Silentium revisited: Pseudorandom Beaver Triple Expansion

Vincent Rieder[0009-0007-8694-7260]
Enrcio Sorbera[1,2][0009-0009-0973-512X]

[1] Fondazione Bruno Kessler, Center for Cybersecurity, Trento, Italy,
[2] University Of Trento, Department of Mathematics, Trento, Italy
enrico.sorbera@unitn.it.

Keywords: Secure Multi-Party Computation, Beaver Triples, Pseudorandom Correlation Generators, Silentium

Abstract. In the line of the SPDZ protocol for secure multi-party computation, the generation of Beaver triples is the most expensive task. Silentium (Rieder, PrivCryp 25) is the implementation of a Pseudorandom Correlation Generator (PCG) for Beaver triples (Boyle et al., Crypto 20). PCGs focus on low-communication costs., e.g. their PCG reduces the communication by one order of magnitude compared to protocols in MP-SPDZ. Silentium is an implementation of their PCG, achieving similar running times than MP-SPDZ. We make three theoretical contributions to Silentium, including an implementation. First, we make a practical proposal how to generate Beaver triples over binary fields $\mathbb{F}_{2^\lambda}$, which extends the previous setting over prime fields. For this, we propose a suitable instantiation of the Number Theoretic Transform. Second, we show how to use the binary triples to construct what we call a Beaver triple expansion scheme, that is we construct a scheme that expands a small batch of Beaver triples into a large batch of Beaver triples, in the sense of recently established oblivious transfer extension schemes. This feature enables an efficient preprocessing stage for the PCG, closing a practical issue of Silentium. Finally, we provide details about the Silentium implementation, by clearing a technical bug in the initial theoretical protocol description.

1 Introduction

Secure multi-party computation (MPC) is a privacy-enhancing technology that allows several parties to evaluate a public function without leaking private inputs and outputs. We consider the two-party setting with active security, i.e., where one malicious party might actively deviate from the computation. As MPC framework, we consider the SPDZ [18] protocol for arithmetic circuit evaluation. The SPDZ protocol initiated a series of work that can be seen as state-of-the art for actively secure MPC arithmetic in the so called preprocessing model.

In the preprocessing model [3], the actual secure function evaluation (online phase) consumes distributed correlated randomness that is provided by a preceding, input-independent, offline phase. While this design allows the construction of online phases with little overhead compared to an unprotected function evaluation, the construction of offline phases is challenging in terms of active security, computation and communication. Concretely for SPDZ, each secure multiplication in the online phases consumes one (authenticated) Beaver triple [2], also known as authenticated multiplication triple.

In our work, we revisit Silentium [35], a recent offline phase implementation for two-party Beaver triples, namely of a Pseudorandom Correlation Generator (PCG) by Boyle et al. [10] (Bt-PCG). Silentium was developed in the context of Carbyne Stack[3] for MPC on cloud servers, where the focus was on low communication. The Silentium implementation proves that the novel PCG approach achieves even good running time efficiency for Beaver triples. However, due to the novelty of PCGs, several points remain open, some of which we address in this work. We make three main contributions (Sections 1.2 to 1.4), including their implementation.

1.1 Related Work

Compared to the most established offline phases for SPDZ, e.g. the protocols in MP-SPDZ [26] based on homomorphic encryption or oblivious transfer extension, Silentium follows another approach, namely that of a PCG. In a nutshell:

Pseudorandom Correlation Generators. PCGs [5,9] can be seen as a distributed generalization of PRGs to expand small (two) private seeds into a large batch of correlated pseudo-randomness (shared between the two parties) that is close to a given distributed target correlation. A PCG consists of two stages: In a first step, private seeds are generated, which, in our use case of an MPC offline phase, takes place as actively secure seed generation protocol. In a second step, the parties can expand their private seeds locally, i.e., without further communication, into a large amount of correlated randomness, e.g., 2^{20} of Beaver triples in case of the Bt-PCG [10]. The point about PCGs is to be compressive in terms of small private seeds, which is tipically achieved with an encoding of the target correlation under variations of the learning with error assumption [33]. Although the compressive property is challenging to achieve, it has two advantages:

- **Low Communication:** When carefully constructed, the small seeds can be generated with low communication, which typically is achieved with function secret sharing [11,12] in the form of distributed point functions (DPF) [22].
- **Reduced Storage:** The storage costs for the seed between the interactive and local phase is cheap, which becomes relevant for large-scale applications with many PCG iterations.

