Generation of Authenticated Secret-Shared Scaled Unit Vectors for Beaver Triples*

Vincent Rieder[1,2][0009−0007−8694−7260]

[1] University of Stuttgart
[2] Bosch Research

For secure multi-party computation in the line of the secret-sharing based SPDZ protocol, actively secure multiplications consume correlated randomness in the form of authenticated Beaver triples, which need to be generated in advance. Although it is a well-studied problem, the generation of Beaver triples is still a bottleneck in practice. In the two-party setting, the best solution with low communication overhead is the protocol by Boyle et al. (Crypto 2020), which is derived from the recent primitive of Pseudorandom Correlation Generators (PCGs) (Crypto 2019). Their protocol requires less than 2 MB of communication to generate about 100 MB of Beaver triples (per party). In this work, we improve their protocol in terms of communication (7%), computation (20% for its interactive phase), and the amount of correlated randomness consumed by internal secure two-party computations (11% storage). To achieve our improvements, we propose a novel actively secure protocol for the efficient generation of (authenticated) secret-shared scaled unit vectors, which in general are the main building blocks of current PCG protocols.

1 Introduction

Secure multi-party computation (MPC) is a privacy-enhancing technology that allows untrusted parties to evaluate a public function on private inputs with the privacy guarantee that nothing is leaked beyond what the parties can learn from their private inputs and outputs. We consider actively secure MPC in the framework of the SPDZ protocol [15] for to evaluate arithmetic circuits over finite fields. The SPDZ protocol has paved the way for practical and efficient MPC protocols based on additive secret-sharing, where the underlying secret-sharing scheme includes an authentication of secret-shared values to achieve active security.

In our work, we consider the two-party setting. More broadly, secure two-party computation with SPDZ induces a scenario for secure multi-party computation, where several clients outsource a large-scale secure computation to a secure two-party computation between two servers [13], e.g., cloud servers. In this scenario, important cost factors are the bandwidth between the servers and how the performance scales with the size of the function to be evaluated.

* This work is funded by the German Federal Ministry of Education and Research https://www.cryptecs.eu

For efficiency reasons, the SPDZ protocol works in the preprocessing model where the secure function evaluation – that takes place in a so-called online phase – consumes distributed correlated randomness that is generated in advance in an input-independent offline phase. While, in general, online phases are designed to be highly efficient, offline phases are typically based on expensive cryptographic constructions. In the scenario of SPDZ, one secure multiplication in an online phase consumes correlated randomness in the form of one (authenticated) Beaver triple, also known as authenticated multiplication triple. While secure multiplications are highly efficient and use only a few bits of communication, offline phases for the generation of Beaver triples are proven to be a bottleneck, in terms of computation and communication [20].

With our work, we contribute to improve the actively secure generation of general-purpose two-party Beaver triples over a finite field. Common primitives to generate Beaver triples are homomorphic encryption, as used by the initial SPDZ protocol [4,14,15,26] and Overdrive [3,22], or oblivious transfer (OT) extensions used, e.g., used by MASCOT [21]. Other approaches benefit from switching from Beaver triples over a field to special-purpose Beaver triples over a ring [17,25,28].

In contrast to all these work, we consider a recent two-party Beaver triple generation protocol by Boyle et al. [10] that is constructed from a Distributed Point Function (DPF) scheme [16,18] and the coding theoretic ring-LPN assumption [10]. The construction follows the paradigm of Pseudorandom Correlation Generators (PCGs) [6,9], which is a recent primitive to generate large batches of correlated randomness with low communication. In this sense, the PCG approach is specifically interesting in the out-sourcing scenario for the secure evaluation of large-scales functions mentioned above. Concretely, for a batch of one million Beaver triples (about 100 MB), the protocol from Boyle et al. [10] reduces the communication from about 2 GB (required by Overdrive) to about 2 MB (per party).

Apart from achieving low communication, the Beaver triple generation protocol [10] is expected to be practical efficient. However, compared to other Beaver triple protocols and their implementation [22], for the practical performance of the protocol by Boyle et al. [10] there exist only estimations. One performance aspect is that their protocol relies on MPC in the preprocessing model, i.e., it consumes different types of correlated randomness that needs to be preprocessed in advance and stored in between [8,10,19]. Since it is an open question how much the preprocessing contributes to the performance in practice, we consider the amount of consumed correlated randomness as a further performance metric, next to computation and communication for the Beaver triple generation protocol itself.

1.1 Why consider authenticated secret-shared scaled unit vectors?

Before we summarize our contributions, we motivate the role of *authenticated secret-shared scaled unit vectors* inside the Beaver triple generation protocol from Boyle et al. [10]. Shortly speaking, their protocol follows a template of

recent PCGs [6,9] that states to encode the distributed target correlation into a batch SUVs. These unit vectors are correlated in terms of their positions and non-zero payloads [6], where the respective compression into their positions and payloads enables to generate the SUVs with low communication. Concretely, the Beaver triple generation protocol [10] consists of two parts: An *interactive phase* to generate the SUVs (following a DPF scheme, see Section 1.3), and a silent *local phase* to expand the SUVs into the target correlation (under the ring-LPN assumption [10]). In our work, we improve the interactive phase by improving the underlying actively secure SUV generation protocol by Boyle et al. [10] – setting up a suitable coding theoretic assumption for an efficient and secure local phase is another story [5,7,29].

Why consider authenticated SUVs instead of SUVs? The term SUV refers to a secret-shared scaled unit vector, where each coefficient is individually additively secret-shared. While this is sufficient to keep its position and payload, an SUV provides only passive security, i.e., a malicious party can introduce errors in subsequent computations without being detected. As an actively secure extension, we introduce *authenticated* SUVs (aSUVs), where each coefficient is additively secret-shared with an additional authentication that can be used to verify subsequent secure computations against potential malicious behavior. Concretely, for the authentication we use the message authentication code (MAC) [14] as introduced with the SPDZ protocol. Then, using aSUVs instead of SUVs enables a more compact presentation of the Beaver triple generation protocol from Boyle et al. [10], where the explicit focus on aSUVs highlights our improvements.

1.2 Contribution

We improve the Beaver triple protocol from Boyle et al. [10] in terms of communication, computation, and consumed correlated randomness. To achieve this, we provide an explicit protocol to generate aSUVs, which improves the implicit method that Boyle et al. [10] provide as part of their Beaver triple protocol. Hereby, our starting point is to improve their SUV protocol.

Improved generation of SUVs: We reduce the communication costs of the SUV protocol [10], while on the same time we keep the computational costs unchanged. Furthermore, our SUV protocol consumes only correlated randomness in the form of Beaver triples: Compared to the initial protocol [10], we avoid the dependency on random VOLE correlations (vector oblivious linear equation) [2]. This has two practical advantages: Firstly, we avoid an actively secure preprocessing stage to generate random VOLE correlations and secondly we avoid the intermediate memory consumption to store the VOLE correlations.

Explicit and improved generation of aSUVs: We extend our improved SUV protocol to an aSUV protocol that improves the proposal by Boyle et

al. [10]. Hereby, our improvement of the SUV protocol can be seen as a preparatory step, which applied to the aSUV generation even reduces the computational costs. Our aSUV protocol only consumes correlated randomness in the form of Beaver triples, avoiding the dependency on VOLE.

Improved generation of Beaver triples: The interactive phase of the Beaver triple protocol from Boyle et al. [10] requires the generation of a few thousand aSUVs to generate one million Beaver triples. The improvements we achieve with our aSUV protocol are: 7% less communicating (a few hundred KB), 11% less consumed correlated randomness (avoiding VOLE correlations with a few MB), and 20% less computational costs for its interactive phase (hundred millions of calls to the AES block cipher and hundred millions of field multiplications) for one million Beaver triples. To give some context: On the one hand, one secure multiplication in an online phase consumes one Beaver triple and takes five field multiplications per party, on the other hand in the offline phase we save hundreds of field multiplications per Beaver triple.

While our aSUV protocol does not consume VOLE correlations, this is initially not true for the Beaver triple protocol [10] itself. To actually avoid a preprocessing stage for VOLE correlations, we introduce an efficient special-purpose aSUV protocol that, compared to the general-purpose aSUV protocol, includes to sample a uniformly random payload of the unit vector to be secret-shared.

We stress that our improvements of the aSUV generation only apply to the interactive part of the Beaver triple protocol. However, we conjecture our improvements as asymptotically effective for the whole Beaver triple protocol, including its local phase and the preprocessing stage for the consumed correlated randomness.

1.3 Context: SUVs, DPFs, and PCGs

Difference between the generation of SUVs and DPF schemes: In order to construct their actively secure Beaver triple protocol, Boyle et al. [10] extend the passively secure DPF key generation protocol from Doerner and Shelat [11,16] to be actively secure, using a consistency check [29]. However, the security guarantees of their protocol are not compatible with the formalism of a DPF scheme. Instead, they use the Universal Composability (UC) framework [12] to formulate security by making a transition from a DPF key generation protocol to an SUV generation protocol. While SUVs and additively secret-shared point functions are equivalent (in terms of their position and payload), the notion of an SUV protocol is conceptually simpler than a DPF key generation protocol: The SUV protocol from Boyle et al. [10] directly outputs a full secret-shared unit vector, and not a respective compression in the form of private DPF keys. As a consequence, the Beaver triple protocol [10] is strictly speaking not in the form of a secure PCG [9]. Instead, the security is formulated in the UC-framework.

Related work for PCGs: Apart from inefficient general purpose constructions [9], and a generalization to the multi-party setting [1], to our knowledge, the

protocol from Boyle et al. [10] is the only two-party Beaver triple protocol in the line of PCGs. Note that this protocol is for authenticated Beaver triples, whereas for multiplication triples without authentication, as sufficient for passively secure MPC, there exist simpler PCG constructions [9,10].

Instead of a DPF scheme, many PCG constructions [8,27,29] use a Pseudorandom Puncturable Function [8], which is a, practically more efficient, relaxation to generate a secret-shared scaled unit vector under the assumption that one party knows the position. However, this relaxation does not apply to the generation of SUVs inside the Beaver triple protocol from Boyle et al. [10], where all SUV positions need to remain secret. Shortly speaking, the reason is that Beaver triples have a multiplicative depth of two due their authentication, and hence their generation turns out to be more challenging and complex compared to linear forms of correlated randomness like VOLE or oblivious transfer (OT).

2 Preliminaries

Let $\mathbb{F}$ be the generic term for a finite ν-bit field, where ν is a security parameter. With $s \xleftarrow{\$} S$, we denote that s is sampled uniformly at random from a set S. We write S^d (or $S^{d\times d}$) for the set of d (or d^2) dimensional vectors over S.

2.1 Secure Arithmetic from Authenticated Secret-Sharing

We restrict our work to the two-party setting and use the index $\sigma \in \{0,1\}$ for secret values known by party P_σ. We use the following two authenticated secret-sharing schemes with induced secure arithmetic:

Secret sharing over $\mathbb{F}$: For secure arithmetic over $\mathbb{F}$, we use secret-sharing as in the SPDZ line of MPC [14,15]. Hereby, we write $[x]$ for an *additively secret-shared* value, i.e., $x = x_0 + x_1 \in \mathbb{F}$ is decomposed uniformly random where P_σ knows x_σ. Then, to achieve active security, an authenticated secret-shared value $[\![x]\!]$ consists of $[x]$ and a *message authentication code* (MAC) $[x'] = [\mathrm{m}x]$, where $\mathrm{m} \xleftarrow{\$} \mathbb{F}$ is a global additively secret-shared MAC key. In Protocol 1 we describe secure arithmetic over $[\![\cdot]\!]$, tailored to our work, e.g., including a definition of *Beaver triples* for secure multiplications and protocols to input and output secret-shared values.

Bit-wise secret sharing of integers: We write $\|\alpha\|_m$ for an m-bit integer that is secret-shared by m individual authenticated bits with a MAC as defined in the TinyOT protocol [24]. For our work, we need a secure integer addition $\|\gamma\|_{m+1} = \|\alpha\|_m + \|\beta\|_m$. Using a binary circuit, this is possible with $2m$ bits of communication, consuming m *AND triples* (Beaver triples with respect to $\|\cdot\|_1$)[10,19,23]. A batch of random secret-shared bits is also known as *random correlated OT* (COT).

