Hardware Private Cubic Circuits

Frederik Reiter and Amir Moradi

Technische Universität Darmstadt, Darmstadt, Germany
{firstname.lastname}@tu-darmstadt.de

Abstract.
Cryptographic hardware implementations often leak secret information through side channels. This can allow attackers to learn secret data, such as a cryptographic key, without any vulnerability in the cryptographic algorithm itself. A popular countermeasure to such attacks is masking, which ensures that processed data is independent of the secrets by splitting them into multiple independent shares, often at the cost of significant overhead in terms of required area, latency, and randomness. The composable PINI notion in the glitch-extended probing model ensures some degree of security against such side-channel analysis attacks, and guarantees that the circuit may be arbitrarily composed with other PINI circuits while maintaining the same security level. This allows for the secure implementation of arbitrary circuits using trivial composition, replacing elementary gates with "gadgets" realizing the same functionality in a PINI-secure manner. Up to now, PINI gadgets at arbitrary security order are limited to quadratic functions, i.e., 2-input gates, with the best known as HPC3.X realizing a 2-input multiplier in one clock cycle.
In this work, we present HPCC, the first low-latency 3-input multiplication gadget for arbitrary fields that maintains a constant latency of one cycle, independent of the number of shares. HPCC additionally allows for the computation of any number of multiplications in a single cycle with relatively little overhead when two of the three operands are identical. When instantiated with two shares and for $\mathbb{F}_2$, HPCC halves the previous record for lowest number of fresh masks required at comparable area cost. With more shares, HPCC is the only single-cycle gadget realizing 3-input multiplications in arbitrary fields. We leverage HPCC to implement the first composable AES S-Box with two cycles of latency with an arbitrary number of shares. This S-Box design significantly outperforms the previous record in terms of area and randomness when instantiated with three shares and stands as the only two-cycle solution for more shares.

Keywords: Side Channel Analysis, Masking, Hardware, PINI, Cubic, HPC

1 Introduction

Due to the physical attacks, developing secure implementations of cryptographic primitives in hardware is known to be challenging, even when the cryptographic primitive itself is secure. In Side-Channel Analysis (SCA) attacks, an adversary with physical access to a device realizing a cryptographic primitive measures certain physical characteristics and behavior of the device, such as timing [Koc96], power consumption [KJJ99] or electromagnetic emanations [GMO01] and makes use of these measurements to deduce information about sensitive data processed, such as a part of the employed symmetric encryption key.

A well-known and extensively studied countermeasure to such attacks is masking, in which any sensitive data is split into multiple independent values, called "shares", such that the sensitive data can only be computed when a certain number of individual shares are

known [CJRR99]. The computation is then performed on these shares individually, forcing the adversary to recover all shares to exploit any information about the sensitive data. This gives implementers an advantage over adversaries, as the complexity of performing SCA attacks grows exponentially in the number of shares [CJRR99], while the overhead introduced into the circuit – in terms of the area footprint, added latency, and required randomness – has only polynomial growth [ISW03].

A circuit computing on shares of sensitive data must be designed with great care to uphold the theoretical security offered by the underlying masking scheme. The d-probing model was introduced in [ISW03] to offer a way of proving the security of a masked circuit under a defined adversary model. In the d-probing model, an adversary is allowed to place up to d probes on any wire in the circuit, and it must be shown that the values observed by these probes do not give any advantage to the adversary to learn any information about the sensitive data. This is determined by showing that the joint distribution of any set of d probed values is independent of the secrets.

The d-probing model can be used to prove some theoretical security of a circuit, but the authors of [MPO05] showed that the assumption that a physical wire only carries information about the value specified by the circuit is not always true. The model actually ignores the presence of temporary intermediate values in the circuit, referred to as "glitches". Any intermediate signal transitions during a clock cycle that are neither intended nor necessary for the functional correctness of the circuit are not present in the idealized circuit of the d-probing model. Nonetheless, these transitions can be observed and used by an adversary in practice to derive information about the processed data. To remedy this shortcoming, the glitch-extended robust d-probing model was introduced in [FGP+18]. In this model, glitches are conservatively assumed to occur at every gate in the circuit, except register cells, and therefore are assumed to leak the value of all input wires of a gate when observing its output. If a circuit is secure under such a model, it ensures that the circuit stays secure in practice even in the presence of worst-case glitches.

Proving the security of a circuit in the glitch-extended robust d-probing model can be a difficult task, due to the high number of possible probe placements that must be considered at higher security orders or for large circuits. The Probe-Isolating Non-Interference (PINI) notion introduced in [CS20] aids in this task. Any circuit secure under the PINI notion can be arbitrarily composed with other PINI circuits, without compromising the security of the composed circuit. This allows for the creation of arbitrarily complex secure circuits by composition, where every logic gate in an insecure circuit is simply replaced by a PINI circuit implementing the same functionality but in the shared (masked) domain. These elementary circuits are commonly referred to as "gadgets". Intuitively, the PINI notion ensures that any probes placed on the output of a gadget can only lead to leakage about the inputs of the gadget from the same share domain. This ensures that any leakage stays "contained" to the probed share domain, even when the input of one gadget is the output of another.

Motivation. The HPC3.X family of multiplication gadgets [KSM22, HB25, CGM+24] all provide a 2-input multiplication PINI gadget at arbitrary security order with only one register stage. The total latency of any circuit composed of such gadgets is therefore at least as high as the maximum number of cascaded multiplication gadgets. The state of the art is also based on a common belief that any PINI multiplication gadget must have at least one register stage inside, and hence the latency of HPC3.X is optimal. The only remaining avenue to further optimize the latency of composed circuits is hence the creation of PINI gadgets that are able to multiply three or more inputs together in a single clock cycle. There are solutions that implement such a gadget, however they are either bounded to first order [VDB+24, SBHM20, KSM22] or their latency increases with higher security orders [VDB+25, LMDM25].

A practical example of a circuit that could benefit from a 3-input multiplication gadget is the 8-bit S-Box of the AES [DR98]. The only non-linear component of the AES S-Box is the inversion in $\mathbb{F}_{2^8}$, which has an algebraic degree of 7. The lowest latency that can be achieved using 2-input PINI multiplication gadgets is therefore 3 cycles, and one such circuit is presented in [HB25]. However, with a 3-input multiplication gadget, it becomes possible to implement the AES S-Box with only 2 cycles of latency, as any function with algebraic degree $3^2 = 9$ can be trivially implemented by two cascaded 3-input multiplication gadgets.

Our Contributions. In this work, we introduce Hardware Private Cubic Circuit (HPCC), a 3-input multiplication gadget for arbitrary fields and arbitrary security orders with only one cycle of latency. To the best of our knowledge, at first security order, HPCC has the lowest randomness requirement among all gadgets capable of implementing a 3-input PINI multiplication in one cycle, halving the lowest known number of 8 required fresh masks [KSM22]. At higher security orders, HPCC is the only PINI gadget implementing a 3-input multiplication in one cycle, at the cost of a higher demand for fresh randomness. We provide a proof for the security of HPCC under the PINI notion in the glitch-extended robust d-probing model at arbitrary security order, and formally verify the security of HPCC up to $d = 3$ using VerifMSI [MT23] and validate the results using PROLEAD [MM22]. We apply HPCC to the AES S-Box to present an implementation with constant two cycles of latency, secure in the glitch-extended d-probing model at arbitrary security order. Our second-order implementation beats the previous lowest required randomness and area for a two-cycle AES S-Box, and is the first design to achieve two cycles of latency at higher arbitrary orders. We validate the security of the AES S-Box using PROLEAD up to $d = 2$ and by FPGA-based experimental analysis up to $d = 3$. All our implementations and analysis results are publicly available on GitHub at

https://github.com/ChairImpSec/HPCC

2 Background

2.1 Notation

We denote an output wire of an arithmetic/logic circuit by its formula, where addition and multiplication correspond to the XOR and AND operations in a Boolean circuit. We denote the presence and position of a register by $\mathrm{Reg}\,[\cdots]$ to imply that the input to the register carries the given formula. We use uppercase greek letters for sets of fresh independent random masks, and lowercase greek letters for individual masks. Specifically, we use Λ and λ to imply "placeholder" masks which will be replaced in the finalized circuit. We denote inputs and outputs of a circuit by capital letters, and use the superscript to indicate multiple inputs in the unshared domain, for example A^0 and A^1. We denote share i of some input A as A_i, where the first share is given by A_0 and the last share is given by A_d. Therefore, every input and output consists of $d+1$ shares, and we refer to d as the security order. We denote sets by calligraphic capital letters, such as $\mathcal{S}$. We use $\not\equiv(\cdots)$ to denote that all arguments are pairwise distinct, for example $\not\equiv(i,j,k) = (i \neq j \wedge i \neq k \wedge j \neq k)$.

2.2 Definitions

Definition 1 (Probe [ISW03])**.** We analyze the security of a circuit by considering probes on the wires of the circuit. A probe represents the distribution of values observed at some wire in a circuit across many executions of the circuit. An internal probe is a probe observing a wire that is not an output of the circuit.

Definition 2 (Simulatability [BBP+16])**.** We say that a set of n probes $\mathcal{P}$ can be simulated by some set $\mathcal{X}$ of primary inputs when there exists a statistical simulator with n outputs such that the distribution of observations made by every probe $P_i \in \mathcal{P}$ is identical to the distribution of the output i of the simulator.

Definition 3 (Extended Probe [FGP+18])**.** We define the glitch-extended probe for some probe to be the set of all circuit inputs and register outputs within the circuit that are reachable via some backwards path through the circuit, starting from the probed wire, without crossing a register stage boundary. Since in this work we deal with only glitch extension of robust probing model [FGP+18], we refer to this by "extended probe" for brevity.

Definition 4 (Probe-Isolating Non-Interference [CS20])**.** Let $\mathcal{I}$ be a set of internal probes and $\mathcal{A}$ a set of probed output share indices so that $|\mathcal{I}| + |\mathcal{A}| \leq d$. Let $\mathcal{O}$ be the set of all output shares with indices in $\mathcal{A}$. A circuit with $d+1$ shares is d-PINI in the robust d-probing model iff there exists a set $\mathcal{B}$ such that $|\mathcal{B}| \leq |\mathcal{I}|$ and the extended probes $\mathcal{I} \cup \mathcal{O}$ can be simulated by inputs with share indices in $\mathcal{A} \cup \mathcal{B}$.

3 Hardware Private Cubic Circuits

HPCC is a generic PINI gadget for the multiplication of three inputs. It computes $Z = ABC$ in the masked domain and is instantiated by HPCC^1 when $d = 1$ and by HPCC^{2+} when $d > 1$. This distinction allows us to perform some optimizations at higher orders which would be incorrect at the first order. In some situations, it can be useful to obtain not only the multiplication of all three inputs to the gadget, but also additionally generate the multiplication of two inputs. It turns out that this is possible with very little overhead in the design of HPCC. Thus, HPCC computes a second output $Y = BC$, which can be easily omitted from the design when it is not needed. Moreover, the multiplication of different A multiplied with the same B and C, such as ABC and $A'BC$, can also be realized with less overhead than adding an entirely distinct second instantiation of HPCC. A practical application of these extra outputs will be demonstrated in Section 4, as the AES S-Box requires the computation of multiple cubic terms with shared operands. We therefore model our gadget such that it takes a generic n number of A as input and denote the unshared inputs as $A^0, \ldots, A^{n-1}$. As an example, A_2^0 indicates the share index 2 of the first A. Accordingly, the gadget actually computes $Z^0, ..., Z^{n-1}$ and Y with $Z^l = A^l BC$ and $Y = BC$. We first construct and prove the basic HPCC gadget with only one Z output and then show that the computation of multiple Z and Y does not change the security properties of the gadget.

Remark 1. The idea of reusing randomness between gadgets performing multiplications with partially the same operands was already discussed and applied in [HB25]. We simply integrate the reuse of randomness directly into the gadget, to allow trivial composition of the gadget.