[3] https://carbynestack.io/

These advantages apply especially to the cloud setting, where, in addition to computational resources, communication bandwidth and storage are important cost factors.

PCGs for Beaver triples. Impressively, the Bt-PCG [9] can generate 2^{20} Beaver triples (128 bit field, worth about 100 MB correlated randomness per party) with only 0.5 to 30 MB per party, which is three orders of magnitude better than for the protocols in MP-SPDZ. Concerning the running time, it is well known that PCGs in general, e.g., for oblivious transfer or in the passive secure case, can achieve very good performances [4,6,7,8,37]. However, the construction of PCGs for Beaver triples turns out to be more challenging due to their multiplicative complexity. So far to our knowledge, apart from initial, but inefficient, approaches [9], there exists only the Bt-PCG [10,34], implemented in Silentium, and a theoretical generalization to the multi-party setting [1].

Silentium. Silentium [35] is, to our knowledge, the only implementation of the Bt-PCG [10], proving that the Bt-PCG can run as fast as competitive standard offline phases approaches in MP-SPDZ [26]. The cryptographic novelty of Silentium is the implementation of two recent primitives [9]:

- An actively secure generation protocol for authenticated secret-shared scaled unit vectors (SUVs) [34].
- The ring-LPN assumption [10] over large fields, generalizing the binary-ring LPN assumption [25]. Hereby the challenge is that the ring-LPN assumption requires large degree polynomial operations, e.g., the Number Theoretic Transform with degree 2^{20} over a 128-bit field. Note that this is at least five orders beyond sizes for comparable applications like the lattice based protocols standardized by NIST[4].

We stress that, at this moment, Silentium is not publicly available, such that protocol details need to remain hidden. Furthermore, we stress that open theoretical aspects about the security of the respective ring-LPN assumption, i.e., on the learning parity noise assumptions, are out of the scope of this work.

1.2 Contribution 1: Algorithmic Correctness

We point out that the initial protocol description for the SUV generation [10] contains a technical bug. While Silentium takes care about this issue from the beginning on, which is our contribution, a theoretical discussion was still pending. We provide this discussion in Section 3, with a comprehensive correctness proof in Section C.

[4] https://csrc.nist.gov/projects/post-quantum-cryptography

1.3 Contribution 2: PCG for Beaver Triples over Binary Fields

We discuss how to extend the Bt-PCG for binary field Beaver triples, i.e., for Beaver triples with respect of fields of size 2^λ. Specifically, we extend Silentium by an implementation for a field $\mathbb{F}_{2^\lambda}$ of size 2^{80}. While this extension was motivated as prerequisite for our Beaver triple expansion (Section 1.4), in general this can be seen as an approach for offline phase in SPDZ protocols over binary fields.

We extend the Silentium implementation, so that one can select between the initial prime field (128 bits) triples and binary field (2^{80} bit) triples at compile time. These selections are encouraged by the versatile design of Silentium, for which our binary PCG add-on can be seen as practical example. The running time and communication costs are hereby similar for both field choices, so our binary field PCG can be stated to be efficient (Section 6).

To instantiate the PCG for binary field Beaver triples, our main contribution is to set up the polynomial arithmetic in the PCG local phase (Section 5). Concretely, our contribution is to set up the NTT over a large binary field in an efficient and suitable way, for which we choose $\mathbb{F}_{2^{80}}$. Running the NTT over a binary field instead of a prime field is a problem of less attention, especially for fields of that large size (see above).