Protocol 1 Secure arithmetic with $[\![\cdot]\!]$[10,14,15]

- The scheme $[\![\cdot]\!]$ is linear, i.e., given shares $[\![x]\!], [\![y]\!]$ and public values $\epsilon, \delta, \rho \in \mathbb{F}$, the parties can locally compute a share of $\epsilon x + \delta \cdot y + \rho$, written $[\![z]\!] = \epsilon \cdot [\![x]\!] + [\![y]\!] + \delta$, where $(z_\sigma, z'_\sigma) = (\epsilon x_\sigma + \delta y_\sigma + \sigma \cdot \rho, \epsilon x'_\sigma + \delta y'_\sigma + \mathrm{m}_\sigma \cdot \rho)$.
- We write $\mathbf{Input}(P_\sigma, x)$ for the functionality to create a secret sharing $[\![x]\!]$ of a value x known by P_σ. Using one entry $u_0 + u_1 = v_\sigma \cdot \mathrm{m}_{1-\sigma}$ of VOLE correlation [2] conditioned on $\mathrm{m}_{1-\sigma}$, this can be achieved with P_σ sending $x - v_\sigma$ to $P_{1-\sigma}$. Then the parties can locally compute their private shares $(x_\sigma, x'_\sigma) = (x, \mathrm{m}_\sigma \cdot x + u_\sigma)$ and $(x_{1-\sigma}, x'_{1-\sigma}) = (0, \mathrm{m}_{1-\sigma} \cdot (x - v_\sigma) + u_{1-\sigma})$ of $[\![x]\!]$
- We write $\mathbf{Open}([\![x]\!])$ for the functionality that securely reveals a secret-shared value $[\![x]\!]$. In a first step, the parties *exchange* x_σ, i.e, P_0 sends x_0 to P_1 and vice versa. Additionally, the parties perform a MAC check to check if the revealed value x was computed correctly and is consistent with $[\![x]\!]$, by checking if $x' = \mathrm{m}x$. Since the MAC check can be efficiently performed for many values at once, we do not consider it in our evaluation.
- For a secure multiplication of two secret-shared values $[\![x]\!], [\![y]\!]$, let $[\![a]\!], [\![b]\!], [\![c]\!]$, conditioned on $a, b \xleftarrow{\$} \mathbb{F}, c = a \cdot b$, be a so called *Beaver triple*. Then, the parties call $\epsilon \leftarrow \mathbf{Open}([\![x]\!] - [\![a]\!]), \delta \leftarrow \mathbf{Open}([\![y]\!] - [\![b]\!])$ and locally compute

$$[\![z]\!] = \delta \cdot [\![x]\!] + \epsilon \cdot [\![y]\!] + [\![c]\!] - \epsilon\delta. \tag{1}$$

- For a secure inversion, i.e. to compute a share $[\![x^{-1}]\!] = [\![x]\!]^{-1}$ from a share $[\![x]\!]$, the parties take a Beaver triple $[\![a]\!], [\![b]\!], [\![c]\!]$. Then they open $x + a$ and compute $[\![y]\!] = (x+a)[\![b]\!] - [\![c]\!]$. Afterwards they open y and locally compute $[\![x^{-1}]\!] = y^{-1} \cdot [\![b]\!]$. If $x = 0$ or $b = 0$, then $y = 0$ causes an abortion.

2.2 (Authenticated) Secret-Shared Scaled Unit Vectors

We extend $[\![\cdot]\!]$ to vectors over $\mathbb{F}^M$, where each coefficient is secret-shared individually, written $[\![\cdot]\!]_{\mathbb{F}^M}$. Then an (a)SUV is a secret-shared vector from the set $\mathcal{U}_M \subset \mathbb{F}^M$ of scaled unit vectors, i.e., from the set of vectors that are zero except for one *position* $\alpha \in [0, M)$ where they take a *payload* $A \in \mathbb{F}$. Formally:

Definition 1. *A secret-shared scaled unit vector (SUV) is a secret-shared vector* $[x]_{\mathcal{U}_M} := [x]_{\mathbb{F}^M}$ *with* $x \in \mathcal{U}_M$*. An authenticated secret-shared scaled unit vector (aSUV) is a secret-shared vector* $[\![x]\!]_{\mathcal{U}_M} \simeq ([x]_{\mathcal{U}_M}, [x']_{\mathcal{U}_M})$ *with* $x \in \mathcal{U}_M$*.*

With the term two-dimensional SUV, we refer to a pair of SUVs $[x]_{\mathcal{U}_M}, [y]_{\mathcal{U}_M}$ that have the same position.

The Protocol $\Pi^{\mathbf{Boyle}}_{\mathbf{SUV}}$: We now summarize the actively secure SUV protocol from Boyle et al.[10], named $\Pi^{\mathrm{Boyle}}_{\mathrm{SUV}}$. Additionally, we provide a formal presentation in Protocol 7, Appendix A.1. The purpose of $\Pi^{\mathrm{Boyle}}_{\mathrm{SUV}}$ is to generate an SUV $[x]_{\mathcal{U}_M}$ of dimension $M = 2^m$, given a secret-shared position $\|\alpha\|_m$ and payload $[\![A]\!]$. For our work, we divide $\Pi^{\mathrm{Boyle}}_{\mathrm{SUV}}$ into three parts: A *tree phase*, which processes the position α, a *verification phase* to detect malicious behavior, and a

payload correction phase, which processes the payload A. The tree phase runs through a PRG-based binary tree of depth m and outputs private *left* and *right* vectors $(t_\sigma^L, t_\sigma^R) \xleftarrow{\$} \mathbb{F}^M \times \mathbb{F}^M$, under the condition that $[t^L]_{\mathcal{U}_M}, [t^R]_{\mathcal{U}_M}$ are two SUVs with the same position α and secret payloads $(u^L, u^R) \xleftarrow{\$} \mathbb{F} \times \mathbb{F}$. We treat the tree phase as a black box, for an intuition see Appendix A.1. Under the premise to keep α and A secret, the next two phases are over $\mathbb{F}$ (*field phase* for short):

1. **Verification:** The tree phase contains passively secure parts that allow an adversary to break the SUV structure of $[t^L]_{\mathcal{U}_M}, [t^R]_{\mathcal{U}_M}$. To prevent such attacks on $[t^L]_{\mathcal{U}_M}$, the field phase uses $[t^R]_{\mathcal{U}_M}$ to verify that $[t^L]_{\mathcal{U}_M} \cong [t^L]_{\mathbb{F}^M}$ is indeed in the form of an SUVs. If not, it aborts. By the means of the authenticated secret sharing scheme $\|\cdot\|_m$, the tree phase then already implies the correctness of the position α. Additionally, the verification procedure provides the payload u^L of t^L in authenticated secret-shared form $[\![u^L]\!]$.
2. **Payload Correction:** To actually transform t^L into the output SUV $[x]_{\mathcal{U}_M}$ with payload A, the random payload u^L has to be corrected. Concretely, the payload correction is a linear operation $[x] = \mathrm{CW}^L \cdot [t^L]_{\mathcal{U}^M}$, where $\mathrm{CW}^L = \frac{A}{u^L}$ is securely computed over $[\![\cdot]\!]$. If it happens that $u^L = 0$, which has negligible probability in ν, the protocol requires a restart.

Costs of $\Pi_{\mathbf{SUV}}^{\mathbf{Boyle}}$: The protocol $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ inherits the asymptotic costs from previous DPF schemes [11,16]. More precisely, the costs to generate an M dimensional SUV with $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ are distributed between the tree phase and the field phase as follows:

- The computational costs of the tree phase are dominated by $2M$ calls to a PRG with security parameter λ, e.g., realized with $4M$ calls to the AES block cipher [16]. The communication costs of the tree phase are $\mathcal{O}(m\lambda)$, and it does not consume any correlated randomness.
- For the field phase, the secure arithmetic over $[\![\cdot]\!]$ takes $\mathcal{O}(\nu)$ communication and consumes a constant number of Beaver triples and VOLE. Roughly 30% of the communication is given by calls of **Input** (Protocol 1). The computation is dominated by $3M$ field multiplications (3 scalar-vector multiplications).

The Protocol $\Pi_{\mathbf{aSUV}}^{\mathbf{Boyle}}$: The actively secure Beaver triple protocol [10] involves the generation of aSUVs. Hence, although Boyle et al. [10] do not formalize aSUVs as a primitive, they provide a method to generate aSUVs (Section 1.1), which we formalize as protocol $\Pi_{\mathrm{aSUV}}^{\mathrm{Boyle}}$ (Protocol 7). Given a position $\|\alpha\|_m$, payload $[\![A]\!]$ and MAC key $[\![\mathrm{m}]\!]$, the aSUV protocol $\Pi_{\mathrm{aSUV}}^{\mathrm{Boyle}}$ computes $[\![A']\!] = [\![\mathrm{m}]\!] \cdot [\![A]\!]$ and then generates a two-dimensional SUV with position α and payload (A, A'). The straightforward approach to generate this two-dimensional SUV is to make two calls to $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$. Instead to be more efficient, Boyle et al [10] remark

Protocol 2 Ideal Functionalities for the SUV and aSUV generation [10]

Blue statements refer only to **aSUV**.

aSUV$(\|\alpha\|_m, [\![A]\!], [\![\mathrm{m}]\!])$:

- If both parties are honest: Exclude the trivial SUV with zero payload, i.e., if $A = 0$, output "$A = 0$" to both parties and abort. Exclude the trivial MAC, i.e., if $\mathrm{m} = 0$, output "$\mathrm{m} = 0$" to both parties and abort. Sample $x_0 \xleftarrow{\$} \mathbb{F}^M$ and let $x_1 = (0, \dots, 0, A, \dots, 0) - x_0$, where A is in position α. Sample $x'_0 \xleftarrow{\$} \mathbb{F}^M$ and let $x_1 = (0, \dots, 0, \mathrm{m}A, \dots, 0) - x_0$, where $\mathrm{m}A$ is in position α. For $\sigma \in \{0, 1\}$ output x_σ, x'_σ to party P_σ. Note that this defines an SUV $[x]_{\mathbb{F}^M}$ (aSUV $[\![x]\!]_{\mathbb{F}^M}$).
- If party P_σ is malicious: Wait for input $x_\sigma \in \mathbb{F}^M$ and $x'_\sigma \in \mathbb{F}$ from P_σ. Wait for a predicate $P : [0, M)] \to \{0, 1\}$ from P_σ. If $P(\alpha) = 0$, abort. If $A = 0$, output "$A = 0$" to both parties and abort. If $\mathrm{m} = 0$, output "$\mathrm{m} = 0$" to both parties and abort. Set $x_{1-\sigma} = (0, \dots, 0, A, 0, \dots 0) - x_\sigma$, where A is in position α. Set $x'_{1-\sigma} = (0, \dots, 0, \mathrm{m}A, 0, \dots 0) - x'_\sigma$, where $\mathrm{m}A$ is in position α. Output "success" to the adversary and $x_{1-\sigma}$, $x'_{1-\sigma}$ to $P_{1-\sigma}$.

how to extend $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$. For this, the tree phase outputs a third SUV $[t^{L'}]_{\mathcal{U}_M}$, which is then mapped to the second output component. In between, the field phase verifies the SUV structure of $[t^{L'}]_{\mathcal{U}_M}$, analogously to $[t^L]_{\mathcal{U}_M}$ with respect to the same $[t^R]_{\mathcal{U}_M}$.

Security of $\Pi_{\mathbf{SUV}}^{\mathbf{Boyle}}$ and $\Pi_{\mathbf{aSUV}}^{\mathbf{Boyle}}$: In the best case, $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ would implement the ideal functionality that samples a random SUV given its position and payload. However, $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ tolerates two sorts of malicious behavior. Since this will be similar for our improved protocols, we present the achieved ideal functionalities **SUV**, **aSUV** in Protocol 2. Firstly, a malicious party can control its share x_σ of the output SUV $[x]_{\mathcal{U}_M}$ and secondly, within the verification step, it can take an arbitrary guess on the position α of x by passing a guess function $P : [0, M) \to \{0, 1\}$. Then, **SUV** aborts if and only if $P(\alpha) = 0$, otherwise it continues. Since in the successful case the information $\alpha \in I = \{\beta \in [0, M) \mid P(\beta) = 1\}$ is leaked, we denote this as *selective failure attack*. We stress that the respective leakage can be caught up in the application of the Beaver triple protocol [10], especially since the risk to be detected is higher for smaller, and hence more interesting, sets I. Formally the security guarantees of $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$, $\Pi_{\mathrm{aSUV}}^{\mathrm{Boyle}}$ are:

Theorem 1 ([10]). *Protocol Π_{SUV}^{Boyle} securely implements the functionality **SUV** with security against malicious adversaries in the UC-model. Similarly, Π_{aSUV}^{Boyle} securely implements **aSUV**.*

3 Improved Generation of SUVs and aSUVs

In this section we construct improved (a)SUV protocols $\Pi_{\mathrm{SUV}}, \Pi_{\mathrm{aSUV}}$ (Protocol 4). We first outline how we modify $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ to achieve our improvements. (Section 3.1). Afterwards we describe our protocols (Section 3.2). We reveal the formal relation to the protocol $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ as part of the security proof in Section 3.3.

3.1 Outline of our Modifications to $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$

To improve the generation of SUVs and aSUVs, we extend $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ into a protocol for the generation of two-dimensional SUVs. Our improvement is better than the respective extension of $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ that Boyle et al. [10] use for $\Pi_{\mathrm{aSUV}}^{\mathrm{Boyle}}$. Concretely, we achieve the following:

1. **Symmetric Verification:** The verification makes only use of two internal SUVs $[t^L], [t^R]$, in a symmetric way. Compared to the two-dimensional extension by Boyle et al. [10], the symmetry allows us to avoid the additional component $t^{L'}$ and the respective computational costs.
2. **Avoiding calls to Input:** We avoid calls to **Input**, i.e., the respective communication, and the storage and preprocessing of VOLE correlations.