3.1 First–Order HPCC

The multiplication of three inputs in one cycle at the first order is already possible using multiple different previous works [VDB+25, KSM22]. Nonetheless, we present a new construction using only four bits of fresh randomness at comparable area cost. It is largely based on HPC3 [KM22], and realizes the multiplication of two distinct share domains using a single blinding mask and a correction term, adding pipelining registers as necessary. The correction terms required by the blinded multiplications can be combined into a single register, which is again blinded by an individual mask. The two output shares Z_0 and Z_1 of HPCC^1 are defined in Equation (1) with four random independent fresh masks $\varsigma, \beta, \tau, \theta$.

$$\begin{aligned} Z_i =& \mathrm{Reg}\left[A_i\right]\mathrm{Reg}\left[B_i\right]\mathrm{Reg}\left[C_i\right] \\ &\oplus \mathrm{Reg}\left[A_i\right]\mathrm{Reg}\left[B_i\right]\mathrm{Reg}\left[C_{1-i}\oplus\varsigma\right] \\ &\oplus \mathrm{Reg}\left[A_i\right]\mathrm{Reg}\left[C_i\right]\mathrm{Reg}\left[B_{1-i}\oplus\beta\right] \\ &\oplus \mathrm{Reg}\left[A_i\right]\mathrm{Reg}\left[B_{1-i}C_{1-i}\oplus\tau\right] \\ &\oplus \mathrm{Reg}\left[A_iB_i\varsigma\oplus A_iC_i\beta\oplus A_i\tau\oplus\theta\right] \end{aligned} \tag{1}$$

Theorem 1. HPCC^1 *is correct.*

Proof. We prove the correctness of HPCC^1 by discarding all register stages and showing that the unmasked value of the output, i.e., the sum of output shares, equals the result of multiplication of the unmasked inputs. Removing the registers of Equation (1), the value of the output share Z_i can be expressed as follows.

$$\begin{aligned} Z_i =\ & A_iB_iC_i \oplus A_iB_i(C_{1-i}\oplus\varsigma) \oplus A_iC_i(B_{1-i}\oplus\beta) \oplus A_i(B_{1-i}C_{1-i}\oplus\tau) \\ & \oplus (A_iB_i\varsigma \oplus A_iC_i\beta \oplus A_i\tau \oplus \theta) \\ =\ & A_iB_iC_i \oplus A_iB_iC_{1-i} \oplus A_iB_i\varsigma \oplus A_iB_{1-i}C_i \oplus A_iC_i\beta \oplus A_iB_{1-i}C_{1-i} \oplus A_i\tau \\ & \oplus A_iB_i\varsigma \oplus A_iC_i\beta \oplus A_i\tau \oplus \theta \\ =\ & A_iB_iC_i \oplus A_iB_iC_{1-i} \oplus A_iB_{1-i}C_i \oplus A_iB_{1-i}C_{1-i} \oplus \theta \end{aligned}$$

It can be trivially seen that the sum of both output shares Z_0 and Z_1 leads to the result of the multiplication of the unmasked input values, proving the correctness of Z in HPCC^1.

$$\begin{aligned} Z_0 \oplus Z_1 =\ & A_0B_0C_0 \oplus A_0B_0C_1 \oplus A_0B_1C_0 \oplus A_0B_1C_1 \oplus \theta \\ & \oplus A_1B_1C_1 \oplus A_1B_1C_0 \oplus A_1B_0C_1 \oplus A_1B_0C_0 \oplus \theta \\ =\ & A_0B_0C_0 \oplus A_0B_0C_1 \oplus A_0B_1C_0 \oplus A_0B_1C_1 \\ & \oplus A_1B_0C_0 \oplus A_1B_0C_1 \oplus A_1B_1C_0 \oplus A_1B_1C_1 \\ =\ & (A_0\oplus A_1)(B_0\oplus B_1)(C_0\oplus C_1) \ =\ A\,B\,C \end{aligned}$$

□

Theorem 2. HPCC^1 *is a PINI gadget in the glitch-extended d-probing model for* $d = 1$.

Proof. Having Definition 4 in mind, when $d = 1$, we know that any set of probes $\mathcal{P} = \mathcal{I} \cup \mathcal{O}$ contains exactly one probe. We indeed only need to consider extended probes placed on the gadget outputs and register inputs, as the extended probe for any other probe would be a subset of such an extended probe.

- First, we consider the case where $\mathcal{I}$ contains a probe, and $\mathcal{O}$ is empty. In such a case, the probe must be placed on a register input. However, there exists no register input that contains shares with more than one share domain, and therefore it is trivial to build a simulator when we add the single share domain contained in the probed register to $\mathcal{B}$.
- In the other case, where $\mathcal{O}$ contains an output probe placed on Z_i, we can observe that the registers $\mathrm{Reg}\left[C_{1-i}\oplus\varsigma\right]$, $\mathrm{Reg}\left[B_{1-i}\oplus\beta\right]$, $\mathrm{Reg}\left[B_{1-i}C_{1-i}\oplus\tau\right]$, and $\mathrm{Reg}\left[A_iB_i\varsigma\oplus A_iC_i\beta\oplus A_i\tau\oplus\theta\right]$ are all blinded by some fresh independent masks, and can therefore all be simulated by a fair coin toss each. The remaining register outputs probed by a probe placed on Z_i all contain only a single primary input with share domain i, and can therefore be simulated easily as well, since all shares in domain i are available to the simulator.

□

3.1.1 Expansion of HPCC[1]

To expand HPCC[1] with multiple Z outputs and the Y output, we take advantage of the fact that when some register outputs are present in the extended probing set of any output probe, they may show up in other output probes with the same share index. This is easy to see based on Definition 4, because all output probes with some probed output share index must be simulated, and if the simulator is able to simulate a register output in the extended probing set for one of the outputs, it can trivially simulate the same register output present in any other output probe by simply copying the simulated value to the other probe.

Therefore, when we construct additional outputs Z_i^l (i.e., the i-th output share index of $Z^l = A^l BC$), we may reuse all registers that do not contain any share of A^l, as B and C are identical across all $Z_i^{0,\dots,n\text{-}1}$. Hence, only the register blinded by θ cannot be reused and must be blinded by a new mask θ^l for every Z_i^l.

To construct the Y output, we take advantage of the fact that Z_0 already contains the multiplication A_0BC, and simply omit the multiplication with A_0 to obtain the desired result. When omitting A_0 from a register input, this register must be blinded with a new mask, which we call κ. The other share Y_1 must then only cancel the blinding mask used for the new register. The outputs Y_0 and Y_1 are defined as follows. We should note that the register considered for Y_1 is for pipelining purposes.

$$\begin{aligned} Y_0 =& \operatorname{Reg}[B_0]\operatorname{Reg}[C_0] \\ &\oplus \operatorname{Reg}[B_0]\operatorname{Reg}[C_1 \oplus \varsigma] \\ &\oplus \operatorname{Reg}[C_0]\operatorname{Reg}[B_1 \oplus \beta] \\ &\oplus \operatorname{Reg}[B_1C_1 \oplus \tau] \\ &\oplus \operatorname{Reg}[B_0\varsigma \oplus C_0\beta \oplus \tau \oplus \kappa] \end{aligned} \qquad Y_1 = \operatorname{Reg}[\kappa] \tag{2}$$

3.2 Higher–Order HPCC

Here we focus on second- and higher-order security. When multiplying three shared values, at most three distinct share domains can be involved in a multiplication term. We call such a multiplication term a "three-domain" multiplication. The challenge of creating a 3-input multiplication gadget in one cycle lies in realizing such a three-domain multiplication securely. The multiplication of values from only one or two distinct share domains is already realized by the existing gadgets implementing the shared 2-input multiplication. Therefore, we first focus on the three-domain multiplication, and add the one- and two-domain multiplication terms at the end. We focus on the implementation of Z for the same reason, deferring the implementation of Y until the end.

3.2.1 Blinded Three-Domain Multiplication

We consider the multiplication of three shares A_i, B_j, C_k with three pairwise distinct domains i, j and k. We use the same structure as in HPC3.X, where any multiplication term with A_i is included in the output share Z_i, and so B_j and C_k must be somehow blinded in the multiplication term. The design of HPCC[2+] should only introduce one cycle of latency, hence a register output can never be the input of another register in the same gadget. Therefore, B_j and C_k must be blinded and stored in separate registers, such as $A_i \operatorname{Reg}[B_j \oplus \bigoplus \Lambda] \operatorname{Reg}[C_k \oplus \bigoplus \Lambda']$ for some set of random independent masks Λ and Λ' with at least one fresh mask each. The HPCC[2+] gadget realizes the multiplication using this structure, which we refer to by "HPC-style".

Proposition 1. *A 3-input HPC-style gadget with only one fresh mask in each Λ and Λ' cannot be implemented optimally with respect to the area or the required randomness.*

Proof. Let us consider the term Z_i. This term must contain the multiplication of B_j and C_k (for any distinct i, j, k), which must be realized in such a way that a single internal probe leaks only one of the two values B_j and C_k, and a probe on Z_i leaks neither of the two. If either B_j or C_k is leaked by a probe placed on Z_i, then a single output probe would lead to leakages at another share domain, which is not allowed by the PINI notion. Let us define an optimal realization of this multiplication as $\mathrm{Reg}\,[B_j \oplus \lambda]\ \mathrm{Reg}\,[C_k \oplus \lambda']$ with two independent fresh masks λ and λ'. Therefore, Z_i must also contain a correction term $\lambda\lambda'$.

Again, this multiplication must be realized in such a way that a single internal probe leaks only one of the two values λ and λ'. Otherwise, such an internal probe and a probe on Z_i would leak information about both share domains j, k. Further, a probe on Z_i must leak neither of λ and λ', as otherwise such a probe on Z_i would leak a share domain other than i which contradicts with the initial requirement. Therefore, this term must also be realized as a blinded multiplication of λ and λ'. The realization of this multiplication however has the exact same requirements as the multiplication of B_j and C_k, in that an internal probe may only leak one of the two multiplicands and a probe on Z_i may leak none. If we assume that there exists an implementation realizing this blinded multiplication of λ and λ' with these requirements, such as by using more than one blinding mask, we can equivalently use this realization directly for B_j and C_k, removing λ and λ' from the circuit. In other words, by using only one blinding mask for each multiplicand, we have made no "progress" and arrived back at the same requirement we tried to solve by blinding with λ and λ' in the first place. This contradicts the assumption that the multiplication was realized optimally. □

We therefore design HPCC^{2+} so that each of the sets Λ and Λ' contains two fresh masks by setting B_j to be blinded by $\beta_{ijk} \oplus \beta'_{ijk}$ and C_k by $\varsigma_{ijk} \oplus \varsigma'_{ijk}$. This means that the product $A_iB_jC_k$ is realized by $A_i\,\mathrm{Reg}\left[B_j \oplus \beta_{ijk} \oplus \beta'_{ijk}\right]\ \mathrm{Reg}\left[C_k \oplus \varsigma_{ijk} \oplus \varsigma'_{ijk}\right]$. Inspired by HPC2 [CGLS21] and HPC3 [KM22], we reuse $\beta_{ijk}, \beta'_{ijk}, \varsigma_{ijk}$ and ς'_{ijk}, by setting $\beta_{ijk} = \beta_{jik}$, $\beta'_{ijk} = \beta'_{kji}$, $\varsigma_{ijk} = \varsigma_{kji}$ and $\varsigma'_{ijk} = \varsigma'_{kji}$. The product $A_i(B_j \oplus \beta_{ijk} \oplus \beta'_{ijk})(C_k \oplus \varsigma_{ijk} \oplus \varsigma'_{ijk})$ contains 8 additional terms, apart from the desired product $A_iB_jC_k$. These "correction terms" must be somehow canceled in the result to ensure correctness of the gadget. In $\mathbb{F}_2$, they may simply be added a second time, as addition is its own inverse in $\mathbb{F}_2$. In arbitrary fields, they must be subtracted. Some of these correction terms are the product of two shares from different share domains, such as $A_iB_j\varsigma_{ijk}$, and thus can only be realized using another blinded multiplication. A naïve approach to these correction terms would be to realize every term individually as a blinded multiplication. However, we can do better by combining certain correction terms into one blinded multiplication. Intuitively, this is possible when a probe on the combined term is not a "stronger" probe than a probe placed on one of the individual terms.

For example, the terms $A_iB_j\varsigma'_{ijk}$ and $A_i\beta'_{ijk}\varsigma'_{ijk}$ should each be realized as a blinded multiplication with A_i such as $A_i\,\mathrm{Reg}\left[B_j\varsigma'_{ijk} \oplus \lambda_{ijk}\right]$ and $A_i\,\mathrm{Reg}\left[\beta'_{ijk}\varsigma'_{ijk} \oplus \lambda'_{ijk}\right]$ with two independent fresh masks λ_{ijk} and λ'_{ijk}. By combining both terms into $A_i\,\mathrm{Reg}\left[(B_j \oplus \beta'_{ijk})\varsigma'_{ijk} \oplus \lambda_{ijk}\right]$, we can omit λ'. A probe placed on the register input of the combined multiplication is exactly as useful as a probe on the register input of $\mathrm{Reg}\left[B_j\varsigma'_{ijk} \oplus \lambda_{ijk}\right]$ because β'_{ijk} is only used to blind B_j, which is already observed by the adversary anyway.

Notably, the actual value of k does not matter for this logic; hence, we can actually combine this term across all k distinct from i and j to form the correction term $A_i\,\mathrm{Reg}\left[\bigoplus_{\not\equiv(i,j,k)}^{\forall k}\left((B_j \oplus \beta'_{ijk})\varsigma'_{ijk}\right) \oplus \lambda_{ij}\right]$ for each j. This is valid, because once B_j is observed by the adversary, all β'_{ijk} no longer serve a purpose as they only blind B_j and may hence be observed by the adversary. The presence of ς'_{ijk} for all k distinct from i and

j in a single register is of more concern, as these masks protect many different shares of C. For now, we note that ς'_{ijk} is of no help to recover C_k without observing ς_{ijk}. We further note that we must carefully reason about any correction term involving ς_{ijk} (for any k) so that only one share of C leaks by each additional internal probe.

Following the same logic, $A_i \operatorname{Reg}\left[\beta'_{ijk} C_k \oplus \lambda'_{ijk}\right]$ and $A_i \operatorname{Reg}\left[\beta'_{ijk}\varsigma_{ijk} \oplus \lambda''_{ijk}\right]$ can be combined into one blinded multiplication across all j to form the correction term $A_i \operatorname{Reg}\left[\bigoplus^{\forall j}_{\not\equiv(i,j,k)} \left((C_k \oplus \varsigma_{ijk})\beta'_{ijk}\right) \oplus \lambda'_{ij}\right]$ for each k. We again note that β'_{ijk} blinds B_j, and that we must ensure that any correction term involving β_{ijk} should be constructed in a way that only one share of B leaks by each additional internal probe.

