 3 combined with any subset of the remaining columns is uniformly distributed requires aforementioned abstraction: Consider $\vec{y}'$ consisting of all but the last share, i.e., the shares corresponding to all but the last column in Figure 3a and b. Showing secret independence requires performing rewrites involving the random variables $r_1, \ldots, r_{m-1}$ for the original scheme in Figure 3a, whereas the variant where the columns are permuted requires an argument involving the random variables $r_2, \ldots, r_m$ in Figure 3b. We emphasize that verifying GCM requires to treat all parameter choices, which includes the two IPM variants in our example. The need for our abstract permute rule Perm and our implementation APerm is now clear.

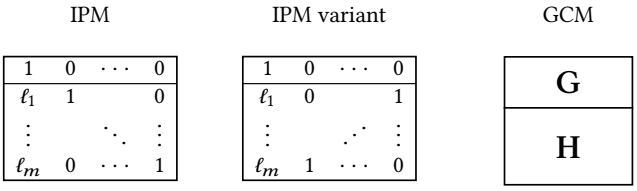


Figure 3: Instantiations of the generic encoder of GCM capture original IPM (a) and a permuted variant (b).

Our approach uses Gröbner bases to verify that no assignment of variables in $\mathcal{V}_C$ results in $\text{rank}(\mathbf{A}) < n$ is possible for the linear randomness matrix $\mathbf{A}^{n \times m}$ with $m \geq n$ randoms and n term-rows. The approach also covers cases where only a subset of rows can be rewritten by APerm and iteratively checks all subsets of $m' \leq m$ columns and $n' \leq m'$ rows, s.t. the m' randoms do not occur outside the n' rows (i.e., the corresponding components of $\mathbf{A}$ must be zero). This can be enumerated efficiently for decreasing n'.

The needed predicate $\text{fullrank}(\mathbf{A}, C_H, C)$ returns true if a matrix $\mathbf{A}$ has full rank by checking $1 \in \text{G}(C \cup C_H \cup \text{lindep}(\mathbf{A}))$ under constraints C, C_H and $\text{lindep}(\mathbf{A})$ explained one after another.

The additional set of constraints C_H enforces secret dependency to avoid false negatives. Consider $e = \alpha s + \alpha r$ as a guiding example. Without any constraint on α the random r does not satisfy $\text{perm}(e, \{r\})$ and hence no rewrites can be performed. However, in the case that $\alpha = 0$ and r vanishes, s also vanishes, rendering the expression independent of secrets. To avoid false negatives arising in such cases, we require that at least one monomial including variables from $\mathcal{V}_H$ has a non-zero coefficient. Let $\mathcal{S}_c$ denote coefficients of secret dependent monomials. This is encoded using the Rabinowitsch trick as $C_H = \left\{\sum_{s_i \in \mathcal{S}_c} u_i s_i - 1\right\}$. As before, this is sound as all coefficients s_i belong to the integral domain $\mathbb{K}[\mathcal{V}_C]$.

We now turn to the function $\text{lindep} : \mathbb{K}(\mathcal{V}_C) \to \mathbb{K}[\mathcal{V}_C]$ that encodes that a matrix $\mathbf{A}$ has linear dependencies in its rows. This perfectly encodes the full rank property as $\mathbf{A} \in (\mathbb{K}(\mathcal{V}_C))^{n' \times m'}$ has full rank $n' \leq m' \leq m$ iff there is no linear combination of its n' rows. Hence, for some auxiliary variables $\vec{\lambda}$ we encode linear dependence as $\vec{\lambda}\mathbf{A} = 0$ and assert at least one λ_i non-zero:

$$\text{lindep}(\mathbf{A}) = \{\vec{\lambda}\mathbf{A}\} \cup \left\{\sum_{\lambda_i \in \vec{\lambda}} v_i \lambda_i - 1\right\}.$$

To use Gröbner-bases, we eliminate fractions in $\mathbf{A}$ by scaling with the least common multiple of the denominators in each row.

The overall constraint amounts to checking if there is a linear combination of rows under constraint C for the encoding that involves at least a partial secret dependence, encoded in C_H. If this is unsatisfiable, matrix $\mathbf{A}$ always has full rank, and consequently also satisfies perm whenever there is a secret dependence. Note, that when APerm returns some rewritten terms $\vec{e}'$ given initial terms $\vec{e}$, then there is no equivalence $\vec{e} \equiv \vec{e}'$ but instead a step is made that asserts that if $\vec{e}' \perp \mathcal{Z}$ then $\vec{e} \perp \mathcal{Z}$ for some $\mathcal{Z} \subseteq \mathcal{V}_H$. This is due to the side-condition ensuring that there is a secret dependency.

3.3.7 Expanding OTP. The purpose of $\text{eOTP}(\vec{e}, C)$ is to combine residue terms, in the hope of establishing the desired continuation φ. In contrast to Perm, the underlying eOTP rule allows establishing not only uniform distribution of $\vec{e}$ but can perform arbitrary

(bijective) substitutions for a random variable and thereby establish more general equivalencies. Consider this example with secrets s_i

$$\vec{e} = \begin{pmatrix} s_1 \\ s_1 - s_2 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}.$$

The coefficient matrix $\mathbf{A}$ and all of its sub-matrices are clearly not invertible. However, the eOTP allows to rewrite r_1 by $e' = r_1 - r_2 - s_1 + s_2$, which satisfies $\text{perm}(e', \{r_1\})$ and results in

$$\vec{e'} = \begin{pmatrix} s_2 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}.$$

This term does not depend on s_1 anymore and hence has reduced the dependency set. We use the rewrite as a means to check the least dependency set before performing Destr. Its implementation relies on the roots predicate to check candidates and some heuristics on the order in which terms $\vec{e}_i$ and random variables are checked.

3.4 Functional Verification

In addition to verifying side-channel resistance, we also verify the functional correctness of gadgets. Let $(\vec{x}^i)_{i\in[l]}$ and $(\vec{y}^i)_{i\in[m]}$ denote encodings of l inputs, respectively m outputs, for gadget G_f implementing function f. Our tool verifies correctness of gadgets for a reference function f provided by users by checking

$$f\left(\text{Dec}(\vec{x}^0), \ldots, \text{Dec}(\vec{x}^{l-1})\right) =_{\mathbb{K}} \left(\text{Dec}(\vec{y}^0), \ldots, \text{Dec}(\vec{y}^{m-1})\right).$$

It relies on Con, or a Gröbner check (using the Rabinowitsch trick) when constraints are involved, both implemented with `SageMath MPolynomials` [33]. Besides strong correctness assurance, this mechanism is invaluable during gadget development to spot flaws.