As a result, if we ignore the second output dimension, we get an SUV protocol Π_{SUV} with decreased communication costs compared to $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ (Item 2). Furthermore, if we restrict the two-dimensional payloads to the form $(A, \mathrm{m}A)$, we get an aSUV protocol Π_{aSUV} with decreased computational and communication costs compared to $\Pi_{\mathrm{aSUV}}^{\mathrm{Boyle}}$ (Item 1). We now outline how we modify the verification phase of $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ to achieve the above improvements.

Symmetric verification: The verification procedure of $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ is asymmetric: While the left component "L" is related to the output, the right component "R" is only used internally for the verification (Section 2.2). For this, $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ reveals the random payload u^R of t^R. Instead, we modify the verification procedure to be symmetric with respect to t^L, t^R such that it does not reveal any information about t^R. As a result, t^R can be additionally mapped to the output, i.e., we make a transition from SUVs to two-dimensional SUVs without the need of an additional component L'. Instead, the additional cost that we need for the transition is only one secure multiplication.

Avoiding calls to Input: To keep α and A secret, the verification of t^L, t^R in $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ takes place as secure computation over $[\![\cdot]\!]$, which requires to translate private information about the unauthenticated SUVs $[t^L], [t^R]$, e.g., sums of their individual private shares, to the scheme $[\![\cdot]\!]$. In $\Pi_{\mathrm{SUV}}^{\mathrm{Boyle}}$ the private unauthenticated inputs are turned into authenticated secret-shared values $[\![\cdot]\!]$ with calls to **Input** (Protocol 1). We point out that any deviation on t^L, t^R will be detected

Protocol 3 Secure multiplication of unauthenticated factors

INPUT: $[x]_{\mathbb{F}}, [y]_{\mathbb{F}}$
OUTPUT: $[\![x]\!], [\![y]\!], [\![z]\!]$ with $z = x \cdot y$.

Let $[\![a]\!], [\![b]\!], [\![c]\!]$ be a Beaver triple, i.e., $c = a \cdot b$.
Locally, the parties compute $\epsilon_\sigma = x_\sigma - a_\sigma$ and $\delta_\sigma = y_\sigma - b_\sigma$. Then they exchange $\epsilon_\sigma, \delta_\sigma$ and locally compute $\epsilon = \epsilon_0 + \epsilon_1, \delta = \delta_0 + \delta_1$. The computation of $[\![z]\!]$ is local and given by Eq. (1). Additionally, the parties locally reshare x, y in authenticated form: $[\![x]\!] = [\![a]\!] + \epsilon, [\![y]\!] = [\![b]\!] + \delta$.

SECURITY: Since $[x]$ and $[y]$ are unauthenticated additive secret-shared, a malicious party might introduce additive errors Δ_x, Δ_y. In this case, the output is $[\![x + \Delta_x]\!], [\![y + \Delta_y]\!], [\![(x + \Delta_x) \cdot (y + \Delta_y)]\!]$, i.e., the outputs are always bounded to each other by the product property.

by the verification procedure itself. Hence, it is redundant to authenticate the information about t^L, t^R in advance of the verification. Instead, we provide a method which implicitly inputs the information about t^L, t^R during secure multiplications that are part of the the verification itself, such that we can avoid previous calls to **Input** without any additional costs.

To incorporate the calls of **Input** into the secure multiplications, we use an adapted form of a secure multiplication. Normally, the purpose of a secure multiplication is to compute $[\![xy]\!]$, given $[\![x]\!], [\![y]\!]$ (see Protocol 1). Instead, we use a secure multiplication that computes $[\![xy]\!], [\![x]\!], [\![y]\!]$, given unauthenticated shares $[x], [y]$ (see Protocol 3). Within $\Pi_{\text{(a)SUV}}$ the context is that the unauthenticated factors are derived from the unauthenticated SUVs $[t^L]_{\mathcal{U}_M}, [t^R]_{\mathcal{U}_M}$. The product $[\![xy]\!]$ is used for the verification of t^L, t^R, while the authenticated factors $[\![x]\!], [\![y]\!]$ are used for the payload correction step afterwards. Hereby x, y are bound to the verification by the means of the authentication. Note that since the inputs $[x], [y]$ are unauthenticated, a malicious party can introduce errors to $[\![x]\!], [\![y]\!]$. However, in the context of $\Pi_{\text{(a)SUV}}$ this turns out not to be an security issue, since, apart from the selective failure attack (Section 2.2), potential errors are detected by the verification (see Section 3.3).

We stress that to avoid all calls of **Input**, we require that the verification is symmetric: Initially, the asymmetric verification within $\Pi^{\text{Boyle}}_{\text{(a)SUV}}$ takes local inversions in $\mathbb{F}$. By making the verification symmetric, we turn these inversions into a secure multiplication, such that we can apply Protocol 3.

3.2 The Protocols Π_{SUV} and Π_{aSUV}

We now describe our protocols $\Pi_{\text{SUV}}, \Pi_{\text{aSUV}}$, which are depicted in Protocol 4. The structure of Π_{SUV} is identical to the structure of $\Pi^{\text{Boyle}}_{\text{SUV}}$. As input Π_{SUV} takes a secret-shared position $\|\alpha\|_m$ and payload $[\![A]\!]$. The first step is to call the tree phase of $\Pi^{\text{Boyle}}_{\text{SUV}}$ to get private vectors t^L_σ, t^R_σ (Appendix A.1). Under

the assumption of an honest execution, these define two SUVs $[t^L]_{\mathcal{U}_M}, [t^R]_{\mathcal{U}_M}$ with position α and uniformly random payloads u^L, u^R. Afterwards, to detect malicious behavior, the verification phase verifies the structure of t^L, t^R, and additionally provides u^L, u^R in secret-shared form $[\![u^L]\!], [\![u^R]\!]$. In the final phase, the payload u^L of t^L is corrected to A, such that the output $[x]_{\mathcal{U}_M}$ is as specified by the inputs α, A. Furthermore, since our verification keeps u^R secret, we get Π_{aSUV} almost for free: It only requires an additional payload correction $[x']_{\mathcal{U}_M} = \frac{Am}{u^R} \cdot [t^R]$ for the MAC component x'.

Symmetric verification: To check that $[t^L]_{\mathbb{F}^M}, [t^R]_{\mathbb{F}^M}$ are indeed SUVs $[t^L]_{\mathcal{U}_M}$, $[t^R]_{\mathcal{U}_M}$ with the same position α, the verification of $\Pi_{\text{(a)SUV}}$ computes a check value $Z \in \mathbb{F}$ that is, up to some negligible probability in the field size ν, zero if and only if t^L, t^R are scaled unit vectors with the same position α. If $Z \neq 0$, the protocols $\Pi_{\text{(a)SUV}}$ abort, else they continue. We now derive the definition of Z. For a description how $\Pi_{\text{(a)SUV}}$ uses Protocol 3 to actually compute Z, we refer to the next paragraph. Remember, that the verification has to be done without revealing information about the secret position α. Hence, the parties operate on sums that uniformly depend on all M coefficients of t^L, t^R. Concretely, given a public random vector $r \xleftarrow{\$} \mathbb{F}^M$, the parties compute local sums $u^L_\sigma, u^R_\sigma, v^L_\sigma, v^R_\sigma$ (see Protocol 4). In other words, they create additively secret-shared sums $[u^L], [u^R], [v^L], [v^R]$, for which, under the assumption that $[t^L]_{\mathcal{U}_M}, [t^R]_{\mathcal{U}_M}$ are honestly computed SUVs with position α, all summands of the form $t_0^{L,i} + t_1^{L,i}, i \neq \alpha$, cancel out. In formulas that means

$$u^L = t^{L,\alpha} \quad u^R = r^\alpha \cdot t^{R,\alpha} \quad v^L = r^\alpha \cdot t^{L,\alpha} \quad v^R = r^\alpha \cdot t^{R,\alpha}. \tag{2}$$

Especially it holds that, if $[t^L], [t^R]$ are in the form of SUVs, then u^L, u^R are equal to their payloads. Furthermore it follows that the check value $Z := u^L v^R - u^R v^L$ fulfills the requirement of the verification: If $[t^L], [t^R]$ are SUVs with the same position α, then $Z = 0$. Conversely, if $t^L, t^R \notin \mathcal{U}_M$, or if they do not have the same position α, then the terms in Eq. (2) do not collapse, in the sense that u^L, u^R, v^L, v^R depend on more than the two coefficients $t^{L,\alpha}, t^{R,\alpha}$. But then, the randomization with r ensures that $Z \neq 0$ with overwhelming probability in ν.

Realization of the verification without calls to Input: To actually compute $Z = u^L v^R - u^R v^L$, the protocols $\Pi_{\text{(a)SUV}}$ make use of the secure multiplication in Protocol 3. Concretely, we separately consider the products $Z^L = u^L v^R, Z^R = u^R v^L$ and use two secure multiplications to compute $[\![Z^L]\!]$ and $[\![Z^R]\!]$, without the need to authenticate the values $u^L_\sigma, u^R_\sigma, v^L_\sigma, v^R_\sigma$ in advance. The first multiplication takes a Beaver triple $[\![a^L]\!], [\![b^L]\!], [\![c^L]\!]$ to compute $[\![Z^L]\!], [\![u^L]\!], [\![v^R]\!]$, given $[u^L]_{\mathbb{F}}, [v^R]_{\mathbb{F}}$. Analogously, the second multiplication takes a Beaver triple $[\![a^R]\!], [\![b^R]\!], [\![c^R]\!]$ to compute $[\![Z^R]\!], [\![u^R]\!], [\![v^L]\!]$, given $[u^R], [v^L]$. Hereby, for security reasons (see Section 3.3), the parties have to commit to the shares $[\![u^L]\!], [\![u^R]\!]$ before sampling the public randomness r. Hence, the verification phase in Protocol 4 distributes the individual steps of Protocol 3 and proceeds as follows:

Protocol 4 Protocols Π_{SUV} and Π_{aSUV} ($\Pi_{\text{(a)SUV}}$)

Blue statements refer only to Π_{aSUV}. All interactive parts are framed.
INPUT: Secret shared position $\|\alpha\|_m$, payload $[\![A]\!]$ and MAC key $[\![\mathrm{m}]\!]$.
OUTPUT: $[x]_{\mathcal{U}_M}$, $[x']_{\mathcal{U}_M}$ with position α and payload A, $\mathrm{m} \cdot A$ (where $M = 2^m$).

Let $[\![a^L]\!], [\![b^L]\!], [\![c^L]\!]$ and $[\![a^R]\!], [\![b^R]\!], [\![c^R]\!]$ be two Beaver triples, e.g., $c^L = a^L \cdot b^L$.

1. TREE PHASE

   Process $\|\alpha\|_m$ to get $t^L_\sigma, t^R_\sigma \in \mathbb{F}^M$ (Section 2.2)

2. VERIFICATION

   $u^L_\sigma = \sum_{j=0}^{M-1} t^{L,j}_\sigma$
   $u^R_\sigma = \sum_{j=0}^{M-1} t^{R,j}_\sigma$

   The parties exchange $u^L_\sigma - a^L_\sigma$ to compute $\epsilon^L = (u^L_0 - a^L_0) + (u^L_1 - a^L_1)$
   The parties exchange $u^R_\sigma - a^R_\sigma$ to compute $\epsilon^R = (u^R_0 - a^R_0) + (u^R_1 - a^R_1)$

   Then let $[\![u^L]\!] = [\![a^L]\!] - \epsilon^L$ and $[\![u^R]\!] = [\![a^R]\!] - \epsilon^R$

   Sample public randomness $(r^0, \ldots, r^{M-1}) \xleftarrow{\$} \mathbb{F}^M$

   $v^L_\sigma = \sum_{j=0}^{M-1} r^j \cdot t^{L,j}_\sigma$
   $v^R_\sigma = \sum_{j=0}^{M-1} r^j \cdot t^{R,j}_\sigma$

   The parties exchange $v^L_\sigma - b^L_\sigma$ to compute $\delta^L = (v^L_0 - b^L_0) + (v^L_1 - b^L_1)$
   The parties exchange $v^R_\sigma - b^R_\sigma$ to compute $\delta^R = (v^R_0 - b^R_0) + (v^R_1 - b^R_1)$

   Then let $[\![v^L]\!] = [\![b^L]\!] - \delta^L$ and $[\![v^R]\!] = [\![b^R]\!] - \delta^R$
   $[\![Z^L]\!] = \delta^R \cdot [\![u^L]\!] + \epsilon^L \cdot [\![v^R]\!] + [\![c^L]\!] - \epsilon^L \cdot \delta^R$
   $[\![Z^R]\!] = \delta^L \cdot [\![u^R]\!] + \epsilon^R \cdot [\![v^L]\!] + [\![c^R]\!] - \epsilon^R \cdot \delta^L$

   $Z \leftarrow \mathbf{Open}([\![Z^L]\!] - [\![Z^R]\!])$

   If $Z \neq 0$, abort

3. PAYLOAD CORRECTION

   $[\![\mathrm{CW}^L]\!] = [\![u^L]\!]^{-1} \cdot [\![A]\!]$, $[\![\mathrm{CW}^R]\!] = [\![u^R]\!]^{-1} \cdot [\![\mathrm{m}]\!] \cdot [\![A]\!]$
   $\mathrm{CW}^L \leftarrow \mathbf{Open}([\![\mathrm{CW}^L]\!])$, $\mathrm{CW}^R \leftarrow \mathbf{Open}([\![\mathrm{CW}^R]\!])$

   If $\mathrm{CW}^L = 0$ or $\mathrm{CW}^R = 0$, abort
   Return $[x]_{\mathbb{F}^M} = \mathrm{CW}^L \cdot [t^L]_{\mathbb{F}^M}$ and $[x']_{\mathbb{F}^M} = \mathrm{CW}^R \cdot [t^R]_{\mathbb{F}^M}$

1. Exchange ϵ^L, ϵ^R, which commits the parties to u_σ^L, u_σ^R. Afterwards, the parties locally compute authenticated shares $[\![u^L]\!], [\![u^R]\!]$.
2. Sample r (to locally compute v_σ^L, v_σ^R).
3. Exchange δ^L, δ^R, which commits the parties to v_σ^L, v_σ^R. Afterwards, the parties locally compute authenticated shares $[\![v^L]\!], [\![v^R]\!]$.
4. Follow Eq. (1) to locally compute $[\![Z^L]\!], [\![Z^R]\!]$. Then open $Z = Z^L - Z^R$ to check if Z is zero.