While there exists many work, e.g. [16,15] to name a few on this extensive topic, it was not straightforward to construct a version which suits into our use case of Silentium. Especially due to the very large degree. Furthermore, the standard fast Fourier transform algorithm (FFT) for degrees of power of two does not apply in our use case of a binary field. However, the Bt-PCG allows some freedom in the parameter regime, and we show that this induces efficient alternatives, using additive fast Fourier transform algorithms (AFFT [20]). Altogether, we believe our construction can bring new insights to applications of the binary field NTT in general.

We stress that a formal security analysis of the ring-LPN assumption in the binary field case is out of the scope of our work. In this sense our construction remains a proposal.

1.4 Contribution 3: Beaver Triple Expansion

We construct a Beaver triple expansion scheme, that is, a scheme which can expand a small set of Beaver triples (over $\mathbb{F}$ and $\mathbb{F}_{2^\lambda}$) into a potentially infinite set of Beaver triples over $\mathbb{F}$. Our scheme is rather simple and makes use of two instances of the Bt-PCG, one instantiated over $\mathbb{F}$ and one over $\mathbb{F}_{2^\lambda}$, see 1. For the instance over $\mathbb{F}_{2^\lambda}$, we apply our second contribution, Section 1.3. The notion of Beaver triple expansion does not only solve the gap of missing protocol for the generation of AND triple generation [10,24] in Silentium, but in fact can be seen as simplification for real-world deployments.

However, to get our scheme, one technical modification of the SUV generation protocol is necessary. This happens on the level of MPC protocol operations inside the PCG itself. These MPC operations are in the processing model, and our contribution targets to simplify the respective preprocessing. What we achieve

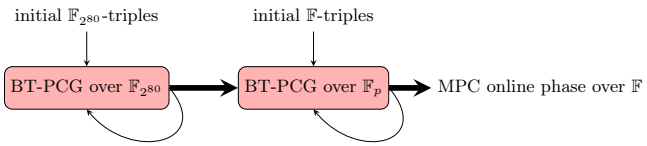


Fig. 1. The Beaver triple expansion framework for Beaver triples over $\mathbb{F}$, consisting of an Bt-PCG instance over $\mathbb{F}_{2^{80}}$ and $\mathbb{F}$. Both instance produce batches of N Beaver triples, of which a small portions are feedback for iterative calls. The very fist rounds needs to be activated on triples from an external source.

is a preprocessiong of the entire PCG, that almost (up to the initial triples for which we e.g. propose to use MP-SPDZ), depends on the PCG construction itself. This is much more compact compared to the initial proposal [10], depending on additional protocols for so called AND triples [9,24] from MPC with TinyOT [31]. Note that we consider the optimized version of the Bt-PCG [34], where the dependency on VOLE [9] correlations is already removed.

More precisely to construct the Beaver triple expansion, we need to exchange the dependency on MPC with TinyOT [31] in the initial Bt-PCG [10] by a dependency on SPDZ over $\mathbb{F}_{2^\lambda}$, that is, we change the MPC protocol to make bitwise operations depending on SPDZ over $\mathbb{F}_{2^\lambda}$ rather than TinyOT. The technical steps are explained in (Section 4).

We provide a feature to Silentium, to select between running the PCG using TinyOT or binary SPDZ. Hereby, the choice between TinyOT and binary SPDZ does not affect the running time. However, the communication cost is increased by a factor 1.5. This is no surprise, since the Bt-PCG construction [10] targets for minimal communication costs. However compared to one-order of magnitude advantage to MP-SPDZ, we see a factor 1.5 acceptable in order to enable the Beaver triple expansion with simplified deployment of the preprocessing (see Section 6).

In Silentium the implementation of the expansion scheme is only sketched and needs to be configured by the user, a full automation is potential future work. However, considering the preprocessing cost analysis of the Bt-PCG [34], we expect the whole Beaver triple expansion scheme to be attractive.

2 Preliminaries

In general, let $\mathbb{F}$ be an arbitrary finite field with q elements, where the bit size $\nu = \log_2(q)$ is a security parameter. Specifically, we write $\mathbb{F}_{2^\lambda}$ for a binary field with bit size, and security parameter, λ. We write $x \xleftarrow{\$} S$ to denote that x is sampled uniformly random from a set S. We use upper indices S^c to denote c independent instances of S. We highlight vectors as $\boldsymbol{s} \in S^c$.

