Polynomial sharings on two secrets: Buy one, get one free

Paula Arnold[1], Sebastian Berndt[2*], Thomas Eisenbarth[1] and Maximilian Orlt[3]

[1] Institute for IT security, University of Lübeck, Lübeck, {p.arnold,thomas.eisenbarth}@uni-luebeck.de
[2] Department of Electrical Engineering and Computer Science, Technische Hochschule Lübeck, Lübeck, sebastian.berndt@th-luebeck.de
[3] Department of Computer Science, Technical University of Darmstadt, Darmstadt, maximilian.orlt@tu-darmstadt.de

Abstract. While passive side-channel attacks and active fault attacks have been studied intensively in the last few decades, strong attackers combining these attacks have only been studied relatively recently. Due to its simplicity, most countermeasures against passive attacks are based on additive sharing. Unfortunately, extending these countermeasures against faults often leads to quite a significant performance penalty, either due to the use of expensive cryptographic operations or a large number of shares due to massive duplication. Just recently, Berndt, Eisenbarth, Gourjon, Faust, Orlt, and Seker thus proposed to use polynomial sharing against combined attackers (CRYPTO 2023). While they construct gadgets secure against combined attackers using only a linear number of shares, the overhead introduced might still be too large for practical scenarios.

In this work, we show how the overhead of nearly all known constructions using polynomial sharing can be reduced by nearly half by embedding *two* secrets in the *coefficients* of one polynomial at the expense of increasing the degree of the polynomial by one. We present a very general framework that allows adapting these constructions to this new sharing scheme and prove the security of this approach against purely passive side-channel attacks, purely active fault attacks, and combined attacks. Furthermore, we present new gadgets allowing us to operate upon the different secrets in a number of useful ways.

Keywords: Polynomial Masking · Parallel Computation · Leakage/Fault Resilience

1 Introduction

Implementations of cryptographic systems are among the most scrutinized pieces of software, as a single small implementation error can lead to a complete security loss. Furthermore, even if the implementation matches the mathematical description perfectly, physical properties of the implementation (outside the scope of the mathematical specifications) can have the same disastrous consequences. Typical examples include non-constant timing behavior or leakage via physical side-channels such as electromagnetic radiation or power consumption that are dependent on sensitive information. To protect against these physical side-channels, the most used countermeasure is *masking*. Here, a sensitive value v is split into a vector $(v_0, \dots, v_{n-1})$ of n values (called *shares*) such that a sufficiently high number of shares is required to reconstruct v. Masking has proved itself to be a very resilient

*Work was partially done while the author was affiliated with the University of Lübeck

countermeasure and is now widely deployed in cryptographic hardware. While several different masking techniques have been developed, the classical techniques of *additive* and *polynomial* masking are still the most widely used. In additive masking, we assume that the sensitive value belongs to a field $\mathbb{F}$ equipped with an addition $+$. Now, for $i = 0, \dots, n-2$, the values v_i are drawn uniformly from $\mathbb{F}$ and v_{n-1} is chosen such that $\sum_{i=0}^{n-1} v_i = v$. For polynomial masking, we construct a polynomial $\mathbf{p}(x) = v + \sum_{i=1}^{d} a_i x^i$ with a_i drawn randomly from $\mathbb{F}$. Then, v_i is given by $\mathbf{p}(\alpha_i)$ for some public support point $\alpha_i \neq 0$. Both of these approaches have several advantages and disadvantages. Probably the most commonly cited advantage of additive masking is that it is relatively efficient regarding complexity and randomness cost to protect against an attacker with access to $t \leq d$ shares. However, additive masking is quite vulnerable to *active* attacks that insert a *fault* into the computation [GIP+14]. Without any additional precautions, adding a value ζ to a single share v_i will simply result in a valid sharing of the value $v + \zeta$. This attack does not look particularly dangerous at first glance but can lead to catastrophic security flaws. A simple example is outputting the product of a secret key and zero. The result does not give any information about the key, however, since the attacker can choose the added value, an additive fault on the zero would reveal the complete key. Even though the example is very contrived, it illustrates the catastrophic effects of such faults. To prevent these attacks, complex verification operations are needed that are often not suited to low hardware resources or require at least $t \cdot \sigma$ shares to protect against t probes and σ faults on the shares [GIP+14, GLO+21, GSZ20, DN20, FRBSG22]. In contrast, polynomial masking has a natural resilience against such fault attacks as adding a value ζ to a single share v_i will result in a polynomial of degree higher than d. This allows to use only $O(d + \sigma)$ shares to achieve the same security as additive masking with $d \cdot \sigma$ shares even against *combined attackers* performing both side-channel probes and adding active faults [SFRES18, BEF+23]. Nevertheless, this linear number of shares (and thus the required area) used by polynomial masking is often too large for practical use cases.

1.1 Our Contribution

In this work, we show that we can adapt (nearly all) known constructions using polynomial masking to share *two* secrets in the *coefficients* of the polynomial, including constructions based on the classical BGW approach [BGW88] and the very recent approach of Berndt et al. [BEF+23]. They proposed a method, where each secret was masked with $d + e + 1$ shares to protect the circuits against d probes and e faults. Compared to other approaches, this significantly reduces the number of shares by nearly a factor of $1/2$. Now, suppose that one wants to perform the same computation using a gadget width depth Δ and n shares, but on two different secrets. Blockciphers are prototypical examples of this, as a round typically consists of the computation of many parallel identical s-boxes. When aiming to minimize the depth of the resulting circuit, we can simply run those two computations in parallel. Clearly, the depth of the resulting circuit is Δ, but we need to double the number of shares to $2n$. On the other hand, if we aim to minimize the number of used shares, we can run the two computations sequentially, but then have a depth of 2Δ. Our contribution shows how to perform these computations for two secrets using $n + 2$ shares and still remain on depth Δ. Hence, instead of computing each masked s-box on its own, we can apply our technique to compute the result of two different s-boxes by computing only a single s-box (although on $n + 2$ shares as we need to increase the degree of the underlying polynomial). Rather than using an area of $2\Delta n$, we now only use an area of $\Delta(n + 2) = \Delta n + 2\Delta$, which, for sufficiently large n, saves a factor of two. Since the s-boxes are the only non-linear part of such blockciphers, they constitute the bottleneck of masked implementations. Our techniques thus allow us to reduce this bottleneck by half in a black-box manner, and can easily be added to existing implementations without large overhead.

Moreover, we construct gadgets that are also secure against combined attackers. Using this construction, we obtain the best of both approaches: We simultaneously have a natural resilience to active attacks *and* a small number of shares. As a consequence of our advancements, the runtime (and randomness cost) associated with concurrent computations involving two secrets, each with a runtime (randomness cost) of $f(n)$, can be significantly reduced from $2f(n)$ to $f(n+2)$. To illustrate, consider the parallel execution of a circuit with a complexity function $f(n) = an^2$. Our construction results in a reduction from $2an^2$ to the more efficient $an^2 + 4an + 2a$.

To show the versatility of our approach, we first generalize several known protocols similar to the classical BGW approach [BGW88] using polynomial masking into a unified framework. Within this framework, we show how to embed both secrets simultaneously and show that in the case of AES, our approach already outperforms the state-of-the-art approaches using one secret per polynomial for small orders $t \geq 4$. We also show how to use the recent new multiplication gadget LaOla of Berndt et al. [BEF+23] with two secrets. Then, we show that our adaptions of the gadgets are (S)NI even when using two secrets. Furthermore, we also show that our approach is resilient against active fault attacks, similar to the setting studied in [SFRES18]. Finally, we show that our gadgets are *fault robust* [BEF+23], a recent security property that allows to securely compose fault resilient gadgets. Namely, we show that up to e faults can be either detected or corrected by our circuit.

Concluding our study, we leverage the analytical framework developed by Berndt et al. [BEF+23] to demonstrate that our constructions attain robust security against combined attacks. This is done by proving *fault-invariance* of our gadgets, a crucial property ensuring that the circuit does not leak secret information, even in the presence of faults. In simpler terms, the degree of information obtained about the secrets from side-channel leakage and the circuit's output remains constant, irrespective of the injected faults.

1.2 Structure of this Work

In Section 2, we establish the needed preliminaries, including our model of circuits, the capabilities of passive and active attackers, our threat model (along with corresponding security definitions), and give an overview about polynomial (or Shamir's) sharing that embeds the secret into the lowest coefficient of a polynomial. We also examine relevant related work. In Section 3, we discuss that the case of embedding the secret in the highest coefficient of a polynomial is also secure, while other positions might not hide the secret sufficiently. Then, we come to the main idea of our paper, called *double sharings*: We show that we can embed one secret into the lowest and another one in the highest coefficient *simultaneously* and only lose one degree of freedom. In Section 4, we focus on presenting gadgets that work on double sharings and guarantee security against passive probing attacks. It is relatively easy to construct a gadget for addition by simply applying the addition in a share-wise manner. Non-linear operations (such as multiplication) are, however, much more complicated. We first consider existing gadgets for non-linear operations that follow the classical approach due to Ben-Or, Goldwasser, and Wigderson [BGW88] (BGW): First, the non-linear transformation is applied locally in a share-wise manner. This results in the degree of the computed polynomial being too high, hence now, a degree reduction is performed where each share is re-shared and then recombined locally. We show that this approach also works in the double sharing setting in a general case. To illustrate the flexibility of our approach, we do not only consider the multiplication operation (which would be sufficient for completeness), but also show how to compute other useful operations in a similar efficient manner. Finally, we also consider another type of gadget that can be used for multiplication: The LaOla gadget of Berndt et al. [BEF+23] that does not follow the classical approach, as it first reduces the polynomials and then computes the non-linear transformation. Nevertheless, we show how to adapt the corresponding multiplication

gadget to also work in the double sharing scenario. Hence, the double sharing technique can also be applied to such non-BGW gadgets. Finally, in Section 5 we consider active attackers and combined attackers that are also allowed to induce faults. Here, we follow the approach used by Seker et al. [SFRES18] and by Berndt et al. [BEF+23]: We construct the gadgets in such a way that invalid (i.e., faulted) inputs will lead to invalid outputs with high probability. To do this, we extract the validity information (which correspond to the higher-order coefficients of the polynomials) and integrate those into the output of the computation. This integration will be performed by using *error propagation polynomials* that we add to the output. We first adapt the non-linear gadgets from the previous section by designing appropriate error propagation polynomials that allow to detect invalid sharings easily. To now show security against active attacks, we make use of the notion of fault invariance introduced in [BEF+23]. This, furthermore, allows us to use the results of Berndt et al. [BEF+23] to show that these gadgets are secure against combined attacks.

2 Preliminaries and Notation

Throughout this work, $n \in \mathbb{N}_{\geq 2}$ will always denote the number of shares on which we operate. Furthermore, $\mathbb{F}$ denotes some finite field with at least $n+1$ elements. If $(v_0, \dots, v_{n-1}) \in \mathbb{F}^n$ is a vector and $I \subseteq \{0, \dots, n-1\}$ is a subset of the indices, we define $v_I = (v_i)_{i \in I}$. For $n \in \mathbb{N}$, we also define $[n] = \{0, \dots, n-1\}$. We write $\delta_{a,b}$ to refer to the Kronecker delta with $\delta_{a,b} = 1$ if $a = b$, and $\delta_{a,b} = 0$ else. The Hamming weight $\mathsf{weight}(v)$ of a vector $v \in \mathbb{F}^n$ denotes the number of non-zero entries of v.

If $X_0, \dots, X_{n-1}$ are random variables, they are *d-wise independent*, if for all $I \subseteq \{0, \dots, n-1\}$ with $|I| \leq d$ and all values $(x_i)_{i \in I}$ in the domain of $(X_i)_{i \in I}$, we have

$$\Pr[\bigwedge_{i \in I} X_i = x_i] = \prod_{i \in I} \Pr[X_i = x_i].$$

2.1 Attacker Models

In this work, we consider both passive and active attackers. Passive attacks are side-channel attacks where the adversary learns some intermediate values in addition to the input-output behavior of a (cryptographic) implementation. A purely active attacker does not learn any intermediate values of the implementation, but can fault some internal values to perform unexpected calculations leading to vulnerabilities. Further, we consider adversaries running *combined* attacks that are allowed to fault and probe intermediate values simultaneously.

Circuits As usual, we represent the functionality that we want to compute as an *arithmetic circuit* C over a finite field $\mathbb{F}$, i.e., a directed acyclic graph $\mathsf{C} = (V, E)$ where each node $v \in V$ is labeled as *input gate*, *output gate*, *addition gate*, a *multiplication gate* or a *random gate*. When starting the computation, the input values $x_1, x_2, \dots$ are assigned to the input gates. Whenever all parents of a gate have a value, we compute the value of the gate by applying the underlying operation (i.e., addition or multiplication) to these values and assign the outcome of this operation to the current gate. Random gates do not have parents and produce a uniformly random element from $\mathbb{F}$. We write $\mathsf{C}(x_1, \dots)$ for the probability distribution of the output gates. We also assume that the circuit can output the *abort symbol* $\perp \notin \mathbb{F}$ to indicate that the computation aborts. If the randomness R used by the circuit is fixed, we denote this deterministic circuit by C^R.

Compiler In order to prevent the attacks, which will be explained in more depth later on, a common approach is to use a *compiler*. Such a compiler transforms a circuit C — possibly

vulnerable to attacks — into a circuit C' that resists these attacks. To do this, each gate $g \in \mathsf{C}$ is transformed into a small circuit G, called a *gadget*. If the inputs to g are $x_1, x_2, \ldots, x_r$ and the output is y, the small circuit G will be given *encodings* of $x_1, x_2, \ldots, x_r$ and produce an encoding of y. We denote these encodings of a value v by $[\![v]\!]$. Depending on the type of attack, these encodings will be chosen such that they resists the attack. For example, to prevent passive attacks using t probes, one could split each x_i into $t+1$ parts called shares such that they are t-wise independent. Hence, an attacker that only obtains t values can not reconstruct the (possibly sensitive) value of any x_i. The main challenge in designing such robust circuits C' is to prevent the leakage of sensitive information via intermediate results produced by the circuit. In order to construct a compiler, it is sufficient to show how to produce the gadgets for addition gates and multiplication gates. However, in this work we will also present more efficient gadgets for operations that we believe to be useful when designing gadgets for parallel computations.

Passive attacks A passive *t-probing attacker* A is given the circuit C and now chooses a subset of at most t wires w_i of this circuit and two inputs x_0 and x_1 to the circuit. Then, a random bit $b \leftarrow\!\!\$\ \{0,1\}$ is chosen and the computation of C on x_b is performed. Afterwards, the value on the t wires chosen by A are given to A and the attacker now outputs a bit b'. If $\Pr[b = b'] = 1/2$ for all attackers A (not necessarily time-bounded), we say that C is *perfectly secure* against t-probing attackers. For example, using Shamir's secret sharing [Sha79] and, e.g., the BGW protocol [BGW88], it is easy to see that we can take an arbitrary arithmetic circuit C on $\mathbb{F}$ and transform it into a circuit C' that is perfectly secure against t-probing attackers as long as $|\mathbb{F}| > 2t+1$. Both Shamir's secret sharing and the BGW protocol will be explained in depth later.

A very useful notion to show perfect security of a circuit is *non-inference* (NI) [BBD+16]. Intuitively, this security notion guarantees that all information gained by probes on intermediate values can already be gained by the same number of probes on the input. Now, if the input is shared with a sufficiently high degree, a subset of the input shares does not reveal anything about the secret input.

Definition 1 (NI [BBD+16])**.** A circuit G is t-NI if for any set of t_1 intermediate variables and any subset O of output indices with $t_1 + |O| \leq t$, there exists a subset of indices I with $|I| \leq t_1 + |O|$ such that the distribution of the t_1 intermediate variables and the output variables in O is perfectly simulatable from I.

At first glance, this definition seems strong enough to argue about security against probing attackers. However, it does not support *composability*, i.e., the concatenation of two t-NI gadgets is not necessarily t-NI. The stronger definition of *strong non-inference* (SNI) [BBD+16] supports composability. It thus suffices to prove that every gadget in the circuit is SNI to conclude that the complete circuit is perfectly secure against up to t probes.

Definition 2 (SNI [BBD+16])**.** A circuit G is t-SNI if for any set of t_1 intermediate variables and any subset O of output indices with $t_1 + |O| \leq t$, there exists a subset of indices I with $|I| \leq t_1$ such that the distribution of the t_1 intermediate variables and the output variables in O is perfectly simulatable from I.

Active attacks An active *σ-faulting attacker* A running a fault attack T is given the circuit C and chooses a subset of at most σ wires w_i along with corresponding values $v_i \in \mathbb{F}$. Furthermore, A chooses an input x to the circuit and the computation of C on x is performed. After one of the chosen wires w_i is computed, the value v_i is added to it. We denote the faulted circuit by $T[\mathsf{C}]$ and $|T| = \sigma$ will refer to the number of faulted wires. Let $y' \leftarrow T[\mathsf{C}]$ be the output of the faulted circuit and $y \leftarrow \mathsf{C}$ be the original output of the unfaulted circuit C (i.e., the one where all v_i are equal to 0). Then, C is

ϵ-*secure* against σ-faulting attackers if $\Pr[y' \in \{y, \perp\}] \geq 1 - \epsilon$ for all attackers A (not necessarily time-bounded). In [SFRES18], the authors propose to use a technique called *infective* computation that intuitively guarantees that errors introduced by the faults will spread over the complete computation. This was refined in [BEF+23] where the notion of *robustness* guarantees that an error does not result in an undetected faulted value. This notion was only introduced for polynomial sharings (which will be thoroughly discussed in the next section). Intuitively, this notion means that every fault attack only changing σ wires will either (i) only change at most σ positions of the encoding or (ii) behave like adding a sufficiently random polynomial to the encoding. In the former case, the bounded number of changes will be detectable due to the error-detecting capabilities of polynomial sharing, as the degree of the resulting polynomial is too high. In the latter case, the result will, with high probability, also be detectable because of the degree of the resulting polynomial being too high.