We similarly combine the correction terms $A_i B_j \varsigma_{ijk}$ and $A_i \beta_{ijk}\varsigma_{ijk}$ to $A_i \operatorname{Reg}\left[(B_j \oplus \beta_{ijk})\varsigma_{ijk} \oplus \lambda''\right]$, but we cannot combine the registers from two terms with different k (like k and k'), as this would otherwise allow an adversary to recover all fresh masks for C_k, $C_{k'}$, A_i, and B_j with only three probes, which is not in accordance with the PINI notion when HPCC^{2+} is instantiated at security order 3 or higher. The first probe would be placed on the output Z_i, the second one on the input of the register $\operatorname{Reg}\left[\bigoplus^{\forall k}_{\not\equiv(i,j,k)} \left((B_j \oplus \beta'_{ijk})\varsigma'_{ijk}\right) \oplus \lambda\right]$, and the third one on the input of the register of the newly combined correction terms. The same holds for the combined multiplication $A_i \operatorname{Reg}\left[(C_k \oplus \varsigma'_{ijk})\beta_{ijk} \oplus \lambda'''\right]$.

The construction of all three-domain multiplications for some output share i (so-called Z'_i) is shown in Equation (3). Note that in the construction of the correction terms, we use blinded multiplications, which require their own correction terms. These are omitted and represented as "$\cdots$" for now.

$$
\begin{aligned}
Z'_i &= \bigoplus^{\forall j,k}_{\not\equiv(i,j,k)} (A_i B_j C_k) && (3)\\
&= A_i \left(\bigoplus^{\forall j,k}_{\not\equiv(i,j,k)} \operatorname{Reg}\left[B_j \oplus \beta_{ijk} \oplus \beta'_{ijk}\right] \operatorname{Reg}\left[C_k \oplus \varsigma_{ijk} \oplus \varsigma'_{ijk}\right] \right) \\
&\oplus A_i \left(\bigoplus^{\forall j}_{\not\equiv(i,j)} \operatorname{Reg}\left[\bigoplus^{\forall k}_{\not\equiv(i,j,k)} \left((B_j \oplus \beta'_{ijk})\varsigma'_{ijk}\right) \oplus \lambda \right] \right) && (4)\\
&\oplus A_i \left(\bigoplus^{\forall k}_{\not\equiv(i,k)} \operatorname{Reg}\left[\bigoplus^{\forall j}_{\not\equiv(i,j,k)} \left((C_k \oplus \varsigma_{ijk})\beta'_{ijk}\right) \oplus \lambda' \right] \right) && (5)\\
&\oplus A_i \left(\bigoplus^{\forall j,k}_{\not\equiv(i,j,k)} \operatorname{Reg}\left[(B_j \oplus \beta_{ijk})\varsigma_{ijk} \oplus \lambda''\right] \right) && (6)\\
&\oplus A_i \left(\bigoplus^{\forall j,k}_{\not\equiv(i,j,k)} \operatorname{Reg}\left[(C_k \oplus \varsigma'_{ijk})\beta_{ijk} \oplus \lambda'''\right] \right) && (7)\\
&\oplus \cdots
\end{aligned}
$$

Next, we note that Equation (3) contains two pairs of sums (Equations (4), (5) and Equations (6), (7)) over the same indices, only differing in the label of the summation variable. We can rename the summation variable so that the "outer" variable is always j

and then merge these two pairs into one summation each, such as

$$\bigoplus_{\not\equiv(i,j)}^{\forall j} \left(\mathrm{Reg}\left[\bigoplus_{\not\equiv(i,k,j)}^{\forall k} \left((B_j \oplus \beta'_{ijk}) \varsigma'_{ijk} \right) \oplus \lambda \right] \oplus \mathrm{Reg}\left[\bigoplus_{\not\equiv(i,j,k)}^{\forall k} \left((C_j \oplus \varsigma_{ikj}) \beta'_{ikj} \right) \oplus \lambda' \right] \right)$$

and

$$\bigoplus_{\not\equiv(i,j,k)}^{\forall j,k} \left(\mathrm{Reg}\Big[\left(B_j \oplus \beta_{ijk} \right) \varsigma_{ijk} \oplus \lambda'' \Big] \oplus \mathrm{Reg}\Big[\left(C_j \oplus \varsigma'_{ikj} \right) \beta_{ikj} \oplus \lambda''' \Big] \right).$$

Both registers in each pair only leak shares from some domain j, and leak some, but not all, blinding masks used for some shares in another domain k. This property is preserved when combining the register pairs into one register, since the paired registers do not contain any "complementary" masks, i.e., masks that are used together to blind some share from domain k in another multiplication. Thus, their combination still only leaks some, but not all, blinding masks for shares from domain k, preserving the PINI property. We may hence merge the register pairs into one register each.

Lastly, we need to consider the masks that should be used in the blinded multiplications of the primary correction terms. We observe that when an adversary places a probe on two output shares Z_i and $Z_{j\neq i}$, all shares from domains i and j may be revealed by the adversary. Hence, we may reuse the mask employed to blind the shares in the first register in Z_j. We call this mask τ and set $\tau_{ij} = \tau_{ji}$. The second register requires more careful consideration, since it cannot be blinded by any single mask that is reused in Z_j, nor a single mask that is reused in Z_k. If the mask would be reused in Z_j, then two probes could reveal not only shares from domain i and j, but also all blinding masks for shares in some domain k, since both the first and second correction term registers can be unblinded. The mask can also not be reused in Z_k, because the register immediately contains shares of domain j, and as such two probes on Z_i and Z_k would reveal shares from domain j. We therefore must blind this register with two shares, one of which is reused in Z_j, and the other one is reused in Z_k. This is to ensure that only when all three outputs Z_i, Z_j, and Z_k are probed can an adversary cancel out the blinding masks and therefore leak information about shares in domains i, j and k. We call these masks ϵ and ϵ' and set $\epsilon_{ijk} = \epsilon_{jik}$ and $\epsilon'_{ijk} = \epsilon'_{kji}$.

The terms required to cancel the extraneous terms from the blinded multiplications in the primary correction terms can be created in a similar manner. First, we blind each term individually with a fresh mask, such as $\mathrm{Reg}\left[A_i \tau_{ij} \oplus \lambda\right]$, and then merge these terms when merging them does not violate the PINI notion. Thus, we add d more masked registers to every Z'_i, each canceling the erroneous terms from a blinded multiplication involving masks used to blind a share from domain j. The new registers are shown in Equation (10). A probe on any of these new registers leaks masks that can be used to unblind some masked register with shares from domain j, and contains some, but not all masks, for some registers containing shares from domain k. We can therefore reuse the blinding mask for these new registers as Z_j, since a probe on Z_j leaks share domain j anyway. We call this mask θ and set $\theta_{ij} = \theta_{ji}$. The final design of the three-domain multiplication terms is shown below.

$$Z'_i = \bigoplus_{\not\equiv(i,j,k)}^{\forall j,k} (A_i B_j C_k) \tag{8}$$

$$= A_i \left(\bigoplus_{\not\equiv(i,j,k)}^{\forall j,k} \left(\mathrm{Reg}\left[B_j \oplus \beta_{ijk} \oplus \beta'_{ijk}\right] \mathrm{Reg}\left[C_k \oplus \varsigma_{ijk} \oplus \varsigma'_{ijk}\right] \right) \right)$$

$$\oplus\ A_i \left(\bigoplus_{\not\equiv(i,j)}^{\forall j} \mathrm{Reg}\left[\bigoplus_{\not\equiv(i,j,k)}^{\forall k} \left((B_j \oplus \beta'_{ijk})\varsigma'_{ijk} \oplus (C_j \oplus \varsigma_{ikj})\beta'_{ikj} \right) \oplus \tau_{ij} \right] \right)$$

$$\oplus\ A_i \left(\bigoplus_{\not\equiv(i,j,k)}^{\forall j,k} \mathrm{Reg}\Big[(B_j \oplus \beta_{ijk})\varsigma_{ijk} \oplus (C_j \oplus \varsigma'_{ikj})\beta_{ikj} \oplus \epsilon_{ijk} \oplus \epsilon'_{ijk} \Big] \right) \tag{9}$$

$$\oplus \bigoplus_{\not\equiv(i,j)}^{\forall j} \mathrm{Reg}\left[A_i \left(\tau_{ij} \oplus \bigoplus_{\not\equiv(i,j,k)}^{\forall k} \left(\epsilon_{ijk} \oplus \epsilon'_{ikj} \right) \right) \oplus \theta_{ij} \right] \tag{10}$$

3.2.2 Blinded Two-Domain Multiplications

The multiplication of two distinct share domains such as $A_iB_iC_j$ could naïvely be realized exactly as in HPCC[1] using one additional blinding mask for C_j. However, we can do better in two ways. Firstly, any multiplication of the form $A_iB_jC_j$ can be realized "for free" by simply including the B_jC_j term in the register in Equation (9). This is valid because the register already contains B_j and C_j; hence, it does not change the extended probing set obtained by a probe placed on the register input. For the multiplications of the form $A_iB_iC_j$ and $A_iB_jC_i$, we reuse an existing register that was instantiated for the three-domain multiplication which already contains the different share domains blinded by some masks, which avoids introducing new blinding masks entirely. We pick an arbitrary distinct index k and use the register for the multiplication $A_iB_jC_k$ (or $A_iB_kC_j$), which is $\mathrm{Reg}\left[B_j \oplus \beta_{ijk} \oplus \beta'_{ijk}\right]$ (or $\mathrm{Reg}\left[C_j \oplus \varsigma_{ikj} \oplus \varsigma'_{ikj}\right]$). We use a function $\tilde{\mathsf{k}}(.,.)$ to derive a share domain $\tilde{k}$ which is distinct from both input arguments. In all places where $\tilde{\mathsf{k}}(.,.)$ is applied, i and j are used in the immediate context as the input arguments. Hence, we omit the input arguments and instead write only $\tilde{k}$. We use the following definition for $\tilde{\mathsf{k}}(.,.)$, which guarantees that $\forall i, j, i \neq j : \tilde{\mathsf{k}}(i,j) \neq i,\ \tilde{\mathsf{k}}(i,j) \neq j$ by construction:

$$\tilde{\mathsf{k}}(i,j) = \begin{cases} j+2 & \text{if } j+1 \equiv i \\ j+1 & \text{otherwise} \end{cases} \pmod{d+1}.$$

The realization of all multiplications of type $A_iB_iC_j$ is shown in Equation (11).

$$A_iB_i \left(\bigoplus_{\not\equiv(i,j)}^{\forall j} \left(\mathrm{Reg}\left[C_j \oplus \varsigma_{i\tilde{k}j} \oplus \varsigma'_{i\tilde{k}j}\right] \right) \right) \tag{11}$$

The correction term $A_iB_i(\varsigma_{i\tilde{k}j} \oplus \varsigma'_{i\tilde{k}j})$ can be realized by adding the term to the register in Equation (10), since this register already contains a share from share domain i and τ_{ij}, which blinds shares from share domain j. The multiplication $A_iB_jC_i$ is realized in the same manner, once again integrating the correction term $A_i(\beta_{ij\tilde{k}} \oplus \beta'_{ij\tilde{k}})C_i$ into Equation (10).

3.2.3 Final Design

We add pipelining registers into the final design and add the multiplication of shares from only one share domain without any blinding to the equivalent output domain, just like

in HPCC[1]. The final circuit for Z is defined in Equation (12), for $d+1$ output shares $Z_{i\in\{0,\dots,d\}}$ with $d>1$. In Appendix A we provide a circuit diagram for Z_0 with $d=2$ and discuss the intuition behind the blinding masks in the design.

$$
\begin{aligned}
Z_i =& \operatorname{Reg}[A_i]\operatorname{Reg}[B_i]\operatorname{Reg}[C_i] \\
&\oplus \operatorname{Reg}[A_i]\operatorname{Reg}[B_i]\left(\bigoplus_{\not\equiv(i,j)}^{\forall j}\operatorname{Reg}\left[C_j \oplus \varsigma_{i\tilde{k}j} \oplus \varsigma'_{i\tilde{k}j}\right]\right) \\
&\oplus \operatorname{Reg}[A_i]\operatorname{Reg}[C_i]\left(\bigoplus_{\not\equiv(i,j)}^{\forall j}\operatorname{Reg}\left[B_j \oplus \beta_{ij\tilde{k}} \oplus \beta'_{ij\tilde{k}}\right]\right) \\
&\oplus \operatorname{Reg}[A_i]\left(\bigoplus_{\not\equiv(i,j,k)}^{\forall j,k}\operatorname{Reg}\left[B_j \oplus \beta_{ijk} \oplus \beta'_{ijk}\right]\operatorname{Reg}\left[C_k \oplus \varsigma_{ijk} \oplus \varsigma'_{ijk}\right]\right) \\
&\oplus \operatorname{Reg}[A_i]\left(\bigoplus_{\not\equiv(i,j)}^{\forall j}\operatorname{Reg}\left[B_jC_j \oplus \bigoplus_{\not\equiv(i,j,k)}^{\forall k}\left((B_j \oplus \beta'_{ijk})\varsigma'_{ijk} \oplus (C_j \oplus \varsigma_{ikj})\beta'_{ikj}\right) \oplus \tau_{ij}\right]\right) \\
&\oplus \operatorname{Reg}[A_i]\left(\bigoplus_{\not\equiv(i,j,k)}^{\forall j,k}\operatorname{Reg}\left[(B_j \oplus \beta_{ijk})\,\varsigma_{ijk} \oplus \left(C_j \oplus \varsigma'_{ikj}\right)\beta_{ikj} \oplus \epsilon_{ijk} \oplus \epsilon'_{ijk}\right]\right) \\
&\oplus \bigoplus_{\not\equiv(i,j)}^{\forall j}\operatorname{Reg}\left[A_i\left(B_i\left(\varsigma_{i\tilde{k}j} \oplus \varsigma'_{i\tilde{k}j}\right) \oplus C_i\left(\beta_{ij\tilde{k}} \oplus \beta'_{ij\tilde{k}}\right) \oplus \tau_{ij} \oplus \bigoplus_{\not\equiv(i,j,k)}^{\forall k}\left(\epsilon_{ijk} \oplus \epsilon'_{ikj}\right)\right) \oplus \theta_{ij}\right]
\end{aligned}
\tag{12}
$$

In this equation, $\beta, \beta', \varsigma, \varsigma', \epsilon, \epsilon', \tau$ and θ are independent fresh masks drawn from a uniform distribution at random, such that $\beta_{ijk} = \beta_{jik}$, $\beta'_{ijk} = \beta'_{kji}$, $\varsigma_{ijk} = \varsigma_{kji}$, $\varsigma'_{ijk} = \varsigma'_{jik}$, $\epsilon_{ijk} = \epsilon_{jik}$, $\epsilon'_{ijk} = \epsilon'_{kji}$, $\tau_{ij} = \tau_{ji}$ and $\theta_{ij} = \theta_{ji}$. As stated, $\tilde{k}$ is the implicit application of $\tilde{\mathsf{k}}(.,.)$ on i and j from the immediate context.