2.1 Secure Computation with Additive-Secret Sharing

We now provide the necessary background for MPC with SPDZ and TinyOT, in the setting of two parties $P_\sigma, \sigma \in \{0,1\}$.

Additive Secret Sharing over $\mathbb{F}$ (SPDZ): We write $[x]$ for a secret-shared value $x \in \mathbb{F}$, where each party P_σ holds a private share x_σ and a private message authentication code (MAC) x'_σ such that $x = x_0 + x_1$ and $\mathrm{m}x = x'_0 + x'_1$, where $\mathrm{m} = \mathrm{m}_0 + \mathrm{m}_1 \in \mathbb{F}$ is an additively secret-shared global MAC key (i.e P_σ privately knows m_σ) [17]. We write $x' = \mathrm{m} \cdot x$ for short. The MAC is used to ensure active security, i.e., on each secret-shared value $[x]$ that is revealed as output, a MAC check is performed to ensure that the value x was computed as intended without any corruption. We extend the notation $[\cdot]$ to vectors and polynomials over $\mathbb{F}$, e.g., $[\cdot]_{\mathbb{F}^N}$, where all coefficients are secret-shared individually. We denote an authenticated secret-shared scaled unit vector, i.e., a secret-shared element in $\mathbb{F}^N$ where exactly one position α has a non-zero payload A, as SUV [34][5].

Secure Arithmetic over $[\cdot]$: The secret-sharing scheme $[\cdot]$ can be used for secure evaluations of arithmetic circuits [2,3]. Due to linearity, additions of secret shared values, additions with a constant, and scalar multiplications can be done locally without communication. However, multiplications $[z] \leftarrow [x] \cdot [y]$ are interactive, consuming one Beaver triple $[c], [a], [b]$, where $a, b \leftarrow \mathbb{F}, c = a \cdot b$, see 1. Furthermore, we write $c \leftarrow \mathbf{Open}([c])$ for a functionality that reveals a secret shared value to both parties with the guarantee of active security. More precisely, this means that the parties exchange their shares c_σ, such that they can learn $c = c_0 + c_1$, and additionally perform a MAC check. Note that in practice, several MAC checks can be performed in one batch to keep the communication costs constant [27].

The command **Open** is used not only to provide the final output, but also for secure multiplications. Hereby, in a first step, parties call $\epsilon \leftarrow \mathbf{Open}([x]-[a])$ and $\delta \leftarrow \mathbf{Open}([y]-[b])$, where the factors of the Beaver triple is used to mask the input factors. Afterwards the parties locally compute the product:

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

Additive Secret-Sharing of Binary Values (TinyOT): We write $[x]_1$ for a secret-shared bit, where both parties P_σ hold a private share x_σ such that $x = x_0 \oplus x_1$ (XOR operation). Furthermore, for active security, the parties hold private MAC values, which are in $\{0,1\}^\lambda$ for a security parameter λ. The definition of the MAC depends on the concrete scheme, for which we consider two different options: $[\cdot]$ over the subset $\mathbb{F}_2 \subset \mathbb{F}_{2^\lambda}$ (Section 4), and TinyOT [31]. In both cases, the parties can locally compute $\oplus$ operations, while AND operations are interactive, following the same pattern as for $[\cdot]$ where we refer to

[5] compared to to [34], we skip the term "authenticated" (aSUV) for simplicity).

Algorithm 1 Direct Generation of Beaver triples (Π_{Bt}) [10,34]

INPUT: A secret-shared MAC key [m] and parameters $d = c \cdot t, N = 2^n, \mathcal{R} = \mathbb{F}[X]/F$, where F has degree N and is fully reducible.
OUTPUT: N Beaver triples over $\mathbb{F}$.