Definition 3 (σ-f-robust [BEF+23, Def. 9])**.** A gadget G with one output sharing and two input sharings and polynomial sharing $[\![\cdot]\!]$ is σ-*fault-robust* with respect to $\mathcal{F}$, if for any valid sharings $(x_0, \ldots, x_{n-1}) \in [\![x]\!]$ and $(x'_0, \ldots, x'_{n-1}) \in [\![x']\!]$, the output $(y_0, \ldots, y_{n-1}) \leftarrow \mathsf{G}((x_0, \ldots, x_{n-1}), (x'_0, \ldots, x'_{n-1}))$ is also valid. Further, it holds for any fault vectors $(v_0, \ldots, v_{n-1}) \in [\![v]\!]$, $(v'_0, \ldots, v'_{n-1}) \in [\![v']\!]$, and any $T \in A(\mathcal{F})$ with $|T| \leq \sigma$ and $(y_i + w_i + w'_i)_{i\in[n]} \leftarrow T[\mathsf{G}]\left((x_i + v_i)_{i\in[n]}, (x'_i + v'_i)_{i\in[n]}\right)$, that there are numbers t_1 and t_2 with $t_1 + t_2 \leq |T|$ such that

(*i*) $\mathsf{weight}(w) \in [0, t_1] \cup [\mathsf{weight}(v + v') - t_1, \mathsf{weight}(v + v') + t_1]$, where $\mathsf{weight}(\cdot)$ of a vector is the number of its non-zero elements,

(*ii*) and $(w'_0, \ldots, w'_{n-1})$ is the zero vector or produced by the following random experiment: A polynomial $\mathbf{p}_{w'} \in \mathbb{F}[x]$ is chosen such that the coefficients of x^{d+1}, x^{d+2}, ..., x^{n-t_2} are drawn uniformly at random from $\mathbb{F}$. Then, $w'_i = \mathbf{p}_{w'}(\alpha_i)$ for some pairwise different points $\alpha_i \in \mathbb{F} \setminus \{0\}$.

We stress here that Definition 9 in [BEF+23] only had the condition $\mathsf{weight}(w) \in [\mathsf{weight}(v + v') - t_1, \mathsf{weight}(v + v') + t_1]$, but the case of $\mathsf{weight}(w) \in [0, t_1]$ is also needed to capture non-linear gadgets and was accidentally ommited [BO]. The analysis presented in [BEF+23], however, also holds for this generalized definition.

As described above, we only consider additive and non-adaptive faults and will denote the set of additive fault functions by $\mathcal{F}^+$. As shown in [BEF+23], it is not sufficient to show both leakage and fault resilience to prove security against combined attacks. However, the authors also presented properties under which individual security against leakages and faults can be lifted to security against combined attack. Intuitively, these properties guarantee that all faults induced in a gadget can either be pushed to inputs of the gadgets or to their outputs. Next, we formalize these properties.

Combined attacks We can also combine the attack types to get a (t, σ)-*combined attacker* that is again given a circuit C. Now, the attacker chooses t wires w_i of the circuit and another σ wires w'_i along with values $v'_i \in \mathbb{F}$. Furthermore, the attacker chooses two inputs x_0 and x_1 for the circuit. Then, a random bit $b \leftarrow\$ \{0, 1\}$ is chosen and the computation of C on x_b is performed. Whenever one of the chosen wires w'_i is computed, the value v'_i is added to it. We denote the output of the faulted circuit by y'_b and by y_b the original output of the circuit (i.e., the one where all v'_i are equal to 0). Now, the attacker is given the values assigned to w_i and $(\delta_{y'_b, y_b}, \delta_{y'_b, \perp})$ and needs to output a bit b'. We say that C is ϵ-*secure* against (t, σ)-combined attackers if both $\Pr[b = b'] = 1/2$ and $\Pr[y'_b \in \{y_b, \perp\}] \geq 1 - \epsilon$ for all attackers A (not necessarily time-bounded).

In the probing-only case, careful composition of (S)NI gadgets allowed to guarantee security of the composition. Similarly, we can use adapted (S)NI properties to achieve

composability results also in the case of an attacker performing combined attacks. The notion of fault-resilience guarantees that the presence of faults in itself can not lead to the leakage of sensitive information, i.e., the (S)NI properties remain true in the presence of faults.

Definition 4 (Fault-resilient SNI [BEF+23, Def. 5])**.** A gadget G is *t-fault-resilient (strong-) non-interfering (t-fr(S)NI)* with respect to $\mathcal{F}$ if $T[\mathsf{G}]$ is t-(S)NI for any fault attack $T \in A(\mathcal{F})$.

In [BEF+23], the authors have shown that faults can destroy the probing-only (S)NI security of gadgets. For this reason, they have established a property that guarantees that faults do not affect the probing security.

Definition 5 (Fault invariance [BEF+23, Def. 8])**.** A circuit C is *fault invariant* with respect to a fault set $\mathcal{F}$ if for any $T \in A(\mathcal{F})$, any intermediate value f in C^R and the according value f' in $T[\mathsf{C}^R]$, there are $\zeta, \zeta_0, \zeta_1, \dots \zeta_{k-1} \in \mathcal{F}$ such that it holds

$$f'^R(x_0, x_1, \dots x_{m-1}) = \zeta(f^R(\zeta_0(x_0), \zeta_1(x_1), \dots \zeta_{m-1}(x_{m-1})))$$

for any input $(x_0, x_1, \dots x_{m-1})$ and randomness R.

More precisely, fault invariance can lift the pure probing security up to security against combined attacks, as it intuitively guarantees that faults applied to a gadget can be pushed either to the input or the output of the gadget. In detail, a gadget is automatically fault-resilient SNI if it is SNI *and* fault invariant.

Lemma 1 ([BEF+23, Cor. 1])**.** *A gadget G that is SNI and fault invariant is fault-resilient SNI.*

Finally, the combination of fault resilience and fault robustness allows us to give meaningful security guarantees against combined attackers.

Lemma 2 ([BEF+23, Thm. 5])**.** *A gadget G that is t-fault-resilient (S)NI and σ-fault-robust is $|\mathbb{F}|^{\omega-\sigma-1}$-secure against (t,ω)-attackers with $\omega \leq \sigma$.*

In this work, we only consider additive faults $\mathcal{F}^+$. However, in [BEF+23] it was shown that fault-resilient (S)NI gadgets with respect to additive faults are also fault-resilient (S)NI with respect to wire-independent faults[1] if the gadget is fault invariant with respect to additive faults and all faults are counted additionally as probes.

2.2 Polynomials

Throughout this work, we will always write polynomials in bold font. If $\mathbf{f} \in \mathbb{F}[x]$ is a polynomial, f_i denotes the i-th coefficient, i.e., $\mathbf{f}(x) = \sum_{i=0}^{\deg(\mathbf{f})} f_i x^i$. Furthermore, we will write F_i for $\mathbf{f}(\alpha_i)$ where $\alpha_0, \dots, \alpha_{n-1} \in \mathbb{F} \setminus \{0\}$ will denote some pairwise distinct elements of $\mathbb{F}$ used as support points of the polynomial sharing. The input of a gadget will usually be denoted by $\mathbf{f}$ (and $\mathbf{g}$ in the case of gadgets with two inputs) and the output by $\mathbf{q}$.

For values $\alpha_0, \dots, \alpha_{n-1} \in \mathbb{F}$, we denote the Vandermonde matrix that has n rows and $k+1$ columns by

$$\mathsf{Vandermonde}_k(\alpha_0, \dots, \alpha_{n-1}) = \begin{pmatrix} 1 & \alpha_0 & \alpha_0^2 & \dots & \alpha_0^k \\ 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^k \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_{n-1} & \alpha_{n-1}^2 & \dots & \alpha_{n-1}^k \end{pmatrix}.$$

[1] Wire-independent faults are faults where the wires can be arbitrarily but independently faulted.

We note here that the *columns* of the Vandermonde are treated as the basis and not the rows. If the matrix is square (i.e., $k = n-1$), we simply write $\mathsf{Vandermonde}(\alpha_0, \dots, \alpha_{n-1})$. Regarding the invertibility of this matrix, it is well-known that $\det(\mathsf{Vandermonde}(\alpha_0, \dots, \alpha_{n-1})) = \prod_{i<j}(\alpha_i - \alpha_j)$. For pairwise distinct points, this matrix is thus invertible and we will denote this inverse matrix by $\mathsf{Vandermonde}^{-1}(\alpha_0, \dots, \alpha_{n-1})$. Further, we write $(\lambda_{0,i-1}, \dots, \lambda_{n-1,i-1})$ to refer to the i^{th} row of $\mathsf{Vandermonde}^{-1}(\alpha_0, \dots, \alpha_{n-1})$. The invertibility of the Vandermonde matrix directly implies the following lemma that we will use in many places throughout this work.

Lemma 3 (Polynomial interpolation [Kal84])**.** *Every polynomial* $\mathbf{f}$ *of degree at most* d *can be interpolated from* $d+1$ *distinct point-value pairs* $(\alpha_i, \mathbf{f}(\alpha_i))$*.*

Note that the Vandermonde matrix can be used both to evaluate a degree-k polynomial $\mathbf{p}(x)$ with coefficients $p_0, \dots, p_k$ at the n points $\alpha_0, \dots, \alpha_{n-1}$ by

$$\mathsf{Vandermonde}_k(\alpha_0, \dots, \alpha_{n-1}) \cdot \begin{pmatrix} p_0 \\ \vdots \\ p_k \end{pmatrix} = \begin{pmatrix} \mathbf{p}(\alpha_0) \\ \vdots \\ \mathbf{p}(\alpha_{n-1}) \end{pmatrix}$$

and to recover all k coefficients from $k+1$ distinct point-value pairs

$$\mathsf{Vandermonde}^{-1}(\alpha_0, \dots, \alpha_k) \cdot \begin{pmatrix} \mathbf{p}(\alpha_0) \\ \vdots \\ \mathbf{p}(\alpha_k) \end{pmatrix} = \begin{pmatrix} p_0 \\ \vdots \\ p_k \end{pmatrix},$$

so a specific coefficient can be computed as

$$p_j = \sum_{i=0}^{k} \lambda_{i,j} \mathbf{p}(\alpha_i). \tag{1}$$

2.3 Shamir's Secret Sharing

In order to share a value $s \in \mathbb{F}$ via polynomial sharing using a polynomial of degree d at coefficient $\chi \in [d+1]$, we choose random elements $(a_i)_{i \in [d+1] \setminus \{\chi\}}$ with $a_i \leftarrow\!\!\$\ \mathbb{F}$ and construct the polynomial $\mathbf{p}_{\chi \to s} \in \mathbb{F}[x]$ with $\mathbf{p}_{\chi \to s}(x) = x^\chi s + \sum_{i \in [d+1] \setminus \{\chi\}} a_i x^i$. Then, we compute the vector $(v_0, \dots, v_{n-1})$ with $v_i = \mathbf{p}_{\chi \to s}(\alpha_i)$. To simplify notation, this complete procedure will be denoted as $(v_0, \dots, v_{n-1}) \leftarrow\!\!\$\ \mathsf{Share}_d(\chi \to s)$. We stress here that $\mathsf{Share}_d(\chi \to s)$ thus constructs a *random* polynomial $\mathbf{p}_{\chi \to s}$, where the randomness is given by the random choices of a_i (but $a_\chi = s$ is fixed). We note that the random variable $\mathcal{P}_{\chi \to s}$ describing this random polynomial is a random variable on the randomness space $\mathbb{F}^d$.

Furthermore, the affine subspace of $\mathbb{F}^n$ induced by the outputs of $\mathsf{Share}_d(\chi \to s)$ is denoted by $[\![\chi \to s]\!]_d$, i.e.,

$$[\![\chi \to s]\!]_d = \mathrm{supp}\left(\mathsf{Share}_d(\chi \to s)\right),$$

where supp denotes the *support* of a probability distribution. To reconstruct s from $(v_0, \dots, v_{n-1}) \in [\![\chi \to s]\!]_d$, we can first reconstruct $\mathbf{p}$ by taking any $d+1$ distinct elements $(v_i)_{i \in I}$ and multiplying them with $\mathsf{Vandermonde}^{-1}(\alpha_I)$. This recovers all coefficients of $\mathbf{p}$ and thus $a_\chi = s$. To simplify notation, this complete procedure will be denoted as $s = \mathsf{Open}_\chi(v_I)$.

For the sake of completeness, the following lemma shows the well known fact that this sharing is secure for $\chi = 0$, in which it is typically known as *Shamir's secret sharing*.

Lemma 4 (Shamir's secret sharing [Sha79])**.** *Let* $s \in \mathbb{F}$ *be a secret, pairwise distinct* $\alpha_0, \dots, \alpha_{n-1} \in \mathbb{F} \setminus \{0\}$ *and* $d < n$*. Then it holds:*

(i) For any subset $I \subseteq \{0, \dots, n-1\}$ with $|I| \leq d$, the set of random variables $\{\mathcal{P}_{0\to s}(\alpha_i)\}_{i\in I}$ is a set of uniformly independent random variables independent of s.

(ii) For any subset $I \subseteq \{0, \dots, n-1\}$ with $|I| > d$, we can reconstruct s via $\mathsf{Open}_0(v_I)$ for all $(v_0, \dots, v_{n-1}) \in [\![0 \to s]\!]_d$.

Proof. Remember that $\mathcal{P}_{0\to s}$ is the random variable on $\mathbb{F}^d$ describing the polynomial used in the computation of $\mathsf{Share}_d(0 \to s)$ by the random choice of the coefficients $a_1, \dots, a_d$. Let $(\mathcal{A}_1, \dots, \mathcal{A}_d)$ be the random variable on $\mathbb{F}^d$ describing these coefficients $a_1, \dots, a_d$. Clearly, $\{\mathcal{A}_i\}_{i\in\{1,\dots,d\}}$ is a set of uniform independent random variables independent from s.

We will first show (i) and thus need to prove that $\{\mathcal{P}_{0\to s}(\alpha_i)\}_{i\in I}$ is a set of uniform independent random variables independent from s for any subset $I \subseteq \{0, \dots, n-1\}$ with $|I| \leq d$. Without loss of generality, we only show this for $|I| = d$, as this clearly also implies the lemma for $|I| < d$. To prove (i), we show that there is a bijection between the outcomes of $\{\mathcal{P}_{0\to s}(\alpha_i)\}_{i\in I}$ and $\{\mathcal{A}_i\}_{i\in\{1,\dots,d\}}$, which directly implies the result due to the randomness of $\{\mathcal{A}_i\}_{i\in\{1,\dots,d\}}$.

Let $V = \mathsf{Vandermonde}(0, \alpha_I)$ be the quadratic $(|I|+1) \times (|I|+1)$-Vandermonde matrix on 0 and the support points indexed by I. Clearly, we have $V \cdot (s, (\mathcal{A}_i)_{i\in\{1,\dots,d\}}) = (s, (\mathcal{P}_{0\to s}(\alpha_i))_{i\in I})$, as the first row of the matrix V is of the form $(1, 0, 0, \dots, 0)$. Furthermore, V is invertible as the α_i are pairwise distinct and not zero. Hence, the described linear transformation is a bijection on $\mathbb{F}^{d+1}$. In other words, for any outcome $(s, (\mathbf{p}_{0\to s}(\alpha_i))_{i\in I})$ exists exactly one outcome $(s, (a_1, \dots, a_d))$ such that

$$(s, (\mathbf{p}_{0\to s}(\alpha_i))_{i\in I}) = V \cdot (s, (a_1, \dots, a_d)) \ .$$

Since V is the identity function in the first coordinate, it follows that for each s', there is also a bijection $V_{s'}$ on $\mathbb{F}^d$ with

$$(\mathbf{p}_{0\to s}(\alpha_i))_{i\in I} = V_{s'} \cdot (a_1, \dots, a_d) \ .$$

As $\{\mathcal{A}_i\}_{i\in\{1,\dots,d\}}$ is a set of uniform independent random variables independent from s, so is $\{\mathcal{P}_{0\to s}(\alpha_i)\}_{i\in I}$.

The correctness of (ii) follows from Lemma 3. Interpolating the polynomial immediately gives the first coefficient $a_0 = s$. □

2.4 Computing on Shares

Our above discussion shows how to embed a secret into a polynomial. Here, we will discuss how to perform operations on such a shared representation.