4 Evaluation

We implement our approach in a universal tool eVer, that is able to verify t-(S)NI and t-PS for all masking schemes[3], validate it on a broad set of 37 masked algorithms, including many previously out of reach, and discuss performance, applicability, and scalability.

4.1 Experimental Setup

Gadgets implementing the field operations, addition and multiplication, and a refresh operation for secure composition are entirely sufficient for protecting any cryptographic scheme, as any other operation can be composed thereof. We evaluate whether approach and tool are able to verify the necessary gadgets instantiated with different security parameters d, e and number of packed secrets k for all schemes. We focus on state-of-the-art and classic gadgets.

All gadget implementations are first verified to be functionally correct, allowing early identification and mitigation of bugs. Combining correctness and security verification rules out attesting security to gadgets not computing correctly, e.g., discarding secrets.

Experiments are conducted on a CPU with two Intel Xeon Gold 6342 processors, totaling 96 cores, and each verification is limited to 60 minutes. The persistent caches for the Gröbner check memoization are emptied before each verification. To check the accuracy of our tool, we verify if it confirms security for the largest t permitted for chosen values of d, e, k. We also check whether it reports vulnerabilities for greater t.

[3] t-Probe-Isolating Non-Interfering (t-PINI) is in development.

Multiplication Gadgets are more complex and expected to attain t-SNI for varying t. For additive masking, we implement the ISW multiplication [25] with $d = t$. For IPM, we instantiate IPMult_L [4, 5] with $d = t$. For PM, we evaluate two approaches BGWMult [1] in Algorithm 10 and our own generalization of LaOlaMult [12] defined in Algorithm 12. Their expected security order is $t \leq d - k + 1$ but LaOlaMult is only proven for even d, even number of shares $n = d + e + k$ and $k \leq \lfloor \frac{d}{2} \rfloor$. Our evaluation also includes instantiations for odd d and n. For GCM, we evaluate two multiplication gadgets from [34] and [36].

Refresh Gadgets must be t-SNI. For additive masking we evaluate ISWRefresh [25], for IPM IPRefresh_L [4, 5], and opt-Refresh [1] for PM, as well as less secure gadgets. The GCM refreshes are CodeL gadgets of [34], respectively [36].

4.2 Verification Results

The results for $\mathbb{K} = \text{GF}(2^8)$ are stated in Table 1. Red crosses indicate that eVer reported vulnerabilities. No false negatives are reported. We analyze the results separately for each scheme in increasing order of algebraic complexity. They clearly indicate that our approach and eVer universally apply to a wide range of gadgets, regardless of the masking scheme. ϵ indicates less than 1 second.

Additive and Inner Product Masking. Addition, multiplication, and refresh gadgets pass t-SNI verification for permitted t, whereas vulnerabilities are reported for $(t' > t)$-SNI.

Polynomial Masking. BGWMult and LaOlaMult verify for a wide parameter range, including $e > 0$, i.e., with fault protection, and packed secret-sharing with $k > 1$. Addition, multiplication, and refresh gadgets verify as t-SNI for permitted t; vulnerabilities are reported for $(t' > t)$-SNI. The original LaOlaMult is defined and proven for even d, n only [12]. Our generalization in Algorithm 12 verifies as t-SNI for $t \leq d-k+1$, including odd instances not covered originally. Surprisingly, the original requirement $k \leq \lfloor \frac{d}{2} \rfloor$ can in some cases be relaxed to $k \leq d$ (Table 1), which was a disadvantage of LaOlaMult compared to BGWMult. It is unclear how far this extends, given that verification rejects 2-SNI for $k = 2, d = 3, e = 0$, which might be false negative. This highlights the impact of our work: formal verification identifies additional secure and more efficient parameters, compared to the options proven by pen-and-paper. Thereby, verification improves the practical usability of PM.

GCM Masking. CodeL-based squaring, refresh, and multiplication gadgets pass t-SNI verification for permitted t, whereas vulnerabilities are reported for $(t' > t)$-SNI. The verification takes longer due to the increased number of Gröbner checks, which is expected since the scheme involves the most complex constraints and highest number of parameters.

4.3 Optimizing Gadgets

Using eVer, we achieve several optimizations for state-of-the-art PM and IPM gadgets.

We adapt LaOlaMult in Algorithm 12, generalizing to odd d, n and verify it as functionally correct and secure. The changes consist of the multiplication for odd d performing one call to SplitRed (see Algorithm 13) with $\lceil \frac{d}{2} \rceil$ and the other one with $\lfloor \frac{d}{2} \rfloor$. In case $d = 1$ SplitRed is invoked only once. Besides, we identify that the state-of-the-art version of the gadget [1] contains a superfluous sequence

Table 1: Benchmark results using one thread for up to 1k (thousand) tuples, 32 threads for up to 1M (million), 64 for up to 1B (billion) and 96 threads otherwise. We verify SNI, unless noted otherwise. ✗ indicates vulnerabilities.

Gadget	**d** with **k** = 1, **e** = 0				
	1	2	3	4	5
ISW Multiplication [24]	ε	ε	1s	38s	-
# tuples	13	435	25k	2M	216M
ISW Refresh [24]	ε	ε	ε	2s	19s
# tuples	5	66	1.5k	52k	2.3M
IPMultiplication [5]	ε	ε	ε	47s	-
# tuples	16	780	67.5k	8.5M	1.4B
NI-IPRefresh [5]	ε	ε	ε	ε	ε
# tuples	6	55	560	6k	66k
SNI-IPRefresh [5]	ε	ε	✗	✗	✗
# tuples	6	55	560	6k	66k
SecIPRefresh [5]	ε	ε	ε	5s	375s
# tuples	10	351	22k	2M	244M
Refresh [21]	ε	ε	1s	9s	832s
# tuples	6	171	10k	865k	10M
OptRefresh [1]	ε	ε	1s	4s	113s
# tuples	6	105	3.3k	149k	9M
SNI-Refresh [32]	ε	ε	✗	✗	✗
# tuples	7	136	4.5k	212k	13M
NI-Refresh [32]	ε	ε	ε	ε	5s
# tuples	7	136	4.5k	212k	13M
BGW [1]	ε	1s	183s	-	-
# tuples	37	6.7k	3M	2.4B	-
LaOla [Algorithm 12]	ε	ε	38s	-	-
# tuples	24	6.5k	1M	541M	157B
GCM Mult [34]	ε	14s	-	-	-
# tuples	40	8.2k	4.3M	-	-
GCM Mult [36]	ε	ε	4s	614s	-
# tuples	29	2.9k	519k	133M	44.4B
GCM Refresh [34]	ε	ε	-	-	-
# tuples	28	3.5k	1M	-	-
GCM Refresh [36]	ε	ε	ε	5s	426s
# tuples	11	406	26.2k	2.4M	298M
GCM Square [34]	ε	1s	-	-	-
# tuples	30	3.7k	1.1M	-	-
GCM Square [36]	ε	ε	1s	6s	560s
# tuples	13	496	32.5k	3M	373M
	(**d**, **e**) with **k** = 1				
	(1,1)	(1,2)	(2,1)	(2,2)	(3,1)
Refresh [21]	ε	ε	ε	ε	3s
# tuples	9	12	378	666	32.5k
OptRefresh [1]	ε	ε	ε	ε	3s
# tuples	9	12	231	406	10.7k
NI-Refresh [32]	ε	ε	ε	ε	ε
# tuples	10	13	231	351	8.4k
BGW [1]	ε	ε	1s	3s	-
# tuples	72	110	19.1k	36.3k	10.9M
LaOla [Algorithm 12]	ε	ε	2s	4s	-
# tuples	54	80	18k	52k	7.1M
	(**k**, **d**, **e**), **t** = **d** − **k** + **1**				
	(2,2,0)	(2,2,1)	(2,2,2)	(2,3,0)	(3,3,0)
BGW [1]	ε	ε	1s	8s	1s
# tuples	101	156	210	29.6k	197
LaOla [Algorithm 12]	ε	1s	2s	✗	1s
# tuples	93	140	225	15k	140