Remark 2. In the implementation of HPCC, the number of required multiplications – especially when an argument is $\operatorname{Reg}[A_i]$ – can be reduced using the inverse distributive law to combine multiplications as far as possible.

3.2.4 Correctness of HPCC^{2+}

Theorem 3. HPCC^{2+} *is correct for all* $d>1$, *i.e.,* $\bigoplus_{\forall i} Z_i = (\bigoplus_{\forall i} A_i)(\bigoplus_{\forall i} B_i)(\bigoplus_{\forall i} C_i)$.

We do not provide a pen-and-paper proof for the correctness of HPCC^{2+} for brevity. Instead, we provide a formal proof of Theorem 3 in the Lean theorem prover [dMKA+15]. Lean is a programming language and theorem prover, capable of formally proving mathematical statements using automation and interactive proof tactics. A proof of Theorem 3 consist mainly of "straightforward" rewrite steps because we must only transform the left side of the equality to the right in order to complete the proof, and it is therefore well-suited for formal verification. We define HPCC^{2+} over an arbitrary field F, and name the number of shares as N. The type `Fin N` contains all natural numbers x with $0 \leq x < N$, and therefore represents all the valid share indices. We define the shared inputs A, B and C as functions `Fin N` $\rightarrow$ `F` and the fresh random masks equivalently as functions with two or three arguments from `Fin N`, such as τ`: Fin N` $\rightarrow$ `Fin N` $\rightarrow$ `F`. Z is then a function mapping the defined inputs, random variables, and a share index i to an element in the field F. Its definition matches the definition given in Equation (12) for $\mathbb{F}_2$ but is extended to arbitrary fields by negating the correction terms where necessary. A

sketch of the required definitions and the theorem are given in Figure 1, but the definition of Z_i and the proof of the theorem are omitted for brevity. The inputs and random masks are defined as explained. It turns out, for correctness, only the reuse of θ is required, which is defined by the hypothesis $h\theta$ given as an argument to the `HPCC_Gen_is_Correct` theorem. It states the assumption that $\theta_{ij} = -\theta_{ji}$, which is a generalization of the mask reuse to fields where the addition is not necessarily its own inverse. The theorem also requires hN, which states that every input is shared into at least three values, i.e., $d > 1$. The theorem then states exactly what is given in Theorem 3, i.e., that the sum of all output shares is equal to the product of the unmasked inputs. The formalized definition and correctness proof is available in the GitHub repository[1] and can be checked by Lean using the command `lake lean HPCC_Gen_Is_Correct.lean`.

```
-- Definitions
variable {N : ℕ}
variable {F : Type*} [Field F]
variable (A B C : Fin N → F)
variable (θ τ : Fin N → Fin N → F)
variable (β β' ς ς' ε ε' : Fin N → Fin N → Fin N → F)
def Z (i : Fin N) (h : N > 2) : F := -- omitted

-- Correctness Theorem
theorem HPCC_Gen_Is_Correct (hN : N > 2) (hθ : ∀ i j, θ i j = -θ j i)
    : Σ i, Z A B C θ τ β β' ς ς' ε ε' i hN
    = (Σ i, A i) * (Σ i, B i) * (Σ i, C i) := by -- proof omitted
```

Figure 1: Formalization of Theorem 3 in Lean

3.2.5 Security of HPCC^{2+}

Theorem 4. HPCC^{2+} *is a PINI gadget under the glitch-extended d-probing model when instantiated with the security parameter* $d > 1$.

Proof. As given in the proof of Theorem 2, we only need to consider the extended probes placed on the gadget outputs and register inputs. Additionally, we can ignore any register input that contains only one primary input, such as $\mathrm{Reg}\,[A_i]$, because such a probe is a subset of the extended probe placed on any wire containing that register output, such as Z_i.

All registers in HPCC^{2+} which contain not only a single primary input are of the form of $\mathrm{Reg}\,[\cdots \oplus \Lambda]$, where Λ is the sum of one or more uniformly random independent fresh masks. We refer to the masks in Λ as the "blinding masks" for a "blinded register" denoted as $[\![\mathcal{P}]\!]_\Lambda$, where $\mathcal{P}$ is the set of probes obtained by a probe placed on the input of the corresponding register. For the purpose of building a simulator, we make two observations about the simulatability of a probe placed on the output of such a register:

- Firstly, when the simulator is able to simulate all probes in $\mathcal{P}$, it is trivial to simulate the probe placed on the register output as well, as the simulator can simply combine the simulated extended probes in exactly the same way as they are combined to form the register output.
- Secondly, when there is any mask in Λ that is not observed by any other extended probe (in the targeted probing set), and is not present in the blinding masks of any other blinded register (which is also among the targeted probing set), the simulator

[1] https://github.com/ChairImpSec/HPCC/tree/main/proof/functional

can toss a fair coin and simulate the probe placed on the output of the register. We refer to this kind of simulation of a probe as "trivial simulation" of the blinded register.

Therefore, in order to show that it is possible to simulate a probe placed on the output of a blinded register, it suffices to show that all probes in $\mathcal{P}$ can be simulated or there exists a mask in Λ which is not required for the simulation of any other probe.

Remark 3. There are more cases where not all probes in $\mathcal{P}$ are required to simulate $[\![\mathcal{P}]\!]_\Lambda$. However, for the purpose of this proof it suffices to consider only the simple special case where there exists a unique blinding mask used only in the simulation of that register. We will be able to show that when this conservative estimation of trivial simulatability is not fulfilled, the simulator has access to the entirety of $\mathcal{P}$ anyway; hence, a more precise but costly analysis of the simulatability is not necessary for this proof.

Any probe placed on the output share or a register input of the circuit realizing Equation (12) fits to one of the following extended probing sets.

$$
\begin{aligned}
\mathcal{P}_B(i,j,k) &= \left\{B_j, \beta_{ijk}, \beta'_{ijk}\right\} \\
\mathcal{P}_C(i,j,k) &= \left\{C_k, \varsigma_{ijk}, \varsigma'_{ijk}\right\} \\
\mathcal{P}_1(i,j) &= \left\{B_j, C_j, \beta'_{ijk}, \beta'_{ikj}, \varsigma_{ikj}, \varsigma'_{ijk}, \tau_{ij} \mid \forall k, k \neq i, k \neq j\right\} \\
\mathcal{P}_2(i,j,k) &= \left\{B_j, C_j, \beta_{ijk}, \beta_{ikj}, \varsigma_{ijk}, \varsigma'_{ikj}, \epsilon_{ijk}, \epsilon'_{ijk}\right\} \\
\mathcal{P}_3(i,j) &= \left\{A_i, B_i, C_i, \beta_{ij\tilde{k}}, \beta'_{ij\tilde{k}}, \varsigma_{i\tilde{k}j}, \varsigma'_{i\tilde{k}j}, \tau_{ij}, \epsilon_{ijk}, \epsilon'_{ikj}, \theta_{ij} \mid \forall k, k \neq i, k \neq j\right\} \\
\mathcal{P}_Z(i) &= \Big\{A_i,\ B_i,\ C_i,\ [\![\mathcal{P}_B(i,j,k)]\!]_{\beta_{ijk},\beta'_{ijk}},\ [\![\mathcal{P}_C(i,j,k)]\!]_{\varsigma_{ijk},\varsigma'_{ijk}},\ [\![\mathcal{P}_1(i,j)]\!]_{\tau_{ij}}, \\
&\qquad [\![\mathcal{P}_2(i,j,k)]\!]_{\epsilon_{ijk},\epsilon'_{ijk}},\ [\![\mathcal{P}_3(i,j)]\!]_{\theta_{ij}} \mid \forall j,k, \not\equiv (i,j,k)\Big\}
\end{aligned}
$$

We denote the set of all possible intermediate probe positions by $\hat{\mathcal{I}}$.

$$
\hat{\mathcal{I}} = \Big\{\mathcal{P}_B(i,j,k), \mathcal{P}_C(i,j,k), \mathcal{P}_1(i,j), \mathcal{P}_2(i,j,k), \mathcal{P}_3(i,j) \mid \forall i,j,k, \not\equiv (i,j,k)\Big\}
$$

For any $\mathcal{I} \subseteq \hat{\mathcal{I}}$ and $\mathcal{A} \subseteq \{0,\ldots,d\}$ with $|\mathcal{I}| + |\mathcal{A}| \leq d$, we first construct $\mathcal{O}$ which – as given in Definition 4 – contains the extended probing sets $\{\mathcal{P}_Z(i) \mid i \in \mathcal{A}\}$. We then construct the set of additional share indices $\mathcal{B}$ required to simulate the probes in $\mathcal{I}$ as follows.

1. Initialize $\mathcal{B} = \emptyset$.
2. For every $\mathcal{P}_B(i,j,k) \in \mathcal{I}$ add b to $\mathcal{B}$ with $b = \begin{cases} i & \text{if } j \in \mathcal{A}, \\ j & \text{otherwise.} \end{cases}$
3. For every $\mathcal{P}_C(i,j,k) \in \mathcal{I}$ add b to $\mathcal{B}$ with $b = \begin{cases} i & \text{if } k \in \mathcal{A}, \\ k & \text{otherwise.} \end{cases}$
4. For every $\mathcal{P}_1(i,j) \in \mathcal{I}$ add b to $\mathcal{B}$ with $b = \begin{cases} i & \text{if } j \in \mathcal{A}, \\ j & \text{otherwise.} \end{cases}$
5. For every $\mathcal{P}_3(i,j) \in \mathcal{I}$ add b to $\mathcal{B}$ with $b = \begin{cases} j & \text{if } i \in \mathcal{A}, \\ i & \text{otherwise.} \end{cases}$
6. For every $\mathcal{P}_2(i,j,k) \in \mathcal{I}$ add b to $\mathcal{B}$ with $b = \begin{cases} k & \text{if } \{i,j\} \subseteq \mathcal{A} \cup \mathcal{B}, \\ i & \text{if } i \notin \mathcal{A} \cup \mathcal{B} \text{ and } j \in \mathcal{A} \cup \mathcal{B}, \\ j & \text{otherwise.} \end{cases}$

We observe that for every probe in $\mathcal{I}$ at most one index is added to $\mathcal{B}$; hence, by construction it holds that $|\mathcal{B}| \leq |\mathcal{I}|$. We now must show that there exists a simulator for all probes in $\mathcal{P} = \mathcal{I} \cup \mathcal{O}$ using only input shares with indices in $\mathcal{S} = \mathcal{A} \cup \mathcal{B}$, thereby proving the d-PINI property of the circuit. Firstly, the simulator creates its own set of uniform and independent random values, to perfectly simulate all required randomness in the circuit. When the circuit reuses some randomness in multiple places, so does the simulator. We observe that every probe in $\mathcal{I}$ can be trivially simulated using just the share indices in $\mathcal{S}$, as these probes do not contain any blinded registers, and we have guaranteed by the construction of $\mathcal{B}$ that the singular share index included in the extended probing set for every probe in $\mathcal{I}$ is included in $\mathcal{S}$.

For every extended probing set $\mathcal{P}_Z(i) \in \mathcal{O}$, the simulations require more careful considerations and can be built as follows.

1. $\mathcal{P}_Z(i) \in \mathcal{O} \implies i \in \mathcal{S}$, hence A_i, B_i and C_i are available to the simulator.

2. For all $j \in [0, d]$ with $j \neq i$, the blinded register $[\![\mathcal{P}_3(i,j)]\!]_{\theta_{ij}}$ can be trivially simulated by θ_{ij} (i.e., tossing a fair coin), unless θ_{ij} is observed by any other extended probe. In such a case, $\mathcal{P}_3(i,j)$ can be simulated in its entirety, because $i \in \mathcal{S}$, and $\mathcal{P}_3(i,j)$ contains only shares with index i. We now consider these cases and show that when $[\![\mathcal{P}_3(i,j)]\!]_{\theta_{ij}}$ cannot be trivially simulated, j is included in $\mathcal{S}$. θ_{ij} (resp. θ_{ji}) can be observed by a probe $\mathcal{P}_Z(j)$, $\mathcal{P}_3(i,j)$, or $\mathcal{P}_3(j,i)$. In all these three cases, it is guaranteed that $j \in \mathcal{S}$ by construction of $\mathcal{B}$. We also need to consider the case where $\mathcal{P}_3(j,i)$ is not observed by a probe directly, but instead required for the simulation of the blinded register $[\![\mathcal{P}_3(j,i)]\!]_{\theta_{ji}}$. This blinded register is only observed by $\mathcal{P}_Z(j)$, and hence it is also guaranteed that $j \in \mathcal{S}$ when $[\![\mathcal{P}_3(j,i)]\!]_{\theta_{ji}}$ is observed.