As polynomial masking is linear, we have

$$[\![\chi \to s]\!]_d + [\![\chi \to s']\!]_d = [\![\chi \to (s + s')]\!]_d.$$

Hence, if we have $(v_0, \dots, v_{n-1}) \in [\![\chi \to s]\!]_d$ and $(v'_0, \dots, v'_{n-1}) \in [\![\chi \to s']\!]_d$, then $(v_0 + v'_0, \dots, v_{n-1} + v'_{n-1}) \in [\![\chi \to (s + s')]\!]_d$. Consequently, addition gates can be computed in a share-wise manner, such that shares with different indices are not mixed. The main complication thus comes from the non-linear multiplication gates. A classical approach to multiply $(v_0, \dots, v_{n-1}) \in [\![0 \to s]\!]_d$ and $(v'_0, \dots, v'_{n-1}) \in [\![0 \to s']\!]_d$, which we will refer to as the BGW[2] protocol, is the following.

First, compute $(w_0, \dots, w_{n-1}) = (v_0 \cdot v'_0, \dots, v_{n-1} \cdot v'_{n-1})$. While the underlying polynomial now has the correct lowest coefficient $s \cdot s'$, the degree is $2d$, i.e., $(w_0, \dots, w_{n-1}) \in [\![0 \to s \cdot s']\!]_{2d}$. To reduce the degree, the shares are re-shared: First, one computes $(w_0^{(i)}, \dots, w_{n-1}^{(i)}) \leftarrow\!\$\ \mathsf{Share}_d(0 \to w_i)$ for $i = 0, \dots, n-1$. Then, it is easy to see that $(\sum_{i=0}^{n-1} \lambda_{i,0} w_0^{(i)}, \dots, \sum_{i=0}^{n-1} \lambda_{i,0} w_{n-1}^{(i)}) \in [\![0 \to s \cdot s']\!]_d$, i.e., it is a valid sharing using a polynomial of degree d. Here, $(\lambda_{0,0}, \dots, \lambda_{n-1,0})$ is the first row of $\mathsf{Vandermonde}^{-1}(\alpha_0, \dots, \alpha_{n-1})$.

[2]To be precise, the protocol described here is an improvement of the one by [BGW88] due to [GRR98].

2.5 Related Work

Packed secret sharing A closely related approach to our technique is known as *packed secret sharing* [FY92] and was used by Grosso et al. [GSF14] to speed up masked implementations. The main differences between our technique and packed secret sharing is where the additional, second secret is hidden. In our approach, we will hide two secrets s_0 and s_1 by constructing a polynomial $\mathbf{p}(x) = s_0 + \sum_{i=1}^{d-1} a_i x^i + s_1 x^d$ for randomly chosen values a_i, i.e., we hide both secrets in the *coefficients* of a polynomial. In contrast, packed secret sharing hides both secrets at the value of the polynomial at specific *support points* of the polynomial, i.e., they construct a sufficiently random polynomial $\mathbf{p}$ such that $\mathbf{p}(0) = s_0$ and $\mathbf{p}(\alpha^*) = s_1$ for some special element α^* (which is then excluded from the set of public support points).

While the differences between these approaches look rather small, they have a large consequence with regard to practicability. First, to share a secret with our technique, we can still simply sample random elements and *evaluate* the corresponding polynomial. In contrast, in packed secret sharing, to share a secret, a polynomial needs to be *interpolated*. From a theoretical point of view, both evaluation and interpolation have the same complexity by using appropriate FFT algorithms, but from a practical point of view, this change in approach requires a large number of modifications and will give worse concrete running time. Second, the *degree reduction* that needs be performed after the share-wise multiplication becomes much more complicated. In the original non-packed version, one could simply erase all of the monomials larger than d (by sharing the resulting shares and performing a corresponding interpolation). As discussed by Grosso et al. [GSF14], this task becomes much more difficult in the packed setting. In order to avoid these drawbacks, Grosso et al. adapt a proposal by Damgård et al. [DIK10] that first blinds the secrets contained in the polynomial, then opens these blinded values, re-shares them, and removes the blinding later on. Unfortunately, opening the blinded values makes this approach vulnerable to active attacks, as these blinded values can now be faulted by an active attacker. Hence, the natural resilience of polynomial sharing to active attacks is lost this way. Furthermore, some optimization such as the fast computation of squarings due to Roche and Prouff [PR11] do not seem to applicable to packed secret sharing.

Code-based masking Embedding multiple secrets into a single sharing has also been used in the context of *code-based masking* [WMCS20]. Here, the authors use the term *amortization* for this idea. In general, the authors use linear error correcting codes to encode the sensitive values. They show that, under certain circumstances, the *generic encoder* corresponding to such codes provides security against passive attacks and present a multiplication gadget working on such encodings. By choosing a high-rate code, they are able to encode multiple secrets into a single codeword, which allows them to apply their multiplication gadget to multiple secrets simultaneously.

The authors also argue that the redundancy in the encodings allows to guarantee security against fault attacks, but only against faulted inputs or outputs of gadgets. The inner workings of the gadgets are assumed to be tamper-resistant (see [WMCS20, Appendix C] for a more thorough discussion) in their security analysis. We stress here that this assumption is absolutely necessary for their security analysis: In order to obtain efficient gadgets, their multiplication gadget first converts the codewords into an additive sharing, then computes an multiplication on this additive sharing, and finally encodes the resulting additive sharing back into a codeword. As additive sharings are highly vulnerable against fault attacks (as described in the introduction), an attacker could easily modify the shared values and thus obtain valid encodings of incorrect values.

3 Security of Two-Secret Sharing

In this section, we show that we can also embed two secrets simultaneously in a polynomial sharing, one in the coefficient a_0 and another one in a_d. For a better understanding, we first discuss the known result that we can also embed a secret s of a polynomial into the highest coefficient a_d instead of the lowest coefficient a_0. Further, we give an example why this, in general, only works securely for the highest and the lowest coefficient. Finally, we contribute our security proof for two simultaneously embedded secrets.

3.1 Hiding the Secret in Another Coefficient

We now discuss which of the coefficients are suitable for embedding a secret.

Using the highest coefficient We first consider to hide s in the highest coefficient a_d, which leads to a secure sharing.

Lemma 5 (E.g., [LD04])**.** *Let $s \in \mathbb{F}$ be a secret, pairwise distinct $\alpha_0, \dots, \alpha_{n-1} \in \mathbb{F} \setminus \{0\}$ and $d < n$. Then it holds*

(i) For any subset $I \subseteq \{0, \dots, n-1\}$ with $|I| \leq d$, the set of random variables $\{\mathcal{P}_{d \to s}(\alpha_i)\}_{i \in I}$ is a set of uniformly independent random variables independent of s.

(ii) For any subset $I \subseteq \{0, \dots, n-1\}$ with $|I| > d$, we can reconstruct s via $\mathsf{Open}_d(v_I)$ for all $(v_0, \dots, v_{n-1}) \in [\![d \to s]\!]_d$.

We stress here that this is a known result, though we also provide our own proof in Appendix A that some readers might find helpful for understanding the very similar proof for embedding two secrets.

Using arbitrary coefficients While the first and the last coefficient can be used to hide a secret in a secret sharing scheme, this, in general, does *not* hold true for the remaining coefficients: If they can be used depends on whether the considered set of support points are in a specific relation to each other [LD04]. The main obstacle here is the fact that the resulting submatrix (which consists of some columns of $\mathsf{Vandermonde}(\alpha_0, \dots, \alpha_{n-1})$) might not be invertible.

To show the severity of using "bad" support points, we consider an example taken from [ph13]. Consider the finite field GF(5) on five elements and $n = 3$ shares with support points $\alpha_0 = 1$, $\alpha_1 = 4$, and $\alpha_2 = 3$. Now, if the attacker obtains (v_0, v_1) of $(v_0, v_1, v_2) \in [\![1 \to s]\!]_2$ such that $v_0 = 1$ and $v_1 = 0$, they can immediately conclude that $s = 3$, as all polynomials $\mathbf{p}(x) = a_2x^2 + a_1x + a_0$ over GF(5) with $\mathbf{p}(1) = 1$ and $\mathbf{p}(4) = 0$ are the following:

$$0x^2 + 3x + 3, \quad 1x^2 + 3x + 2, \quad 2x^2 + 3x + 1, \quad 3x^2 + 3x + 0, \quad 4x^2 + 3x + 4.$$

Hence, knowledge about the two shares v_0 and v_1 might completely give information about the secret s embedded in a_1 in certain situations.

3.2 Two for One: Two Secrets in One Polynomial

Previously, we have seen that it is possible to embed the secret s in the coefficient of the lowest monomial $\chi = 0$ (as sx^0) or the highest monomial $\chi = d$ (as sx^d). In the following, we will embed two secrets s_0 and s_1 *simultaneously* into the coefficients of the polynomial where we will embed s_0 in position $\chi = 0$ and s_1 in position $\chi = d$. We thus generalize our notation in a straightforward way by writing $(v_0, \dots, v_{n-1}) \leftarrow \mathsf{Share}_d((0, d) \to (s_0, s_1))$ for

the embedding, $(s_0, s_1) = \mathsf{Open}_{0,d}(v_I)$ for the reconstruction, and $[\![(0,d) \to (s_0,s_1)]\!]_d$ for the affine subspace induced by this sharings. Note that the existence of $\mathsf{Open}_{0,d}$ directly follows from Lemma 3, as interpolation reconstructs the complete polynomial.

Lemma 6. *Let $s_0, s_1 \in \mathbb{F}$ be two secrets, pairwise distinct $\alpha_0, \dots, \alpha_{n-1} \in \mathbb{F} \setminus \{0\}$ and $d < n$. Then it holds*

(i) For any subset $I \subseteq \{0, \dots, n-1\}$ with $|I| \leq d-1$, the set of random variables $\{\mathcal{P}_)(\alpha_i)\}_{i \in I}$ is a set of uniformly independent random variables independent of s_0, s_1.

(ii) For any subset $I \subseteq \{0, \dots, n-1\}$ with $|I| > d$, we can reconstruct s_0 and s_1 via $\mathsf{Open}_{0,d}(v_I)$ for all $(v_0, \dots, v_{n-1}) \in [\![(0,d) \to (s_0,s_1)]\!]_d$.

(iii) For any subset $I \subseteq \{0, \dots, n-1\}$ with $|I| = d$, we can reconstruct s_{1-b} if s_b and v_I is known for all $(v_0, \dots, v_{n-1}) \in [\![(0,d) \to (s_0,s_1)]\!]_d$.

Proof. Let $\mathcal{P}_{(0,d)\to(s_0,s_1)}$ be the random variable on $\mathbb{F}^{d-1}$ describing the polynomial used in the computation of $\mathsf{Share}_d((0,d) \to (s_0,s_1))$ by the random choice of the coefficients $a_1, \dots, a_{d-1}$. Let $(\mathcal{A}_1, \dots, \mathcal{A}_{d-1})$ be the random variable on $\mathbb{F}^{d-1}$ describing these coefficients $a_1, \dots, a_{d-1}$. Clearly, $\{\mathcal{A}_i\}_{i \in \{1,\dots,d-1\}}$ is a set of uniform independent random variables independent from s_0 and s_1.

We will first show (i) and thus need to prove that $\{\mathcal{P}_{(0,d)\to(s_0,s_1)}(\alpha_i)\}_{i \in I}$ is a set of uniform independent random variables independent from s_0 and s_1 for any subset $I \subseteq \{0, \dots, n-1\}$ with $|I| \leq d-1$. Without loss of generality, we only show this for $|I| = d-1$, as this clearly also implies the lemma for $|I| < d-1$. To prove (i), we show that there is a bijection between the outcomes of $\{\mathcal{P}_{(0,d)\to(s_0,s_1)}(\alpha_i)\}_{i \in I}$ and $\{\mathcal{A}_i\}_{i \in \{1,\dots,d-1\}}$, which directly implies the result. Let $V' = \mathsf{Vandermonde}_d(0, \alpha_I)$ be the $d \times (d+1)$-Vandermonde matrix and V be the $(d+1) \times (d+1)$-matrix where we add a first row of $(0, 0, \dots, 0, 1)$ to V'. To show that V is invertible, we use the Laplace expansion and expand along the first row. The resulting determinant is $(-1)^{d+2} \cdot \det(\mathsf{Vandermonde}(0, \alpha_I))$ and thus not zero, as all α_i are distinct and non-zero. Clearly, we have $V \cdot (s_0, (\mathcal{A}_i)_{i \in \{1,\dots,d-1\}}, s_1) = ((\mathcal{P}_{(0,d)\to(s_0,s_1)}(s_0, \alpha_i), s_1)_{i \in I})$, as the first row of the matrix V is of the form $(0, 0, \dots, 0, 1)$. Hence, the described linear transformation is a bijection on $\mathbb{F}^{d+1}$. In other words, for any outcome $((\mathbf{p}_{(0,d)\to(s_0,s_1)}(\alpha_i))_{i \in I}, s)$, there exists exactly one outcome $((a_1, \dots, a_{d-1}), s)$ such that

$$((\mathbf{p}_{(0,d)\to(s_0,s_1)}(s_0, \alpha_i))_{i \in I}, s_1) = V \cdot (s_0, (a_1, \dots, a_{d-1}), s_1) \ .$$

Since V is the identity function in both the first and the last coordinate, it follows that for each $(s_0', s_1') \in \mathbb{F}^2$, there is also a bijection $V_{s_0', s_1'}$ on $\mathbb{F}^{d-1}$ with

$$(\mathbf{p}_{(0,d)\to(s_0,s_1)}(\alpha_i))_{i \in I} = V_{s_0', s_1'} \cdot (a_1, \dots, a_{d-1}) \ .$$

As $\{\mathcal{A}_i\}_{i \in \{1,\dots,d-1\}}$ is a set of uniform independent random variables independent from s_0, s_1, so is $\{\mathcal{P}_{(0,d)\to(s_0,s_1)}(\alpha_i)\}_{i \in I}$.

Similar to the proof above, Lemma 3 implies that (ii) holds true as any set of at least $d+1$ points allows interpolating the underlying polynomial, which in turn gives both the first and the last coefficient.

For (iii), we consider the *linear* relation between the coefficients $s_0, a_1, \dots, a_{d-1}, s_1$ and the point-values $(\mathbf{p}_{(0,d)\to(s_0,s_1)}(\alpha_0), \mathbf{p}_{(0,d)\to(s_0,s_1)}(\alpha_1), \dots, \mathbf{p}_{(0,d)\to(s_0,s_1)}(\alpha_{n-1}))$. If $I = \{i_0, \dots, i_{d-1}\}$, we have

$$\mathsf{Vandermonde}(\alpha_{i_0}, \dots, \alpha_{i_{d-1}}) \cdot \begin{pmatrix} s_0 \\ a_1 \\ \vdots \\ a_{d-1} \\ s_d \end{pmatrix} = \begin{pmatrix} \mathbf{p}_{(0,d)\to(s_0,s_1)}(\alpha_{i_0}) \\ \vdots \\ \mathbf{p}_{(0,d)\to(s_0,s_1)}(\alpha_{i_{d-1}}) \end{pmatrix} .$$

Hence, we can write the relation as the system

$$\begin{aligned} s_0 + \alpha_{i_0} a_1 + \alpha_{i_0}^2 a_2 + \dots + \alpha_{i_0}^d s_1 &= \mathbf{p}_{(0,d)\to(s_0,s_1)}(\alpha_{i_0}) \\ s_0 + \alpha_{i_1} a_1 + \alpha_{i_1}^2 a_2 + \dots + \alpha_{i_1}^d s_1 &= \mathbf{p}_{(0,d)\to(s_0,s_1)}(\alpha_{i_1}) \\ &\vdots \\ s_0 + \alpha_{i_{d-1}} a_1 + \alpha_{i_{d-1}}^2 a_2 + \dots + \alpha_{i_{d-1}}^d s_1 &= \mathbf{p}_{(0,d)\to(s_0,s_1)}(\alpha_{i_{d-1}}). \end{aligned}$$

Note that the coefficients in this system are the publicly known support points α_{i_j} and the a_i (resp. s_0 or s_1) are the unknowns. This system thus has $d+1$ unknowns and d equations. As the system is thus under-determined, the secrets (s_0, s_1) cannot be identified uniquely. However, once either of these secrets is known, the system has only d unknowns and can thus be solved uniquely since the equations are linearly independent. □

We stress here the differences between Item (i) in Lemma 6 and in Lemma 4 and Lemma 5. In the case of embedding a single secret, random variables in a set of size $t \le d$ are independent. However, due to the possible dependency between the two secrets, we can only guarantee this independence for sets of size $t \le d-1$ when embedding two secrets. Hence, to guarantee security against an attacker allowed to make t probes, we need use a polynomial of degree $t+1$ (instead of t when considering a single secret). To embed a second secret for a fixed security order t, we thus need to increase the number of shares by one. However, as discussed in the introduction, this is a much smaller overhead compared to the alternatives of parallel or sequential computation of two identical gadgets.

More than two secrets Finally, it might also be possible to embed more than two secrets using a similar approach. This would require the aforementioned complicated relations between the support points, as discussed in [LD04]. Furthermore, a very useful property of the lowest and the highest coefficient of a polynomial is the fact that the lowest (resp. highest) coefficient of a product $\mathbf{p} \cdot \mathbf{p}'$ of two polynomials is exactly the product of the lowest (resp. highest) coefficients of $\mathbf{p}$ and $\mathbf{p}'$. This is not true for the other coefficients due to the convolution of the indices used in polynomial multiplication. Hence, embedding multiple secrets seems to require a much more complicated multiplication.