of ZEnc calls that can be omitted without impacting security. This improves performance and randomness consumption significantly.

In our evaluation we include gadgets known to be insecure. The refresh gadget proven by pen-and-paper in [32] fails to achieve 3-SNI as first published in [12], which we confirm with eVer. Besides, eVer confirms that the refresh gadget is t-SNI for $t < 3$, cf. Table 1. This confirms that the gadget can be used, despite the broken proof, for $t < 3$ instead of secure t-SNI variants in [1, 12, 21] which are more expensive. The same optimization, using only a single zero encoding for $t < 3$, also applies to both the state-of-the-art refresh gadget from [1] and the IPM refresh from [5], reducing computation and randomness by up to half. Our results show that optimizing gadgets for fixed orders can provide a significant gain and is worth pursuing for other gadgets. Again, eVer strongly supports gadget design and optimization.

4.4 Discussion

Verification time depends clearly on the number of leakages and, hence, the size of the gadgets, since security notions require a security predicate φ to hold for any combination of leaking terms (and outputs for t-SNI). The exponential complexity of the number of such tuples is depicted for multiplication and refresh gadgets in Figure 4 and explains why the verification of IPM, PM, and GCM gadgets is slower than additive masking: the larger number of leakages in these gadgets require to invoke the Verify procedure more often. eVer is slower in verifying additive masking than existing tools since it tries to apply rewrite rules only needed for the more complex schemes. For the BGWMult and LaOlaMult multiplications, the number of tuples quickly approaches 1 Billion, which makes enumerative approaches unlikely to succeed, even if the verification time was as fast as for additive masking schemes.

Another interesting aspect is whether the number of parameter variables impacts verification time and scalability. Verifying a security notion requires checking that the notion holds for any valid choice of parameters. In Table 2, the number of possible assignments of parameter variables for BGWMult and LaOlaMult is depicted, which is $p = \binom{|\mathbb{K}|}{m}$ where $m = n + k$ is the number of support points in packed secret sharing. It is evident that already for the AES field $\mathrm{GF}(2^8)$ the number of parameter-tuple combinations quickly exceeds the count of tuples of an ISW multiplication at order $d = 5$. Assuming there was an unreasonable fast tool that could verify an instance of BGWMult or LaOlaMult in 0.001s for concrete parameters, this tool would still take longer than eVer to re-verify for any valid choice of parameters, for all settings in Table 1. This underlines the strength of our approach of treating parameters symbolically.

Likely, the count of parameter variables influences our verification time, however, the exact impact remains unclear. The GCM multiplication of [34] is significantly affected by Gröbner checks. There, the APerm rule is invoked in the verification of every tuple that is not trivially secure. However, it is unclear whether its slower verification is inherent to the gadget structure or rather caused by a side effect of FDedup on the linear randomness matrix **A** that might lead to suboptimal Gröbner checks. We observe that variable ordering can significantly influence performance and leave its optimization to future work.

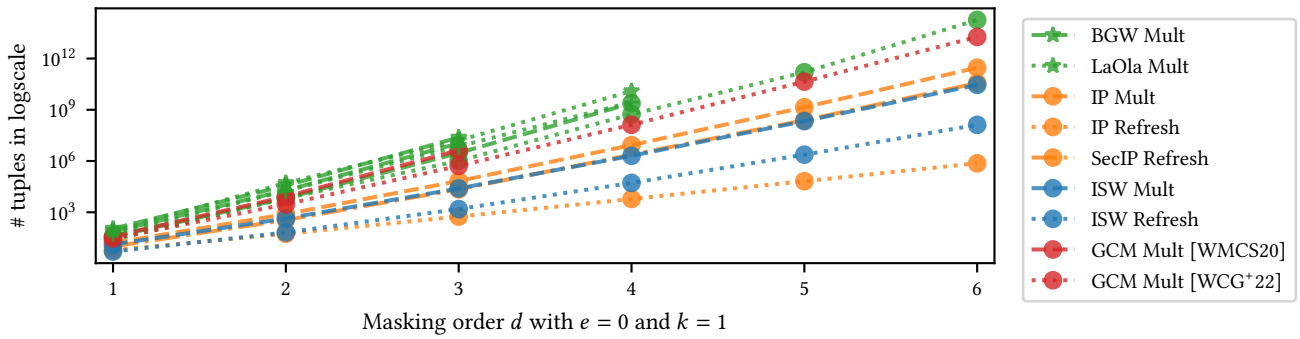


Figure 4: The number of tuples to be verified at specific security orders d for selected gadgets and parameters e, k. The masking schemes with higher algebraic complexity have more tuples that need to be verified.

Table 2: Number of valid assignments of parameter variables in BGWMult and LaOlaMult.

Gadget	(k, d, e)	GF(7)	GF(2^8)	GF(3329)
LaOlaMult	(1, 1, 0)	30	$6.5 \cdot 10^4$	$1.1 \cdot 10^7$
LaOlaMult	(1, 1, 1)	120	$1.6 \cdot 10^7$	$3.7 \cdot 10^{10}$
LaOlaMult	(1, 2, 0)	120	$1.6 \cdot 10^7$	$3.7 \cdot 10^{10}$
LaOlaMult	(2, 2, 0)	720	$1.0 \cdot 10^{12}$	$4.1 \cdot 10^{17}$
BGWMult	(1, 1, 0)	120	$1.6 \cdot 10^7$	$3.7 \cdot 10^{10}$
BGWMult	(1, 1, 1)	360	$4.1 \cdot 10^9$	$1.2 \cdot 10^{14}$
BGWMult	(1, 2, 0)	720	$1.0 \cdot 10^{12}$	$4.1 \cdot 10^{17}$
BGWMult	(2, 2, 0)	-	$6.5 \cdot 10^{16}$	$4.5 \cdot 10^{24}$

Table 3: Benchmark results evaluating SNI for different AES S-box'es, using one thread for d = 1 and 32 threads for d = 2 with a timeout of 1 hour.