3. Next, we show that for all $j \in [0, d]$ with $j \neq i$, the blinded register $[\![\mathcal{P}_1(i,j)]\!]_{\tau_{ij}}$ can either be trivially simulated (by tossing a fair coin for τ_{ij}) or $j \in \mathcal{S}$, so $\mathcal{P}_1(i,j)$ can be simulated entirely. τ_{ij} (resp. τ_{ji}) can be observed by a probe $\mathcal{P}_Z(j)$, $\mathcal{P}_1(i,j)$, $\mathcal{P}_1(j,i)$, $\mathcal{P}_3(i,j)$, or $\mathcal{P}_3(j,i)$. In all these five cases, $j \in \mathcal{S}$ by construction of $\mathcal{S}$. If there is no probe of type $\mathcal{P}_1(j,i)$ and $\mathcal{P}_3(j,i)$ but the blinding mask is used in the simulation of a blinded register in probe $\mathcal{P}_Z(j)$, then $j \in \mathcal{S}$. Additionally, if there is no probe of type $\mathcal{P}_3(i,j)$ but it is used in the simulation of the blinded register $[\![\mathcal{P}_3(i,j)]\!]_{\theta_{ij}}$, we know that $j \in \mathcal{S}$ according to the statement given in Item 2 above. Therefore, when $[\![\mathcal{P}_1(i,j)]\!]_{\tau_{ij}}$ cannot be trivially simulated by τ_{ij}, we know that $j \in \mathcal{S}$ and hence all share indices in $\mathcal{P}_1(i,j)$ can be used for simulation of the blinded register.

4. We can similarly show that for all $j, k \in [0, d]$ with $\not\equiv (i,j,k)$, the blinded register $[\![\mathcal{P}_2(i,j,k)]\!]_{\epsilon_{ijk}, \epsilon'_{ijk}}$ can either be simulated by tossing a fair coin for $\epsilon_{ijk} \oplus \epsilon'_{ijk}$ or $j \in \mathcal{S}$. The blinded register cannot be trivially simulated only when both ϵ_{ijk} and ϵ'_{ijk} are used in the simulation of another probe. However, ϵ_{ijk} (resp. ϵ_{jik}) can only be observed by a probe $\mathcal{P}_2(i,j,k)$, $\mathcal{P}_2(j,i,k)$, $\mathcal{P}_3(i,j), \mathcal{P}_3(j,i)$, or $\mathcal{P}_Z(j)$, all of which guarantee that $j \in \mathcal{S}$. We additionally know that if all probes in $\mathcal{P}_3(i,j)$ are used in the simulation of the blinded register in $\mathcal{P}_Z(i)$, then $j \in \mathcal{S}$ according to the statement given in Item 2 above. If $\mathcal{P}_2(j,i,k)$ or $\mathcal{P}_3(j,i)$ is used in the simulation of a blinded register, then $\mathcal{P}_Z(j) \in \mathcal{O}$ and hence still $j \in \mathcal{S}$.

5. We now show that any blinded register $[\![\mathcal{P}_C(i,j,k)]\!]_{\varsigma_{ijk}, \varsigma'_{ijk}}$ can either be trivially simulated by tossing a fair coin for $\varsigma_{ijk} \oplus \varsigma'_{ijk}$ or we know that $k \in \mathcal{S}$. The relevant probe positions, where ς'_{ijk} (resp. ς'_{jik}) can be observed and k is not trivially in $\mathcal{S}$ by construction, are $\mathcal{P}_Z(j), \mathcal{P}_1(i,j)$ and $\mathcal{P}_1(j,i)$. When any of these three probes is placed, we know that $\{i,j\} \subseteq \mathcal{S}$ by construction of $\mathcal{S}$. The relevant probe positions where ς_{ijk} (resp. ς_{kji}) can be observed but k is not trivially in $\mathcal{S}$, are $\mathcal{P}_2(i,j,k)$ and $\mathcal{P}_2(k,j,i)$. However, Step 6 of the construction of $\mathcal{B}$ ensures that when two of

the three arguments of $\mathcal{P}_2$ are in $\mathcal{S}$, then the third one is added. We know that $\{i, j\} \subseteq S$ when k is not trivially in $\mathcal{S}$. Therefore, when $\mathcal{P}_2(i, j, k)$ or $\mathcal{P}_2(k, j, i)$ is probed, we know that $k \in \mathcal{S}$ by construction. If $\mathcal{P}_2(k, j, i)$ is observed through a blinded register, then $\mathcal{P}_Z(k) \in \mathcal{O}$ and hence $k \in \mathcal{S}$. Lastly, we need to consider the case where $\mathcal{P}_2(i, j, k)$ is simulated for the blinded register $[\![\mathcal{P}_2(i, j, k)]\!]_{\epsilon_{ijk}, \epsilon'_{ijk}}$. When this blinded register cannot be trivially simulated by $\epsilon_{ijk} \oplus \epsilon'_{ijk}$, it must be the case that ϵ'_{ijk} is observed by some other probe. However, there is no probe position where ϵ'_{ijk} (resp. ϵ'_{kji}) can be observed without k being included in $\mathcal{S}$ by construction. Therefore, we know that when the blinded register $[\![\mathcal{P}_C(i, j, k)]\!]_{\varsigma_{ijk}, \varsigma'_{ijk}}$ cannot be trivially simulated, all input shares with index k are available to the simulator and $[\![\mathcal{P}_C(i, j, k)]\!]_{\varsigma_{ijk}, \varsigma'_{ijk}}$ can be entirely simulated.

6. Lastly, we similarly show that any blinded register $[\![\mathcal{P}_B(i, j, k)]\!]_{\beta_{ijk}, \beta'_{ijk}}$ can be simulated by tossing a fair coin for $\beta_{ijk} \oplus \beta'_{ijk}$ or $j \in \mathcal{S}$. The relevant probe positions, where β'_{ijk} (resp. β'_{kji}) can be observed, are $\mathcal{P}_Z(k), \mathcal{P}_1(i, k)$ and $\mathcal{P}_1(k, i)$. Once again, in these cases j is not trivially in $\mathcal{S}$, but we know that $\{i, k\} \subseteq \mathcal{S}$. The relevant probe positions, where β_{ijk} (resp. β_{jik}) can be observed but j is not trivially in $\mathcal{S}$, are $\mathcal{P}_2(i, k, j)$ and $\mathcal{P}_2(j, k, i)$. When $\mathcal{P}_2(i, k, j)$ or $\mathcal{P}_2(j, k, i)$ is probed and $\{i, k\} \subseteq \mathcal{S}$, then $j \in \mathcal{S}$. We again need to consider the case where $\mathcal{P}_2(i, k, j)$ is not probed, but is instead used to simulate a blinded register $[\![\mathcal{P}_2(i, k, j)]\!]_{\epsilon_{ikj}, \epsilon'_{ikj}}$. However, once again, there is no probe position where ϵ'_{ikj} (resp. ϵ'_{jki}) can be observed without j being included in $\mathcal{S}$ by construction. Therefore, we have shown that the blinded register $[\![\mathcal{P}_B(i, j, k)]\!]_{\beta_{ijk}, \beta'_{ijk}}$ can be trivially simulated by $\beta_{ijk} \oplus \beta'_{ijk}$ or $j \in \mathcal{S}$; hence, $\mathcal{P}_B(i, j, k)$ can be entirely simulated.

We have shown that all blinded registers in $\mathcal{P}_Z(i)$ can be simulated by the blinding masks alone, or the share indices contained in the blinded probes are available for the simulator. Therefore, we have constructed the set $\mathcal{B}$ in such a way that the observations from probes in $\mathcal{P} = \mathcal{I} \cup \mathcal{O}$ can be simulated with the input shares with indices in $\mathcal{S} = \mathcal{A} \cup \mathcal{B}$ and $|\mathcal{B}| \leq |\mathcal{I}|$ and therefore shown that HPCC^{2+} is d-PINI. □

3.2.6 Expansion of HPCC^{2+}

The expansion of HPCC^{2+} to compute multiple Z^l and the extra Y output follows the same principle as discussed in Section 3.1.1. We can use all registers in any Z_i^l as long as the register does not contain any share of A^l. This is only the case for the registers blinded by θ, and thus all masks θ_{ij} must be replaced by unique θ_{ij}^l in every Z_i^l.

The implementation of Y follows the same pattern as in HPCC^1, by observing that Z_0 already contains the desired BC product. We therefore construct Y_0 by omitting all multiplications with A_0. In the final register, where A_0 is absent, we must replace the mask $\theta_{0,j}$ by a unique mask κ_j, so that $\theta_{0,j}$ is not used to blind two different registers. As a result, Y_0 is given below, and the other $d - 1$ shares of Y must then simply cancel all κ_j. Hence, we assign $Y_{j \in \{1,\ldots,d\}} = \mathrm{Reg}\,[\kappa_j]$.

$$
\begin{aligned}
Y_0 =& \operatorname{Reg}[B_0]\operatorname{Reg}[C_0] \\
&\oplus \operatorname{Reg}[B_0]\left(\bigoplus_{\not\equiv(0,j)}^{\forall j} \operatorname{Reg}\left[C_j \oplus \varsigma_{0,\tilde{k},j} \oplus \varsigma'_{0,\tilde{k},j}\right]\right) \\
&\oplus \operatorname{Reg}[C_0]\left(\bigoplus_{\not\equiv(0,j)}^{\forall j} \operatorname{Reg}\left[B_j \oplus \beta_{0,j,\tilde{k}} \oplus \beta'_{i,j,\tilde{k}}\right]\right) \\
&\oplus \left(\bigoplus_{\not\equiv(0,j,k)}^{\forall j,k} \operatorname{Reg}\left[B_j \oplus \beta_{0,j,k} \oplus \beta'_{0,j,k}\right]\operatorname{Reg}\left[C_k \oplus \varsigma_{0,j,k} \oplus \varsigma'_{0,j,k}\right]\right) \\
&\oplus \bigoplus_{\not\equiv(0,j)}^{\forall j} \operatorname{Reg}\left[B_jC_j \oplus \bigoplus_{\not\equiv(0,j,k)}^{\forall k}\left((B_j \oplus \beta'_{0,j,k})\varsigma'_{0,j,k} \oplus (C_j \oplus \varsigma_{0,k,j})\beta'_{0,k,j}\right) \oplus \tau_{0,j}\right] \\
&\oplus \bigoplus_{\not\equiv(0,j,k)}^{\forall j,k} \operatorname{Reg}\left[\left(B_j \oplus \beta_{0,j,k}\right)\varsigma_{0,j,k} \oplus \left(C_j \oplus \varsigma'_{0,k,j}\right)\beta_{0,k,j} \oplus \epsilon_{0,j,k} \oplus \epsilon'_{0,j,k}\right] \\
&\oplus \bigoplus_{\not\equiv(0,j)}^{\forall j} \operatorname{Reg}\left[B_0\left(\varsigma_{0,\tilde{k},j} \oplus \varsigma'_{0,\tilde{k},j}\right) \oplus C_0\left(\beta_{0,j,\tilde{k}} \oplus \beta'_{0,j,\tilde{k}}\right) \oplus \tau_{0,j} \oplus \bigoplus_{\not\equiv(0,j,k)}^{\forall k}\left(\epsilon_{0,j,k} \oplus \epsilon'_{0,k,j}\right) \oplus \kappa_j\right]
\end{aligned}
\tag{13}
$$

4 Application to the Inversion in $\mathbb{F}_{2^8}$

4.1 Structure of the Inversion

In this section, we challenge ourselves with the implementation of a secure AES S-Box, achieving two cycles of latency at arbitrary order. The AES S-Box is an 8-bit bijective map and is computed by taking the input as an element in $\mathbb{F}_{2^8}$, computing its inverse, and finally applying an affine transformation to the result. The affine transformation can be trivially realized in the shared domain, without incurring any latency cost. The implementation of the inversion in the masked domain is the challenging part. To this end, one option is the decomposition of the inversion into two cubic monomials, meaning two values x, y with a Hamming weight of 3 each, such that $(a^x)^y = a^{-1}$. All possible tuples of these values are given in [WM18] and can be easily verified with a brute force. This approach would allow for the implementation of the inversion using two cascaded instantiations of HPCC in $\mathbb{F}_{2^8}$, as the exponentiation to a cubic monomial can be computed by the product of the contained squares, such as $x^{13} = x^8x^4x$. This does indeed give a latency of only two cycles for the full AES S-Box but the area and randomness requirement of HPCC when instantiated for $\mathbb{F}_{2^8}$ is undesirable.

Instead, we focus on the inversion presented in [HB25], which achieves a multiplicative depth of 3 using only 2-input multipliers in $\mathbb{F}_{2^4}$ and $\mathbb{F}_{2^2}$, and adapt it so that some 2-input multiplications are directly adjacent and can be instantiated by one instance of HPCC. Their inversion is based on Canright's $\mathbb{F}_{2^8}$ inversion [Can05]; both use a tower field decomposition of $\mathbb{F}_{2^8}$ to represent any element in $\mathbb{F}_{2^8}$ as a vector of two elements in $\mathbb{F}_{2^4}$, any element of $\mathbb{F}_{2^4}$ as two elements of $\mathbb{F}_{2^2}$, and finally any element of $\mathbb{F}_{2^2}$ as a vector of two elements in $\mathbb{F}_2$. The derivation of the structure for the inversion is given in [HB25] and is not reiterated here.