4 Computing on Two-Secret Sharings Against Passive Attackers

In this section, we will generalize several known algorithms secure against passive attacks into a unified algorithm and show that our double-sharing approach can be used easily while still giving security against passive attacks in this context. To do so, we show that the unified algorithms are also (S)NI. For the analyzed circuits, we would only need to consider addition and multiplication; however, we will show that our approach can also easily support other operations that we believe to be useful when designing parallel gadgets.

In the following, we always consider double sharings. For the sake of readability, we will thus write $[\![s_0, s_1]\!]_d$ instead of $[\![(0,d) \to (s_0, s_1)]\!]_d$ and, similarly, $\mathsf{Share}_d(s_0, s_1)$ and $\mathsf{Open}(v_I)$. As discussed in Section 3.2, to guarantee security against a t-probing attacker, we need to use at least a polynomial of degree $d \ge t+1$ here.

In general, given two sharings $(F_0, \dots, F_{n-1}) \in [\![s_0, s_1]\!]_d$ and $(G_0, \dots, G_{n-1}) \in [\![s_0', s_1']\!]_d$, the goal is to compute a sharing $(Q_0, \dots, Q_{n-1}) \in [\![\varphi_0(s_0, s_1, s_0', s_1'), \varphi_1(s_0, s_1, s_0', s_1')]\!]_d$ for some functions φ_0 and φ_1.

4.1 Share-Wise Gadgets

We first analyze the security of any share-wise gadget, i.e., those corresponding to *linear* functions φ_0 and φ_1. For the sake of simplicity, we consider the addition gadget $\varphi_i(s_0, s_1, s_0', s_1') = s_i + s_i'$ as a running example. As

$$[\![s_0, s_1]\!]_d + [\![s_0', s_1']\!]_d = [\![s_0 + s_0', s_1 + s_1']\!]_d,$$

we can easily perform the addition by adding the shares of the sharing as shown in Algorithm 1 where $(F_i)_{i\in[n]} \in [\![s_0, s_1]\!]_d, (G_i)_{i\in[n]} \in [\![s_0', s_1']\!]_d$. Hence, it only remains to prove leakage resilience.

Algorithm 1: Addition-Gadget

Input: Degree-d shares of s_0, s_1 as $(F_i)_{i\in[n]}$ and shares of s_0', s_1' as $(G_i)_{i\in[n]}$.
Result: Degree-d shares of $q_0 = s_0 + s_0', q_1 = s_1 + s_1'$ as $(Q_i)_{i\in[n]}$.

1 initialize Q_i
2 **forall** $i \leftarrow 0$ **to** $n-1$ **do**
3 $\quad Q_i \leftarrow F_i + G_i$ `// share-wise transformation`
4 **return** $(Q_0, \ldots, Q_{n-1})$

Lemma 7. *Algorithm 1 is t-NI for $n > d$ when using polynomials of degree $d \geq t+1$.*

Proof. It is easy to see that the NI property follows from the fact that the gadget is share-wise, and the new double sharing does not affect the NI property. Each probe Q_i, F_i, and G_i can be simulated with F_i and G_i. This observation immediately results in the NI property because each probe can be perfectly simulated with at most one share of each input sharing. □

Note that this proof is not limited to a specific share-wise transformation in Algorithm 1, and also holds for any other share-wise transformation ϕ with $Q_i \leftarrow \phi(F_i, G_i)$. Hence, the claim holds for any share-wise gadget, as mentioned at the beginning of this section.

The Frobenius Optimization A very useful optimization for the computation of certain exponentiations was first presented by Roche and Prouff [PR11]. Let p be the characteristic of the field $\mathbb{F}$ underlying the arithmetic circuit that we aim to protect. The method of Roche and Prouff then allows computing the operation $x \mapsto x^p$ in a very efficient manner. In contrast to the packed secret sharing of Grosso et al. [GSF14], our approach can also use this improvement. Such operations are, for example, very useful when considering the AES s-box, where an efficient squaring operation is very helpful when computing the GF(2)-affine transformation. It requires the support points α_i to fulfill a condition called *stability over Frobenius automorphism*, which says that for every support point α_i, there is some support point $\alpha_{j(i)}$ such that $\alpha_{j(i)} = \alpha_i^p$.

Consider the polynomial $\mathbf{p} \in \mathbb{F}[x]$ of degree d that embeds two secrets s_0 and s_1, i.e.,

$$\mathbf{p}(x) = s_0 + \left(\sum_{k=1}^{d-1} a_k x^k\right) + s_1 x^d$$

and the following polynomial embedding the exponentiation of power p on both secrets s_0 and s_1:

$$\mathbf{p}'(x) = s_0^p + \left(\sum_{k=1}^{d-1} a_k^p x^k\right) + s_1^p x^d.$$

Note that if the coefficients $a_1, \dots, a_{d-1}$ of $\mathbf{p}$ are distributed uniformly at random, the coefficients $a_1^p, \dots, a_{d-1}^p$ of $\mathbf{p}'$ are also distributed uniformly at random, as the mapping $x \mapsto x^p$ is the Frobenius automorphism in fields of characteristic p. Hence, we only need to consider how to obtain a sharing $(V_0', \dots, V_{n-1}')$ with $V_i' = \mathbf{p}'(\alpha_i)$ from the sharing $(V_0, \dots, V_{n-1})$ with $V_i = \mathbf{p}(\alpha_i)$. Fortunately, this can be done simply by setting $V'_{j(i)} = V_i^p$, which is a completely share-wise operation.

To see that these shares indeed describe the desired polynomial, i.e., $V'_{j(i)} = \mathbf{p}'(\alpha_{j(i)})$, one can easily verify that

$$V'_{j(i)} = V_i^p = \left(s_0 + \left(\sum_{k=1}^{d-1} a_k \alpha_i^k\right) + s_1 \alpha_i^d\right)^p = s_0^p + \left(\sum_{k=1}^{d-1} a_k^p \alpha_i^{kp}\right) + s_1^p \alpha_i^{dp}.$$

Here, the last equality is due to the identity $(a+b)^p = a^p + b^p$ in fields of characteristic p, which is sometimes called *freshman's dream*. Now, due to the stability condition $\alpha_{j(i)} = \alpha_i^p$, we have

$$s_0^p + \left(\sum_{k=1}^{d-1} a_k^p \alpha_i^{kp}\right) + s_1^p \alpha_i^{dp} = s_0^p + \left(\sum_{k=1}^{d-1} a_k^p \alpha_{j(i)}^{k}\right) + s_1^p \alpha_{j(i)}^{d} = \mathbf{p}'(\alpha_{j(i)}).$$

Hence, from a sharing $(V_0, \dots, V_{n-1})$ of the secrets s_0 and s_1, we can compute a sharing $(V_0', \dots, V_{n-1}')$ of the secrets s_0^p and s_1^p in a share-wise manner, using at most $2n \cdot \log(p)$ field multiplications.

4.2 Non-Linear Operations

Non-linear operations, however, are more complicated. For example, while two sharings $[\![s_0, s_1]\!]_d, [\![s_0', s_1']\!]_d$ of two degree-d sharings can still be multiplied share-wise, the product polynomial is of degree $2d$, i.e.,

$$[\![s_0, s_1]\!]_d \cdot [\![s_0', s_1']\!]_d = [\![s_0 \cdot s_0', s_1 \cdot s_1']\!]_{2d}.$$

Hence, a subsequent step is required that reduces the degree back to (at most) d. As the degree of is increased to $2d$, we need to require $n > 2d$ to guarantee correctness of the computation, just like in the classical BGW scheme [BGW88]. Nearly every solution on how to handle non-linear operations when using polynomial sharing follows this basic principle: First perform a share-wise computation to obtain a high-degree encoding of the correct value and then perform a degree reduction. We call such gadgets *BGW-like*. The remaining solutions (such as Grosso et al. [GSF14] or Wang et al. [WMCS20]) improve the performance by changing temporarily to an alternate encoding. However, these alternate encodings typically do not have the same resilience against active fault attackers compared to the polynomial sharing and are thus vulnerable to active attacks. To the best of our knowledge, the only non-BGW-like approach that changes to a fault-resilient encoding is due to Berndt et al. [BEF+23]. As our goal is to protect against passive and active attacks, we thus focus on BGW-like gadgets and on the LaOla-gadget of Berndt et al.

Before focusing on concrete gadgets, we introduce the auxiliary gadgets $\mathbf{ZEnc}_n^d$ and $\mathbf{sZEnc}_n^d$ based on the two algorithms in [BEF+23] that will be useful when describing the gadgets for non-linear operations. The first algorithm shown in Algorithm 2 generates a random zero sharing, i.e. a random polynomial g such that both $g_0 = 0$ and $g_d = 0$. This is, however, not SNI, due to a lack of sufficient independence between the output shares. However, one can extend this to achieve independence by multiple executions as shown in Algorithm 3. Note that Algorithm 2 has a slightly different structure than the identically named gadget in [BEF+23], but as all intermediate values are the same with only their order changing, this has no impact on the security.

Algorithm 2: $\mathbf{ZEnc}_n^d$

Result: A randomized sharing of zero $[\![g]\!]_d$.

initialize G_j
$(r_1, \dots, r_{d-1}) \leftarrow\!\!\$\ \mathbb{F}$
forall $j \leftarrow 0$ **to** $n-1$ **do**
  forall $k \leftarrow 1$ **to** $d-1$ **do**
    $G_j \leftarrow G_j + r_k \cdot \alpha_j^k$
return $[\![g]\!]_d$

Algorithm 3: $\mathbf{sZEnc}_n^d$

Result: A randomized sharing of zero $[\![y]\!]_d$.

initialize $[\![y]\!]_d$
forall $j \leftarrow 0$ **to** d **do**
  $[\![g]\!]_d \leftarrow \mathbf{ZEnc}_n^d$
  $[\![y]\!]_d \leftarrow [\![y]\!]_d + [\![g]\!]_d$
return $[\![y]\!]_d$

4.2.1 BGW-Like Gadgets

While there is a variety of different non-linear gadgets, we first concentrate on a subset of them, all having a similar structure that goes back to the original work of Ben-Or, Goldwasser, and Wigderson [BGW88].

In general, the computation performed by these gadgets can be grouped into two different phases: In the first phase, the non-linear function is computed *locally* on each share. While this produces a correct result, it also increases the degree of the underlying polynomial. Hence, in the second phase, a *degree reduction* needs to be performed. We generalize this construction such that we allow the following representation for a sharing $(Q_0, \dots, Q_{n-1})$:

$$(Q_0, \dots, Q_{n-1}) = (\sum_{i=0}^{n-1} \ell_{i,0}^{(\varphi_0,\varphi_1)}(F_i, G_i), \dots . \sum_{i=0}^{n-1} \ell_{i,n-1}^{(\varphi_0,\varphi_1)}(F_i, G_i))$$

Here, the share-wise operations $\ell_{i,j}^{(\varphi_0,\varphi_1)}$ are a compact description of both phases of the computation. Hence, each output share Q_j depends on a linear combination of the share-wise operations $\ell_{0,j}^{(\varphi_0,\varphi_1)}(F_0, G_0), \ell_{1,j}^{(\varphi_0,\varphi_1)}(F_1, G_1), \dots, \ell_{n-1,j}^{(\varphi_0,\varphi_1)}(F_{n-1}, G_{n-1})$, namely $\sum_{i=0}^{n-1} \ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)$.

This approach is sufficient to model many different functions φ_0 and φ_1. Examples for interesting functions φ_0, φ_1 that we believe to be useful with

$$(Q_0, \dots, Q_{n-1}) \in [\![\varphi_0(s_0, s_1, s_0', s_1'), \varphi_1(s_0, s_1, s_0', s_1')]\!]_d$$

are given in Table 1. Some of these functions allow multiplication, i.e., the computation of $(\varphi_0(s_0, s_1, s_0', s_1'), \varphi_1(s_0, s_1, s_0', s_1')) = (s_0 \cdot s_0', s_1 \cdot s_1')$ while others allow re-combinations of the secrets, i.e., $(\varphi_0(s_0, s_1, s_0', s_1'), \varphi_1(s_0, s_1, s_0', s_1')) = (s_0, s_0')$.

A formal description of this algorithm is given in Algorithm 4. Note that because of the very general modeling, share-wise operations can also be represented this way, though this has some overhead in comparison to the straightforward computation discussed in Section 4.1.

Lemma 8. *Algorithm 4 is t-SNI when using $n > 2d$ shares and polynomials of degree $d \geq t + 1$.*

Algorithm 4: (φ_0, φ_1)-Gadget

Input: Degree-d shares of s_0, s_1 as $(F_i)_{i\in[n]}$ and shares of s_0', s_1' as $(G_i)_{i\in[n]}$.
Result: Degree-d shares of $q_0 = \varphi_0(s_0, s_1, s_0', s_1'), q_1 = \varphi_1(s_0, s_1, s_0', s_1')$ as $(Q_i)_{i\in[n]}$.

initialize Q_j
forall *i←0* **to** *n-1* **do**
  $(\tilde{Q}_{i,0}, \dots, \tilde{Q}_{i,n-1}) \leftarrow \mathbf{ZEnc}_n^d$
  forall *j←0* **to** *n-1* **do**
    $\tilde{Q}_{i,j} \leftarrow \tilde{Q}_{i,j} + \ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)$ `// share-wise transformation and reduction`
    $Q_j \leftarrow Q_j + \tilde{Q}_{i,j}$ `// referred to as` $Q_{j,i}$
return $(Q_0, \dots, Q_{n-1})$

Proof. Let t_1 be the number of internal probes and t_2 the number of output probes with $t_1 + t_2 < t$. We first have to find input sets I and J of size at most t_1 and afterwards construct a perfect simulator that can simulate all t probed intermediate and output variables only using elements of I and J. Depending on the intermediate variables that are probed, the two sets I and J are constructed as follows:

- If F_i or G_i is probed, add i to I or J, respectively.
- If any value is probed during the computation of $\ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)$, add i to I and J.
- If r_j in loop i (denoted in the following as $r_{i,j}$), some intermediate value of $\mathbf{ZEnc}_n^d$ (denoted in the following as $\tilde{Q}_{i,j}^k$ where $k \in \{1, \dots, d-1\}$), or the sum with the share-wise transformation (denoted in the following as $\tilde{Q}_{i,j}^d$) is probed, add i to I and J.

According to our selection, we add at most one index to I and J for each internal probe and, therefore, $|I| \leq t_1$ and $|J| \leq t_1$.

We now construct the simulator. Note that whenever a coefficient $r_{i,j}$ is required for the simulation of other variables, it will be sampled by the simulator. This will fix the value of $r_{i,j}$.

1. The simulation of probed variables F_i, G_i, and the internal probes in $\ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)$ are straightforward. Since $i \in I$, the share F_i is a known value and we can perfectly simulate F_i. The same holds true for G_i, since $i \in J$. Similarly, we can simulate all internal values of $\ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)$, as $i \in I \cap J$ and $\ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)$ only depends on public values and F_i, G_i.
2. If $\tilde{Q}_{i,j}^k$ with $k \in \{1, \dots, d-1\}$ is probed, the simulator has to simulate the value $\tilde{Q}_{i,j}^k = \sum_{l=1}^{k} r_{i,l} \cdot \alpha_j^l$. All unknown $r_{i,l}$ with $1 \leq l \leq k$ can be sampled uniformly at random, allowing a perfect simulation.
3. The simulation of $\tilde{Q}_{i,j}^d$ is solely depending on $\tilde{Q}_{i,j}^{d-1}$ and $\ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)$ as $\tilde{Q}_{i,j}^d = \ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i) + \sum_{l=1}^{d-1} r_{i,l} \cdot \alpha_j^l$. Like before, if any of the $r_{i,l}$ with $1 \leq l \leq d-1$ is unknown, it can be sampled uniformly at random. Further, since $i \in I \cap J$, the value $\ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)$ is known. This allows a perfect simulation of $\tilde{Q}_{i,j}^d$.

Table 1: Instantiations of the different functions ℓ of Algorithm 4 to protect the gadgets against passive probes. The input sharings are $(F_i)_{i\in[n]}$ and $(G_i)_{i\in[n]}$ and embed the secrets s_0, s_1 and s'_0, s'_1, respectively. The output sharing is $(Q_i)_{i\in[n]}$. If $\ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)$ does not use $(G_i)_{i\in[n]}$, we consider the gadget as gadget with only one input sharing.

(φ_0, φ_1)	$\ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)$	Output	
(s_0, s'_0)	$\lambda_{i,0}F_i + \lambda_{i,0}\alpha_j^d G_i$	$q_0 = f_0$	$q_d = g_0$
(s_0, s'_1)	$\lambda_{i,0}F_i + \lambda_{i,d}\alpha_j^d G_i$	$q_0 = f_0$	$q_d = g_d$
(s_1, s'_0)	$\lambda_{i,d}F_i + \lambda_{i,0}\alpha_j^d G_i$	$q_0 = f_d$	$q_d = g_0$
(s_1, s'_1)	$\lambda_{i,d}F_i + \lambda_{i,d}\alpha_j^d G_i$	$q_0 = f_d$	$q_d = g_d$
(s_0, s_1)	$(\lambda_{i,0} + \lambda_{i,d}\alpha_j^d)F_i$	$q_0 = f_0$	$q_d = f_d$
$(s_0 \cdot s'_0, 0)$	$\lambda_{i,0}F_iG_i$	$q_0 = f_0 \cdot g_0$	$q_d = 0$
$(0, s_1 \cdot s'_1)$	$(\lambda_{i,2d}\alpha_j^d)F_iG_i$	$q_0 = 0$	$q_d = f_d \cdot g_d$
$(s_0 \cdot s'_0, s_1 \cdot s'_1)$	$(\lambda_{i,0} + \lambda_{i,2d}\alpha_j^d)F_iG_i$	$q_0 = f_0 \cdot g_0$	$q_d = f_d \cdot g_d$

4. Let's consider the case that $Q_{j,i}$ is probed. The value is computed as $Q_{j,i} = \sum_{k=0}^{i} \tilde{Q}_{k,j}^d$ or can be rewritten as $Q_{j,i} = Q_{j,\ell} + \sum_{k=\ell+1}^{i} \tilde{Q}_{k,j}^d$ if $Q_{j,\ell}$ with $\ell < i$ has already been probed. In the latter case, only the simulations of $\tilde{Q}_{k,j}^d$ where $\ell < k \leq i$ have to be considered, otherwise all of them are needed.