Gadget	d with k = 1, e = 0	
	1	2
IP AES S-box [5]	2s	1393s
PM AES S-box [1]	3s	-
GCM20 AES S-box [34]	20s	-
GCM22 AES S-box [36]	1s	51s

Another interesting question is whether the verification performance depends on the underlying field $\mathbb{K}$ or is field-agnostic. In Figure 5 we show the verification time of selected multiplication gadgets for varying fields GF(2), GF(2^k), GF(p) and several p, k. The main observation is that, while there is some difference it is independent of the field size of $\mathbb{K}$. This is in stark contrast to model counting and BDD approaches like SILVER [26] where performance is inherently decreasing with the field size and number of variables. The verification time differs between characteristics GF(2), GF(2^k) and GF(p), but these differences stem from `SageMath`, where the GF(2^k) extension fields appear to be heavily optimized, as our algorithms and implementation do not distinguish these cases. Hence, our approach is agnostic to the underlying field.

Does the approach allow to verify realistic compositions of gadgets? We implement IPM, PM and GCM masked AES S-box'es using aforementioned gadgets with $k = 1, e = 0$ and Frobenius squaring when available. Their verification passes at first order. Notably, IPM and GCM22 pass at higher order for $d = t = 2$, cf. Table 3. Due to GCM22 requiring many Perm applications, we conclude that the approach, especially our topological application for Destr, conceptually extends to realistic compositions, even at higher order. The PM and GCM20 S-Box'es with $d = 2$ and all S-Box'es with $d > 2$ fail *before* verification due to circuit size for a different reason: Our tool eagerly computes canonical representatives which is computationally expensive for large terms but rarely used for verifying compositions. This problem will be addressed in the future.

4.5 Towards Verification of Concrete Implementations and Practical Security

eVer intentionally provides as main entry point a Python API for verifying security notions on sets of terms. In our experience, it is easier to debug gadgets in Python than in DSLs which often lack debugging capabilities. We provide many examples for implementing gadgets in Python within the artifact.

To verify concrete implementations and conversion gadgets, the term algebra needs to support multi-sorted signatures with casts between different fields, bit-slicing and shifts. Such a change voids Theorems 2.2 and 2.4 since there is no longer a unique canonical representation. However, this is not critical: one has to search for the linear randomness matrix $\mathbf{A}$ (Section 3.3.2) directly within the term algebra, and ensure that its elements fall into $T(\Sigma^{\times}, \mathcal{V}_C)$ only if Gröbner checks should be applied. The new problem arising here is choosing the right $\mathbf{A}$ among many congruent terms, whereas there is a single representative in our work. Note, that this problem is orthogonal to our work. Once resolved, the approach of scVerif [10] can be applied immediately to verify concrete implementations.

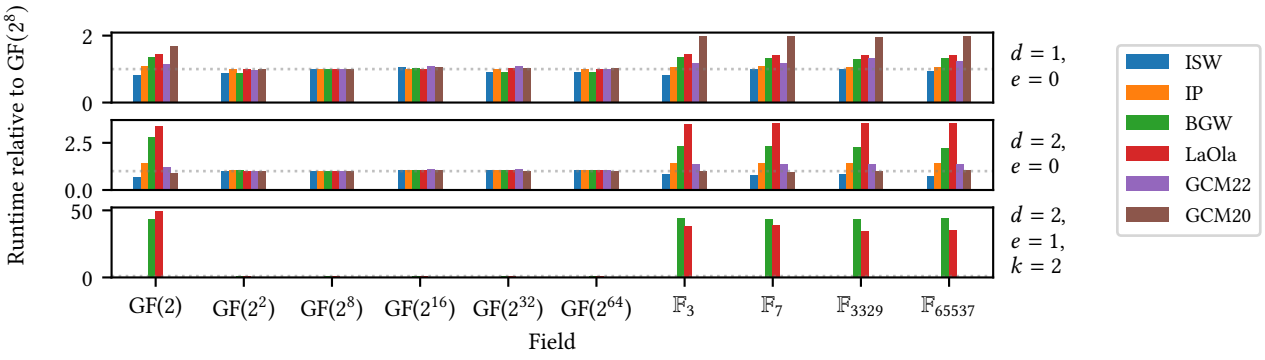


Figure 5: Verification times for selected gadgets and parameters for varying fields $\mathbb{K}$, relative to GF(2^8).

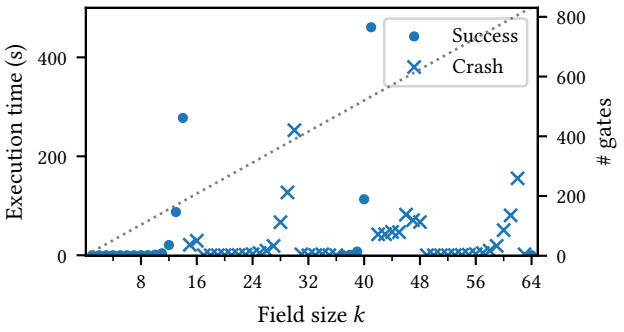


Figure 6: The time (left scale, blue marks) until SILVER has verified or crashed during checking probing security for the Boolean expression $(((a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c) \oplus x_0) \oplus x_1)$ with random variables a, b, c and shares x_0, x_1 in $GF(2^k)$ for increasing field size k. While circuit size increases linearly (gray line, right scale), SILVER inconsistently crashes. The expression can be verified in time independent of the field using a sequence of two eOTPs.

Besides this, providing an API allows to re-use our approach for verifying random probing security as outlined in [11]. We emphasize that this enables to evaluate and quantify, via formal verification, the stronger practical resilience of IPM, PM and GCM. This is currently supported only by less dependable simulations.

5 Related Work

We compare to state-of-the-art tools for verifying side-channel security. No prior work handles symbolic parameters with constraints as required for verifying IPM, PM, and the GCM variants.

Our approach does not degrade with the field size (Section 4.4). In contrast, model counting and BDD approaches like SILVER [26] degrade with increasing field size, as highlighted by an experiment in Figure 6. Approaches mandating to bit-blast to Boolean logic, like [14, 22], degrade similarly.