Altogether, the protocol $\Pi_{\text{(a)SUV}}$ aborts, if there is a malicious behavior, i.e., if

- $Z \neq 0$, which can have two reasons: Either there was a corruption about t^L, t^R in the tree phase, i.e., if the SUVs $[t^L], [t^R]$ are broken (see Eq. (2) cf.), or a malicious party P_σ provided erroneous values for $u_\sigma^L, u_\sigma^R, v_\sigma^L, v_\sigma^R$. For the converse situation, i.e., if it can hold $Z = 0$ although the values $u_\sigma^L, u_\sigma^R, v_\sigma^L, v_\sigma^R$ are corrupted, we refer to Section 3.3.
- the opening of Z failed, i.e., if a malicious party cheated during the secure computation with $[\![\cdot]\!]$ (see **Open**, Protocol 1).

Otherwise the verification was successful, i.e., it detects no malicious behavior, and the protocol $\Pi_{\text{(a)SUV}}$ continues with the payload correction phase.

Payload correction step: The payload correction phase of Π_{SUV} is identical to the linear payload correction of $\Pi_{\text{SUV}}^{\text{Boyle}}$ (Section 2.2). In total, Π_{aSUV} has the following extended payload correction phase:

$$[x]_{\mathcal{U}_M} = \text{CW}^L \cdot [t^L]_{\mathcal{U}_M} = \frac{A}{u^L} \cdot [t^L]_{\mathcal{U}_M}, \quad [x']_{\mathcal{U}_M} = \text{CW}^R \cdot [t^R]_{\mathcal{U}_M} = \frac{\text{m}A}{u^R} \cdot [t^R]_{\mathcal{U}_M}.$$

The actual computation of CW^L, CW^R takes place as secure computation, given $[\![u^L]\!], [\![u^R]\!]$ from the verification phase and the inputs $[\![A]\!], [\![\text{m}]\!]$. A special case is if $\text{CW}^L = \text{CW}^R = 0$. Since this leaks that $A = 0$ or $\text{m} = 0$, the protocol aborts[3]. Apart from that, CW^L and CW^R do not leak information about the target payloads, since u^L, u^R are uniformly random and independent of each other by the means of the tree phase (Appendix A.1).

3.3 Active Security (Reduction to $\Pi_{\text{SUV}}^{\text{Boyle}}$)

We now prove that our protocols $\Pi_{\text{SUV}}, \Pi_{\text{aSUV}}$ are correct and actively secure, i.e., we prove an equivalence of Theorem 1:

Theorem 2. *Protocol Π_{SUV} securely implements the functionality **SUV** with security against malicious adversaries in the UC-model. Similarly Π_{aSUV} securely implements **aSUV**.*

[3] If $A = 0$, the output (a)SUV secret-shares the zero vector, which is anyway disadvantageous in applications like the Beaver triple generation. If $\text{m} = 0$, the secure arithmetic with $[\![\cdot]\!]$ is broken.

Our proof strategy is a reduction to Theorem 1. By this, we give an formal relation to the protocols $\Pi^{\text{Boyle}}_{\text{(a)SUV}}$, and avoid to repeat technical arguments of their respective proof, e.g., about the tree phase. We now outline the two steps of our reduction proof, following the two modifications described in Section 3.1. In the outline, we only cover a reduction from Π_{aSUV} to $\Pi^{\text{Boyle}}_{\text{SUV}}$, since the SUV generation can be seen as a simplification of the aSUV generation (Section 3.1, Protocol 4). For a comprehensive proof, we refer to Appendix A.2.

Step 1, Symmetric verification: In the first reduction step, we extend $\Pi^{\text{Boyle}}_{\text{SUV}}$ to a protocol for aSUVs with a symmetric verification procedure. Both protocols employ two internal unit vectors $t^L, t^R \in \mathbb{F}^M$ with secret-shared payloads $[\![u^L]\!], [\![u^R]\!]$. While $\Pi^{\text{Boyle}}_{\text{SUV}}$ maps t^L to the output SUV $[x]_{\mathcal{U}_M}$, the second component t^R is used for a verification against malicious parties. Using the notation of Protocol 4, $\Pi^{\text{Boyle}}_{\text{SUV}}$ reveals u^R and continues if and only if $[\![Z_{\text{Boyle}}]\!] = [\![u^L]\!] \cdot (\frac{1}{u^R}[\![v^R]\!]) - [\![v^L]\!]$ is zero. Instead, we keep the payload u^R of t^R secret, which enables us to use $\text{CW}^R = \frac{\text{m}A}{u^R}$ for the payload correction of the MAC component without leaking information about mA. Under this condition, the term $\frac{1}{u^R}$ in Z_{Boyle} needs to be replaced by $[\![u^R]\!]^{-1}$. In preparation of the second reduction step, we scale Z_{Boyle} with u^R and use $Z = u^L v^R - u^R v^L$ as check value. At this point, scaling is secure since it does not affect if $Z = 0$ or $Z \neq 0$, apart form the case $u^R = 0$ which we exclude in the payload correction step (the inversion $[\![u^R]\!]^{-1}$ fails). The payload correction itself is secure, since it is similar to the payload correction in $\Pi^{\text{Boyle}}_{\text{aSUV}}$.

Step 2, Removing calls to Input: Starting with $\Pi^{\text{Boyle}}_{\text{SUV}}$, the protocol resulting from Step 1 calls **Input** to create shares $[\![u^L]\!], [\![u^R]\!], [\![v^L]\!], [\![v^R]\!]$ that are subsequently used to compute $[\![Z]\!] = [\![u^L]\!] \cdot [\![v^r]\!] - [\![u^R]\!] \cdot [\![v^L]\!]$ with two multiplications according to Protocol 1. Instead, Π_{aSUV} avoids the calls to **Input** by using two calls of the secure multiplication according to Protocol 3. As mentioned in Section 3.2, these two calls are distributed inside Π_{aSUV}, which in fact preserves the order of $\Pi^{\text{Boyle}}_{\text{SUV}}$. We then use that $[\![\cdot]\!]$ is additive and provides active security to show that any potential manipulation in Π_{aSUV} can be reduced to a manipulation in the local values $u^L_\sigma, u^R_\sigma, v^L_\sigma, v^R_\sigma$ (Appendix A.2). Since these local values are similar for the protocol resulting from the previous reduction step, our modification does not give any advantage to a malicious party, especially since potential errors are detected by the respective verification procedure. Hereby, Π_{aSUV} aborts whenever $\Pi^{\text{Boyle}}_{\text{SUV}}$ aborts, since $Z_{\text{Boyle}} = 0$ implies $Z = 0$ in the previous reduction step.

Attacker for the selective failure attack: The ideal functionalities SUV and aSUV allow an adversary to execute a selective failure attack on the position α (see Section 2.2). In order to demonstrate that the selective failure attack is not merely a theoretical feature, we construct a corresponding attacker (see Appendix A.3) that is applicable to $\Pi_{\text{(a)SUV}}$ and $\Pi^{\text{Boyle}}_{\text{(a)SUV}}$.

Protocol 5 Generation of Beaver triples (Π_{Bt}) [10]

INPUT: A secret-shared MAC key $[\![\mathrm{m}]\!]$ and parameters $d = c \cdot t, N = 2^n, F$. A public randomness $\vec{u} \xleftarrow{\$} \mathcal{R}^d$.
OUTPUT: N Beaver triples over $\mathbb{F}$.

Interactive Phase:

1. For $i = 0, \ldots, d-1$ call $([\![\vec{x}_i]\!]_{\mathbb{F}^N}, \|\vec{\alpha}_i\|_n, [\![\vec{A}_i]\!]) \leftarrow \mathbf{aSUV}^*([\![\mathrm{m}]\!], n)$
2. For $i = 0, \ldots, d-1$ call $([\![\vec{y}_i]\!]_{\mathbb{F}^N}, \|\vec{\beta}_i\|_n, [\![\vec{B}_i]\!]) \leftarrow \mathbf{aSUV}^*([\![\mathrm{m}]\!], n)$
3. For $i, j = 0, \ldots, d-1$: $\|\vec{\gamma}_{i,j}\|_{n+1} = \|\vec{\alpha}_i\|_n + \|\vec{\beta}_j\|_n$
4. For $i, j = 0, \ldots, d-1$: $[\![\vec{C}_{i,j}]\!] = [\![\vec{A}_i]\!] \cdot [\![\vec{B}_j]\!]$
5. For $i, j = 0, \ldots, d-1$ call $[\![\vec{z}_{i,j}]\!]_{\mathbb{F}^{2N}} \leftarrow \mathbf{aSUV}(\|\vec{\gamma}_{i,j}\|_{n+1}, [\![\vec{C}_{i,j}]\!], [\![\mathrm{m}]\!])$

Local phase:

Let $\mathcal{T}_{c,t} : \mathcal{R}^c \times \mathcal{R}^d \to \mathbb{F}^N, (f, g) \mapsto \psi(\langle f, \bar{g} \rangle_c)$ where $\bar{g}_i = \sum_{k=0}^{t-1} g_{t \cdot i + k}$
Let $\mathcal{S}_{c,t} : \mathcal{R}^c \times \mathcal{R}^{d \times d} \to \mathbb{F}^N, (f, g) \mapsto \psi(\langle f \otimes f, \hat{g} \rangle_{c \times c})$, where $\hat{g}_{i,j} = \sum_{k,l=0}^{t-1} g_{t \cdot i + k, t \cdot j + l}$
Return $\mathcal{T}_{c,t}(\vec{u}, [\![\vec{x}]\!]_{\mathbb{F}^N}), \mathcal{T}_{c,t}(\vec{u}, [\![\vec{y}]\!]_{\mathbb{F}^N}), \mathcal{S}_{c,t}(\bar{u}, [\![\vec{z}]\!]_{\mathbb{F}^N})$, where $\vec{x}, \vec{y}$ are interpreted in $\mathcal{R}^d$ and $\vec{z}$ is interpreted in $\mathcal{R}^{d \times d}$ (i.e. modulo F).

4 Application: Generation of Beaver Triples with $\Pi_{\mathbf{aSUV}}$

We will now demonstrate the application of our protocol Π_{aSUV} to the actively secure Beaver triple protocol Π_{Bt} by Boyle et al. [10] (Protocol 5). The main difference to their initial presentation is that we make the usage of aSUVs explicit in order to apply our improved protocol Π_{aSUV}. Additionally, we introduce a special purpose protocol Π^*_{aSUV} to avoid calls to **Input**, i.e., the dependency on VOLE, for Π_{Bt} itself. The protocol Π_{Bt} consists of two parts:

1. An *interactive phase* (Section 4.1) that generates three vectors of aSUVs

$$[\![\vec{x}]\!]_{\mathcal{U}_N} \otimes [\![\vec{y}]\!]_{\mathcal{U}_N} = [\![\vec{z}]\!]_{\mathcal{U}_{2N}} \quad \vec{x}, \vec{y} \xleftarrow{\$} \mathcal{U}_N^d, N = 2^n, \tag{3}$$

   where $\otimes$ denotes a polynomial tensor product, i.e., $\vec{z} \in (\mathbb{F}^{2N})^{d \times d}, \vec{z}_{i,j} = \vec{x}_i \cdot \vec{y}_j$ with multiplication in $\mathbb{F}[X]$.
2. A silent *local phase* (Section 4.2) to transform the private shares of the three aSUVs into private shares of N many Beaver triples, i.e., into $[\![a]\!]_{\mathbb{F}^N} \cdot [\![b]\!]_{\mathbb{F}^N} = [\![h]\!]_{\mathbb{F}^N}$, where $a, b \xleftarrow{\$} \mathbb{F}^N$ and $h = a \cdot b$ coefficient-wise in $\mathbb{F}$.