   - If for all necessary k (either $\ell < k \leq i$ or $0 \leq k \leq i$), $\ell_{k,j}^{(\varphi_0,\varphi_1)}(F_k, G_k)$ is a known value, all $\tilde{Q}_{k,j}^d$ can be perfectly simulated as shown in step 3, which in turn allows the perfect simulation of $Q_{j,i}$.
   - Else, there is at least one unknown $\ell_{k,j}^{(\varphi_0,\varphi_1)}(F_k, G_k)$. Note that this implies that, for any l, no $r_{k,l}$ was probed or sampled in a previous step of the simulation. Hence, $(\tilde{Q}_{k,j}^d)_{j\in[n]}$ with $\tilde{Q}_{k,j}^d = \ell_{k,j}^{(\varphi_0,\varphi_1)}(F_k, G_k) + \sum_{l=1}^{d-1} r_{k,l} \cdot \alpha_j^l$ has d unknowns: $\ell_{k,j}^{(\varphi_0,\varphi_1)}(F_k, G_k)$ and all $r_{k,l}$. Since $r_{k,l}$ are uniform random, the tuple $(\tilde{Q}_{k,j}^d)_{j\in[n]}$ represents a $d-1$-wise independent set of random variables. Recall that the number of allowed probes t is by definition smaller than d and each probe consists of at most one of the random values. Consequently, $Q_{j,i}$ can be simulated by a value chosen uniformly at random since it is randomized by the sum of $\tilde{Q}_{k,j}^d$.

As could be seen in step 4 above, $Q_{j,i}$ can be simulated without any index being added to I or J. The simulation is hence independent of the selected inputs, and the probed output shares $Q_j = Q_{j,n-1}$ can be perfectly simulated. □

To show the generality of this approach, Table 1 does not only consider the BGW-like multiplication (corresponding to the last row of the table with gadget $(s_0 \cdot s'_0, s_1 \cdot s'_1)$), but also various permutations of secrets to merge and permute multiple secrets in one polynomial that we believe to be useful when designing parallel gadgets: The gadgets

$(s_0 \cdot s'_0, 0)$ and $(0, s_1 \cdot s'_1)$ multiply the lower or upper secrets, respectively, while the (s_a, s'_b) gadget allows, depending on the choice of $a, b \in \{0, 1\}$, to generate a output sharing containing any secret of the first input in one and any of the second input in the other position. Additionally, the sole gadget (s_0, s_1) with only one input refreshes this input sharing without changing the secrets.

Lemma 9. *Algorithm 4 is correct for all instantiations of $\ell_{i,j}^{(\varphi_0,\varphi_1)}(\cdot,\cdot)$ shown in Table 1 when using $n > 2d$ shares and polynomials of degree d.*

Proof. In the following, we will prove that the output Q_j of Algorithm 4 describes the polynomial value of α_j as given in Table 1, so $Q_j = \sum_{i=0}^{n-1} q_i \alpha_j^i$ with q_0 and q_d as is stated in the table for the respective gadget. For a specific j, after each loop iteration over i in the first loop of Algorithm 4 we get

$$\tilde{Q}_{i,j}^d = \ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i) + \sum_{k=1}^{d-1} r_{i,k} \cdot \alpha_j^k$$

and this finally results in the output $Q_j = \sum_{i=0}^{n-1} \tilde{Q}_{i,j}^d$. Hence, we get

$$Q_j = \sum_{i=0}^{n-1} \left(\ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i) + \sum_{k=1}^{d-1} r_{i,k} \cdot \alpha_j^k \right) = \underbrace{\sum_{i=0}^{n-1} \ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)}_{\hat{=} q_0 + q_d \alpha_j^d} + \underbrace{\sum_{k=1}^{d-1} \left(\sum_{i=0}^{n-1} r_{i,k} \right) \cdot \alpha_j^k}_{\hat{=} \sum_{k=1}^{d-1} q_k \cdot \alpha_j^k} .$$

For correctness, we are interested in q_0 and q_d, and only consider the term $\sum_{i=0}^{n-1} \ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_i)$ in the following.

- We first analyze the computation of $(s_0 \cdot s'_0, 0)$, $(0, s_1 \cdot s'_1)$, and $(s_0 \cdot s'_0, s_1 \cdot s'_1)$. Using the latter as an example, the gadget computing the secret-wise multiplication of the double sharing is initialized by $\ell_{i,j}^{(s_0 \cdot s'_0, s_1 \cdot s'_1)}(F_i, G_i) = (\lambda_{i,0} + \lambda_{i,2d} \alpha_j^d) F_i G_i$, and this results in

$$\begin{aligned} \sum_{i=0}^{n-1} \ell_{i,j}^{(s_0 \cdot s'_0, s_1 \cdot s'_1)}(F_i, G_i) &= \sum_{i=0}^{n-1} (\lambda_{i,0} + \lambda_{i,2d} \alpha_j^d) F_i G_i \\ &= \underbrace{(\sum_{i=0}^{n-1} \lambda_{i,0} F_i G_i)}_{\hat{=} q_0} \alpha_j^0 + \underbrace{(\sum_{i=0}^{n-1} \lambda_{i,2d} F_i G_i)}_{\hat{=} q_d} \alpha_j^d . \end{aligned}$$

  Since the (unfaulted) polynomials $(F_i)_{i \in [n]}$ and $(G_i)_{i \in [n]}$ have degree d with $\sum_{i=0}^{d} f_i x^i$ and $\sum_{i=0}^{d} g_i x^i$, respectively, it follows that the share-wise multiplication $(F_i \cdot G_i)$ results in a polynomial of degree $2d$ such that the highest monomial is $(f_d g_d) x^{2d}$ and the lowest one is $(f_0 g_0) x^0$. With Equation (1) it immediately follows that $q_0 = \sum_{i=0}^{n-1} \lambda_{i,0} F_i G_i = f_0 g_0$ and $q_d = \sum_{i=0}^{n-1} \lambda_{i,2d} F_i G_i = f_d g_d$. This proves the correctness for $(s_0 \cdot s'_0, s_1 \cdot s'_1)$, the correctness of $(s_0 \cdot s'_0, 0)$ and $(0, s_1 \cdot s'_1)$ follows with the same argument.

- When considering (s_0, s_1), the proof is similar to the previous one except for the fact that there is no share-wise multiplication and the secrets are still described by the monomials of degree d and 0. Hence, the instantiation $\ell_{i,j}^{(s_0,s_1)}(F_i)$ multiplies F_i with $(\lambda_{i,0} + \lambda_{i,d} \alpha_j^d)$ instead of $(\lambda_{i,0} + \lambda_{i,2d} \alpha_j^d)$, and we get $q_0 = \sum_{i=0}^{n-1} \lambda_{i,0} F_i = f_0$ and $q_d = \sum_{i=0}^{n-1} \lambda_{i,d} F_i = f_d$.

- For the (s_a, s'_b) gadget with $a, b \in \{0,1\}$, we initialize $\ell_{i,j}^{(s_a,s'_b)}(F_i, G_i) = \lambda_{i,a\cdot d} F_i + \lambda_{i,b\cdot d}\alpha_j^d G_i$, and this results in

$$\sum_{i=0}^{n-1} \ell_{i,j}^{(s_a,s'_b)}(F_i, G_i) = \sum_{i=0}^{n-1} \lambda_{i,a\cdot d} F_i + \lambda_{i,b\cdot d}\alpha_j^d G_i = (\underbrace{\sum_{i=0}^{n-1} \lambda_{i,a\cdot d} F_i}_{\hat{=} q_0})\alpha_j^0 + (\underbrace{\sum_{i=0}^{n-1} \lambda_{i,b\cdot d} G_i}_{\hat{=} q_d})\alpha_j^d \ .$$

  With Equation (1), it immediately follows that $q_0 = \sum_{i=0}^{n-1} \lambda_{i,a\cdot d} F_i = f_{a\cdot d}$ and $q_d = \sum_{i=0}^{n-1} \lambda_{i,b\cdot d} G_i = g_{b\cdot d}$. This proves the correctness of all (s_a, s'_b) gadgets with $a, b \in \{0,1\}$.

This concludes the proof. □

4.3 Protecting AES

In the following, we will show how our approach can decrease the cost of masking when considering the block cipher AES-128 compared to using the classical BGW-gadgets. This cipher consists of ten repetitions of a round transformation on an internal state of length 128, which is interpreted as 16 elements of $\mathrm{GF}(2^8)$. Each round begins with a key addition AddRoundKey that adds the current round key (also of length 128) to the internal state. Then, the non-linear SubBytes operation is applied to every byte of the internal state. The SubByte operation first applies the function $y \mapsto y^{254}$ to every byte, followed by an $\mathrm{GF}(2)^8$-affine transformation τ_A, which can be expressed as

$$\begin{aligned}\tau_A(y) = \texttt{0x63} + (\texttt{0x05}\cdot y) + (\texttt{0x09}\cdot y^2) + (\texttt{0xf9}\cdot y^4) + (\texttt{0x25}\cdot y^8) + \\ (\texttt{0xf4}\cdot y^{16}) + (\texttt{0x01}\cdot y^{32}) + (\texttt{0xb5}\cdot y^{64}) + (\texttt{0x8f}\cdot y^{128}).\end{aligned}$$

A derivation of this is described, e.g., by Roche and Prouff [PR11]. Finally, two linear transformations, called ShiftRows and MixColumns are then applied to the internal state. More concretely, for these linear transformations, the elements of $\mathrm{GF}(2^8)$ are put into a 4×4-matrix

$$A = \begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix}.$$

Then, ShiftRows transforms this matrix into the matrix

$$A' = \begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_5 & a_9 & a_{13} & a_1 \\ a_{10} & a_{14} & a_2 & a_6 \\ a_{15} & a_3 & a_7 & a_{11} \end{pmatrix}.$$

Finally, MixColumns multiplies each column of A' with a fixed MDS matrix M to obtain the matrix A''.

Following the approach of Roche and Prouff [PR11], which is also used by Seker et al. [SFRES18], we can now compute the number of additions, affine operations, squarings, and multiplications which are needed for each operation within a single round. A summary of this is shown in Table 2 which depends on the number of used sharings. To represent the 16 elements of $\mathrm{GF}(2^8)$, we require $B = 8$ sharings when using our double sharing approach and $B = 16$ otherwise. The running times for the individual operations when using polynomials of degree d and the minimal number of shares $n = 2d + 1$ are shown in Table 3.

Table 2: The number of calls to different gadgets to mask the different operations of AES when operating on $B \in \{8, 16\}$ polynomials.

	AddRoundKey	$y \mapsto y^{254}$	η_A	MixColumns	ShiftRows
Add(d)	B		$7B$	$(3/4)B$	
Affine(d)			$8B$	B	
Square(d)		$7B$	$7B$		
Mult(d)		$6B$			

Table 3: The number of field operations of the different gadgets when using $n = 2d + 1$ shares. For the squaring operation, we use the Frobenius approach of [PR11].

	Add(d)	Affine(d)	Square(d)	Mult(d)
Complexity	$2d + 1$	$2d + 1$	$2d + 1$	$8d^3 + 16d^2 + 10d + 2$

Combining all of these information means that a masked version using polynomials of degree d that does not use our double sharing approach needs about

$$\begin{aligned} T_{\text{nds}}(d) &= 140\mathsf{Add}(d) + 144\mathsf{Affine}(d) + 224\mathsf{Square}(d) + 96\mathsf{Mult}(d) \\ &= 768d^3 + 1536d^2 + 1976d + 192 \end{aligned}$$

field operations when using polynomials of degree d and the minimal number of shares $n = 2d + 1$.

When applying our double sharing approach, both AddRoundKey and SubBytes are easily handled, as the computations of each element of $\mathrm{GF}(2^8)$ are independent of each other. For MixColumns, we need to be more careful, as the computation now mixes different elements. However, all elements that are mixed come from the same column, the computations for each column are independent of each other and all elements from one row of the resulting matrix are computed using the same row of the MDS matrix. We thus group the secrets such that each of the eight double-sharing polynomials contains two secrets from the same row: The element from the first column will be encoded together with the one from second column and the element from the third column together with the one from the fourth column. More formally, if a polynomial encodes two values a_i and a_j, then $(i, j) \in \{(0, 4), (5, 9), (10, 14), (15, 3), (8, 12), (13, 1), (2, 6), (7, 11)\}$.

The only complication when keeping this structure of the polynomials comes from the ShiftRows operations. However, it is easy to see that both the first and the third row keep this structure, as the elements are not shifted at all or shifted by two (where a simple swap of the polynomials is sufficient). To handle the second and the fourth row, we use the (s_1, s_0') and (s_1', s_0) gadget of Algorithm 4 to combine the two sharings such that the secrets are shifted. For the second row, containing the sharings $[\![a_1, a_5]\!]_d$ shared in $\mathbf{f}$ and $[\![a_9, a_{13}]\!]_d$ shared in $\mathbf{g}$, the (s_1, s_0')-gadget instantiation on inputs $\mathbf{f}, \mathbf{g}$ gives a new sharing containing $[\![a_5, a_9]\!]_d$ and the (s_1', s_0)-gadget instantiation on inputs $\mathbf{g}, \mathbf{f}$ gives a sharing of $[\![a_{13}, a_1]\!]_d$. The fourth row can be computed accordingly. Hence, we now have the proper presentation of the double-sharing polynomials where MixColumns can be computed. Each call to Algorithm 4 takes time

$$\mathsf{Phi0Phi1}(d) = 8d^3 + 32d^2 + 26d + 6.$$

Hence, the total number of field operations when using our double-sharing approach

Table 4: Comparison of $T_{\text{nds}}(t)$ and $T_{\text{ds}}(t+1)$ for $t = 1, \dots, 6$.

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
$T_{\text{nds}}(t)$	4472	16432	40680	81824	144472	233232
$T_{\text{ds}}(t+1)$	9470	22946	45702	80234	129038	194610

for polynomials of degree d and the minimal number of shares $n = 2d + 1$ is

$$\begin{aligned} T_{\text{ds}}(d) &= 70\mathsf{Add}(d) + 72\mathsf{Affine}(d) + 112\mathsf{Square}(d) + 48\mathsf{Mult}(d) + 4\mathsf{Phi0Phi1}(d) \\ &= 416d^3 + 896d^2 + 1092d + 374. \end{aligned}$$

Note, however, that due to the need to increase the degree of the underlying polynomial by one when using the double-sharing approach, we need to compare $T_{\text{nds}}(t)$ with $T_{\text{ds}}(t+1)$ when considering security against a t-probing attacker. As can be seen in Table 4, a simple calculation then shows that the double-sharing approach already outperforms the single-sharing approach for $t = 4$. By contrast, the packed secret sharing approach of Grosso et al. [GSF14] only started to outperform the single-sharing approach for $t \geq 10$.

4.4 LaOla Gadget

In [BEF+23], the authors give a new multiplication gadget, called *LaOla*. This gadget does not fall in the class of BGW-gadgets, as it aims to avoid the doubling of the degree of the intermediate polynomials. To do this, LaOla first *splits* a polynomial $\mathbf{f}$ with degree d that embeds a secret s into two polynomials $\mathbf{f}'$ and $\mathbf{f}''$ of high degree, i.e., $\deg(\mathbf{f}') = \deg(\mathbf{f}'') = d$. However, the sum of these polynomials has lower degree, but also embeds s, i.e., $\deg(\mathbf{f}' + \mathbf{f}'') = d/2$, and $\mathbf{f}'(0) + \mathbf{f}''(0) = s$. This operation is called **SplitRed**. After this was performed on $\mathbf{f}$ and $\mathbf{g}$ to obtain $\mathbf{f}'$, $\mathbf{f}''$, $\mathbf{g}'$, and $\mathbf{g}''$, one only needs to compute the polynomial

$$\mathbf{f}' \cdot \mathbf{g}' + \mathbf{f}' \cdot \mathbf{g}'' + \mathbf{f}'' \cdot \mathbf{g}' + \mathbf{f}'' \cdot \mathbf{g}''$$

of degree d that embeds the secret $\mathbf{f}(0) \cdot \mathbf{g}(0)$ in a share-wise manner. In this section, we show how to (slightly) modify the multiplication gadget of [BEF+23] to also allow to use double sharings for this more efficient gadget.