Two state-of-the-art language-based tools are MaskVerif [6] and IronMask [11]. While both tools outperform eVer in additive masking, cf. Table 4, they are, as any existing tool, incapable of verifying IPM, PM, and GCM. MaskVerif relies on the OTP and a limited variant of eOTP, which are insufficient to verify any of these schemes (Section 1.1 and 2.2.2). IronMask is based on eOTP and Gaussian elimination, similar to our PiPerm but neither supporting symbolic, nor polynomial, coefficients, hence mandating exhaustive re-verification for parameter choices, which becomes quickly intractable and is outperformed by our approach (Section 4.4). Its underlying eOTP is intrinsically insufficient to verify GCM as exemplified in Section 3.3.6 and the tool itself is restricted to GF(2^k), making eVer the first to support arbitrary fields. Notably, IronMask is proven complete for a specific structure of gadgets, called *non-linear randomness (NLR)* that requires randomness to occur linearly in in- and outputs but not within the circuit. Our linear randomness representation is flexibly computed on a best-effort basis, driven by Destr, making our approach apply to compositions of gadgets, which often violate NLR (cf. S-Box'es in Section 4.4).

Table 4: Performance comparison of MaskVerif, IronMask, SILVER, and eVer for ISW AND and ISW Refresh in GF(2) with $d = 1, \ldots, 5$ on 32 threads.

Gadget	Tool	d				
		1	2	3	4	5
ISW AND	MaskVerif	ε	ε	ε	ε	6 s
	IronMask	ε	ε	ε	ε	1 s
	SILVER	ε	ε	5 s	27 min	-
	eVer	ε	ε	ε	28 s	-
ISW Refresh	MaskVerif	ε	ε	ε	ε	ε
	IronMask	ε	ε	ε	ε	ε
	SILVER	ε	ε	ε	2 s	85 s
	eVer	ε	ε	ε	ε	17 s

Acknowledgments

This research was supported by the Deutsche Forschungsgemeinschaft (DFG, German research Foundation) as part of the Excellence Strategy of the German Federal and State Governments – EXC 2092

CASA – 390781972; from the German Federal Ministry of Research, Technology and Space (BMFTR) under the projects Sec4IoMT and RILKOSAN; and by the European Research Council (ERC) Advanced Grant BRIDGE (101096871). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Jorge Andresen, Paula Arnold, Sebastian Berndt, Thomas Eisenbarth, Sebastian Faust, Marc Gourjon, Eric Landthaler, Elena Micheli, Maximilian Orlt, Pajam Pauls, Kathrin Wirschem, and Liang Zhao. 2025. UP TO 50% OFF: Efficient Implementation of Polynomial Masking. Cryptology ePrint Archive, Paper 2025/2088. https://eprint.iacr.org/2025/2088

[2] Paula Arnold, Sebastian Berndt, Thomas Eisenbarth, Sebastian Faust, Marc Gourjon, Elena Micheli, Maximilian Orlt, Pajam Pauls, Kathrin Wirschem, and Liang Zhao. 2025. Improving the Fault Robustness of Polynomial Masking. Cryptology ePrint Archive, Paper 2025/1275. https://eprint.iacr.org/2025/1275

[3] Paula Arnold, Sebastian Berndt, Thomas Eisenbarth, and Maximilian Orlt. 2024. Polynomial sharings on two secrets: Buy one, get one free. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2024, 3 (2024), 671–706. doi:10.46586/TCHES.V2024.I3.671-706

[4] Josep Balasch, Sebastian Faust, and Benedikt Gierlichs. 2015. Inner Product Masking Revisited. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9056)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, 486–510. doi:10.1007/978-3-662-46800-5_19

[5] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, Clara Paglialonga, and François-Xavier Standaert. 2017. Consolidating Inner Product Masking. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, 724–754. doi:10.1007/978-3-319-70694-8_25

[6] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. 2019. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11735)*, Kazue Sako, Steve A. Schneider, and Peter Y. A. Ryan (Eds.). Springer, 300–318. doi:10.1007/978-3-030-29959-0_15

[7] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. 2015. Verified Proofs of Higher-Order Masking. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9056)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, 457–485. doi:10.1007/978-3-662-46800-5_18

[8] 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, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 116–129. doi:10.1145/2976749.2978427

[9] Gilles Barthe, Marion Daubignard, Bruce Kapron, Yassine Lakhnech, and Vincent Laporte. 2010. On the equality of probabilistic terms. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*. Springer, 46–63.

[10] Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, and Lars Porth. 2021. Masking in Fine-Grained Leakage Models: Construction, Implementation and Verification. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 2 (2021), 189–228. doi:10.46586/TCHES.V2021.I2.189-228

[11] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. 2022. IronMask: Versatile Verification of Masking Security. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 142–160. doi:10.1109/SP46214.2022.9833600

[12] Sebastian Berndt, Thomas Eisenbarth, Sebastian Faust, Marc Gourjon, Maximilian Orlt, and Okan Seker. 2023. Combined Fault and Leakage Resilience: Composability, Constructions and Compiler. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 14083)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, 377–409. doi:10.1007/978-3-031-38548-3_13

[13] Roderick Bloem, Barbara Gigerl, Marc Gourjon, Vedad Hadzic, Stefan Mangard, and Robert Primas. 2022. Power Contracts: Provably Complete Power Leakage Models for Processors. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM, 381–395. doi:10.1145/3548606.3560600

[14] 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 *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 321–353.

[15] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. 2021. Masking Kyber: First- and Higher-Order Implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 4 (2021), 173–214. doi:10.46586/TCHES.V2021.I4.173-214

[16] Julien Bringer, Claude Carlet, Hervé Chabanne, Sylvain Guilley, and Houssem Maghrebi. 2014. Orthogonal Direct Sum Masking - A Smartcard Friendly Computation Paradigm in a Code, with Builtin Protection against Side-Channel and Fault Attacks. In *Information Security Theory and Practice. Securing the Internet of Things - 8th IFIP WG 11.2 International Workshop, WISTP 2014, Heraklion, Crete, Greece, June 30 - July 2, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8501)*, David Naccache and Damien Sauveron (Eds.). Springer, 40–56. doi:10.1007/978-3-662-43826-8_4

[17] Claude Carlet and Sylvain Guilley. 2015. Complementary Dual Codes for Counter-Measures to Side-Channel Attacks. In *Coding Theory and Applications*, Raquel Pinto, Paula Rocha Malonek, and Paolo Vettori (Eds.). Springer International Publishing, Cham, 97–105.