The structure of the inversion operating on the two $\mathbb{F}_{2^4}$ elements of the input is shown in Figure 2. In this figure, $\bigoplus$ is the addition of two elements, $\bigotimes$ their multiplication, and Π and Σ are constants. The $\bigcirc^2$ operator squares its input, and the $\bigcirc^4$ operator squares

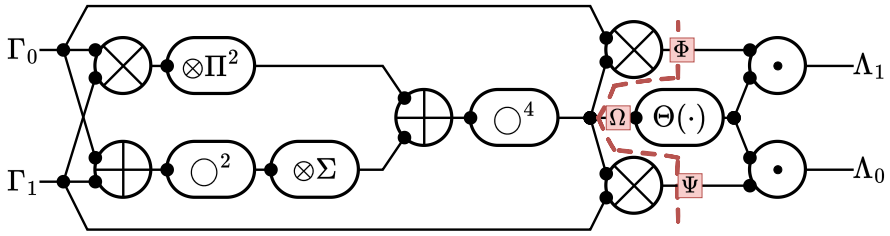


Figure 2: Structure of the $\mathbb{F}_{2^8}$ inversion presented in [HB25]. $\Theta(\cdot)$ and $\odot$ are $\mathbb{F}_{2^2}$ operators which are applied on two elements, each belonging to $\mathbb{F}_{2^4}$.

its input twice. Notably, the $\Theta(\cdot)$ and $\odot$ operators are actually defined to operate on two $\mathbb{F}_{2^2}$ elements as their inputs, implicitly splitting each $\mathbb{F}_{2^4}$ inputs into their two parts. They are defined in [HB25] for any $\mathbb{F}_{2^2}$ elements $\omega_0, \omega_1, \phi_0, \phi_1$ as follows.

$$\Theta(\omega_1 Z^4 + \omega_0 Z) = \left(\omega_0\omega_1 \oplus (\omega_0 \oplus \omega_1)^2 \sigma\pi\right)^2 (Z^4 + Z)$$

$$(\phi_1 Z^4 + \phi_0 Z) \odot (\omega_1 Z^4 + \omega_0 Z) = \phi_1\lambda_1\pi Z^4 + \phi_0\lambda_0\pi Z$$

We can never combine a multiplier in $\mathbb{F}_{2^4}$ with a multiplier in $\mathbb{F}_{2^2}$, which makes the red line shown in Figure 2 a natural place to "cut" the structure, ensuring that all $\mathbb{F}_{2^4}$ multipliers are on the left side of the cut, and all $\mathbb{F}_{2^2}$ multipliers (hidden inside the $\Theta(\cdot)$ and $\odot$ operators) are on the right side. In the left half, there are two cascaded $\mathbb{F}_{2^4}$ multipliers in the computation of Φ and Ψ, which must be moved in the computation diagram such that they are directly adjacent. This can be achieved as follows.

$$\begin{aligned}
\Phi &= \Gamma_0 \left(\Gamma_0\Gamma_1\Pi^2 \oplus (\Gamma_0 \oplus \Gamma_1)^2 \Sigma\right)^4 \\
&= \Gamma_0 \left(\left(\Gamma_0\Gamma_1\Pi^2\right)^4 \oplus \left((\Gamma_0 \oplus \Gamma_1)^2 \Sigma\right)^4\right) \\
&= \Gamma_0\Gamma_0^4 \left(\Gamma_1\Pi^2\right)^4 \oplus \Gamma_0 \left((\Gamma_0 \oplus \Gamma_1)^2 \Sigma\right)^4, \\
\Psi &= \Gamma_1 \left(\Gamma_0\Gamma_1\Pi^2 \oplus (\Gamma_0 \oplus \Gamma_1)^2 \Sigma\right)^4 \\
&= \Gamma_1\Gamma_0^4 \left(\Gamma_1\Pi^2\right)^4 \oplus \Gamma_1 \left((\Gamma_0 \oplus \Gamma_1)^2 \Sigma\right)^4
\end{aligned}$$

Ω is already computed with a multiplicative depth of only one multiplication in the inversion of [HB25], but we adjust it slightly to allow for a more efficient implementation in the end. We rewrite Ω so that we can use the partial product $\Gamma_0^4 \left(\Gamma_1\Pi^2\right)^4$ used in the computation of Φ and Ψ as follows.

$$\begin{aligned}
\Omega &= \left(\Gamma_0\Gamma_1\Pi^2 \oplus (\Gamma_0 \oplus \Gamma_1)^2 \Sigma\right)^4 \\
&= \Gamma_0^4 \left(\Gamma_1\Pi^2\right)^4 \oplus \left((\Gamma_0 \oplus \Gamma_1)^2 \Sigma\right)^4
\end{aligned}$$

This allows us to eventually save one multiplication gadget in the masked implementation, as we can use the Y output of HPCC when computing $\Gamma_0\Gamma_0^4 \left(\Gamma_1\Pi^2\right)^4$. The new computation of the left half is shown in Figure 3. The right half involves the $\Theta(\cdot)$ and $\odot$ operators, which operate on $\mathbb{F}_{2^2}$ elements. Therefore, we split the three outputs of

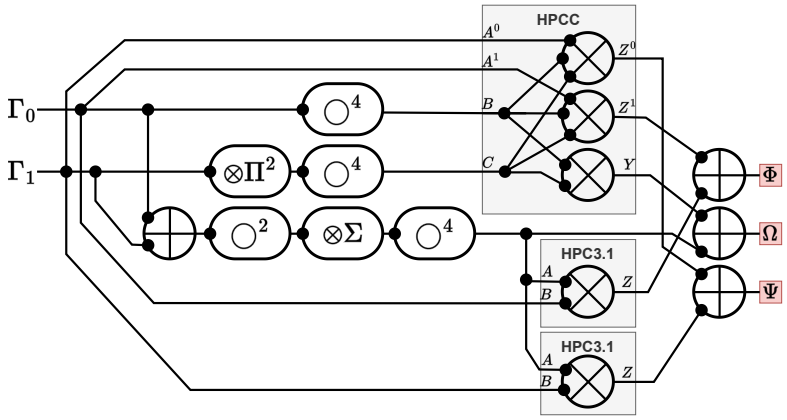


Figure 3: Structure of the left half of the adjusted $\mathbb{F}_{2^8}$ inversion.

the left half into their respective $\mathbb{F}_{2^2}$ elements as $\Phi = \phi_1 Z^4 + \phi_0$, $\Omega = \omega_1 Z^4 + \omega_0$ and $\Psi = \psi_1 Z^4 + \psi_0$. We must merge the two $\Theta(\cdot)$ and $\odot$ operators into one equation and rewrite it so that the two contained multiplications are directly adjacent. We consider the output $\Lambda_0 = \lambda_1 Z^4 + \lambda_0$ first and rewrite λ_0 as follows.

$$\begin{aligned}\lambda_0 &= \psi_0\pi\left(\omega_0\omega_1 \oplus (\omega_0 \oplus \omega_1)^2 \sigma\pi\right)^2 \\ &= \psi_0\pi\left(\omega_0^2\omega_1^2 \oplus \left((\omega_0 \oplus \omega_1)^2 \sigma\pi\right)^2\right) \\ &= \pi\left(\psi_0\omega_0^2\omega_1^2 \oplus \psi_0\left((\omega_0 \oplus \omega_1)^2 \sigma\pi\right)^2\right)\end{aligned}$$

The structure of this computation is shown in Figure 4. The output λ_1 can be rewritten in the same way as

$$\lambda_1 = \pi\left(\psi_1\omega_0^2\omega_1^2 \oplus \psi_1\left((\omega_0 \oplus \omega_1)^2 \sigma\pi\right)^2\right),$$

and the output $\Lambda_1 = \lambda_1' Z^4 + \lambda_0' Z$ can also be rewritten in a similar manner.

$$\begin{aligned}\lambda_0' &= \pi\left(\phi_0\omega_0^2\omega_1^2 \oplus \phi_0\left((\omega_0 \oplus \omega_1)^2 \sigma\pi\right)^2\right) \\ \lambda_1' &= \pi\left(\phi_1\omega_0^2\omega_1^2 \oplus \phi_1\left((\omega_0 \oplus \omega_1)^2 \sigma\pi\right)^2\right)\end{aligned}$$

These adjustments actually increase the multiplicative depth of the circuit to four, but we can now merge pairs of 2-input multiplications easily to achieve a latency of only two cycles with HPCC.

4.2 Masking the Inversion with HPCC

In the process of preparing the structure of the inversion for the implementation with HPCC, we distributed multiplications over additions to move the 2-input multiplications in the inversion of [HB25] so that they are directly adjacent and can be computed with a single 3-input multiplication. This results in many more required multiplication gadgets than in the original inversion, and hence increases the required area and fresh masks. However, we can note that in the left half (Figure 3) the two 3-input multiplications actually only differ in one operand and can therefore be realized all at once with one

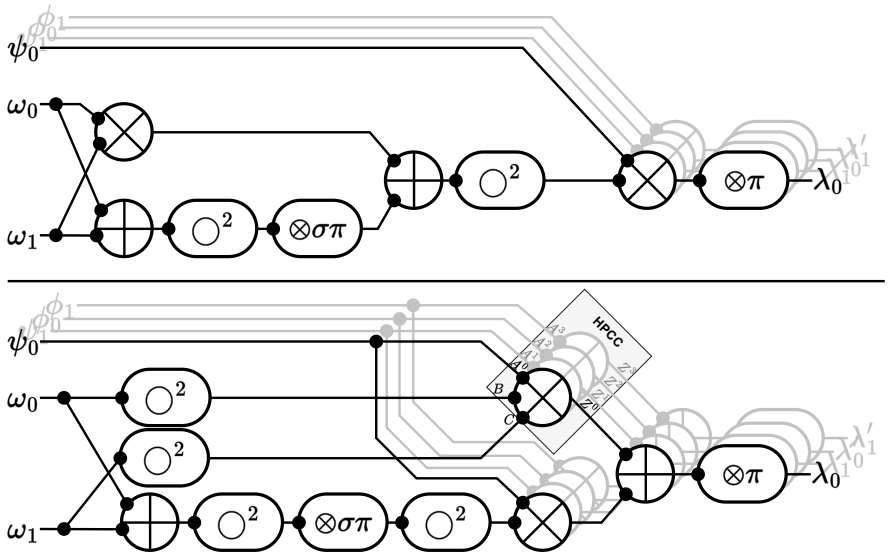


Figure 4: Structure of the right half of the $\mathbb{F}_{2^8}$ inversion in [HB25] (top) and the adjusted design (bottom). The figure focuses on the computation of λ_0, i.e., one of the elements of Λ_0. The other three $\mathbb{F}_{2^2}$ outputs are shown in gray.

HPCC gadget instantiated for two Z outputs. The gadget can also compute $\Gamma_0^4 \left(\Gamma_1 \Pi^2\right)^4$ by using the Y output.

Similarly, both 2-input multiplications of the left half (Figure 3) share an operand. We realize these multiplications with HPC3.1 [HB25], which allows us to reuse the internal blinding randomness for both instances of HPC3.1, as shown in [HB25]. In total, this means that the left half requires one instance of HPCC for $\mathbb{F}_{2^4}$ with two Z outputs and the extra Y output, and two instances of HPC3.1 in $\mathbb{F}_{2^4}$ which share their internal randomness.

The right half requires four 3-input multiplications in $\mathbb{F}_{2^2}$, and four 2-input multiplications, which are realized with HPCC and HPC3.1, respectively. All 3-input multiplications share ω_0^2 and ω_1^2 as their two of the three operands, and may thus all be realized by a single HPCC instance with four outputs as shown in Figure 4. Similarly, the 2-input multiplication gadgets share an input, which can therefore be blinded with shared randomness once again. In this case, the Y output of HPCC is not needed and can hence be omitted.

With these optimizations in mind, we can then use trivial composition to build the two-cycle masked AES S-Box. All linear operations can be replaced by their corresponding share-wise operations, and the non-linear multiplications stages are realized as discussed.

5 Evaluations

In this section, we first present the performance results, providing an overview of the area and randomness requirements when HPCC gadgets are employed. Subsequently, the results of the security assessments are outlined.

5.1 Performance Figures

Below, we first present the performance figures of HPCC gadgets under different settings at first and higher orders compared with other similar PINI gadgets. This is followed by the AES S-Box as the main target compared with state-of-the-art low-latency masked designs.

Table 1: Cost of HPCC in $\mathbb{F}_2$ with various number of Z outputs (n); y indicates the presence of the extra Y output.

n	y	**Rand.** [bit]	**Area** [GE]
1	0	4	150
	1	5	160
2	0	5	180
	1	6	195
4	0	7	249
	1	8	265

(a) $d = 1$

n	y	**Rand.** [bit]	**Area** [GE]
1	0	24	560
	1	26	591
2	0	27	642
	1	29	674
4	0	33	815
	1	35	835

(b) $d = 2$

n	y	**Rand.** [bit]	**Area** [GE]
1	0	84	1652
	1	87	1717
2	0	90	1848
	1	93	1866
4	0	102	2169
	1	105	2191

(c) $d = 3$

5.1.1 Gadgets

The randomness cost of HPCC depends not only on the security order, but also on the number of desired Z outputs, and if the extra Y output is needed. Let d be the security order, n the number of desired Z outputs, and $y = 1$ when Y should be instantiated and $y = 0$ otherwise. The number of required random field elements is then given by $3 + n + y$ for a first-order design and $3(d-1)d(d+1) + (n+1)\frac{d(d+1)}{2} + yd$ for higher security orders. This is equivalent to the number of random bits in $\mathbb{F}_2$ but must be multiplied by the field size for larger fields. The cost of HPCC under various configurations in $\mathbb{F}_2$ is listed in Table 1. When not stated otherwise, the designs and gadgets compared in the tables of this paper were synthesized by Yosys [2] using the NanGate45 design cell library.