Algorithm 5: LaOlaMult

Input: Degree-d shares of s_0, s_1 as $(F_i)_{i\in[n]}$ and shares of s_0', s_1' as $(G_i)_{i\in[n]}$.
Result: Degree-d shares of $q_0 = s_0 \cdot s_0', q_1 = s_1 \cdot s_1'$ as $(Q_i)_{i\in[n]}$.

$\left((F_i')_{i\in[n]}, (F_i'')_{i\in[n]}\right) \leftarrow$ **SplitRed**$((F_i)_{i\in[n]})$
$\left((G_i')_{i\in[n]}, (G_i'')_{i\in[n]}\right) \leftarrow$ **SplitRed**$((G_i)_{i\in[n]})$
$(Q_i)_{i\in[n]} \leftarrow \mathbf{sZEnc}_n^d$
forall $j \leftarrow 0$ **to** $n-1$ **do**
  $H_j^0 \leftarrow F_j'G_j'$; $H_j^1 \leftarrow F_j'G_j''$; $H_j^2 \leftarrow F_j''G_j'$; $H_j^3 \leftarrow F_j''G_j''$;
  $Q_j \leftarrow (((Q_j + H_j^0) + H_j^1) + H_j^2) + H_j^3$
return $(Q_0, \dots, Q_{n-1})$

The gadget **SplitRed** is also further modified to accommodate for the fact that we only consider passive attacks here; the resulting gadget is shown in Algorithm 5. In total, we modify $\ell_{i,j}^{(\sim)}(F_i)$ used within **SplitRed** in Algorithm 6 to equal $(\lambda_{i,0} + \lambda_{i,d}\alpha_j^{d/2})F_i$ instead

of $(\lambda_{i,0} + \sum_{k>d} \lambda_{i,k}\alpha_i^k)F_i$. Later on, when we also consider active attacks, we will also consider the remaining entries. For the sake of completeness, the complete description of **SplitRed** can be found in Appendix B.

The following lemma shows that **SplitRed** can then compute two multiplications in parallel:

Lemma 10. *Algorithm 5 is correct if **SplitRed** is initialized with* $\ell_{i,j}^{(\sim)}(F_i) = (\lambda_{i,0} + \lambda_{i,d}\alpha_j^{d/2})F_i$ *and uses* $n \geq d+1$ *shares and polynomials of degree* d.

Proof. The gadget **SplitRed** outputs $(F_i')_{i\in[n]}, (F_i'')_{i\in[n]}, (G_i')_{i\in[n]}, (G_i'')_{i\in[n]}$ describing the four polynomials $\sum f_i'x^i, \sum f_i''x^i, \sum g_i'x^i, \sum g_i''x^i$ with $f_0' + f_0'' = s_0$, $f_{d/2}' + f_{d/2}'' = s_1$, and $g_0' + g_0'' = s_0'$, $g_{d/2}' + g_{d/2}'' = s_1'$ such that both addition polynomials embedded within $(F_i' + F_i'')_{i\in[n]}$ and $(G_i' + G_i'')_{i\in[n]}$ have degree $d/2$. Hence, the share-wise multiplication $((F_i' + F_i'') \cdot (G_i' + G_i''))_{i\in[n]}$ results in a polynomial $\sum q_i x^i$ with degree d and $q_0 = s_0 \cdot s_0'$, $q_d = s_1 \cdot s_1'$. Taking into account that it holds

$$((F_i' + F_i'') \cdot (G_i' + G_i''))_{i\in[n]} = \left(H_i^0\right)_{i\in[n]} + \left(H_i^1\right)_{i\in[n]} + \left(H_i^2\right)_{i\in[n]} + \left(H_i^3\right)_{i\in[n]},$$

this results in the claim of the lemma. Note that the polynomial $\hat{\mathbf{q}}(x)$ described by $(Q_i)_{i\in[n]}$ generated in line 3 of **LaOlaMult** does not change $q_0 = s_0 \cdot s_0'$, $q_d = s_1 \cdot s_1'$ because it is a polynomial with degree $d-1$ and $\hat{\mathbf{q}}(0) = 0$. □

Next we argue, why the parallelization of **LaOlaMult** does not affect the security of the gadget. For this reason, we analyze the security of **SplitRed** first.

Lemma 11. *Algorithm 6 is t-NI when using $n \geq d+1$ shares and polynomials of degree $d > t+1$.*

Proof. The only differences to [BEF+23] are the modified (public) constants $\ell_j^i := (\lambda_{i,0} + \lambda_{i,d}\alpha_j^{d/2})$. Hence, the proof is the same as in [BEF+23]. □

The gadget **SplitRed** is the only modified part of our multiplication gadgets. Since *Lemma 11* proves that the modification does not affect its security, we get the following result.

Lemma 12. *Algorithm 5 is t-SNI when using $n \geq d+1$ shares and polynomials of degree $d > t+1$.*

Proof. The only difference between our gadget the one in [BEF+23] is the modification of the public values in **SplitRed**. Hence, it only remains to prove that our modified **SplitRed** has the same security properties as the original one in [BEF+23]. In particular, [BEF+23] shows that **LaOlaMult** is t-SNI if **SplitRed** is t-NI. In Lemma 11, we argue why our modified **SplitRed** is t-NI as well. □

5 Computing on Two-Secret Sharings Against Active Attackers

In this section, we consider active adversaries that can probe as well as induce faults. As before, t will denote the number of allowed probes and σ the number of allowed faults. Recall that the multiplication of two shares raises the degree of the embedded polynomial. We continue considering polynomials of degree d; so as to not lose any information before the degree reduction of the product polynomial, we require $n > 2d$ in the case of BGW-like gadgets or $n > d$ in the case of LaOla. Additionally, to be able to detect σ faults occurring during the same run, our analysis requires $n > 2d + \sigma$ for BGW-like gadgets and $n > d + \sigma$ for LaOla as well as $d > \sigma$.

5.1 Error Propagation

When considering active attacks, i.e., the introduction of faults, there are roughly two possible approaches that make use of redundancy within the sharing. In the first approach, one tries to *correct* the errors introduced by the faults. In the second approach, one tries to *detect* the presence of errors. A common technique to detect the presence of errors is called *error propagation*. Intuitively, this technique aims to guarantee that once an error is introduced, it spreads fast and wide. This way, an active attacker can not hide the presence of this error by, e.g., faulting the error-detection routine.

In this work, we also focus on error detection using error propagation. We will show that simple modifications (the addition of an *error propagation polynomial*) of the gadgets described in Table 1 allows to embed two secrets in a secure manner in the presence of faults. One can think of the error term added by the induced faults as an additional error polynomial $\boldsymbol{\zeta}(x)$ that is added to the original unfaulted polynomial $\mathbf{f}(x)$ resulting in the faulty polynomial $\mathbf{f}'(x) = \mathbf{f}(x) + \boldsymbol{\zeta}(x)$. In the following, $\mathbf{f}(x)$ (resp. $\mathbf{g}(x)$) will thus always denote the polynomial resulting from a non-faulted run of the computation and $\mathbf{f}'(x)$ (resp. $\mathbf{g}'(x)$) will denote the (possibly faulted) new polynomials. Due to the highly symmetric nature of the polynomial sharing, it is easy to see that all shares F'_i behave the same. Since there are no ordering requirements regarding the public support points $(\alpha_i)_{i\in I}$, all effects that an attacker might gain from faulting some arbitrary positions $I \subseteq \{0, \dots, n-1\}$ with $|I| \leq \sigma$ can also be obtained by rearranging the values of $(\alpha_i)_{i\in I}$ such that the faulted positions are now $\{0, \dots, |I|-1\}$. We thus assume throughout this section that the faults $\boldsymbol{\zeta}$ are induced in the first σ shares, which gives the following equation

$$V^{-1}\begin{pmatrix} F'_0 \\ \vdots \\ F'_{\sigma-1} \\ F'_\sigma \\ \vdots \\ F'_{n-1} \end{pmatrix} = V^{-1}\left(\begin{pmatrix} F_0 \\ \vdots \\ F_{\sigma-1} \\ F_\sigma \\ \vdots \\ F_{n-1} \end{pmatrix} + \begin{pmatrix} Z_0 \\ \vdots \\ Z_{\sigma-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}\right) = \begin{pmatrix} f_0 \\ \vdots \\ f_d \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \begin{pmatrix} \zeta_0 \\ \vdots \\ \zeta_d \\ \zeta_{d+1} \\ \vdots \\ \zeta_{n-1} \end{pmatrix} = \begin{pmatrix} f'_0 \\ \vdots \\ f'_d \\ f'_{d+1} \\ \vdots \\ f'_{n-1} \end{pmatrix}.$$

If any ζ_i for $i \in \{d+1, \dots, n-1\}$ is non-zero, the degree of the error polynomial increases and in turn the degree of the faulty polynomial $\mathbf{f}'(x)$ does so as well.

Lemma 13. *Let $\mathbf{f}' = \mathbf{f} + \boldsymbol{\zeta}$ be the sum of a degree-d sharing $(F_i)_{i\in[n]}$ and a polynomial $\boldsymbol{\zeta} \neq \mathbf{0}$ generated by $\omega \leq \sigma$ faults. Then $\mathbf{f}'$ has degree at least $n - \omega$.*

Proof. The faulty polynomial $\mathbf{f}'(x)$ is the sum of the original polynomial $\mathbf{f}(x)$ and the error polynomial $\boldsymbol{\zeta}(x)$. The latter is generated by n points, ω of which are the induced faults, while the others are zeros. By the fundamental theorem of algebra, a polynomial having at least $n - \omega$ zeros either has a degree of at least $n - \omega$ or is equal to zero. The latter is only possible if the induced faults $Z_i = 0$ for all $0 \leq i < \sigma$, which would imply that $\mathbf{f}'(x) = \mathbf{f}(x)$, i.e., $\boldsymbol{\zeta} = \mathbf{0}$. □

By requiring $n > 2d + \sigma$ or $n > d + \sigma$, the degree of such a faulty polynomial is consequently higher than the degree of any unfaulted one when the attacker can only introduce σ faults. Hence, by adding a completely random polynomial of degree d to the current polynomial and interpolating the coefficients $d+1, \dots, n-1$, as described by Seker et al. [SFRES18], the presence of the introduced faults is easily detectable.

We want the gadgets to also preserve the at most σ faults induced by the active adversary. Our goal is to achieve error propagation such that if a fault is induced at some point in the computation, with high probability, it will either be preserved throughout making the output shares invalid or disappear, leading to the original unfaulted output.

As shown in [SFRES18], for the gadgets regarding the linear operations *addition* or the *exponentation* to the power of p over a finite field of characteristic p, or *affine transform*, if

the inputs of the gadgets are faulty, this error is propagated to the output of the gadget with a high probability. The argumentation is easily transferable from Shamir's secret sharings to our double sharings.

However, non-linear operations again require more complexity as they include a reduction step that necessarily loses information (e.g., in the form of the coefficients $d+1, \dots, 2d$ for the BGW-like gadgets). It is thus vital that we preserve the important information about the occurrence of a fault. To assure this, we follow the gadget design of [SFRES18, BEF+23], and embed this information in the output of a gadget by adding some error propagation term to the shares. Intuitively, one can think of the added error propagation term as adding some polynomial $\boldsymbol{\rho}(x)$ to the polynomial defined by the output of the gadget without error propagation $\mathbf{q}(x)$. We require the following properties of the *error propagation polynomial* $\boldsymbol{\rho}(x)$:

1. If no fault was introduced into the computation, $\boldsymbol{\rho}$ should be zero.
2. If some faults were introduced into the computation, $\boldsymbol{\rho}$ should have a degree higher than d with high probability.

Using these properties, we thus know that the sum $\mathbf{q}'(x)$ of the original (possibly faulted) polynomial $\mathbf{q}(x)$ and the error propagation polynomial $\boldsymbol{\rho}(x)$ has degree at most d iff no faults were introduced into the computation (with high probability following property 2). In order to avoid confusion between the adversarially chosen fault polynomial and the error propagation polynomial, we always denote the former by $\boldsymbol{\zeta}$ and the latter by $\boldsymbol{\rho}$. Hence, we have

$$\begin{aligned}\mathbf{q}'(x) &= \mathbf{q}(x) + \boldsymbol{\rho}(x)\\ &= \mathbf{f}'(x) \circ \mathbf{g}'(x) + \boldsymbol{\rho}(x)\\ &= \mathbf{f}(x) \circ \mathbf{g}(x) + \boldsymbol{\zeta}^{\mathbf{f}\circ\mathbf{g}}(x) + \boldsymbol{\rho}(x)\\ &= \mathbf{f}(x) \circ \mathbf{g}(x) + \mathbf{f}(x) \circ \boldsymbol{\zeta}^{\mathbf{g}}(x) + \mathbf{g}(x) \circ \boldsymbol{\zeta}^{\mathbf{f}}(x) + \boldsymbol{\zeta}^{\mathbf{f}}(x) \circ \boldsymbol{\zeta}^{\mathbf{g}}(x) + \boldsymbol{\rho}(x)\,.\end{aligned}$$

Here, $\circ$ denotes the operation performed on the polynomials by the gadget.

As usual, linear or affine operations are easy to handle, as the respective fault polynomials simply add up. Hence, there is no need to propagate the errors explicitly. In the following, we thus concentrate on how to construct the error propagation polynomial for non-linear operations.

5.2 Two-Input Non-Linear Gadgets

We first start with considering the two-input operations presented in Table 1 and choose the error propagation polynomial such that its shares consist of the highest coefficients of the product of the two input polynomials. If no fault was induced, the coefficients should be zero and, hence, not change the computation result. If, however, a non-zero fault did occur, we will show that with high probability, at least one of these coefficients is non-zero. Hence, adding the error propagation term will most likely change at least one share, invalidating the whole sharing.

More formally, given the (possibly faulted) input shares $(F'_i)_{i\in[n]}, (G'_i)_{i\in[n]}$, the error propagation term we add during the computation is $E_{i,j} \cdot F'_i \cdot G'_i$ with

$$E_{i,j} = \begin{cases} \lambda_{i,n-j-1} & \text{if } 0 \le j < \sigma,\\ 0 & \text{if } \sigma \le j \le n-1. \end{cases}$$

We call this method *multiplicative error propagation* as it adds the error of the share-wise multiplication. We stress here that this error propagation is independent of the secrets

contained in $\mathbf{f}'$ and $\mathbf{g}'$ and independent of the specific operation of the gadget. The reason for this is that the only goal of the multiplicative error propagation is to forward the higher-order coefficients of $\mathbf{f}'$ and $\mathbf{g}'$ (which can only be non-zero due to a fault) to the output of the non-linear computation. Hence, it can be used for all two-input gadgets of Table 1 regardless of whether the gadget itself computes a multiplication, though they have different success probabilities. The new gadgets employing the error polynomial are denoted by a subscript $+$. For example, $(s_0 \cdot s_0', 0)_+$ is the implementation of the multiplication gadget where the error propagation term is added to the specific instantiation as $\ell_{i,j}^{(s_0 \cdot s_0', 0)_+}(F_i', G_i') = \ell_{i,j}^{(s_0 \cdot s_0', 0)}(F_i', G_i') + E_{i,j} \cdot F_i' \cdot G_i'$. The complete error-preserving operations can then be computed with the gadget instantiations of Algorithm 4.

Correctness We first observe that the addition of the error polynomial does not change the correctness of the gadgets on unfaulted inputs.

Lemma 14. *Given one of the two-input (φ_0, φ_1)-gadgets introduced in Table 1, let the result of the operation on some polynomials $\mathbf{f}'$ and $\mathbf{g}'$ without error preservation be denoted by $(Q_0, \dots, Q_{n-1})$ and the ones of the operation with error preservation using the $(\varphi_0, \varphi_1)_+$-gadget from Table 5 as $(Q_0', \dots, Q_{n-1}')$. If $n \geq d + \sigma + 1$ and using polynomials of degree d, then for any Q_i', we have*

$$Q_i' = \begin{cases} Q_i + h_{n-i-1}' & \text{if } 0 \leq i < \sigma, \\ Q_i & \text{if } \sigma \leq i \leq n-1, \end{cases}$$

where h_k' denotes the k-th coefficient of the polynomial $\mathbf{h}' = \mathbf{f}' \cdot \mathbf{g}'$.

Proof. Following the algorithm where $Q_{k,i}'$denotes the loop iteration results using $\ell_{k,i}^{(\varphi_0,\varphi_1)_+}$, we get

$$\begin{aligned} Q_i' &= \sum_{k=0}^{n-1} \tilde{Q}_{k,i}' = \sum_{k=0}^{n-1} \left(\ell_{k,i}^{(\varphi_0,\varphi_1)_+}(F_k', G_k') + \sum_{l=1}^{d-1} r_{k,l} \cdot \alpha_i^l \right) \\ &= \sum_{k=0}^{n-1} \left(\ell_{k,i}^{(\varphi_0,\varphi_1)}(F_k', G_k') + \sum_{l=1}^{d-1} r_{k,l} \cdot \alpha_i^l + E_{k,i} \cdot F_k' \cdot G_k' \right) \\ &= \sum_{k=0}^{n-1} \left(\tilde{Q}_{k,i} + E_{k,i} \cdot F_k' \cdot G_k' \right) = Q_i + \sum_{k=0}^{n-1} E_{k,i} \cdot F_k' \cdot G_k'. \end{aligned}$$

For all $i \geq \sigma$, the statement follows immediately since, for all k, we have $E_{k,i} = 0$. For i with $0 \leq i < \sigma$, with Equation (1) we have

$$Q_i' = Q_i + \sum_{k=0}^{n-1} E_{k,i} \cdot F_k' \cdot G_k' = Q_i + \sum_{k=0}^{n-1} \lambda_{k,n-i-1} \cdot F_k' \cdot G_k' = Q_i + h_{n-i-1}'. \qquad \square$$

The shares of the error propagation polynomial $\boldsymbol{\rho}(x)$ are thus $(h_{n-1}', \dots, h_{n-\sigma}', 0, \dots, 0)$. It is easy to see that this lemma implies the correctness of our construction.