4.1 The Interactive Phase

The task of the interactive phase is to generate a random instance of Eq. (3), i.e., $2d$ *small* N-dimensional aSUVs sharing $\vec{x}, \vec{y}$, and d^2 *large* $2N$-dimensional aSUVs, with correlated positions and payloads, sharing the product $\vec{z}$.

Protocol 6 Payload correction for Π^*_{aSUV} (Alternative step 3 in Protocol 4)

$[\![\text{CW}^R]\!] = [\![u^R]\!]^{-1} \cdot [\![\text{m}]\!] \cdot [\![u^L]\!]$, $\text{CW}^R \leftarrow \textbf{Open}([\![\text{CW}^R]\!])$

If $\text{CW}^R = 0$, abort. Else return $[x]_{\mathbb{F}^M} = [t^L]_{\mathbb{F}^M}$ and $[x']_{\mathbb{F}^M} = \text{CW}^R \cdot [t^R]_{\mathbb{F}^M}$.

Small aSUV instances (generated with aSUV*): In Π_{Bt} we write **aSUV*** for a functionality that, given a depth n and secret-shared MAC $[\![\text{m}]\!]$, samples a uniformly random n-bit position and payload, calls **aSUV** and additionally returns the position and payload in secret-shared form. With **aSUV*** the generation of $[\![\vec{x}]\!]_{\mathbb{F}^N}$ and $[\![\vec{y}]\!]_{\mathbb{F}^N}$ is straightforward: Call d instances of **aSUV*** to get $[\![\vec{x}]\!]_{\mathbb{F}^N}$ with uniformly random positions $\|\vec{\alpha}\|_n$ and payloads $[\![\vec{A}]\!]$. Then call d instances of **aSUV*** to get $[\![\vec{y}]\!]_{\mathbb{F}^N}$ with uniformly random positions $\|\vec{\beta}\|_n$ and payloads $[\![\vec{B}]\!]$.

To actually sample the n-bit integer position, we follow Boyle et al. [10] and take preprocessed correlated randomness in the form of COT (Section 2.1). For the sampling of the payloads, our compact formulation with **aSUV*** enables a new optimization. Boyle et al.[10] propose to sample the random shares $[\![\vec{A}]\!], [\![\vec{B}]\!]$ with explicit calls to **Input** inside Π_{Bt}. Instead, we use a variation Π^*_{aSUV} of Π_{aSUV} that includes to uniformly sample a new payload, without further costs. Concretely we achieve even more, namely a more efficient payload correction inside Π_{aSUV} (Protocol 6): Instead of sampling a new position A, we propose to take over u^L, which is already uniformly random (Section 2.2). In other words, we set $\text{CW}^L = 1$, which removes all respective operations in Π_{aSUV}, and we set $\text{CW}^R = \frac{u^L \text{m}}{u^R}$. Since u^L, u^R are secret-shared, uniformly random and independent of each other by the means of the tree phase (Section 1.1), Theorem 2 already implies that Π^*_{aSUV} securely implements **aSUV***.

Large aSUV instances: For the product $[\![\vec{z}]\!]_{\mathcal{U}_N}$, the protocol Π_{Bt} calls d^2 instances of **aSUV** with dimension $2N$. Following the polynomial tensor product, the positions of $\vec{z}$ are given by a tensor sum

$$\vec{\gamma} = \vec{\alpha} \boxplus \vec{\beta} \in [0, 2N)^{d \times d} \text{ with } \vec{\gamma}_{i,j} = \vec{\alpha}_i + \vec{\beta}_j, \tag{4}$$

i.e, Π_{Bt} calls d^2 many integer additions $\|\vec{\gamma}_{i,j}\|_{n+1} = \|\vec{\alpha}_i\|_n + \|\vec{\beta}_j\|_n$ (Section 2.1). Similarly, the payloads of $\vec{z}$ are given by a tensor product

$$\vec{C} = \vec{A} \otimes \vec{B} \in \mathbb{F}^{d \times d} \text{ with } \vec{C}_{i,j} = \vec{A}_i \cdot \vec{B}_j, \tag{5}$$

which is in terms of d^2 many secure multiplications $[\![\vec{C}_{i,j}]\!] = [\![\vec{A}_i]\!] \cdot [\![\vec{B}_j]\!]$.

4.2 The Local Phase

The problem to be solved with the local phase is: Can an instance of Eq. (3) be locally transformed into N many Beaver triples? The answer is yes, under the security of the ring-LPN assumption [10]. The ring-LPN assumption is formulated

over a polynomial ring $\mathcal{R} = \mathbb{F}[X]/F$, where in our case F is a fully reducible polynomial of degree N such that there exists an isomorphism $\psi : \mathbb{F}[X] \simeq \mathbb{F}^N$. Let $\bar{x} \in \mathcal{R}^c$ be a vector of c many t-sparse polynomials. In other words, each coefficient of $\bar{x}^i$ can be written as a sum of t many scaled unit vectors of dimension N. Let $\mathcal{R}_t$ be the respective subset of $\mathcal{R}$, and let c, t, N depend on a security parameter λ_{LPN}. Then the ring-LPN assumption states that the distributions

$$\{(\bar{u}, \langle \bar{u}, \bar{x} \rangle_c) \mid \bar{u} \xleftarrow{\$} \mathcal{R}^c, \bar{x} \xleftarrow{\$} \mathcal{R}_t^c\}, \quad \{(\bar{u}, e) \mid \bar{u} \xleftarrow{\$} \mathcal{R}^c, e \xleftarrow{\$} \mathcal{R}\} \tag{6}$$

are computationally indistinguishable, where $\langle \cdot, \cdot \rangle_c$ denotes the c-dimensional scalar product, i.e., $\langle \bar{u}, \bar{x} \rangle_c = \sum_{i=0}^{c-1} \bar{u}_i \cdot \bar{x}_i$.

The strategy of Π_{Bt} is to aggregate $\vec{x} \in \mathcal{U}_M^d$, interpreted in $\mathcal{R}^d$, into a vector $\bar{x} \in \mathcal{R}^c$ of t-sparse polynomials and then to apply a transformation $\mathcal{T} : \bar{x} \mapsto \langle \bar{u}, \bar{x} \rangle_c \in \mathcal{R} \simeq_\psi \mathbb{F}^N$. Under the ring-LPN assumption, the resulting vector $a = \langle \bar{u}, \bar{x} \rangle_c$ is indistinguishable from uniformly random. Furthermore, $\mathcal{T}$ is linear. Hence by the linearity of $[\![\cdot]\!]$, the parties can locally transform the additively shared aSUVs $[\![\vec{x}]\!]_{\mathcal{U}_N}, [\![\vec{y}]\!]_{\mathcal{U}_N}$ into the secret-shared, uniformly random, factors $[\![a]\!]_{\mathbb{F}^N}, [\![b]\!]_{\mathbb{F}^N}$ of the output Beaver triples. Finally, the transformation $\mathcal{T}$ induces a two-dimensional transformation $\mathcal{S}$ that allows to locally compute the product $[\![h]\!]_{\mathbb{F}^N}$ from $[\![\vec{z}]\!]_{\mathcal{U}_N}$ (see Protocol 5).

4.3 Comments on Π_{Bt}

Regular variant: Boyle et al. [10] introduce a regular variant of Π_{Bt}, which reduces costs by reducing the dimension of the aSUVs. Hereby, the coefficients of $\bar{x}, \bar{y}$, which are in the form of t-sparse vectors, are replaced by regular t-sparse vectors, which consist of t many blocks with sparsity one. Then for each coefficient, instead of generating t aSUVs with dimension N, it is sufficient to generate t SUVs of smaller dimension $\frac{N}{t}$. Similarly, for the coefficients of $\bar{z}$ the dimension $\frac{2N}{t}$ is sufficient. For details, especially about the adaption of the ring-LPN assumption, we refer to Boyle et al. [10].

Security: For secure parameter choices $(N, d = c \cdot t, F)$ under the ring-LPN assumption, we refer to Boyle et al. [10], e.g., the parameters we use in our evaluation (Section 5.2). The two influences that a malicious party has inside **aSUV** translate to Π_{Bt} as follows:

- The selective failure attack can be addressed under the ring-LPN assumption with static leakage [10].
- A malicious party is allowed to choose its shares of the Beaver triples. However, since Beaver triples are additively secret-shared objects, this is uncritical for many MPC applications, e.g., in the sense of Section 2.1.

Table 1. Costs (per party) for the generation of SUVs and aSUVs depending on their dimension $M = 2^m$, the field size ν, and the security parameter λ of the tree phase. We give the number of AES calls, the estimated number of field multiplications, the number of consumed Beaver triples (Bt), the size of the consumed correlated randomness (CR) in bits and the communication in bits. The absolute numbers for the communication are in bytes and refer to different parameter triples (m, λ, ν). For the AES calls we assume that $86 \leq \nu \leq 128,\ \lambda < 128$.

	AES	Mult.	Bt	CR	communication	(15, 127, 124)	(18, 127, 124)	(17, 80, 80)
$\Pi_{\text{SUV}}^{\text{Boyle}}$	$4 \cdot 2^m$	$3 \cdot 2^m$	3	30ν	$m(2\lambda+2)+13\nu$	682	778	474
Π_{SUV}	$4 \cdot 2^m$	$3 \cdot 2^m$	4	24ν	$m(2\lambda+2)+10\nu$	635	731	444
$\Pi_{\text{aSUV}}^{\text{Boyle}}$	$5 \cdot 2^m$	$5 \cdot 2^m$	7	60ν	$m(2\lambda+2)+24\nu$	837	933	574
Π_{aSUV}	$4 \cdot 2^m$	$4 \cdot 2^m$	7	42ν	$m(2\lambda+2)+17\nu$	744	840	514

5 Evaluation

5.1 Evaluation of the SUV and aSUV Generation

In Table 1 we compare the costs of our protocols $\Pi_{(\text{a})\text{SUV}}$ (Protocol 4) with the costs of the protocols $\Pi_{(\text{a})\text{SUV}}^{\text{Boyle}}$ (Appendix A.1). At this point, note that an aSUV has the double size compared to an SUV. However, since the aSUV protocols are efficiently build on top of the SUV protocols, the additional costs for their generation are below a factor two. We now separately discuss the numbers for the computation, consumed correlated randomness and communication:

Computational Costs: For $\Pi_{\text{SUV}}^{\text{Boyle}}, \Pi_{\text{SUV}}$, and Π_{aSUV}, the tree phase is identical, with costs dominated by M many calls to a $\text{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ and M many calls to a $\text{PRG} : \{0,1\}^\lambda \to \mathbb{F}^2$ (Appendix A.1). Under the assumption that $\lambda < 127, \nu \leq 128$, both PRGs can be efficiently realized with two calls to the 128-bit AES block cipher [11]. Instead, the tree phase of $\Pi_{\text{aSUV}}^{\text{Boyle}}$ extends the second PRG to outputs in $\mathbb{F}^3$, for which, if $3\nu < 256$, we count one additional AES call. Due to our symmetric verification procedure (Section 3.1), Π_{aSUV} avoids the extension to $\mathbb{F}^3$, which gives an advantage of 20% AES calls.

As a measure for the computational costs of the field phases we take the number of field multiplications given by M dimensional scalar-vector-multiplications. Concretely, this refers to the payload correction with CW^L and to masking t^L, t^R with the randomness $r \in \mathbb{F}^M$. On top of that, the aSUV protocols take M multiplications with CW^R to correct the payload of the MAC component. Furthermore, $\Pi_{\text{aSUV}}^{\text{Boyle}}$ takes M more field multiplications to mask the additional component $t^{L'}$ with r. Our protocol Π_{aSUV} avoids the costs for the additional component, which gives an advantage of 20%.

Correlated Randomness: The protocols $\Pi_{\text{SUV}}^{\text{Boyle}}, \Pi_{\text{aSUV}}^{\text{Boyle}}$ consume correlated randomness in the form of Beaver triples, for secure multiplications and inver-

sions, and in the form of VOLE correlations, to support calls of **Input** (Protocol 1). Instead, since our protocols $\Pi_{\text{(a)SUV}}$ avoid calls to **Input**, we avoid the dependency on VOLE. In Table 1, we give the size of consumed correlated randomness per party, counting 6 field elements for a Beaver triple and 1.5 field elements for one entry of a VOLE correlation (ignoring the constant scalar m_σ). By avoiding calls to **Input**, we can reduce this size by 20% for the SUV generation and 30% for the aSUV generation.