To compare HPCC with other gadgets, we instantiate HPCC with $n = 1$ and $y = 0$. Following [HB25], we estimate the required area for a Trivium PRNG to generate the required random bits at 39.4 GE per random bit. The results of the comparison are listed in Table 2. At higher orders and in larger fields than $\mathbb{F}_2$, HPCC is the only gadget achieving a latency of one clock cycle. Some schemes, such as TSM [VDB+24] and GHPC_{LL} [KSM22], allow for the realization of arbitrary circuits at the first order with only one cycle of latency, and could hence also be used to realize a 3-input multiplication for larger fields. In both cases, we expect the area and required random bits to be larger than HPCC, but we did not synthesize any of such implementation because no implementation is publicly available for the TSM, and because of prohibitive area costs for the GHPC_{LL}. GLM [GIB18] is not a PINI gadget, and thus also excluded from this comparison. Similarly, LLTI [AZN21] is not a PINI gadget and operates on $td + 1$ shares, where t is the algebraic degree of the function. It even uses higher number of input shares to remove the register between composed TI modules. For the implementation with compress [CGM+24], we used the multiplier code provided in [HB25] as the input to the tool, which was then masked to the desired security order using various PINI gadgets, targeting a latency of two cycles. Therefore the compress designs are bit-level masked designs of multipliers, even when the multiplication happens in a larger field. The first-order HPCC design for a 3-input $\mathbb{F}_2$ multiplier outperforms all other gadgets with one cycle of latency in terms of randomness at a slightly larger area footprint. At higher orders and in larger fields, the randomness and area cost exceeds the cost of other gadgets, but is the only gadget achieving a latency of one clock cycle.

[2] `Yosys 0.59+134 (git sha1 9871e9b17, clang++ 17.0.0 -fPIC -O3)`

Table 2: Implementation cost of a masked 3-input multiplication in various fields and security orders with different gadgets.

Field	d	Gadget	Latency	Rand.	Area [GE]		Ref.
			[cycle]	[bit]	standalone	w. PRNG	
$\mathbb{F}_2$	1	HPCC	1	4	150	308	[new]
		GHPC_{LL}	1	8	129	444	[KSM22]
		TSM	1	10	166	560	[VDB+24]
		GHPC	2	1	182	221	[KSM22]
		compress	2	3	116	234	[CGM+24]
		2×HPC3.1	2	4	129	278	[HB25]
	2	HPCC	1	24	560	1506	[new]
		HO-TSM	2	30	472	1654	[VDB+25]
		compress	2	9	321	676	[CGM+24]
		2×HPC3.1	2	12	318	791	[HB25]
	3	HPCC	1	84	1652	4962	[new]
		compress	2	18	643	1352	[CGM+24]
		2×HPC3.1	2	24	587	1533	[HB25]
$\mathbb{F}_{2^2}$	1	HPCC	1	8	405	720	[new]
		compress	2	9	381	736	[CGM+24]
		2×HPC3.1	2	8	335	650	[HB25]
	2	HPCC	1	48	1683	3574	[new]
		compress	2	27	1012	2076	[CGM+24]
		2× HPC3.1	2	24	779	1725	[HB25]
	3	HPCC	1	168	5340	11959	[new]
		compress	2	54	1994	4122	[CGM+24]
		2×HPC3.1	2	48	1413	3304	[HB25]
$\mathbb{F}_{2^4}$	1	HPCC	1	16	4906	5536	[new]
		compress	2	27	1207	2271	[CGM+24]
		2×HPC3.1	2	16	891	1521	[HB25]
	2	HPCC	1	96	4917	8699	[new]
		compress	2	81	3132	6323	[CGM+24]
		2×HPC3.1	2	48	1981	3872	[HB25]
	3	HPCC	1	336	13781	27019	[new]
		compress	2	162	6110	12493	[CGM+24]
		2×HPC3.1	2	96	3571	7353	[HB25]

5.1.2 AES S-Box

A comparison of the implementation costs of the AES S-Box at various security orders is shown in Table 3. We exclude any S-Box implementations from this comparison with a latency of four cycles or more, as the main focus of this work is on lowering the latency. At the first order, the HPCC-based construction uses 7 times more randomness than GHPC [KSM22] but GHPC is roughly 23 times larger than the HPCC-based construction. Although the two-cycle TSM design requires 14 fewer random bits, its total area including the PRNG cost exceeds that of the HPCC-based design. At the second order, a HO-TSM S-Box [VDB+25] achieves two cycles of latency but uses more than 3 times as much randomness and area as the HPCC-based construction. At higher orders, the HPCC-

Table 3: Implementation cost of the masked AES S-Box using different gadgets.

d	Gadget	Latency	Rand.	Area [GE]		Ref.
		[cycle]	[bit]	standalone	w. PRNG	
1	TSM [a]	1	262	14352	24675	[VDB+24]
	GHPC_{LL} [a]	1	2048	64111	144802	[KSM22]
	HPCC-based [b]	**2**	**60**	**3234**	**5598**	**[new]**
	TSM [a]	2	46	4020	5832	[VDB+25]
	GHPC [a]	2	8	77145	77460	[KSM22]
	All-HPC3.1 [b]	3	32	1875	3136	[HB25]
2	**HPCC-based** [b]	**2**	**248**	**10436**	**20207**	**[new]**
	HO-TSM [a]	2	786	33700	64668	[VDB+25]
	All-HPC3.1 [b]	3	96	4176	7958	[HB25]
3	**HPCC-based** [b]	**2**	**708**	**25953**	**53848**	**[new]**
	All-HPC3.1 [b]	3	192	7687	15252	[HB25]

[a] Forward S-Box only, using Synopsys Design Compiler.
[b] Bidirectional S-Box, using Yosys.

based construction is the only design with two cycles of latency, although at the cost of significantly higher randomness and area requirements than the three-cycle implementation. In short, our HPCC-based AES S-Box requires 60 fresh random bits at the first order, and $18d^3 + 22d^2 + 8d$ bits for any security order $d > 1$. We provided RTL source code of our HPCC-based AES S-Box for arbitrary order in the GitHub repository.

5.2 Security Assessments

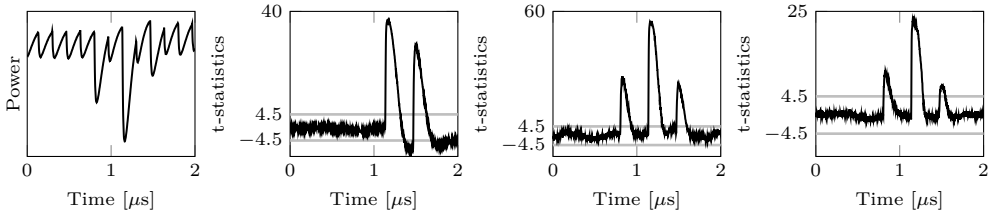
5.2.1 Tool-Based Analyses

We employed VerifMSI [MT23] to formally verify the PINI property of the HPCC gadgets with $d = 1$, $d = 2$, and $d = 3$. The gadget was instantiated with two Z outputs and including the Y output in $\mathbb{F}_2$ and $\mathbb{F}_{2^2}$. Since VerifMSI is provided as a Python library without native support for synthesized netlists, we developed a custom parser for the JSON netlist format produced by Yosys. This approach guarantees that the verified model accurately reflects the synthesized hardware. The parser and verification script are provided in the artifact and can be executed by `make verif BIT_WIDTH=1 NUM_SHARES=2`.

We used PROLEAD [MM22] to statistically validate the PINI property of HPCC in $\mathbb{F}_2$ with two Z outputs and Y output up to $d = 2$ with 500 million simulations. The second-order simulation, due to excessive memory usage, was performed only using compact distributions. A third-order design with no expansions, i.e., $n = 1$ and $y = 0$, was statistically validated with 300 million simulations. The AES S-Box implementation was statistically validated in the glitch-extended probing model with PROLEAD using compact distributions up to $d = 2$. We did not employ VerifMSI to formally verify the S-Box. The designs, netlists, configurations and results of our tool-based analyses are provided in the GitHub repository.

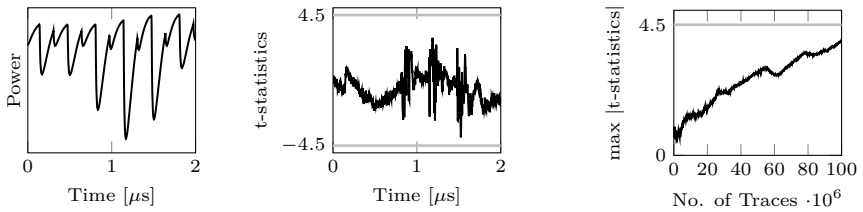
5.2.2 Experimental Analyses

We utilized the FPGA-based platform SAKURA-X [SAK16], designed for SCA evaluations, to conduct experimental security assessments. To facilitate these evaluations, we collected power consumption traces from the Kintex-7 FPGA on the SAKURA-X board. This was achieved by using a digital oscilloscope to monitor the voltage drop across a shunt resistor



(a) Mean, 1000 traces. (b) t-test, first order. (c) t-test, second order. (d) t-test, third order.

Figure 5: Experimental leakage assessment results, using 0.5 million traces, first-order HPCC-based AES S-Box, PRNG off.



(a) Mean, 1000 traces. (b) t-test, first order. (c) t-test, first order, over no. of traces.

Figure 6: Experimental leakage assessment results, using 100 million traces, first-order HPCC-based AES S-Box.

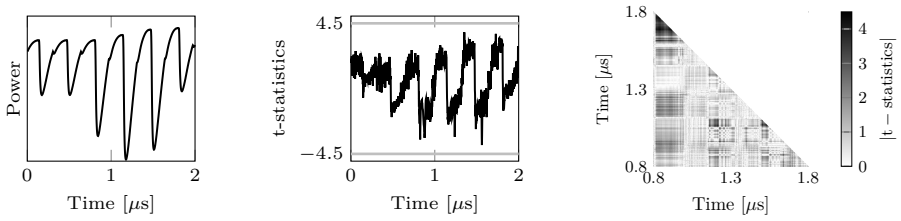
placed in the Vdd path of the target FPGA. To improve signal quality and reduce noise, we inserted a 10 dB AC amplifier between the board and the oscilloscope.

The signal was sampled at a rate of 500 MS/s, and the FPGA clock was driven by a stable, jitter-free source with a frequency of 3 MHz. To generate fresh randomness for each design, we instantiated a highly efficient, FPGA-specific 31-bit Linear Feedback Shift Register (LFSR) [MMW18] for each required fresh mask bit. These LFSRs were seeded randomly during the power-up cycle of the Kintex-7 FPGA. To evaluate the security of the designs, we performed a well-established fixed-vs-random t-test, also known as the Test Vector Leakage Assessment (TVLA) [CDG+13, SM15]. For this analysis, the target circuit was provided either with zero as the fixed input or a randomly selected input, determined by a coin flip for each measurement. In both cases, the input was freshly masked before being fed into the circuit using 2, 3, or 4 shares depending on the target design.

For the analysis, we selected three target designs: the first-, second-, and third-order AES S-Box circuits, which were implemented using our HPCC gadgets, whose performance results are detailed in Table 3. Consistent with state-of-the-art practice, we collected 100 million traces following the above-explained scenario and estimated the t-statistics for each corresponding t-test.

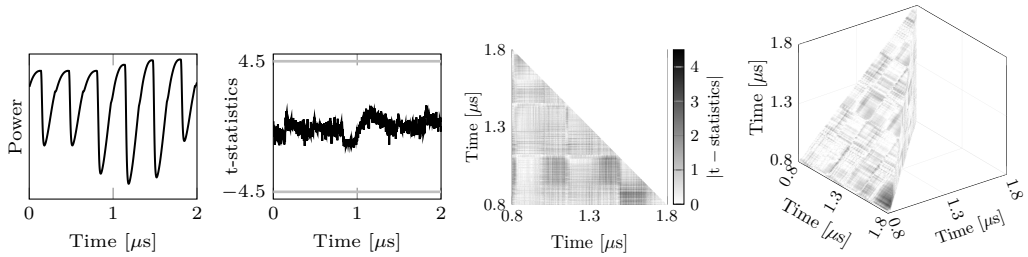
To validate our setup, we first evaluated the first-order design with the PRNG turned off, which effectively set all fresh masks to 0. The results, obtained using just 0.5 million traces, including a mean trace and t-statistics (up to the third order), are presented in Figure 5. These results confirm the correctness of our evaluation framework, and identify the cycles between 0.8 μs and 1.8 μs relevant to the activity of the combinational circuits of the targeted S-Box.

Naturally, we continue our analysis by examining the same design while the PRNG is operating properly. The corresponding results using 100 million traces are depicted in Figure 6 where the maximum of t-statistics over the number of traces is additionally shown. These results are in line with our expectations and with those of PROLEAD (see Section 5.2.1),



(a) Mean, 1000 traces. (b) t-test, first order. (c) t-test, second order.

Figure 7: Experimental leakage assessment results, using 1000 million traces, second-order HPCC-based AES S-Box.



(a) Mean, 1000 traces. (b) t-test, first order. (c) t-test, second order. (d) t-test, third order.

Figure 8: Experimental leakage assessment results, using 1000 million traces, third-order HPCC-based AES S-Box.

i.e., no detectable first-order leakage. Note that second- and third-order leakages are instantly detected similar to the case with the deactivated PRNG. This is natural and expected since the entire design as well as the S-Box input is based on two shares.

When evaluating the higher-order designs, we extended our analysis to cover multivariate leakages. To this end, we focused on the aforementioned window (0.8 μs and 1.8 μs) for conducting bi-variate second-order and tri-variate third-order t-tests. Following the state of the art and as given in [SM15], we combined every two (or every three) possible sample points in the aforementioned window by mean-free product prior to conducting the t-tests. This means, $\binom{500}{2} = 124,750$ t-tests for the bi-variate second-order analysis and $\binom{500}{3} = 20,708,500$ t-tests for the tri-variate third-order analysis. The corresponding results are shown in Figure 7 and Figure 8 respectively, which are consistent with our expectations, i.e., no first- and no second-order leakage in our higher-order designs, and additionally no third-order leakage in our third-order design.