Interactive Phase:

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

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 $\mathcal{T}_{c,t}(\boldsymbol{u}, [\boldsymbol{x}]), \mathcal{T}_{c,t}(\boldsymbol{u}, [\boldsymbol{y}]), \mathcal{S}_{c,t}(\bar{u}, [\boldsymbol{z}])$, where $\boldsymbol{x}, \boldsymbol{y}$ are interpreted in $\mathcal{R}^d$ and $\boldsymbol{z}$ is interpreted in $\mathcal{R}^{d \times d}$ (i.e. modulo F).

the consumed triples as AND triple. We extend $[\cdot]_1$ to secret share n-bit integers $[\alpha]_n$ where each bit is shared individually with $[\cdot]_1$. Then, we write $[\alpha_i]_1$ for the individual bits, where $\alpha_{m-1} \ldots \alpha_0$ is the binary representation of α. We write $[c]_{n+1} \leftarrow [a]_n + [b]_n$ for the secure evaluation of a respective bit-wise integer addition, which can be done with n (interactive) AND operations [28].

2.2 The Bt-PCG

In Protocol 1 we provide an algorithmic presentation of the Beaver triple generation protocol Π_{Bt}[10], in the following we give a step by step description. For a more conceptual description, we refer to previous work [10,34]. In general, the purpose of Π_{Bt} is to generate $N = 2^n$ many Beaver triples $[x], [y], [z]$, these three components are reflected inside the whole Bt-PCG. In the sense of a PCG, Π_{Bt} consists of an interactive - and local phase, which we describe separately. The core algorithm of the interactive phase is a protocol Π_{SUV}, and Π^*_{SUV} [34] as variation thereof, for the generation of SUVs.

Interactive Phase: For the x and y components, the interactive phases starts to generate two vectors of d many N dimensional SUVs with uniformly random positions and payloads. In Protocol 1, this refers to steps 1. and 2., where a protocol Π^*_{SUV} is called. In the notation of one entry of the x-component, Π^*_{SUV} takes a depth n and the secret-shared MAC key [m], authenticated with itself, as input, and outputs a uniformly random SUV $\boldsymbol{x}_i$, i.e., a uniformly random additive secret sharing of a N-dimensional unit vector with a uniformly random integer position $\boldsymbol{\alpha}_i \in [0, N)$ and uniformly random payload $\boldsymbol{A}_i \in \mathbb{F}$. Note that the output is not only $[\boldsymbol{x}_i]$ but also covers a secret-sharing of the position $[\boldsymbol{\alpha}_i]_n$

and payload $[\boldsymbol{A}_i]$, where the choice of the schemes $[\cdot]_n, [\cdot]$ is set by the specific operations for the concrete protocol Π^*_{SUV} [10].

Once the x and y component are sampled, the parties perform secure computations on their positions and payloads to compute secret shares of $\boldsymbol{\gamma}_{i,j} = \boldsymbol{\alpha}_i \cdot \boldsymbol{\beta}_i, \boldsymbol{C}_{i,j} = \boldsymbol{A}_i \cdot \boldsymbol{B}_i$ (steps 3., 4.). In other words, the parties securely compute a tensor sum $\boldsymbol{\gamma} = \boldsymbol{\alpha} \boxplus \boldsymbol{\beta} \in \mathbb{Z}^{d \times d}$ and a tensor product $\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B} \in (\mathbb{F}^N)^{c \times c}$.

The most expensive step in the interactive phase is the generation of vector $[\boldsymbol{z}]$ of quadratic many SUVs related to the z-component. This takes place in step 5. by calls to Π_{SUV}. The difference between Π_{SUV} and Π^*_{SUV} is that Π_{SUV} generates an uniformly random SUV, given a secret shared position and secret-shared payload as input rather than providing them as output.

Local Phase: The purpose of the local phase is to transform the vectors of SUVs $[\boldsymbol{x}], [\boldsymbol{y}], [\boldsymbol{z}]$ into a set of N many Beaver triples. For this, the parties locally apply a linear transformation that expands sparse vectors into dense elements over $\mathbb{F}$, where $\langle \cdot, \cdot \rangle$ denotes the standard scalar product (here over vectors in $\mathcal{R}$). To ensure pseudo-randomness of the output Beaver triples, the security of the transformation relies on the ring-LPN assumption [10]. For completeness, we recap the ring-LPN assumption in Section A.