Note that the additional Beaver triple consumed by Π_{SUV}, compared to $\Pi_{\text{SUV}}^{\text{Boyle}}$, is the price we pay for making the verification procedure symmetric, which in turn is necessary to avoid calls to **Input** in $\Pi_{\text{(a)SUV}}$ (Section 3.1).

Communication Costs: All SUV and aSUV protocols have the communication costs of the tree phase in common, i.e., $m(2\lambda + 2)$ bits per party [10]. On top of that, the field phase has linear communication costs in ν, referring to the secure arithmetic, Protocol 1, and our adapted multiplication, Protocol 3.[4] By avoiding calls to **Input**, we reduce the communication of the field phase (Section 3.1). The improvement for the whole SUV and aSUVs protocols depends on the relation between m, ν, λ. In Table 1, we provide absolute numbers for different parameter choices that are realistic for Π_{Bt} in the regular variant. In general, our improvements for the communication costs are more effective for the aSUV generation (10% in Table 1) then for the SUV generation (6% in Table 1), since for the SUV generation we avoid 8 calls to **Input**, whereas for the aSUV generation we avoid 12 calls.

5.2 Evaluation of the Beaver Triple Generation

We now evaluate the effect of our optimized protocols Π_{aSUV} on the interactive phase of the Beaver triple protocol Π_{Bt}. According to Section 4.1, for the small aSUV instances we on top consider the optimizations implied by Π_{aSUV}^*.[5] The interactive phase of Π_{Bt} consists of:

- $2d$ calls of Π_{aSUV}^* with *small* regular depth $m = \lceil \log(\frac{N}{t}) \rceil$.[6]
- d^2 calls of Π_{aSUV} with *large* depth $m + 1$.
- d^2 secure integer additions (Eq. (5)) and secure multiplications (Eq. (4)).

In Table 2 we give absolute costs for the generation of 2^{20} Beaver triples with Π_{Bt} (interactive phase). We take the parameter choices over from Boyle et al. [10], which represent secure parameters for the ring-LPN assumption[7].

[4] We follow Bolye et al. [10] and do not count the costs to sample r.

[5] We do not separately evaluate Π_{aSUV}^* in Section 5.1 since the costs are dominated by the large aSUVs with Π_{aSUV}. Each call of Π_{aSUV}^* gives an advantage of M many saved field multiplications, two saved Beaver triples and 5ν bits less of communication, compared to the straightforward realization of $\mathbf{SUV}^*$ with Π_{aSUV}.

[6] i.e., for the regular variant $M = 2^m$ is $\frac{N}{t}$ rounded to the next power of two as required by the tree phase.

[7] The parameters refer to different choices for $d = c \cdot t$, affecting the costs of the interactive- and local phase of Π_{Bt}. See [10] for a discussion.

Table 2. Costs per party for the generation of 2^{20} Beaver triples over a 124-bit field with the regular variant of Π_{Bt} (interactive phase), instantiated with either $\Pi_{\mathrm{aSUV}}^{\mathrm{Boyle}}$ or our protocols $\Pi_{\mathrm{aSUV}}, \Pi_{\mathrm{aSUV}}^{*}$. We set $\lambda = 127$, while λ_{LPN} is the security level of the ring-LPN assumption. "Storage" refers to the accumulated amount of consumed correlated randomness (Beaver triples, AND triples, COT, VOLE).

			AES in billion		Mul. in billion		storage in MB		communication in MB	
λ_{LPN}	d	m	Boyle	Our work	Boyle	Our work	Boyle	Our work	Boyle [10]	Our work
80	96	15	3.05	2.44	3.05	2.43	22.07	19.40	8.77	8.16
80	40	17	2.15	1.72	2.15	1.71	4.21	3.72	1.68	1.57
80	32	18	2.77	2.21	2.77	2.20	2.81	2.50	1.13	1.05
128	152	14	3.81	3.05	3.81	3.04	52.85	46.25	20.97	19.46
128	64	16	2.73	2.18	2.73	2.17	10.27	9.06	4.09	3.80
128	40	18	4.30	3.44	4.30	3.42	4.37	3.88	1.75	1.63
Improvement				20.0%		20.4%		11.7%		7.0%

The costs advantage of reduced AES calls and field multiplications (20%) inside Π_{aSUV} directly translates to an advantage inside the interactive phase of Π_{Bt}, i.e., hundred millions in absolute numbers (between 400 and 800 per generated Beaver triple). On top of that, using Π_{aSUV}^{*} saves up to 20 million multiplications.

Altogether Π_{Bt} consumes correlated randomness in the form of Beaver triples, VOLE, AND triples and COT (Section 2.1). We stress that we avoid the dependency on VOLE, by the means of Π_{aSUV} and Π_{aSUV}^{*}. By avoiding the dependency on VOLE, we reduce the amount of consumed correlated randomness by about 11%, which are a few MB in absolute numbers.

For our absolute communication cost numbers, we set $\lambda = 127$, i.e., the maximum security parameter to instantiate the PRG in the tree phase with two AES calls. We stress that the numbers from Boyle et al. [10] include the costs to generate AND triples, hence we make the same assumption for our numbers. In this scenario, we reduce the communication costs by about 7%. For a more comprehensive evaluation, we provide numbers excluding the AND triple generation, and numbers including more of the preprocessing, in Appendix B.

Asymptotic relation to the local phase: Using the regular variant to generate N Beaver triples, the interactive phase of Π_{Bt} has computational costs of $\mathcal{O}(c^2tN)$ AES calls and field multiplications (dominated by the *large* aSUVs), whereas the non-interactive local phase is dominated by c^2 many multiplications in $\mathcal{R}$. For appropriate choices of $\mathcal{R}$, these multiplications can implemented with $\mathcal{O}(N \log N)$ field multiplications, e.g., using the Number Theoretic Transform [10]. Therefore, for the whole protocol Π_{Bt}, our improvements asymptotically scale with $\mathcal{O}(c^2N)$, depending on the relation between t and $\log(N)$ according to the ring-LPN assumption. For example, the numbers in Table 2 refer to $\log(N) = 20$ and different $4 \leq t \leq 76$. In this sense, we conjecture our computational improvements, on the level of the aSUV generation, to be effective for the whole protocol Π_{Bt}.

References

1. Damiano Abram and Peter Scholl. Low-communication multiparty triple generation for spdz from ring-lpn. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography – PKC 2022*, pages 221–251, Cham, 2022. Springer International Publishing.
2. Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 223–254, Cham, 2017. Springer International Publishing.
3. Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using topgear in overdrive: A more efficient zkpok for spdz. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography – SAC 2019*, pages 274–302, Cham, 2020. Springer International Publishing.
4. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 169–188, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
5. Maxime Bombar, Geoffroy Couteau, Alain Couvreur, and Clément Ducros. Correlated pseudorandomness from the hardness of quasi-abelian decoding. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 567–601, Cham, 2023. Springer Nature Switzerland.
6. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector ole. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 896–912, New York, NY, USA, 2018. Association for Computing Machinery.
7. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part II*, page 603–633, Berlin, Heidelberg, 2022. Springer-Verlag.
8. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round ot extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 291–308, New York, NY, USA, 2019. Association for Computing Machinery.
9. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. In *CRYPTO*, volume 11694 of *LNCS*, pages 489–518. Springer, 2019.
10. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient Pseudorandom Correlation Generators from Ring-LPN. In *CRYPTO*, volume 12171 of *LNCS*, pages 387–416. Springer, 2020.
11. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function Secret Sharing: Improvements and Extensions. In *CCS*, pages 1292–1303. ACM, 2016.
12. R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.
13. Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential benchmarking based on multiparty computation. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security*, pages 169–187, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

14. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure mpc for dishonest majority – or: Breaking the spdz limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security – ESORICS 2013*, pages 1–18, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
15. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417 of *LNCS*, pages 643–662. Springer, 2012.
16. Jack Doerner and Abhi Shelat. Scaling ORAM for Secure Computation. In *CCS*, pages 523–535. ACM, 2017.
17. Daniel Escudero, Chaoping Xing, and Chen Yuan. More efficient dishonest majority secure computation over $\mathbb{Z}_{2^k}$ via galois rings. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 383–412, Cham, 2022. Springer Nature Switzerland.
18. Niv Gilboa and Yuval Ishai. Distributed Point Functions and Their Applications. In *EUROCRYPT*, volume 8441 of *LNCS*, pages 640–658. Springer, 2014.
19. Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round mpc combining bmr and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 598–628, Cham, 2017. Springer International Publishing.
20. Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1575–1590, New York, NY, USA, 2020. Association for Computing Machinery.
21. Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: Faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 830–842, New York, NY, USA, 2016. Association for Computing Machinery.
22. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ Great Again. In *EUROCRYPT*, volume 10822 of *LNCS*, pages 158–189. Springer, 2018.
23. Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, editors, *Cryptology and Network Security*, pages 1–20, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
24. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 681–700, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
25. Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. Overdrive2k: Efficient secure mpc over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, pages 254–283, Cham, 2020. Springer International Publishing.
26. Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for spdz. *Journal of Cryptology*, 2021.
27. Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-ole: Improved constructions and implementation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1055–1072, New York, NY, USA, 2019. Association for Computing Machinery.

28. Hasler Sebastin, Reisert Pascal, Rivinius Marc, and Küsters Ralf. Multipars: Reduced-communication mpc over z2k. In *Proceedings on Privacy Enhancing Technologies*, 2024.
29. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast Extension for Correlated OT with Small Communication. In *CCS*, pages 1607–1626. ACM, 2020.

A Details on the SUV and aSUV Generation

A.1 Presentation of $\Pi^{\text{Boyle}}_{\text{SUV}}$, $\Pi^{\text{Boyle}}_{\text{aSUV}}$

We present $\Pi^{\text{Boyle}}_{\text{SUV}}$ and $\Pi^{\text{Boyle}}_{\text{aSUV}}$ in Protocol 7. To be consistent with our work, we slightly deviate from the notation by Boyle et al. [10], e.g., to avoid situations where for an additive secret-shared value $[x]$ it holds $x = x_0 - x_1$ instead of $x = x_0 + x_1$. Furthermore to get a better framework to describe our modifications, we split the protocol into a tree phase and field phase (verification and payload correction). Hereby, we shift the computation of $[\![\text{CW}^L]\!] = [\![u^L]\!]^{-1} \cdot [\![A]\!]$ from the verification step into the payload correction step.

We remark that Boyle et al. [10] make in fact two proposals how to extend $\Pi^{\text{Boyle}}_{\text{SUV}}$ for two-dimensional SUVs, which are a special cases of an extension to multi-dimensional SUVs with coefficients in $\mathbb{F}^l$ instead of $\mathbb{F}$ (cf. Remark 5.2 [10]). For our work, we built the aSUV generation on the second one, which they consider to be more efficient since it avoids to work over an extension field of $\mathbb{F}$. To make our notation consistent with the aSUV generation, we write "$L; L'; R$" instead of "$L, 0; L, 1; R$", for the three internal SUV components.

Intuition of the tree phase: For the tree phase, the parties start with private seeds $s^{0,0}_\sigma \in \{0,1\}^\lambda$ and run through a binary tree $s^{i,j}_\sigma, 0 \leq i \leq m+1, 0 \leq j < 2^i$ by using a length doubling PRG to generate $s^{i+1,2j}_\sigma, s^{i+1,2j+1}_\sigma$ from $s^{i,j}_\sigma$. After each tree level $i \leq m$, the parties run an interactive correction to maintain a tree invariant: $s^{i,j}_0 = s^{i,j}_1$ if and only if $j \neq \alpha_{m-1} \cdots \alpha_i$, where $\alpha_{m-1} \cdots \alpha_0$ is the binary representation of the position α. To support the correction, the PRG outputs two additional bits, i.e. it is of the form $\text{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$. After the last tree round, they convert their node values $s^{m+1,2k}_\sigma$ into pseudo-random, independent left and right field elements $s^{L,k}_\sigma, s^{R,k}_\sigma \in \mathbb{F}$, using a public conversion function, e.g., the last round of PRGs is replaced by a PRG $\{0,1\}^\lambda \to \mathbb{F}^2$. Due to the tree invariant and the randomization with the PRGs, the 2^m dimensional secret-shared vectors $s^L_0 - s^L_1, s^R_0 - s^R_1$ are SUVs with random payloads and the same position α. We then define $t^L_\sigma = (-1)^\sigma s^L_\sigma, t^R_\sigma = (-1)^\sigma s^R_\sigma$. However, while a malicious party can not deviate from the secret-shared position $\|\alpha\|_m$, it can deviate from the private PRG outputs to break the unit-vector structure of t^L, t^R. Hereby, the tree phase is constructed in such a way, that a malicious party can not learn anything about α and the payloads u^L, u^R of t^L, t^R.