Lemma 15. *Algorithm 4 with $\ell_{i,j}^{(\varphi_0,\varphi_1)_+}(\cdot,\cdot)$ is correct for all two-input instantiations of Table 5 for $n > 2d + \sigma$ when using polynomials of degree d.*

Proof. Recall that $\ell_{i,j}^{(\varphi_0,\varphi_1)_+}(F_i', G_i') = \ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i', G_i') + E_{i,j} \cdot F_i' \cdot G_i'$ is the version including the error propagating term. For the $(\varphi_0, \varphi_1)_+$-gadget, we rely on the correctness proof for the (φ_0, φ_1)-gadget in Lemma 9, therefore, it remains to prove that the added error propagation terms $E_{i,j} \cdot F_i' \cdot G_i'$ do not change the outcome as long as the inputs that share

the two polynomials $\mathbf{f}'$ and $\mathbf{g}'$ are unfaulted. As can be seen in Lemma 14, the value added to a share is generated by the sum over the error propagation terms and results in either h'_{n-j-1} or zero depending on the index j of the share. Since $\mathbf{f}'$ and $\mathbf{g}'$ are not faulted, they are both of degree d and $\mathbf{h}'$ is thus of degree at most $2d$. Due to the construction of the error propagation terms, only the coefficients between h'_{n-1} and $h'_{n-\sigma}$ are added. Due to $n > 2d + \sigma$, we know $n - \sigma > 2d$. Hence, all added coefficients are zero and the output is thus correct. □

In the next segment, we give a security analysis for the case that the attacker introduced faults into the computation. To do so, we follow the approach described in [BEF+23]. First, we prove that, given a faulty pair of input polynomials, if no fault is induced in the gadget, the output polynomial will with high probability also be faulty. Then, faults in the gadget are considered and we will show that all of those faults can be modeled with faults in the input or the output shares. Combining these aspects shows that our gadgets are fault robust and, due to the analysis of the previous section, this allows us to conclude the security against combined attackers.

Faulted inputs without faults in the gadget In the following lemma, we consider the scenario where at least one of the input polynomials $\mathbf{f}'$ or $\mathbf{g}'$ is faulted, i.e., at least one of $\mathbf{f}' \neq \mathbf{f}$ or $\mathbf{g}' \neq \mathbf{g}$ does hold. We also assume that no faults are introduced by the attacker during the computation.

Lemma 16. *Let $(F'_i)_{i\in[n]}, (G'_i)_{i\in[n]}$ denote the input sharings of the two-input error propagation $(\varphi_0, \varphi_1)_+$-gadget of Table 5 and let $(Q'_i)_{i\in[n]}$ be the output sharing. If $n \geq d + \sigma + 1, n > 2d$ and the polynomials have a degree d, then at least one of the input polynomials is invalid due to the insertion of at most σ faults affecting $\omega \leq \sigma$ many share indices, and no faults are introduced during the computation of the gadget, either the faults are ineffective meaning $\mathbf{q}' = \mathbf{q}$ or the coefficients $q'_{n-\omega+1}$, $q'_{n-\omega+2}$, ..., q'_n are uniformly and independent randomly distributed elements of $\mathbb{F}$. In the case of a multiplication gadget, one of these elements is guaranteed to be non-zero.*

Proof. For the output polynomial to be faulty, the error propagation polynomial $\boldsymbol{\rho}(x)$ should have a degree higher than d. Using Lemma 13, it is sufficient to show at least one of the added coefficients h'_j is non-zero. At the same time, if all of these coefficients are zero, the error propagation polynomial itself would be zero. Note that the faulty product polynomial $\mathbf{h}'$ is the sum of the original product polynomial and its own error polynomial $\boldsymbol{\zeta}^{\mathbf{h}}$, i.e., $\mathbf{h}'(x) = \mathbf{f}'(x) \cdot \mathbf{g}'(x) = \mathbf{h}(x) + \boldsymbol{\zeta}^{\mathbf{h}}(x)$. The coefficients of $\mathbf{h}'(x)$ are thus computed by

$$V^{-1}\begin{pmatrix} H'_0 \\ \vdots \\ H'_{\sigma-1} \\ H'_\sigma \\ \vdots \\ H'_{n-1} \end{pmatrix} = V^{-1}\left(\begin{pmatrix} H_0 \\ \vdots \\ H_{\sigma-1} \\ H_\sigma \\ \vdots \\ H_{n-1} \end{pmatrix} + \begin{pmatrix} Z_0^H \\ \vdots \\ Z_{\sigma-1}^H \\ 0 \\ \vdots \\ 0 \end{pmatrix}\right) = \begin{pmatrix} h_0 \\ \vdots \\ h_{2d} \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \begin{pmatrix} \zeta_0^h \\ \vdots \\ \zeta_{2d}^h \\ \zeta_{2d+1}^h \\ \vdots \\ \zeta_{n-1}^h \end{pmatrix} = \begin{pmatrix} h'_0 \\ \vdots \\ h'_{2d} \\ h'_{2d+1} \\ \vdots \\ h'_{n-1} \end{pmatrix}.$$

For $n - \sigma \leq j < n$, this gives $h'_j = \zeta_j^h$. We thus have the following equation system for the last σ coefficients

$$\begin{aligned} \lambda_{0,n-\sigma} Z_0^H + \lambda_{1,n-\sigma} Z_1^H + ... + \lambda_{\sigma-1,n-\sigma} Z_{\sigma-1}^H &= \zeta_{n-\sigma}^h \\ &\vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots \\ \lambda_{0,n-1} Z_0^H + \lambda_{1,n-1} Z_1^H + ... + \lambda_{\sigma-1,n-1} Z_{\sigma-1}^H &= \zeta_{n-1}^h. \end{aligned}$$

Since the support points α_i are pairwise independent, the columns of the submatrix

$$\hat{V}^{-1} = \begin{pmatrix} \lambda_{0,n-\sigma} & \lambda_{1,n-\sigma} & \cdots & \lambda_{\sigma-1,n-\sigma} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{0,n-1} & \lambda_{1,n-1} & \cdots & \lambda_{\sigma-1,n-1} \end{pmatrix}$$

of the inverse V^{-1} are by construction [MS58] linearly independent. Hence, $\hat{V}^{-1}$ has full rank.

Multiplication: For a multiplication gadget, this property has the following very useful implication: The error propagation term mirrors the computation done by the gadget exactly. Hence, $\boldsymbol{\rho}(x)$ and $\boldsymbol{\zeta}^{\mathbf{h}}(x)$ directly depend on the same values Z_i^H. Thus, if $Z_i^H = 0$ for all $0 \leq i < \sigma$, we know that the product polynomial $\mathbf{q}'$ equals the unfaulted one $\mathbf{q}$, as

$$\mathbf{q}'(x) = \mathbf{q}(x) + \boldsymbol{\rho}(x) = \mathbf{q}(x) = \mathbf{h}'(x) = \mathbf{h}(x) + \boldsymbol{\zeta}^{\mathbf{h}}(x) = \mathbf{h}(x).$$

Consequently, this gives us perfect security for the multiplication gadgets as injected faults either can be identified due to an increased degree of the output sharing or the faults become ineffective, i.e., they lead to the same output as their faulted equivalent.

Other gadgets: For other non-linear gadgets that do not compute a multiplication operation, the connection between the error propagation polynomial $\boldsymbol{\rho}(x)$ and error polynomial $\boldsymbol{\zeta}^{\mathbf{h}}(x)$ is not as strong unfortunately. Furthermore, such general (φ_0, φ_1)-gadgets might not even have *any* ineffective faults and thus allowing the adversary to influence the error propagation terms in such a way that the output might be a valid sharing of the wrong value.

To show that this can only happen very rarely, we will now analyze the probability that an adversary can set all necessary Z_i^H to zero. In our multiplicative error propagation, Z_i^H can be concretely computed by

$$H_i' = F_i' \cdot G_i' = (F_i + Z_i^F) \cdot (G_i + Z_i^G) = F_i \cdot G_i + \underbrace{F_i \cdot Z_i^G + Z_i^F \cdot G_i + Z_i^F \cdot Z_i^G}_{\hat{=} Z_i^H},$$

where Z_i^F and Z_i^G are zero for all $i \geq \sigma$.

To zero out Z_i^H, the attacker needs to choose Z_i^F and Z_i^G such that

$$0 \overset{!}{=} F_i \cdot Z_i^G + Z_i^F \cdot G_i + Z_i^F \cdot Z_i^G. \tag{2}$$

As the adversary is non-adaptive, they have to choose the respective faults Z_i^F and Z_i^G before the run of the computation. Hence, due to the uniform distribution of F_i and G_i, if at least one of Z_i^F or Z_i^G is non-zero, the probability that $Z_i^H = 0$ for a given i is $|\mathbb{F}|^{-1}$, as there are no non-trivial zero divisors in fields:

Z_i^F	Z_i^G	reduced eq. (2)	probability
$= 0$	$= 0$	$0 = 0$	1
$= 0$	$\neq 0$	$0 = F_i$	$\lvert\mathbb{F}\rvert^{-1}$
$\neq 0$	$= 0$	$0 = G_i$	$\lvert\mathbb{F}\rvert^{-1}$
$\neq 0$	$\neq 0$	$G_i = -\frac{F_i \cdot Z_i^G}{Z_i^F} - Z_i^G$	$\lvert\mathbb{F}\rvert^{-1}$

Because the number of indices i where Z_i^F or Z_i^G is non-zero is ω, the probability that $Z_i^H = 0$ holds for all i is at most $|\mathbb{F}|^{-\omega}$. This concludes the proof of the lemma, as it implies that the coefficients $q'_{n-\omega+1}, \ldots, q'_n$ are uniformly distributed. □

Non-faulted inputs with faults in the gadget In the following lemma, we consider the scenario where none of the input polynomials $\mathbf{f}'$ or $\mathbf{g}'$ is faulted, i.e., we have $\mathbf{f}' = \mathbf{f}$ and $\mathbf{g}' = \mathbf{g}$, but a fault is introduced by the attacker during the computation. To analyze this situation, we make use of the notion of *fault invariance* introduced in Definition 5, which guarantees that all faults introduced during the computation by the attacker can be pushed either into the inputs or the outputs of the gadget.

Lemma 17. *All gadgets introduced in Table 1 are fault invariant with respect to* $\mathcal{F}^+$.

Proof. As all faults in $\mathcal{F}^+$ are additive faults, linear and affine operations are trivially fault invariant. A close inspection of the (φ_0, φ_1)-gadget presented in Algorithm 4 reveals that the only non-affine operations can appear in the calculation of $\ell_{i,j}^{(\varphi_0,\varphi_1)}(F_i, G_j)$. But, all concrete instantiations in Table 1 share the property that they are the only non linear operation in the gadget. Hence, all faults before the operation of $\ell_{i,j}^{(\varphi_0,\varphi_1)}$ can be moved to the input of the gadget and all faults after the operation of $\ell_{i,j}^{(\varphi_0,\varphi_1)}$ can be moved to the output of the gadget as in [BEF+23]. Formally, moving the linear transformation of an additive fault e on an intermediate value (e.g. $(a+b)$ with input values a and b) such that the fault $((a+b)+e)$ can be described as a fault on the input $((a+e)+b)$. Hence, the intermediate fault can be moved to the input a. The same holds for outputs. □

Faulted inputs with faults in the gadget Finally, we need to consider the most general scenario where the input polynomials $\mathbf{f}'$ and $\mathbf{g}'$ might be faulted *and* faults might be introduced during the computation. The main idea to analyze this situation is to combine Lemma 16 and the fault resilience of the encoding to show that an adversary cannot make the faulty output sharing valid again by faulting the shares (with high probability).

Lemma 18. *The two-input* $(\varphi_0, \varphi_1)_+$*-gadgets of Table 5 are* σ*-f-robust.*

Proof. Let $(Q'_i)_{i\in[n]}$ be the invalid output sharing of a two-input $(\varphi_0, \varphi_1)_+$-gadget of Table 5 where σ_1 faults were injected on previous computations and inputs, and σ_2 faults are injected in the internal computation.

Lemma 16 proves that the standalone σ_1 input faults are either ineffective or cause an error polynomial such that q'_{n-s+1}, q'_{n-s+2}, ..., q'_n are uniformly and independent randomly distributed elements of $\mathbb{F}$. If the faults are ineffective, the first property of σ-f-robust (Def. 3 (i)) is fulfilled. Else, using the notation of Definition 3, we have $(y_i + w'_i)_{i\in[n]} \leftarrow T[\mathsf{G}]\left((x_i + v_i)_{i\in[n]}, (x'_i + v'_i)_{i\in[n]}\right)$ such that $(w'_0, \ldots, w'_{n-1})$ is produced by the following random experiment: A polynomial $p_{w'} \in \mathbb{F}[x]$ is chosen such that the coefficients of x^{d+1}, x^{d+2}, ..., x^{n-t_2} are drawn uniformly at random from $\mathbb{F}$. Then, $w'_i = p_{w'}(\alpha_i)$ for some pairwise different points $\alpha_i \in \mathbb{F} \setminus \{0\}$. Hence, the second property of σ-f-robust (Def. 3 (ii)) is fulfilled in this case.

It remains to prove that the σ_2 intermediate faults do not lead to any problems. We will show that they fulfill the second property of (Def. 3 (i)) such that

$$(y_i + w_i + w'_i)_{i\in[n]} \leftarrow T[\mathsf{G}]\left((x_i + v_i)_{i\in[n]}, (x'_i + v'_i)_{i\in[n]}\right)$$

with $\mathsf{weight}(w) \leq \sigma_2$. The essential observation is that each fault (except the input faults) can only affect one single output share, e.g. $Q_i = y_i + w_i + w'_i$. Consequently there are at most σ_2 output shares in $(Q'_i)_{i\in[n]}$ that differ from $(y_i + w'_i)_{i\in[n]}$. In other words, the final output of the internally faulted gadget can be described with $(y_i + w_i + w'_i)_{i\in[n]}$ such that $\mathsf{weight}(w) \leq \sigma_2$, This concludes the proof as w' is either zero or randomized by the input faults, and the weight of w is restricted by σ_2. □

Table 5: Instantiations of Algorithm 4 to protect the gadgets against active faults. The input sharings are $(F_i)_{i\in[n]}$ and $(G_i)_{i\in[n]}$ and embed the secrets s_0, s_1 and s'_0, s'_1, respectively. The output sharing is $(Q_i)_{i\in[n]}$, and $a, b \in \{0,1\}$. If $\ell_{i,j}^{(\varphi_0,\varphi_1)_+}(F_i, G_i)$ does not use $(G_i)_{i\in[n]}$, we consider the gadget as a gadget with only one input sharing.

$(\varphi_0,\varphi_1)_+$	$\ell_{i,j}^{(\varphi_0,\varphi_1)_+}(F_i,G_i)$	Output	
$(s_a, s'_b)_+$	$\lambda_{i,a\cdot d}F_i + \lambda_{i,b\cdot d}\alpha_j^d G_i + E_{i,j}F_iG_i$	$q_0 = f_{a\cdot d}$	$q_d = g_{b\cdot d}$
$(s_0, s_1)_+$	$(\lambda_{i,0} + \lambda_{i,d}\alpha_j^d + E_{i,j})F_i$	$q_0 = f_0$	$q_d = f_d$
$(s_0 \cdot s'_0, 0)_+$	$(\lambda_{i,0} + E_{i,j})F_iG_i$	$q_0 = f_0 \cdot g_0$	$q_d = 0$
$(0, s_1 \cdot s'_1)_+$	$(\lambda_{i,2d}\alpha_j^d + E_{i,j})F_iG_i$	$q_0 = 0$	$q_d = f_d \cdot g_d$
$(s_0 \cdot s'_0, s_1 \cdot s'_1)_+$	$(\lambda_{i,0} + \lambda_{i,2d}\alpha_j^d + E_{i,j})F_iG_i$	$q_0 = f_0 \cdot g_0$	$q_d = f_d \cdot g_d$

5.3 One-Input Non-Linear Gadgets

We have to also show that the only introduced one-input gadget (s_0, s_1) has an error propagating equivalent $(s_0, s_1)_+$.

Lemma 19. *The one-input $(\varphi_0,\varphi_1)_+$-gadgets of Table 5 are σ-f-robust.*

The proof is similar to the one for two-input gadgets as they are similar to a multiplication gadget with constant second input $G_i = 1$. We briefly sketch the proof in the following.