2.3 The Number Theoretic Transform (NTT)

With NTT, we denote a generalization of the Discrete Fourier Transform over finite fields. In this sense, we assume the NTT as well known and established. We now recap the basics for our work. Let N be a vector degree and let the field size q of $\mathbb{F}$ be such that $q \equiv 1 \mod N$, i.e., N divides $q-1$, then there exists a primitive N-root of unity $\omega \in \mathbb{F}$, that is $\omega^N = 1$ and $\omega^n \neq 1$ for all $0 < n < N$. In such a setting, the NTT is defined as the map

$$\text{NTT} : \mathbb{F}^N \to \mathbb{F}^N \quad (f_i)_{i=0}^{N-1} \mapsto (\hat{f}_i)_{i=0}^{N-1} \quad \hat{f}_i = \sum_{j=0}^{N-1} f_j \cdot \omega^{i \cdot j}, \tag{2}$$

If we endow the vector space $\mathbb{F}^N$ with a ring structure, that is, we consider the polynomial ring $\mathcal{R} = \mathbb{F}[X]/(X^N - 1)$ to be the domain of such map, then, the NTT is a ring-isomorphism between $(\mathcal{R}, +, \cdot)$ and $(\mathcal{R}, +, *)$, where $\cdot$ is the polynomial multiplication, while $*$ is the coefficient wise multiplication.This gives rise to an NTT style polynomial multiplication in $\mathcal{R}$, e.g., $f \cdot g = \text{NTT}^{-1}(\text{NTT}(f) * \text{NTT}(g))$. If we instantiate the ring $\mathcal{R}$ in Π_{Bt} as $\mathcal{R} = \mathbb{F}[X]/(X^N - 1)$, its local phases reduces to NTT calls. In fact, a secure choice is to operate on $\mathcal{R} = \mathbb{F}[X]/(X^N - 1)$ [9], where ring multiplications can be reduce to NTT calls with little overhead [30].

We denote both, the NTT map, and NTT algorithms to compute $\hat{f} = (\hat{f}_i)_{i=0}^{N-1}$, simply as NTT. It is a well known result that the NTT can be computed in asymptotic time $\mathcal{O}(N \log(N))$, for which there exists a variety of Fast Fourier Transform Algorithms. We now recap a few standard methods, differentiating between assumptions on N.

Cooley-Tukey and Radix-p. Let $N = N_1 \cdot N_2$, be some non-trivial integer factorization of N. The most common approach to efficiently compute the N points NTT, is to decompose the NTT into N_1 many NTTs of degree N_2, N_2 many NTTs of degree N_1, and $N_2 \cdot (N_1 - 1)$ many field multiplications with so called twiddle factors (powers of ω) and multiplications. If N_1, N_2 are further decompose into smaller factors, this reduction step can be recursively continued down to NTTs of prime degree. Depending on the concrete integer indexing, these methods are known as Cooley-Tukey [14] or Gentleman-Sande [21]. The most common case is that $N = p^k$ is a prime power, where specifically for small p the prime NTT is trivial in complexity. This approach for computing NTT in such a setting is generally denoted as Radix-p NTT. We recap general prime degree NTT algorithms below.

Multi-dimensional NTT. The decomposition step for the Cooley-Tukey NTT works for any factorization $N = N_1 \cdot N_2$. However, whenever N_1, N_2 are co-prime one can do even better and avoid the multiplications with the twiddle factors. This approach is called two-dimensional NTT [13], or multi-dimensional NTT referring to further recursive decompositions (down to the co-prime factors of N).