To additionally provide $t^{L'}$, for $\Pi^{\text{Boyle}}_{\text{aSUV}}$ the PRG in the last tree round is extended to a $\text{PRG} : \{0,1\}^\lambda \to \mathbb{F}^3$ that outputs three independent field elements $s^{L,k}_\sigma, s^{L',k}_\sigma, s^{R,k}_\sigma$.

Protocol 7 Protocols $\Pi^{\text{Boyle}}_{\text{SUV}}, \Pi^{\text{Boyle}}_{\text{aSUV}}$ ([10] cf. $\Pi_{\text{c-SUV}}$, Remark 5.2)

Blue statements refer only to $\Pi^{\text{Boyle}}_{\text{aSUV}}$. All interactive parts are framed.
INPUT: Secret shared position $\|\alpha\|_m$, payload $[\![A]\!]$, and MAC key $[\![\mathrm{m}]\!]$.
OUTPUT: $[x]_{\mathcal{U}_M}$, $[x']_{\mathcal{U}_M}$ with position α and payload A, $\mathrm{m} \cdot A$ (where $M = 2^m$).

1. TREE PHASE

   Process $\|\alpha\|_m$ to get $t^L_\sigma, t^{L'}_\sigma, t^R_\sigma \in \mathbb{F}^M$ (Appendix A.1)

2. VERIFICATION

   $u^L_\sigma = \sum_{j=0}^{M-1} t^{L,j}_\sigma$, $u^{L'}_\sigma = \sum_{j=0}^{M-1} t^{L',j}_\sigma$
   $u^R_\sigma = \sum_{j=0}^{M-1} t^{R,j}_\sigma$

   For both P_σ let $[\![u^L_\sigma]\!] \leftarrow \mathbf{Input}(u^L_\sigma, P_\sigma)$, $[\![u^{L'}_\sigma]\!] \leftarrow \mathbf{Input}(u^{L'}_\sigma, P_\sigma)$
   For both P_σ let $[\![u^R_\sigma]\!] \leftarrow \mathbf{Input}(u^R_\sigma, P_\sigma)$

   $[\![u^L]\!] = [\![u^L_0]\!] + [\![u^L_1]\!]$, $[\![u^{L'}]\!] = [\![u^{L'}_0]\!] + [\![u^{L'}_1]\!]$, and $[\![u^R]\!] = [\![u^R_0]\!] + [\![u^R_1]\!]$

   $\mathrm{CW}^R \leftarrow \mathbf{Open}([\![u^R]\!])$
   Sample public randomness $(r^0, \ldots, r^{M-1}) \xleftarrow{\$} \mathbb{F}^M$

   $v^L_\sigma = \sum_{j=0}^{M-1} r^j \cdot t^{L,j}_\sigma$, $v^{L'}_\sigma = \sum_{j=0}^{M-1} r^j \cdot t^{L',j}_\sigma$
   $w^R_\sigma = \frac{1}{\mathrm{CW}^R} \sum_{j=0}^{M-1} r^j \cdot t^{R,j}_\sigma$

   For both P_σ let $[\![v^L_\sigma]\!] \leftarrow \mathbf{Input}(v^L_\sigma, P_\sigma)$, $[\![v^{L'}_\sigma]\!] \leftarrow \mathbf{Input}(v^{L'}_\sigma, P_\sigma)$
   For both P_σ let $[\![w^R_\sigma]\!] \leftarrow \mathbf{Input}(w^R_\sigma, P_\sigma)$

   $[\![v^L]\!] = [\![v^L_0]\!] + [\![v^L_1]\!]$, $[\![v^{L'}]\!] = [\![v^{L'}_0]\!] + [\![v^{L'}_1]\!]$, and $[\![w^R]\!] = [\![w^R_0]\!] + [\![w^R_1]\!]$

   $[\![Z]\!] = [\![u^L]\!] \cdot [\![w^R]\!] - [\![v^L]\!]$, $[\![Y]\!] = [\![u^{L'}]\!] \cdot [\![w^R]\!] - [\![v^{L'}]\!]$
   $Z \leftarrow \mathbf{Open}([\![Z]\!])$, $Y \leftarrow \mathbf{Open}([\![Y]\!])$

   If $Z \neq 0$ or $Y \neq 0$, abort

5. PAYLOAD CORRECTION

   $[\![\mathrm{CW}^L]\!] = [\![u^L]\!]^{-1} \cdot [\![A]\!]$, $[\![\mathrm{CW}^{L'}]\!] = [\![u^{L'}]\!]^{-1} \cdot [\![\mathrm{m}]\!] \cdot [\![A]\!]$
   $\mathrm{CW}^L \leftarrow \mathbf{Open}([\![\mathrm{CW}^L]\!])$, $\mathrm{CW}^{L'} \leftarrow \mathbf{Open}([\![\mathrm{CW}^{L'}]\!])$

   If $\mathrm{CW}^L = 0$ or $\mathrm{CW}^{L'} = 0$, abort
   Return $[x]_{\mathbb{F}^M} = \mathrm{CW}^L \cdot [t^L]_{\mathbb{F}^M}$ and $[x']_{\mathbb{F}^M} = \mathrm{CW}^{L'} \cdot [t^{L'}]_{\mathbb{F}^M}$

A.2 Proof of Theorem 2: Security of $\Pi_{\text{(a)SUV}}$

For the proof of Theorem 2, we follow the two reduction steps as outlined in Section 3.3.

First reduction step:

Theorem 3. *Protocol Π^{sym}_{SUV} (with symmetric verification phase, Protocol 8) securely implements the functionality **SUV** with security against malicious adversaries in the UC-model. Similarly, Π^{sym}_{aSUV} securely implements **aSUV**.*

To prove Theorem 3, we go through all modifications between Protocol 7 and Protocol 8 and show that they neither affect the expected abortion behavior and output structure, nor leak any additional information.

- For $\Pi^{\text{sym}}_{\text{aSUV}}$, we remove the L' component in the tree phase and verification. This is secure, since it only removes statements.
- Similarly, it is secure to remove the statement $\text{CW}^R \leftarrow \mathbf{Open}([\![u^R]\!])$.
- The value w^R is replaced by v^R. This is secure, since it is a local modification.
- The definition of Z is modified from $Z = u^L w^R - v^L$ to $Z = u^L v^R - u^R v^L$. By definition of v^R, w^R, this can be seen as scaling the old value of Z with u^R (CW^R in $\Pi^{\text{Boyle}}_{\text{SUV}}$ Protocol 7). Scaling however, does not affect if $Z = 0$ or $Z \neq 0$ as long as $u^R \neq 0$. In Protocol 7 the case $u^R = 0$ is excluded since the parties fail to locally compute $\frac{1}{\text{CW}^R}$. Instead in Protocol 8, the case $u^R = 0$ causes an abortion since then the secret-shared inversion $[\![u^L]\!]^{-1}$ in the payload correction step fails. Hence with respect to Z, Protocol 7 aborts if and only if Protocol 8 aborts.
- The payload correction phases differs only for the aSUV generation. For $\Pi^{\text{sym}}_{\text{aSUV}}$, the output for the MAC component is $[x'] = \frac{Am}{u^R} \cdot [t^R]$. Since $\Pi^{\text{step}}_{\text{aSUV}}$ is symmetric with respect to the L - and R component this step is secure, if the respective correction $[x] = \frac{A}{u^L} \cdot [t^L]$ of the left component is secure: The only difference is the additional term m, which is analogously processed in $\Pi^{\text{Boyle}}_{\text{aSUV}}$. Hence, the security and correctness for $[x']$ follows, since the L components are identical for $\Pi^{\text{Boyle}}_{\text{SUV}}$ and $\Pi^{\text{sym}}_{\text{SUV}}$ (with the only difference in the definition of Z, which is independent of the output).

Second reduction step:

We now reduce the security of Protocol 4 to Theorem 3. The computation of $[\![Z]\!]$ in Protocol 8 requires two Beaver multiplications $[\![Z^L]\!] := [\![u^L]\!] \cdot [\![v^R]\!]$ and $[\![Z^R]\!] := [\![u^R]\!] \cdot [\![v^L]\!]$. For this, let $[\![a^L]\!], [\![b^L]\!], [\![c^L]\!]$ and $[\![a^R]\!], [\![b^R]\!], [\![c^R]\!]$ be two Beaver triples. Applying Eq. (1) to compute $[\![Z^L]\!]$ means to compute

$$\epsilon^L = \mathbf{Open}([\![u^L]\!] - [\![a^L]\!]), \quad \delta^L = \mathbf{Open}([\![v^R]\!] - [\![b^L]\!])$$
$$[\![Z^L]\!] = \delta^L \cdot [\![u^L]\!] + \epsilon^L \cdot [\![v^R]\!] + [\![c^L]\!] - \epsilon^L \cdot \delta^L.$$

The computation of $[\![Z^R]\!]$ is analogue. Then, we point out the following:

Protocol 8 Protocols $\Pi^{\text{sym}}_{\text{SUV}}, \Pi^{\text{sym}}_{\text{aSUV}}$

Blue statements refer only to $\Pi^{\text{sym}}_{\text{aSUV}}$. All interactive parts are framed.
INPUT: Secret shared position $\|\alpha\|_m$, payload $[\![A]\!]$, and MAC key $[\![\mathrm{m}]\!]$.
OUTPUT: $[x]_{\mathcal{U}_M}$, $[x']_{\mathcal{U}_M}$ with position α and payload A, $\mathrm{m} \cdot A$ (where $M = 2^m$).

1. TREE PHASE

   Process $\|\alpha\|_m$ to get $t^L_\sigma, t^R_\sigma \in \mathbb{F}^M$ (Appendix A.1)

2. VERIFICATION

   $u^L_\sigma = \sum_{j=0}^{M-1} t^{L,j}_\sigma$
   $u^R_\sigma = \sum_{j=0}^{M-1} t^{R,j}_\sigma$

   For both P_σ let $[\![u^L_\sigma]\!] \leftarrow \mathbf{Input}(u^L_\sigma, P_\sigma)$
   For both P_σ let $[\![u^R_\sigma]\!] \leftarrow \mathbf{Input}(u^R_\sigma, P_\sigma)$

   $[\![u^L]\!] = [\![u^L_0]\!] + [\![u^L_1]\!]$ and $[\![u^R]\!] = [\![u^R_0]\!] + [\![u^R_1]\!]$

   Sample public randomness $(r^0, \ldots, r^{M-1}) \xleftarrow{\$} \mathbb{F}^M$

   $v^L_\sigma = \sum_{j=0}^{M-1} r^j \cdot t^{L,j}_\sigma$
   $v^R_\sigma = \sum_{j=0}^{M-1} r^j \cdot t^{R,j}_\sigma$

   For both P_σ let $[\![v^L_\sigma]\!] \leftarrow \mathbf{Input}(v^L_\sigma, P_\sigma)$
   For both P_σ let $[\![v^R_\sigma]\!] \leftarrow \mathbf{Input}(v^R_\sigma, P_\sigma)$

   $[\![v^L]\!] = [\![v^L_0]\!] + [\![v^L_1]\!]$ and $[\![v^R]\!] = [\![v^R_0]\!] + [\![v^R_1]\!]$

   $[\![Z]\!] = [\![u^L]\!] \cdot [\![v^R]\!] - [\![u^R]\!] \cdot [\![v^L]\!]$
   $Z \leftarrow \mathbf{Open}([\![Z]\!])$

   If $Z \neq 0$, abort

3. PAYLOAD CORRECTION

   $[\![\mathrm{CW}^L]\!] = [\![u^L]\!]^{-1} \cdot [\![A]\!]$, $[\![\mathrm{CW}^R]\!] = [\![u^R]\!]^{-1} \cdot [\![\mathrm{m}]\!] \cdot [\![A]\!]$
   $\mathrm{CW}^L \leftarrow \mathbf{Open}([\![\mathrm{CW}^L]\!])$, $\mathrm{CW}^R \leftarrow \mathbf{Open}([\![\mathrm{CW}^R]\!])$

   If $\mathrm{CW}^L = 0$ or $\mathrm{CW}^R = 0$, abort
   Return $[x]_{\mathbb{F}M} = \mathrm{CW}^L \cdot [t^L]_{\mathbb{F}M}$ and $[x']_{\mathbb{F}M} = \mathrm{CW}^R \cdot [t^R]_{\mathbb{F}M}$

– Since ϵ^L is masked with the factor a^L of a Beaver triple, no information is leaked to the parties. We point out that the computation of ϵ^L can be done directly after $[\![u^L]\!]$ is created, i.e., several step earlier in Protocol 8: Since a^L is only used for this purpose, the order of operations is not relevant. Similarly, the values $\epsilon^R, \delta^L, \delta^R$ can be computed directly after $[\![v^L]\!], [\![v^L]\!], [\![v^R]\!]$ are created.
– When applying the reordering as described in the previous point, the computations

$$\begin{aligned} [\![u_0^L]\!] &\leftarrow \mathbf{Input}(u_0^L, P_0), \quad [\![u_1^L]\!] \leftarrow \mathbf{Input}(u_1^L, P_1) \\ [\![u^L]\!] &= [\![u_0^L]\!] + [\![u_1^L]\!], \quad \epsilon^L \leftarrow \mathbf{Open}([\![u^L]\!] - [\![a^L]\!]) \end{aligned} \tag{7}$$

can be seen as one statement, which in the honest case can be simplified to:

$$\begin{aligned} \text{The parties exchange } u_\sigma^L - a_\sigma^L \text{ to compute } \epsilon^L &= (u_0^L - a_0^L) + (u_1^L - a_1^L) \\ [\![u^L]\!] &= [\![a^L]\!] + \epsilon^L. \end{aligned} \tag{8}$$

This step shows how to remove the calls of **Input** for the value $[\![u^L]\!]$. The same argument can be applied to the computation of $u^R, v^L, v^R, \delta^L, \epsilon^R, \delta^R$, which finally gives Protocol 4.