[18] Wei Cheng, Sylvain Guilley, Claude Carlet, Jean-Luc Danger, and Sihem Mesnager. 2021. Information Leakages in Code-based Masking: A Unified Quantification Approach. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 3 (2021), 465–495. doi:10.46586/TCHES.V2021.I3.465-495

[19] Jean-Sébastien Coron, Emmanuel Prouff, and Thomas Roche. 2012. On the Use of Shamir's Secret Sharing against Side-Channel Analysis. In *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 7771)*, Stefan Mangard (Ed.). Springer, 77–90. doi:10.1007/978-3-642-37288-9_6

[20] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. 2018. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 89–120. doi:10.13154/TCHES.V2018.I3.89-120

[21] Sebastian Faust, Maximilian Orlt, Kathrin Wirschem, and Liang Zhao. 2025. All-You-Can-Compute: Packed Secret Sharing for Combined Resilience. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2025, 2 (2025), 420–459. doi:10.46586/TCHES.V2025.I2.420-459

[22] Barbara Gigerl, Robert Primas, and Stefan Mangard. 2023. Formal Verification of Arithmetic Masking in Hardware and Software. In *Applied Cryptography and Network Security - 21st International Conference, ACNS 2023, Kyoto, Japan, June 19-22, 2023, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13905)*, Mehdi Tibouchi and Xiaofeng Wang (Eds.). Springer, 3–32. doi:10.1007/978-3-031-33488-7_1

[23] Louis Goubin and Ange Martinelli. 2011. Protecting AES with Shamir's Secret Sharing Scheme. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6917)*, Bart Preneel and Tsuyoshi Takagi (Eds.). Springer, 79–94. doi:10.1007/978-3-642-23951-9_6

[24] Dahmun Goudarzi, Anthony Journault, Matthieu Rivain, and François-Xavier Standaert. 2018. Secure Multiplication for Bitslice Higher-Order Masking: Optimisation and Comparison. In *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10815)*, Junfeng Fan and Benedikt Gierlichs (Eds.). Springer, 3–22. doi:10.1007/978-3-319-89641-0_1

[25] Yuval Ishai, Amit Sahai, and David A. Wagner. 2003. Private Circuits: Securing Hardware against Probing Attacks. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2729)*, Dan Boneh (Ed.). Springer, 463–481. doi:10.1007/978-3-540-45146-4_27

[26] David Knichel, Pascal Sasdrich, and Amir Moradi. 2020. SILVER - Statistical Independence and Leakage Verification. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12491)*, Shiho Moriai and Huaxiong Wang (Eds.). Springer, 787–816. doi:10.1007/978-3-030-64837-4_26

[27] Romain Poussier, Qian Guo, François-Xavier Standaert, Claude Carlet, and Sylvain Guilley. 2017. Connecting and Improving Direct Sum Masking and Inner Product Masking. In *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10728)*, Thomas

Eisenbarth and Yannick Teglia (Eds.). Springer, 123–141. doi:10.1007/978-3-319-75208-2_8

[28] Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. 2014. On the Practical Security of a Leakage Resilient Masking Scheme. In *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8366)*, Josh Benaloh (Ed.). Springer, 169–182. doi:10.1007/978-3-319-04852-9_9

[29] Emmanuel Prouff and Thomas Roche. 2011. Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6917)*, Bart Preneel and Tsuyoshi Takagi (Eds.). Springer, 63–78. doi:10.1007/978-3-642-23951-9_5

[30] JL Rabinowitsch. 1930. Zum Hilbertschen Nullstellensatz. *Math. Ann.* 102, 1 (1930), 520–520.

[31] Matthieu Rivain and Emmanuel Prouff. 2010. Provably Secure Higher-Order Masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6225)*, Stefan Mangard and François-Xavier Standaert (Eds.). Springer, 413–427. doi:10.1007/978-3-642-15031-9_28

[32] Okan Seker, Abraham Fernandez-Rubio, Thomas Eisenbarth, and Rainer Steinwandt. 2018. Extending Glitch-Free Multiparty Protocols to Resist Fault Injection Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 394–430. doi:10.13154/TCHES.V2018.I3.394-430

[33] The Sage Developers. 2022. *SageMath, the Sage Mathematics Software System (Version 9.5.0)*. `https://www.sagemath.org`.

[34] Weijia Wang, Pierrick Méaux, Gaëtan Cassiers, and François-Xavier Standaert. 2020. Efficient and Private Computations with Code-Based Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020, 2 (2020), 128–171. doi:10.13154/TCHES.V2020.I2.128-171

[35] Weijia Wang, François-Xavier Standaert, Yu Yu, Sihang Pu, Junrong Liu, Zheng Guo, and Dawu Gu. 2016. Inner Product Masking for Bitslice Ciphers and Security Order Amplification for Linear Leakages. In *Smart Card Research and Advanced Applications - 15th International Conference, CARDIS 2016, Cannes, France, November 7-9, 2016, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10146)*, Kerstin Lemke-Rust and Michael Tunstall (Eds.). Springer, 174–191. doi:10.1007/978-3-319-54669-8_11

[36] Qianmei Wu, Wei Cheng, Sylvain Guilley, Fan Zhang, and Wei Fu. 2022. On Efficient and Secure Code-based Masking: A Pragmatic Evaluation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 3 (2022), 192–222. doi:10.46586/TCHES.V2022.I3.192-222

A Improved Polynomial Masking Gadgets

The algorithms for our generalized and improved LaOlaMult adapted from [1] are stated in Algorithm 12, respectively in Algorithm 13 for SplitRed. They are evaluated for correctness and security using eVer, cf. Table 1. We also list other algorithms implemented for our evaluation.

Algorithm 1: SWAdd for $n = d + e + 1$, and $k \leq d$ [12, Algorithm 1]

Input : Encodings $\mathbf{F}$ and $\mathbf{G}$
Output : Encoding $\mathbf{Q}$ s.t. $\mathrm{Dec}(\mathbf{Q}) = \mathrm{Dec}(\mathbf{F}) + \mathrm{Dec}(\mathbf{G})$

for $i \in [n]$ **do**
  $Q_i \leftarrow F_i + G_i$
return $\mathbf{Q}$

Algorithm 2: SWSub for $n = d + e + 1$, and $k \leq d$

Input : Encodings $\mathbf{F}$ and $\mathbf{G}$
Output : Encoding $\mathbf{Q}$ s.t. $\mathrm{Dec}(\mathbf{Q}) = \mathrm{Dec}(\mathbf{F}) - \mathrm{Dec}(\mathbf{G})$

for $i \in [n]$ **do**
  $Q_i \leftarrow F_i - G_i$
return $\mathbf{Q}$

Algorithm 3: SWMult for $n = d + e + 1$ [12, Algorithm 2]

Input : Encodings $\mathbf{F}$ and $\mathbf{G}$
Output : Encoding $\mathbf{Q}$ s.t. $\mathrm{Dec}(\mathbf{Q}) = \mathrm{Dec}(\mathbf{F}) \cdot \mathrm{Dec}(\mathbf{G})$

for $i \in [n]$ **do**
  $Q_i \leftarrow F_i \cdot G_i$
return $\mathbf{Q}$

Algorithm 4: ZEnc_n^d for $n = d + e + 1$ and $k \leq d$ [1, Algorithm 3]