Proof sketch. Note that the proof of Lemma 18 also works for one-input non-linear gadgets if we set the fault vector v' to zero and all G_i of the second input to one. The reason is that the proof of Lemma 18 does not use the properties of the public factors $\ell_{i,j}^{(\varphi_0,\varphi_1)_+}$ (Table 5), and the structure of the multiplication is consequently the same with $G_i = 1$. It only remains to prove Lemma 16 for one-input gadgets. If at most σ faults were induced into the input polynomial $\mathbf{f}(x)$ of the gadget, the respective error polynomial $\boldsymbol{\zeta}(x)$ is either a zero polynomial or has a degree of at least $n - \sigma$. The case of it being equal to zero can be ignored in the further analysis, as this gives $\mathbf{f}'(x) = \mathbf{f}(x)$ and the correctness of the gadget thus implies the robustness. Otherwise, we consider the error propagation term $E_{i,j} \cdot F'_i$ with $E_{i,j}$ as defined above. The uniform distribution of the higher coefficients $q'_{n-\omega+1}$, $q'_{n-\omega+2}$, ..., q'_n follows from the uniform distribution of F_i. Note that the input polynomials also include the freshly drawn coefficients from the zero encoding and are thus randomized. The correctness follows with the same argument as before. Because of its degree, it again suffices to only consider the highest σ coefficients and setting $\mathbf{h}'(x) = \mathbf{f}'(x)$ allows to adapt the previously stated lemmas to also hold for this gadget. Note that because here $Z_i^H = Z_i^F$ and $\mathbf{f}'(x)$ is invalid, at least one of the highest σ coefficients of $\mathbf{h}'(x)$ in Lemma 16 must be non-zero. □

It follows that all gadgets introduced in Table 5 can be easily changed to propagate at most σ errors with high probability.

5.4 LaOla Gadget

It remains to analyze the LaOla multiplication where its instantiation $\ell_{i,j}^{(\sim)}(F_i)$ is called for the two input sharings $(F_i)_{i\in[n]}, (G_i)_{i\in[n]}$ separately. Again, the approach as well as the necessary proofs is very similar to the ones in Section 5.2 and Section 5.3. We thus only provide a sketch:

Proof sketch. As in Section 5.3, we will define $\ell_{i,j}^{(\sim)_+}(F_i)$ to have the error propagation term $E_{i,j} \cdot F_i$. Hence, we have $\ell_{i,j}^{(\sim)_+}(F_i) = (\lambda_{i,0} + \lambda_{i,d}\alpha_j^{d/2} + E_{i,j})F_i$. In the computation of Algorithm 5 with the adjusted instantiation, let the outputs of the **SplitRed** gadget be $\left(\tilde{F}_i'\right)_{i\in[n]}$, $\left(\tilde{F}_i''\right)_{i\in[n]}$, $\left(\tilde{G}_i'\right)_{i\in[n]}$, and $\left(\tilde{G}_i''\right)_{i\in[n]}$ which describe the underlying polynomials $\sum \tilde{f}_i' x^i$, $\sum \tilde{f}_i'' x^i$, $\sum \tilde{g}_i' x^i$, and $\sum \tilde{g}_i'' x^i$. Because of the added error propagation term, we also have

$$\tilde{F}_i' + \tilde{F}_i'' = \begin{cases} F_i' + F_i'' + f_{n-i-1} & \text{if } 0 \le i < \sigma, \\ F_i' + F_i'' & \text{if } \sigma \le i \le n-1. \end{cases}$$

The same structure also holds for $\tilde{G}_i' + \tilde{G}_i''$ with $\tilde{F}_i' + \tilde{F}_i''$ and $\tilde{G}_i' + \tilde{G}_i''$ being the output shares of the unmodified **SplitRed** gadget. Since the LaOla multiplication now computes their product, it follows that

$$(\tilde{F}_i' + \tilde{F}_i'') \cdot (\tilde{G}_i' + \tilde{G}_i'') = \begin{cases} (F_i' + F_i'') \cdot (G_i' + G_i'') + h_{n-i-1} & \text{if } 0 \le i < \sigma, \\ (F_i' + F_i'') \cdot (G_i' + G_i'') & \text{if } \sigma \le i \le n-1, \end{cases}$$

where h_k denotes the k-th coefficient of the possibly faulty polynomial $\mathbf{h} = \mathbf{f} \cdot \mathbf{g}$.

The correctness of using this instantiation follows with the same argument as in Lemma 15, while the proof of its robustness is similar to Lemma 18. □

5.5 Combined Attacks

The previous sections considered attacks against purely passive probing attacks (Section 4) or purely active faulting attacks (Section 5.2, Section 5.3, and Section 5.4). Here, we consider *combined* attackers that can simultaneously perform probing attacks and faulting attacks. Using combined attacks, an adversary can try to lead the computation to a wrong result or to break the secrecy of the protocol. In [BEF+23], it was rigorously shown that standalone leakage and fault resilient constructions do not imply resilience against combined attacks. We follow their approach and use their observation that those constructions remain secure against combined attacks when they are fault invariant as well.

Theorem 1. *All gadgets constructed in this work are $|\mathbb{F}|^{\omega-\sigma-1}$-secure against (t,ω)-attackers as defined in Section 2.1 with $\omega \le \sigma$.*

Proof. Lemma 18, on the one hand, shows that all constructions are σ-fault-robust. On the other hand, Lemma 8 and Lemma 11 show that the presented gadgets are (S)NI. Hence, all gadgets are secure against standalone probing and faulting attacks. Furthermore, Lemma 17 shows that all constructions are fault invariant, and consequently, Lemma 1 implies that they are fault-resilient (S)NI as well. Finally, Lemma 2 allows us to conclude our theorem about the security of our constructions. □

In essence, we can leverage the same constructions as presented in [BEF+23] to concurrently compute two functions, requiring only minor adjustments to the public λ-parameters. This improvement only incurs the cost of one additional share, which almost halves both the run time and the cost of randomness.

6 Conclusion

In this paper, we introduced the concept of *double sharings*, i.e., embedding two different secrets into a single polynomial sharing. After we proved that embedding a second secret only requires to add another single share to guarantee perfect security, we studied how to extend existing single-secret gadgets to the double-sharing setting. We first focus on gadgets using a BGW-like approach of local computation followed by a degree reduction and show how these gadgets can easily support computations on double sharings. Afterwards, we showed how to extend the recent LaOla multiplication gadget to also work on double sharings. After studying the security of these gadgets against purely passive attacks, we showed how to extend them by introducing error propagation polynomials to also prevent active and combined attacks.

Our construction shows how one can drastically improve the cost (computational and randomness) of existing solutions using polynomial sharing when considering parallel computations and still guarantee security even against very strong combined attackers. The only overhead compared to the single-secret setting is the fact that we need to increase the number of shares by one.

We only presented the foundational work of double sharing in this paper. A very interesting follow-up work upon this would be to perform a practical evaluation of our method, applied to a real-world scenario. As we also guarantee security against combined attackers, this would also require to develop a meaningful simulation of such attackers to obtain relevant experimental results.

In this work, we only consider t-probing attackers. However, as discussed in [BEF+23], the LaOla gadget also guarantees security in the *region probing* model [DDF19]. As our analysis basically mirrors the original analysis of LaOla, the corresponding double-sharing gadget will also be secure in the region probing model.

Another interesting question is whether one can extend our approach beyond two secrets. Due to the special properties of the lowest and highest coefficients of polynomials, we were able to easily extend existing gadgets. Hence, such an extension seems to require the development of completely new gadgets.

Acknowledgments

We thank the anonymous reviewers for their useful feedback. This work has been funded by the German Research Foundation (DFG) CRC 1119 CROSSING (project S7), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE, and by the ERC Grant 101044770 (CRYPTOLAYER). This work was supported by funding from the German Federal Ministry of Education and Research (BMBF) under the project “Sec4IoMT” (ID 16KIS1693).

A Proofs for Section 3 (Security of Two-Secret Sharing)

Proof of Lemma 5. Remember that $\mathcal{P}_{d\to s}$ is the random variable on $\mathbb{F}^d$ describing the polynomial used in the computation of $\mathsf{Share}_d(d \to s)$ by the random choice of the coefficients $a_0, \dots, a_{d-1}$. Let $(\mathcal{A}_0, \dots, \mathcal{A}_{d-1})$ be the random variable on $\mathbb{F}^d$ describing these coefficients $a_0, \dots, a_{d-1}$. Clearly, $\{\mathcal{A}_i\}_{i\in\{0,\dots,d-1\}}$ is a set of uniform independent random variables independent from s.

We will first show (i) and thus need to prove that $\{\mathcal{P}_{d\to s}(\alpha_i)\}_{i\in I}$ is a set of uniform independent random variables independent from s for any subset $I \subseteq \{0, \dots, n-1\}$ with $|I| \leq d$. Without loss of generality, we only show this for $|I| = d$, as this clearly also implies the lemma for $|I| < d$. To prove (i), we show that there is a bijection between the outcomes of $\{\mathcal{P}_{d\to s}(\alpha_i)\}_{i\in I}$ and $\{\mathcal{A}_i\}_{i\in\{0,\dots,d-1\}}$, which directly implies the result.

The proof of (i) is very similar to the one for Shamir's secret sharing given in Lemma 4. The only difference is that instead of considering $\mathsf{Vandermonde}(0, \alpha_I)$, we need to consider a slightly different matrix. Let $V' = \mathsf{Vandermonde}_d(\alpha_I)$ be the $d \times (d+1)$-Vandermonde matrix and V be the $(d+1) \times (d+1)$-matrix where we add a first row of $(0, 0, \dots, 0, 1)$ to V'. Note that V is *not* a Vandermonde matrix. Nevertheless, it is still invertible: Using the Laplace expansion, we expand along the first row. The resulting determinant is $(-1)^{d+2} \cdot \det(\mathsf{Vandermonde}(\alpha_I))$ and thus not zero, as all α_i are distinct. Clearly, we have $V \cdot ((\mathcal{A}_i)_{i\in\{0,\dots,d-1\}}, s) = ((\mathcal{P}_{0\to s}(\alpha_i), s)_{i\in I})$, as the first row of the matrix V is of the form $(0, 0, \dots, 0, 1)$. Hence, the described linear transformation is a bijection on $\mathbb{F}^{d+1}$. In other words, for any outcome $((\mathbf{p}_{d\to s}(\alpha_i))_{i\in I}, s)$ exists exactly one outcome $((a_0, \dots, a_{d-1}), s)$ such that

$$((\mathbf{p}_{d\to s}(\alpha_i))_{i\in I}, s) = V \cdot ((a_0, \dots, a_{d-1}), s) \; .$$

Since V is the identity function in the last coordinate, it follows that for each s', there is also a bijection $V_{s'}$ on $\mathbb{F}^d$ with

$$(\mathbf{p}_{d\to s}(\alpha_i))_{i\in I} = V_{s'} \cdot (a_0, \dots, a_{d-1}) \; .$$

As $\{\mathcal{A}_i\}_{i\in\{0,\dots,d-1\}}$ is a set of uniform independent random variables independent from s, so is $\{\mathcal{P}_{d\to s}(\alpha_i)\}_{i\in I}$.

From Lemma 3, it follows that (ii) holds true since again the polynomial can be reconstructed, directly giving the last coefficient $a_d = s$. □

B The SplitRed gadget

Algorithm 6: SplitRed

Input: Shares of f_0, f_d as $(F_i)_{i\in[n]}$.

Result: Shares of $f'_0, f'_{d/2}$ as $(F'_i)_{i\in[n]}$ and shares of $f''_0, f''_{d/2}$ as $(F''_i)_{i\in[n]}$ such that $f_0 = f'_0 + f''_0$ and $f_d = f'_{d/2} + f''_{d/2}$.

initialize $\left(G_i^j\right)_{i\in[n]}, \left(\Phi_i^j\right)_{i\in[n]}, (F'_i)_{i\in[n]}, (F''_i)_{i\in[n]}$

forall $j \leftarrow 0$ **to** $\frac{n}{2} - 1$ **do**

  $\left(\hat{G}_i^j\right)_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^d$

forall $j \leftarrow 0$ **to** $\frac{n}{2} - 1$ **do**

  $\left(\tilde{G}_i^j\right)_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^{\frac{d}{2}}$; $\left(G_i^j\right)_{i\in[n]} \leftarrow \left(\tilde{G}_i^j\right)_{i\in[n]} + \left(\hat{G}_i^j\right)_{i\in[n]}$

forall $j \leftarrow 0$ **to** $\frac{n}{2} - 1$ **do**

  forall $i \leftarrow 0$ **to** $n - 1$ **do**

    $\Phi_i'^j \leftarrow \ell_{j,i}^{(\sim)}(F_j)$

forall $j \leftarrow 0$ **to** $\frac{n}{2} - 1$ **do**

  $\left(\Phi_i^j\right)_{i\in[n]} \leftarrow \left(\Phi_i'^j\right)_{i\in[n]} + \left(G_i^j\right)_{i\in[n]}$

forall $j \leftarrow 0$ **to** $\frac{n}{2} - 1$ **do**

  $(F'_i)_{i\in[n]} \leftarrow (F'_i)_{i\in[n]} + \left(\Phi_i^j\right)_{i\in[n]}$

forall $j \leftarrow 0$ **to** $\frac{n}{2} - 1$ **do**

  $\left(\tilde{G}_i^{j+\frac{n}{2}}\right)_{i\in[n]} \leftarrow \mathbf{ZEnc}_n^{\frac{d}{2}}$; $\left(G_i^{j+\frac{n}{2}}\right)_{i\in[n]} \leftarrow \left(\tilde{G}_i^j\right)_{i\in[n]} - \left(\hat{G}_i^j\right)_{i\in[n]}$

forall $j \leftarrow 0$ **to** $\frac{n}{2} - 1$ **do**

  forall $i \leftarrow 0$ **to** $n - 1$ **do**

    $\Phi_i'^{j+\frac{n}{2}} \leftarrow \ell_{j+\frac{n}{2},i}^{(\sim)}(F_{j+\frac{n}{2}})$

forall $j \leftarrow 0$ **to** $\frac{n}{2} - 1$ **do**

  $\left(\Phi_i^{j+\frac{n}{2}}\right)_{i\in[n]} \leftarrow \left(\Phi_i'^{j+\frac{n}{2}}\right)_{i\in[n]} + \left(G_i^{j+\frac{n}{2}}\right)_{i\in[n]}$

forall $j \leftarrow 0$ **to** $\frac{n}{2} - 1$ **do**

  $(F''_i)_{i\in[n]} \leftarrow (F''_i)_{i\in[n]} + \left(\Phi_i^{j+\frac{n}{2}}\right)_{i\in[n]}$

return $(F'_i)_{i\in[n]}, (F''_i)_{i\in[n]}$

References

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016.

[BEF+23] Sebastian Berndt, Thomas Eisenbarth, Sebastian Faust, Marc Gourjon, Maximilian Orlt, and Okan Seker. Combined fault and leakage resilience: Composability, constructions and compiler. In *CRYPTO (3)*, volume 14083 of *Lecture Notes in Computer Science*, pages 377–409. Springer, 2023.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BO] Sebastian Berndt and Maximilian Orlt. personal communication.

[DDF19] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. *Journal of Cryptology*, 32(1):151–177, January 2019.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 445–465. Springer, Heidelberg, May / June 2010.

[DN20] Siemen Dhooghe and Svetla Nikova. My gadget just cares for me - how NINA can prove security against combined attacks. In Stanislaw Jarecki, editor, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 35–55. Springer, Heidelberg, February 2020.

[FRBSG22] Jakob Feldtkeller, Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. CINI MINIS: Domain isolation for fault and combined security. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1023–1036. ACM Press, November 2022.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710. ACM Press, May 1992.

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In David B. Shmoys, editor, *46th ACM STOC*, pages 495–504. ACM Press, May / June 2014.

[GLO+21] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. ATLAS: Efficient and scalable MPC in the honest majority setting. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 244–274, Virtual Event, August 2021. Springer, Heidelberg.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *17th ACM PODC*, pages 101–111. ACM, June / July 1998.

[GSF14] Vincent Grosso, François-Xavier Standaert, and Sebastian Faust. Masking vs. multiparty computation: how large is the gap for AES? *Journal of Cryptographic Engineering*, 4(1):47–57, April 2014.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority MPC. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 618–646. Springer, Heidelberg, August 2020.

[Kal84] Dan Kalman. The generalized Vandermonde matrix. *Mathematics Magazine*, 57(1):15–21, 1984.

[LD04] Chun-Pong Lai and Cunsheng Ding. Several generalizations of Shamir's secret sharing scheme. *Int. J. Found. Comput. Sci.*, 15(2):445–458, 2004.

[MS58] Nathaniel Macon and Abraham Spitzbart. Inverses of Vandermonde matrices. *The American Mathematical Monthly*, 65(2):95–100, 1958.

[ph13] poncho (https://crypto.stackexchange.com/users/452/poncho). Coefficients in Shamir's secret sharing scheme. Cryptography Stack Exchange, 2013. URL:https://crypto.stackexchange.com/a/12553 (version: 2023-03-25).

[PR11] Emmanuel Prouff and Thomas Roche. Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 63–78. Springer, Heidelberg, September / October 2011.

[SFRES18] Okan Seker, Abraham Fernandez-Rubio, Thomas Eisenbarth, and Rainer Steinwandt. Extending glitch-free multiparty protocols to resist fault injection attacks. *IACR TCHES*, 2018(3):394–430, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7281.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[WMCS20] Weijia Wang, Pierrick Méaux, Gaëtan Cassiers, and François-Xavier Standaert. Efficient and private computations with code-based masking. *IACR TCHES*, 2020(2):128–171, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8547.