Prime Degree NTTs and Rader Reduction. If $N = p$ is a small prime, e.g., $p < 20$, it is efficient to compute the NTT with the formula as given in 2. Hereby, we point out that by arithmetic properties of $\mathcal{R}$, it holds $\sum_{i=0}^{p-1} \omega^i = -1$, which can be used to reduce the number of field multiplications by substituting the largest power ω^{p-1} by smaller powers (butterfly substitution). Since this method has costs of $\mathcal{O}(p^2)$ it is not attractive for larger primes. In this case one can use Rader's algorithm [32], which reduces the p-dimensional degree to a polynomial multiplication of degree $p - 1$. Since $p - 1$ is composite, one can solve this multiplication recursively (see next paragraph).

Cyclotomic NTT. Further optimization is possible when N equals a prime p, such that $p = 2^\lambda - 1$ for some integer λ, for which one can use a cyclotomic NTT algorithm [36]. This algorithm computes a p points NTT by exploiting cyclotomic classes and linearized polynomials. Whenever cyclotomic classes have the same cardinality m, the cyclotomic NTT takes $(l-1)m$ multiplications in $\mathbb{F}_{2^\lambda}$, where l is the number of cyclotomic classes.

Karatsuba Multiplication. As a building block for the NTT, e.g. Rader's method, we use the Karatsuba multiplication algorithm of n-degree polynomials. Although it has not the best asymptotic complexity, e.g., in the step-3 case achieves $\mathcal{O}(n^{\log(3)})$ compared to standard algorithms with $\mathcal{O}(n \log(n))$, Karatsubas reductions turn out to be sufficient for our use cases on small degrees n. For details, see Section B.

3 Clarifications on the Protocol Π_{SUV}

In the following, we present Π_{SUV} in the form as implemented within Silentium [35]. The task of Π_{SUV} is to generate an SUV, given a secret-shared position $[\alpha]_n$, payload $[A]$ and MAC key [m] that need to remain secret. The protocol Π_{SUV} consists of two parts [34]: A tree phase Π^{tree}, Protocol 2, processing the position α and a field phase Π^{field}, Section C.1, processing the payload and MAC. While the field phase is identical as described in previous work [34], our presentation of $\Pi^{\text{tree}}_{\text{SUV}}$, enhances the previous presentation [10] as follows:

- We fix a bug and provide an adapted comprehensive correctness proof. The bug is about proper indexing.
- We adapt the description of the used PRG interactive phase to the implementation in Silentium.

Section 3.1 provides an intuition of $\Pi^{\text{tree}}_{\text{SUV}}$, while Section 3.2 provides a detailed step by step description. The differences to the initial protocol [34,9] are discussed in Section 3.3, while the formal correctness proof is in the Appendix,Section C.2.

3.1 Intuition of the Tree Phase $\Pi^{\text{tree}}_{\text{SUV}}$

In the following we consider the generation of $M = 2^m$-dimensional SUVs, where for the Bt-PCG $m \in \{n_b, n_b + 1\}$, for the x, y or z-components respectively. We focus on Π_{SUV}, since Π^{*}_{SUV} is a minor optimization specially introduced for the Bt-PCG, where the bug is analogue.

The task of $\Pi^{\text{tree}}_{\text{SUV}}$ is to generate private left- and right vectors, denoted as $t^L_\sigma, t^R_\sigma \in \mathbb{F}^M$, where party P_σ only learns s^L_σ, s^R_σ and such that $t^L = t^L_0 + t^L_1$ and $t^R = t^R_0 + t^R_1$ are scaled unit vectors with respective uniformly random payloads u^L, u^R at the same position α. The tree phase employs the following PRGs and sub-routine, where λ is the security parameter of $[\cdot]_1$:

- A (length-doubling) PRG : $\{0,1\}^\lambda \to \{0,1\}^{2\lambda+2} \simeq \{0,1\}^\lambda \times \{0,1\}^\lambda \times \{0,1\} \times \{0,1\}$.
- A PRG Conv : $\{0,1\}^\lambda \to \mathbb{F} \times \mathbb{F}$.
- A functionality $\textbf{MixedMult}(x_0, x_1, [\delta]_1)$ that receives $x_\sigma \in \{0,1\}^\lambda$ from P_σ and a secret-shared bit $[\delta]_1$ (as identifier). It then computes $z = \delta \cdot (x_0 \oplus x_1)$ Finally it randomly decomposes $z = z_0 + z_1$ and sends z_σ to P_σ.