Output : Randomized encoding $\mathbf{G}$ s.t. $\mathrm{Dec}(\mathbf{g}) = (0_i)_{[k]}$

$(r_k, \ldots, r_d) \leftarrow_{\$} \mathbb{F}^{d+1-k}$;
for $i \in [n]$ **do**
  for $j \in [k, d+1]$ **do**
    $G_i \leftarrow G_i + r_j M_{i,j}$
return $\mathbf{G}$

Algorithm 5: sZEnc_n^d for $n = d + e + 1$ and $k \leq d$ [21, Algorithm 4]

Output : Randomized encoding $\mathbf{G}$ s.t. $\mathrm{Dec}(\mathbf{g}) = (0_i)_{[k]}$

$\mathrm{Y} \leftarrow \mathsf{ZEnc}_n^d$
for $j \in [1, d+1-k]$ **do**
  $\mathrm{G} \leftarrow \mathsf{ZEnc}_n^d$
  $\mathrm{Y} \leftarrow \mathsf{SWAdd}(\mathrm{Y}, \mathrm{G})$
return Y

Algorithm 6: Refresh for $n = d + e + 1$, and $k \leq d$ [21, Algorithm 6]

Input : Encoding $\mathbf{F}$
Output : Encoding $\mathbf{Q}$ s.t. $\mathrm{Dec}(\mathbf{Q}) = \mathrm{Dec}(\mathbf{F})$

$\mathrm{G} \leftarrow \mathsf{sZEnc}_n^d$
$\mathrm{Q} \leftarrow \mathsf{SWAdd}(\mathbf{F}, \mathrm{G})$
return Q

Algorithm 7: $\mathsf{opt\text{-}ZEnc}_n^{d,\mathrm{Offset}}$ for $n = d + e + 1$ and $k \leq d$

Output : Randomized encoding $\mathbf{G}$ s.t. $\mathrm{Dec}(\mathbf{g}) = (0_i)_{[k]}$

$(G_0, \ldots, G_{\mathrm{Offset}-1}) \leftarrow 0^{\mathrm{Offset}}$
$(G_{\mathrm{Offset}}, \ldots, G_{d-k}) \leftarrow_{\$} \mathbb{F}^{d+1-k}$;
for $i \in [d+1-k, n]$ **do**
  for $j \in [\mathrm{Offset}, \ldots, d+1-k]$ **do**
    $G_i \leftarrow G_i + G_j \cdot \tilde{A}_{\mathrm{opt}}[i, j]$
return $\mathbf{G}$

Algorithm 8: opt-sZEnc$_n^d$ for $n = d + e + 1$ and $k \leq d$

Output: Randomized encoding $\mathbf{Y}$ s.t. Dec $(\mathbf{Y}) = (0_i)_{[k]}$

$\mathbf{Y} \leftarrow$ opt-ZEnc$_n^{d,\text{Offset}\leftarrow 0}$
for $j \in [1, d + 1 - k]$ **do**
  $\mathbf{G} \leftarrow$ opt-ZEnc$_n^{d,\text{Offset}\leftarrow j}$
  $\mathbf{Y} \leftarrow \text{SWAdd}(\mathbf{Y}, \mathbf{G})$
return $\mathbf{Y}$

Algorithm 9: opt-Refresh for $n = d + e + 1$, and $k \leq d$ [21, Algorithm 5]

Input : Encoding $\mathbf{F}$
Output: Encoding $\mathbf{Q}$ s.t. Dec $(\mathbf{Q})$ = Dec $(\mathbf{F})$

$\mathbf{G} \leftarrow$ opt-sZEnc$_n^d$
$\mathbf{Q} \leftarrow \text{SWAdd}(\mathbf{F}, \mathbf{G})$
return $\mathbf{Q}$

Algorithm 10: BGW for $n = 2d + e + 1$, $k \leq d$

Input : Encodings $\mathbf{F}, \mathbf{G}$
Output: Encoding $\mathbf{H}$ s.t. Dec $(\mathbf{H})$ = Dec $(\mathbf{F})$ · Dec $(\mathbf{G})$

$\mathbf{Q} \leftarrow \text{SWMult}(\mathbf{F}, \mathbf{G})$;
for $j \in [n]$ **do**
  $d_{enc} \leftarrow \begin{cases} d - j & \text{if } j < d + 1 - k, \\ d & \text{otherwise.} \end{cases}$
  $\mathbf{Y}^j \leftarrow$ opt-ZEnc$_n^{d_{enc}}$
for $i \in [n]$ **do**
  for $j \in [n]$ **do**
    $H_{i,j} \leftarrow \hat{\lambda}_{j,i}^{(\text{DegRed})} \cdot Q_j + Y_i^{(j)}$
    $H_i \leftarrow H_i + H_{i,j}$
return $\mathbf{H}$

Algorithm 11: SWComp for $n = d + e + 1$ and $k \leq d$ [1, Algorithm 9]

Input : Four sharings $\mathbf{H}^{(j)}$, with $j \in [4]$
Output: Sharing $\mathbf{Q}$ s.t. Dec $(\mathbf{Q}) = \sum_{j\in[4]} \text{Dec}\left(\mathbf{H}^{(j)}\right)$

$\mathbf{Q} \leftarrow$ opt-sZEnc$_n^d$
for $j \in [4]$ **do**
  $\mathbf{Q} \leftarrow \text{SWAdd}\left(\mathbf{Q}, \mathbf{H}^{(j)}\right)$
return $\mathbf{Q}$

Algorithm 12: LaOlaMult for $n = d + e + 1$ with $k \leq \lfloor \frac{d}{2} \rfloor$.