It remains to show that the transition from Eq. (7) to Eq. (8) is secure in the malicious case. In Eq. (7), the only thing that a malicious party, say P_0, can achieve is to add some linear error Δ on u^L by calling $[\![u_0^L + \Delta]\!] = \mathbf{Input}(u_0^L + \Delta, P_0)$. All other operations are secured by $[\![\cdot]\!]$. In Eq. (8) a malicious party P_0 can only run attacks of the form $\epsilon_0^L = u_0^L - a_0^L + \Delta$, where it is possible to base Δ on $u_1^L - a_1^L$. However the erroneous term can be written as $(u_0^L + \Delta) - a_0^L$ and for any malicious P_0 the term $u_1^L - a_1^L$ looks uniformly random since a_1^L is uniformly random. Together this implies: The only options that an attacker has to deviate in the case of Eq. (8) are equivalent to options it already has in case of Eq. (7). Hence, it is secure to replace Eq. (7) by Eq. (8), and the proof of Theorem 2 is complete.

A.3 Selective Failure Attack

We now give a selective failure attack on Π_{aSUV} that allows a malicious party to guess if the secret position α is in a certain subset $J \subset [0, M)$ or not. This is equivalent to a malicious party sending the guess function $P : [0, M) \to \{0, 1\}, P(\alpha) = 1 \Leftrightarrow \alpha \in J$ to **aSUV**. We present the attack for Π_{aSUV} – as expected by Theorem 1, similar attacks apply to the other protocols $\Pi_{\text{SUV}}, \Pi_{\text{(a)SUV}}^{\text{Boyle}}$. Hereby, the SUV generation can be seen a special case of the aSUV generation. And due to the symmetry of our verification phase, achieved through scaling Z (Section 3.3), the presentation of the attacker is more instructive for Π_{aSUV}.

Let $I \cup J = [0, M)$ be a disjoint decomposition. Let $t_I^L = (t^{L,i})_{i \in I}$ and similar for t^R, J. We then use the short notation $u_I^L, u_I^R, v_I^L, v_I^R, u_J^L, u_J^R, v_J^L, v_J^R$ for the respective partial sums. Let $Z_I = u_I^L v_I^R - u_I^R v_I^L, Z_J = u_J^L v_J^R - u_J^R v_J^L$ be the Z values defined for these smaller vectors. We now show that, under the assumption that $\alpha \in J$, it is possible to deviate from the values t_I^L, t_I^R such that it still holds $Z = 0$. On the other hand it will hold $Z \neq 0$ if the assumption $\alpha \in J$ was wrong in which case the attacker risks an abortion. However, if the protocol passes the verification with $Z = 0$, an attacker can conclude that $\alpha \in J$.

Given the decomposition $I \cup J = [0, M)$ it holds

$$\begin{aligned} Z &= (u_I^L + u_J^L)(v_I^R + v_J^R) - (u_I^R + u_J^R)(v_I^L + v_J^L) \\ &= Z_I + Z_J + u_I^L v_J^R + u_J^L v_I^R - u_I^R v_J^L - u_J^R v_I^L \\ &= Z_I + Z_J + \sum_{j \in J} t^{L,j} \sum_{i \in I} (r^i - r^j) t^{R,i} + \sum_{j \in J} t^{R,j} \sum_{i \in I} (r^j - r^i) t^{L,i}. \end{aligned}$$

Under the assumption that $\alpha \in J$, an attacker knows that $t^{L,i}, t^{R,i} = 0$ for all $i \in I$, i.e., the attacker can fully control the additively shared values $t_I^{L,i}, t_I^{R,i}, i \in I$. For the selective failure attack, it can try replace them by some malicious values $\bar{t}_I^{L,i}, \bar{t}_I^{R,i}$, such that still $Z = 0$. A key observation is to set $\bar{t}_I^i := \bar{t}_I^{L,i} = \bar{t}_I^{R,i}$, since then the malicious version $\bar{Z}_I = \bar{u}_I^L \bar{v}_I^R - \bar{u}_I^R \bar{v}_I^L$ of Z_I is zero and cancels out. If the attacker is consistent for t_J^L, t_J^R, which are unit vectors, it holds as well $Z_J = 0$. Then, for the erroneous value $\bar{Z}$ it follows

$$\bar{Z} = \sum_{j \in J} (t^{L,j} - t^{R,j}) \sum_{i \in I} (r^i - r^j) \bar{t}_I^i.$$

Hereby, the attacker can not control $t_J^{L,j}, t_J^{R,j}$ since they are independent and secret-shared, i.e., since they depend on $t_{1-\sigma}^{L,\alpha}, t_{1-\sigma}^{R,\alpha}$. But it has control over $\bar{t}_I^i$ and can use this to cancel out all sums $\sum_{i \in I} (r^i - r^j) \bar{t}^i$ which already implies $\bar{Z} = 0$ as desired. Concretely, an attacker can achieve this by solving a linear equation system $\Omega \cdot \bar{t}_I = 0$, where the entries of $\Omega \in \mathbb{F}^{|J| \times |I|}$ are given by $\Omega_{j,i} = r^i - r^j$.

A careful reader might point out two issues:

1. The presented attack might harm the correctness of Π_{aSUV} since the malicious values $\bar{t}_I^L, \bar{t}_I^R$ result in erroneous terms $\bar{u}^L, \bar{u}^R$ for u^L, u^R. Consequently the payload correction actually corrects towards malicious payloads $\bar{A}, \bar{A}'$ instead of A, A'. How is this in accordance with the functionality **aSUV**?
2. On the one hand, inside Π_{aSUV} the value r is sampled after the parties compute $[\![u^L]\!], [\![u^R]\!]$, which is binding. On the other hand, for the attack the terms $\bar{t}_I^L, \bar{t}_I^R$, and hence the malicious terms $\bar{u}^L, \bar{u}^R$, depend on r. Is the selective failure attack indeed possible?

To address both issues, we write $E := \sum_{i \in I} \bar{t}_I^i$, for the linear error term that the attacker adds to u^L, u^R when deviating from $t_I^i = 0$. We then distinguish between $|J| = 1$ and $|J| > 1$.

- If $|J| > 1$ we will show that the equation system $\Omega \cdot \bar{t}_I = 0$ already implies $E = 0$. This already solves the second issue, since $E = 0$ is independent of r, and the first issues, since $E = 0$ means that $u^R = \bar{u}^R, u^R = \bar{u}^R$ are correct. To prove $E = 0$, consider two different indices $j_0, j_1 \in J$. From $\Omega \cdot \bar{t}_I = 0$, it follows that $\sum_{i \in I}(r^i - r^{j_0})\bar{t}_I^i = 0$ and $\sum_{i \in I}(r^i - r^{j_1})\bar{t}_I^i = 0$. Together, this implies $(r^{j_0} - r^{j_1}) \sum_{i \in I} \bar{t}_I^i = (r^{j_0} - r^{j_1})E = 0$. Since $r \xleftarrow{\$} \mathbb{F}^M$ is uniformly random, we can assume that there exists j_0, j_1 with $r^{j_0} \neq r^{j_1}$ (for ν, M large). Hence, it follows $E = 0$.
- The case $|J| = 1$ is a special case for which $E \neq 0$ is possible. To address the second issue, we point out that any E is possible, since for any particular solution $\hat{t}_I$ of $\Omega \cdot \hat{t}_I = 0$, the solution $\frac{E}{\sum_{i \in I} \hat{t}_I} \cdot \hat{t}_I$ is in accordance with E. The difference to the case $|J| > 1$, is that at least one solution $\hat{t}^I$ with $\sum_{i \in I} \hat{t}_I \neq 0$ exists (except the corner case $r^i = r^\alpha$ for all $i \in I$). For the first issue, note that by the attack, the outputs $[\bar{x}]_{\mathcal{U}_M}, [\bar{x}']_{\mathcal{U}_M}$ are corrupted with erroneous payloads $\bar{A} = A + E, \bar{A}' = A' + E$. For $|J| = 1$, the selective failure attacker means nothing else that a malicious party, say P_0, learns α. Hence P_0 can locally correct the payloads to A, A': It just needs to replace $t_0^{L,\alpha}, t_1^{R,\alpha}$ by $t_0^{L,\alpha} - E, t_1^{R,\alpha} - E$. With other words, the correctness of the output depends only on local manipulations by P_0. However, the UC-security guarantees are indifferent about the output for the malicious party. Hence the selective failure attack is in accordance with the ideal functionality **aSUV**.

B Communication Costs of Π_{Bt} including Preprocessing

The evaluation of Π_{Bt} in Section 5.2 partially takes care about the communication costs for the preprocessing, i.e., for AND triples. In Table 3 we provide more comprehensive numbers for the communication costs, covering more of the preprocessing, as well as numbers excluding the costs for the AND triples. Following Boyle et al. [10], the preprocessing stage for Π_{Bt} is as follows:

- the Beaver triples that Π_{Bt} consumes can be taken from a previous call of Π_{Bt}. We denote the ratio between consumed and generated Beaver triples as recursion rate R. Note that the self-recursion assumes that $[\![\mathrm{m}]\!]$ is authenticated with respect to itself. For simplicity, we assume that $[\![\mathrm{m}]\!]$ is given as input.
- the PCG for VOLE [8] to support the calls of **Input**(only for the initial instantiation with $\Pi_{\mathrm{aSUV}}^{\mathrm{Boyle}}$). In Table 2, we make a slightly different assumption and count the costs to preprocess randoms shares instead of VOLE, that can be use alternatively to realize calls to **Input** [15]. Random shares can be generated by restricting Π_{Bt} to the x-component. We do this for simplicity, and to bind the MAC m to several recursive calls of Π_{Bt}.
- the AND triple protocol [19] to support integer additions with $\|\cdot\|_m$. According to [10,19], the costs for one AND triple are 32 bits of communication per party and 54 random COT. The numbers in Table 1 include the 32 bits per AND triple.

Table 3. Communication cost of Π_{Bt} including the preprocessing. This table extends Table 1. "without" refers to numbers without preprocessing, while the other columns include preprocessing. For COT we take two different assumptions. "Overhead" refers to the additional costs induced by the prepossessing: The numbers for our work without preprocessing related to the numbers for our our work where the costs for COT are according to [8]. R is the recursion rate for Beaver triples.

				without		COT with [29]		COT with [8]		
λ_{LPN}	d	t	R	Boyle	Our work	Boyle	Our work	Boyle	Our work	Overhead
80	96	48	7%	8.22	7.60	9.90	9.22	9.56	8.87	17%
80	40	10	1%	1.57	1.46	1.78	1.67	1.72	1.61	10%
80	32	4	1%	1.06	0.98	1.20	1.12	1.15	1.07	10%
128	152	76	18%	19.68	18.17	26.73	24.84	25.83	23.94	32%
128	64	16	3%	3.83	3.54	4.43	4.13	4.27	3.97	12%
128	40	5	1%	1.63	1.51	1.86	1.73	1.79	1.67	10%
Average reduction					7.5%		6.8%		7.0%	

– a PCG for COT, as necessary for the AND triple generation [19] and to sample the positions of the small aSUV instances. To generate COT, Boyle et al. [10] propose the PCG [8]. In our estimated evaluation below, we either assume this PCG with communication costs of 101 bits per COT (with malicious security) or FerretOT [29] with higher costs of 0.44 per COT, but better runtime efficiency.

To model the costs for the preprocessing, we compute the total communication costs C of Π_{Bt} according to $C = C_{\mathrm{Bt}} + C_{\mathrm{VOLE}} + C_{\mathrm{AND}} + C_{\mathrm{COT}} + R \cdot C$, where C_{Bt} refers to Π_{Bt} without preprocessing and the other values refer to the costs for the respective preprocessing. This reflects to a large scale application, where the self-recursion of Beaver triples is modelled as a stream running infinite rounds of Π_{Bt}. In this sense, we do not include initialization costs, e.g., for the very first round of Beaver triples.

The numbers in Table 3 show that the preprocessing increases the costs by at least 10%. The main reasons is due to generation of AND triples. However, since the recursion rate for Beaver triples grows with d^2, the costs for the recursive calls of Π_{Bt} become more and more dominant. Furthermore, the different options for the COT generation are significant.