6 Conclusions

In this work, we presented HPCC, a generic PINI gadget under the glitch-extended probing model for the multiplication of three values in arbitrary fields. It is the first PINI gadget realizing such a multiplication (i.e., a cubic function) with only one clock cycle of latency at arbitrary order. At the first order, where other comparable gadgets exist, it offers a new record for the lowest number of required random bits. Additionally, HPCC is very flexible and allows for the instantiation of any number of multiplications when two of the three operands are identical, and can additionally provide the product of just two operands, with comparatively little extra overhead. This allows for optimized implementations under trivial composition. We applied HPCC to the AES S-Box and presented a composable implementation of the S-Box with two cycles of latency. The first-order design offers a new

performance record in terms of area, and the second-order design offers a new record in terms of both randomness and area. At higher orders, our design is the only one achieving two cycles of latency.

We should highlight that HPCC does not fulfill the O-PINI notion [CS21]. Implementations with at least two register layers, such as the AES S-Box designs presented here, inherently avoid transitional leakages when the consecutive inputs have independent sharings. However, HPCC may not be used in single-cycle looped circuits, e.g., single-cycle-per-round implementation of lightweight ciphers with cubic S-Boxes like PRESENT [BKL+07] and PRINCE [BCG+12]. A trivial way to achieve conformity with the O-PINI notion is the addition of an extra register layer at the end of the gadget, refreshing the output shares with fresh independent randomness, which is obviously not in line with its low-latency feature. Extending HPCC to satisfy the requirements to be O-PINI while keeping the latency at one clock cycle is among our future works.

Acknowledgments

The work described in this paper has been supported in part by the German Research Foundation (DFG) through the projects 535533866 (SSIMA) and 549340884 (MatSec).

References

[AZN21] Victor Arribas, Zhenda Zhang, and Svetla Nikova. LLTI: low-latency threshold implementations. *IEEE Trans. Inf. Forensics Secur.*, 16:5108–5123, 2021.

[BBP+16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[Can05] David Canright. A very compact s-box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1,*

2005, Proceedings, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

[CDG+13] Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Pankaj Rohatgi, et al. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*, volume 20, 2013.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.

[CGM+24] Gaëtan Cassiers, Barbara Gigerl, Stefan Mangard, Charles Momin, and Rishub Nagpal. Compress: Generate small and fast masked pipelined circuits. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(3):500–529, 2024.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):136–158, 2021.

[dMKA+15] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015.

[DR98] Joan Daemen and Vincent Rijmen. The block cipher rijndael. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Smart Card Research and Applications, This International Conference, CARDIS '98, Louvain-la-Neuve, Belgium, September 14-16, 1998, Proceedings*, volume 1820 of *Lecture Notes in Computer Science*, pages 277–284. Springer, 1998.

[FGP+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic Low-Latency Masking in Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[HB25] Vedad Hadzic and Roderick Bloem. Efficient and composable masked AES s-box designs using optimized inverters. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(1):656–683, 2025.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KM22] David Knichel and Amir Moradi. Low-Latency Hardware Private Circuits. In *CCS 2022*, pages 1799–1812. ACM, 2022.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[KSM22] David Knichel, Pascal Sasdrich, and Amir Moradi. Generic Hardware Private Circuits Towards Automated Generation of Composable Secure Gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):323–344, 2022.

[LMDM25] Daniel Lammers, Nicolai Müller, Siemen Dhooghe, and Amir Moradi. Constant-cycle hardware private circuits. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(4):172–214, 2025.

[MM22] Nicolai Müller and Amir Moradi. PROLEAD A probing-based hardware leakage detection tool. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):311–348, 2022.

[MMW18] Lauren De Meyer, Amir Moradi, and Felix Wegener. Spin me right round rotational symmetry for fpga-specific AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):596–626, 2018.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully Attacking Masked AES Hardware Implementations. In *CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[MT23] Quentin L. Meunier and Abdul Rahman Taleb. Verifmsi: Practical verification of hardware and software masking schemes implementations. In Sabrina De Capitani di Vimercati and Pierangela Samarati, editors, *Proceedings of the 20th International Conference on Security and Cryptography, SECRYPT 2023, Rome, Italy, July 10-12, 2023*, pages 520–527. SCITEPRESS, 2023.

[SAK16] SAKURA. Side-channel Attack User Reference Architecture. http://satoh.cs.uec.ac.jp/SAKURA/index.html, 2016.

[SBHM20] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. Low-Latency Hardware Masking with Application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):300–326, 2020.

[SM15] Tobias Schneider and Amir Moradi. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015.

[VDB+24] Dilip Kumar S. V., Siemen Dhooghe, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. Time Sharing - A Novel Approach to Low-Latency Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(3):249–272, 2024.

[VDB+25] Dilip Kumar S. V., Siemen Dhooghe, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. Higher-order time sharing masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2025(2):235–267, 2025.

[WM18] Felix Wegener and Amir Moradi. A first-order SCA resistant AES without fresh randomness. In Junfeng Fan and Benedikt Gierlichs, editors, *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings*, volume 10815 of *Lecture Notes in Computer Science*, pages 245–262. Springer, 2018.

A Instantiation of Z_0 for $d = 2$

In this section, we illustrate the instantiation of HPCC for $d = 2$ and discuss the intuition behind each mask used in the design. The construction of Z_0 is shown in Figure 9, which illustrates Equation (12), with the added optimization of using the inverse distributive law to combine multiplications. First, we realize the three-domain multiplications as blinded multiplications using two blinding masks for B and C (yellow). These blinded multiplications compute not only the desired multiplication, but also erroneous terms which must be canceled by a first "level" of correction terms (red), such as B_1 multiplied with the blinding randomness for C_2. These correction terms may contain two different share domains, such as A_0 and B_1, and thus, these correction terms must also be realized using blinded multiplications. The second level of correction terms (purple) cancel the blinding randomness used in these blinded multiplications and only contain terms with a single share domain.

We blind B_1 with two masks $\beta_{0,1,2}$ and $\beta'_{0,1,2}$. Intuitively, β is used to prevent B_1 from being leaked directly by a probe on Z_0, just like in quadratic HPC gadgets, while β' is used to split the correction term with C_2 into two registers. If β' was not used, a single internal probe on the correction term $C_2\beta_{0,1,2}$ and a second probe on Z_0 would reveal C_2 and A_0. Consequently, B_1 is revealed since $\beta_{0,1,2}$ is known. This would violate the PINI notion, as three share domains are revealed by only two probes. By using two blinding masks for every share in the three-domain multiplication, we ensure that three probes must be placed (one on the output, and one on each blinding register) to learn all three domains involved in the three-domain multiplication. Indeed, when three probes are placed, we may reveal all three shares involved in the three-domain multiplication. As long as we can ensure this property, some of the 8 individual correction terms may be computed in the same register. This observation motivates the trick we use to reduce the number of fresh masks and registers by combining some correction terms, such as combining the correction term $B_1\varsigma_{0,1,2}$ with the correction terms $\beta_{0,1,2}\varsigma_{0,1,2}$, $C_1\beta_{0,2,1}$ and $\varsigma'_{0,2,1}\beta'_{0,2,1}$. This ultimately results in two correction terms per three-domain multiplication. If there was only one correction term, a probe placed inside the correction term and an output probe on Z_0 would violate the PINI notion.

We now consider the blinding masks required to realize the correction terms. In HPC3.X, masks in the correction terms are shared between the domains where the correction term is added, and the output domain which corresponds to the domain blinded by the blinding mask in the correction term. For example, in HPC3.X, if some mask λ is used to blind B_1 in Z_0 and B_0 in Z_1, then $\mathrm{Reg}\,[A_0\lambda + \lambda']$ and $\mathrm{Reg}\,[A_1\lambda + \lambda']$ in Z_0 and Z_1 are blinded by the same mask. This is motivated by the observation that when two output probes are placed on domains 0 and 1, input share domains 0 and 1 are available to the simulator by

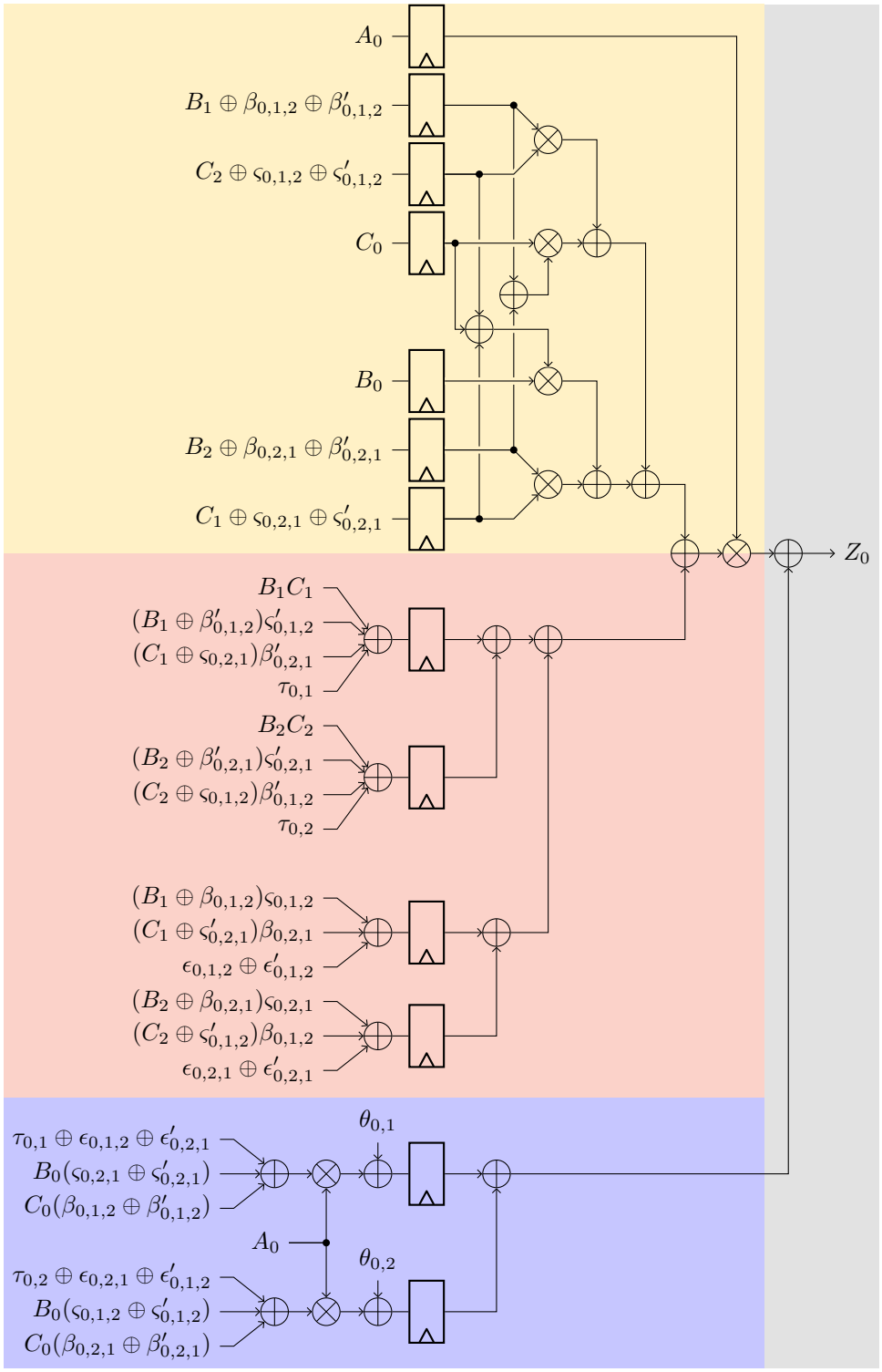


Figure 9: Instantiation of second-order HPCC, showing just the computation of Z_0. It consists of the primary multiplications (yellow), a first level of correction terms (red), and a second level of correction terms (purple).

the definition of PINI, and so the blinding masks used to protect shares of these domains become effectively "useless". We can apply a similar principle to the correction terms of a three-domain multiplication. When two output probes are placed on share domains 0 and 1, any masks blinding share domain 0 or 1 may be revealed. However, the blinding masks for share domain 2 must stay secret, and so we cannot reveal all correction terms with only two probes on output share domains 0 and 1. Hence, at least one correction term must be properly blinded unless three probes are placed. We arbitrarily choose one of the two correction terms and blind it with $\epsilon_{0,1,2} \oplus \epsilon'_{0,1,2}$, so that we can reuse $\epsilon_{0,1,2}$ for $\epsilon_{1,0,2}$ and $\epsilon'_{0,1,2}$ for $\epsilon'_{2,1,0}$. The other correction term may be revealed by two probes, and as such is blinded by $\tau_{0,1}$, which is reused for $\tau_{1,0}$. This also motivates the further merging of correction terms at higher orders. If they were not merged, multiple registers in Z_0 would be blinded by the same mask $\tau_{0,1}$, which would immediately violate the PINI notion.

The second level of correction terms is constructed like the correction terms in quadratic HPC gadgets, where each correction term contains only share domain 0 and blinding masks protecting a single other domain. Therefore, these final terms may be revealed by two probes on 0 and the other protected domain. We therefore blind the final level of correction terms by $\theta_{0,1}$ and $\theta_{0,2}$, which are reused for $\theta_{1,0}$ and $\theta_{2,0}$ respectively. This mask reuse is required to ensure correctness, so that each blinding mask appears twice in the addition of all output shares.