Input : Encodings $\mathbf{F}$ and $\mathbf{G}$
Output: Encoding $\mathbf{Q}$ s.t. Dec $(\mathbf{Q})$ = Dec $(\mathbf{F})$ · Dec $(\mathbf{G})$

$\mathbf{F}', \mathbf{F}'' \leftarrow \text{SplitRed}\left(\mathbf{F}, \lceil \frac{d}{2} \rceil\right)$
$\mathbf{G}', \mathbf{G}'' \leftarrow \text{SplitRed}\left(\mathbf{G}, \lfloor \frac{d}{2} \rfloor\right)$
$\mathbf{H}^{(0)} \leftarrow \text{SWMult}(\mathbf{F}', \mathbf{G}')$
$\mathbf{H}^{(1)} \leftarrow \text{SWMult}(\mathbf{F}', \mathbf{G}'')$
$\mathbf{H}^{(2)} \leftarrow \text{SWMult}(\mathbf{F}'', \mathbf{G}')$
$\mathbf{H}^{(3)} \leftarrow \text{SWMult}(\mathbf{F}'', \mathbf{G}'')$
$\mathbf{Q} \leftarrow \text{SWComp}\left(\mathbf{H}^{(0)}, \mathbf{H}^{(1)}, \mathbf{H}^{(2)}, \mathbf{H}^{(3)}\right)$
return $\mathbf{Q}$

Algorithm 13: SplitRed for $n = d + e + 1$ and $k \leq \lfloor \frac{d}{2} \rfloor$

Input : Encoding $\mathbf{F}$ and target degree d'
Output: Encodings $\mathbf{F}', \mathbf{F}''$ s.t. Dec (SWAdd $(\mathbf{F}', \mathbf{F}'')$) = Dec $(\mathbf{F})$ and Deg (SWAdd $(\mathbf{F}', \mathbf{F}'')$) = d'

if $d' = d$ **then**
  return $\mathbf{F}$, $\mathbf{0}$
for $j \in \lceil n/2 \rceil$ **do**
  $d_{enc} \leftarrow \begin{cases} d - j & \text{if } j < d + 1 - k, \\ d & \text{otherwise.} \end{cases}$
  $\hat{\mathbf{G}}^j \leftarrow$ opt-ZEnc$_n^{d_{enc}}$
for $j \in \lfloor n/2 \rfloor$ **do**
  $\mathbf{G}^j \leftarrow \hat{\mathbf{G}}^j$
if $n \bmod 2 = 1$ **then**
  $\mathbf{G}^{\lfloor n/2 \rfloor - 1} \leftarrow \text{SWAdd}\left(\mathbf{G}^{\lfloor n/2 \rfloor - 1}, \hat{\mathbf{G}}^{\lceil n/2 \rceil - 1}\right)$
for $j \in \lfloor n/2 \rfloor$ **do**
  for $i \in [n]$ **do**
    $\mathcal{F}_i'^j \leftarrow \hat{\lambda}_j^i \cdot F_j$
  $\mathcal{F}^j \leftarrow \text{SWAdd}\left(\mathcal{F}'^j, \mathbf{G}^j\right)$
  $\mathbf{F}' \leftarrow \text{SWAdd}\left(\mathbf{F}', \mathcal{F}^j\right)$
for $j \in \lceil n/2 \rceil$ **do**
  $d'_{enc} \leftarrow \begin{cases} d' - j & \text{if } j < d' + 1 - k, \\ d' & \text{otherwise.} \end{cases}$
  $\mathbf{G}^{j+\lfloor n/2 \rfloor} \leftarrow \text{SWSub}\left(\text{opt-ZEnc}_n^{d'_{enc}}, \hat{\mathbf{G}}^j\right)$
  for $i \in [n]$ **do**
    $\mathcal{F}_i'^{j+\lfloor n/2 \rfloor} \leftarrow \hat{\lambda}^i_{j+\lfloor n/2 \rfloor} \cdot F_{j+\lfloor n/2 \rfloor}$
  $\mathcal{F}^{j+\lfloor n/2 \rfloor} \leftarrow \text{SWAdd}\left(\mathcal{F}'^{j+\lfloor n/2 \rfloor}, \mathbf{G}^{j+\lfloor n/2 \rfloor}\right)$
  $\mathbf{F}'' \leftarrow \text{SWAdd}\left(\mathbf{F}'', \mathcal{F}^{j+\lfloor n/2 \rfloor}\right)$
return $\mathbf{F}'$, $\mathbf{F}''$

Algorithm 14: IPAdd_L for $n = d + 1$ [4, Algorithm 4]

Input : Sharings $\mathbf{X}$ and $\mathbf{Y}$
Output: Sharing $\mathbf{Z}$, such that $\langle \mathbf{L}, \mathbf{Z} \rangle = \langle \mathbf{L}, \mathbf{X} \rangle + \langle \mathbf{L}, \mathbf{Y} \rangle$

for $i \in [n]$ **do**
  $Z_i \leftarrow X_i + Y_i$
return $\mathbf{Z}$

Algorithm 15: $\mathsf{IPRefresh}_L$ for $n = d + 1$ [5, Algorithm 5]

Input : Sharing $\mathbf{X}$
Output: Sharing $\mathbf{X}'$, such that $\langle \mathbf{L}, \mathbf{X}' \rangle = \langle \mathbf{L}, \mathbf{X} \rangle$

$(A_1, \ldots, A_n) \leftarrow_{\$} \mathbb{F}$
$A_0 \leftarrow \sum_{j \in [n]} A_i \cdot L_i$
$\mathbf{X}' \leftarrow \mathsf{IPAdd}_L(\mathbf{X}, \mathbf{A})$
return $\mathbf{X}'$

Algorithm 16: $\mathsf{SecIPRefresh}_L$ for $n = d + 1$ [5, Algorithm 6]

Input : Sharing $\mathbf{X}$
Output: Sharing $\mathbf{Y}$, such that $\langle \mathbf{L}, \mathbf{Y} \rangle = \langle \mathbf{L}, \mathbf{X} \rangle$

$\mathbf{Y}_0 \leftarrow \mathbf{X}$
for $i \in [n]$ **do**
  $\mathbf{Y}_i \leftarrow \mathsf{IPRefresh}_L(\mathbf{Y}_{i-1})$
$\mathbf{Y} \leftarrow \mathbf{Y}_n$
return $\mathbf{Y}$

Algorithm 17: IPMult_L for $n = d + 1$ [5, Algorithm 3]

Input : Sharings $\mathbf{A}$ and $\mathbf{B}$
Output: Sharing $\mathbf{C}$, such that $\langle \mathbf{L}, \mathbf{C} \rangle = \langle \mathbf{L}, \mathbf{A} \rangle \cdot \langle \mathbf{L}, \mathbf{B} \rangle$

for $i \in [n]$ **do**
  for $j \in [n]$ **do**
    $T_{i,j} \leftarrow A_i \cdot B_j \cdot L_j$
for $i \in [n]$ **do**
  $U'_{i,i} \leftarrow 0$
  for $j \in [n]$ **do**
    if $i < j$ **then**
      $U_{i,j} \leftarrow_{\$} \mathbb{F}$
    if $i > j$ **then**
      $U_{i,j} \leftarrow -U_{j,i}$
    $U_{i,j} \leftarrow U'_{i,j} \cdot \delta_{i,j} L_i^{-1}$
for $i \in [n]$ **do**
  for $j \in [n]$ **do**
    $V_{i,j} \leftarrow T_{i,j} + U_{i,j}$
for $i \in [n]$ **do**
  $C_i \leftarrow \sum_{j \in [n]} V_{i,j}$
return $\mathbf{C}$