The tree phase follows a GGM [23] binary tree, where each party runs through nodes $s^{i,j}_\sigma \in \{0,1\}^\lambda,\ i = 0, \lfloor, m,\ j = 0, \rfloor, 2^i - 1$. Using the interaction in the first framed box of $\Pi^{\text{tree}}_{\text{SUV}}$, the parties maintain a tree invariant [10]. The tree invariant states that their node values $s^{i,j}_\sigma$ differ exactly on the tree path corresponding to α. In formulas:

$$s^{i,j}_0 = s^{i,j}_1 \Leftrightarrow j \neq \lfloor \frac{\alpha}{2^{m-i}} \rfloor \tag{3}$$

Algorithm 2 Tree phase $\Pi^{\text{tree}}_{\text{SUV}}$

The interactive parts of $\Pi^{\text{tree}}_{\text{SUV}}$ are framed.

INPUT: $[\alpha]_m$. This means especially that for each bit α_j, party P_σ holds $\alpha_{j,\sigma}$ with $\alpha_j = \alpha_{j,0} \oplus \alpha_{j,1}$.

1. SETUP
   $s_\sigma^{0,0} \xleftarrow{\$} \{0,1\}^\lambda$
   $T_\sigma^{0,0} \leftarrow \sigma$.
2. LOOP THROUGH TREE LEVELS
   for $i = 1, \ldots, m$ **do**
   for $j = 0, \ldots, 2^{i-1} - 1$ **do**
   $(\bar{s}_\sigma^{i,2j}, \bar{s}_\sigma^{i,2j+1}, t_\sigma^{i,2j}, t_\sigma^{i,2j+1}) \leftarrow \text{PRG}(s_\sigma^{i-1,j})$
   end for
   For $b \in \{0,1\} : u_\sigma^{i,b} \leftarrow \bigoplus_{j=0}^{2^{i-1}-1} t_\sigma^{i,2j+b}$
   For $b \in \{0,1\} : z_\sigma^{i,b} \leftarrow \bigoplus_{j=0}^{2^{i-1}-1} \bar{s}_\sigma^{i,2j+b}$

   The parties exchange $u_\sigma^{i,0} \oplus \alpha_{m-i,\sigma}$ to compute $\tau^{i,0} = u^{i,0} \oplus \alpha_{m-i} \oplus 1$
   The parties exchange $u_\sigma^{i,1} \oplus \alpha_{m-i,\sigma}$ to compute $\tau^{i,1} = u^{i,1} \oplus \alpha_{m-i}$
   $(y_0, y_1) \leftarrow \textbf{MixedMult}(z_0^{i,0} \oplus z_0^{i,1}, z_1^{i,0} \oplus z_1^{i,1}, [\alpha_{m-i}]_1)$
   The parties exchange $y_\sigma \oplus z_\sigma^{i,1}$ to compute $\text{CW}^i = y \oplus z^{i,1}$

   for $j = 0, \ldots, 2^{i-1} - 1$ **do**
   For $b \in \{0,1\} : T_\sigma^{i,2j+b} \leftarrow t_\sigma^{i,2j+b} \oplus T_\sigma^{i-1,j} \cdot \tau^{i,b}$
   For $b \in \{0,1\} : s_\sigma^{i,2j+b} \leftarrow \bar{s}_\sigma^{i,2j+b} \oplus T_\sigma^{i,2j+1-b} \cdot \text{CW}^i$
   end for
   end for
3. CONVERSION
   for $j = 1, \ldots, 2^{i-1} - 1$ **do**
   $(s_\sigma^{L,j}, s_\sigma^{R,j}) \leftarrow \text{Conv}(s_\sigma^{m,j})$
   end for
   return $t_\sigma^L = (-1)^\sigma \cdot s_\sigma^L$ and $t_\sigma^R = (-1)^\sigma \cdot s_\sigma^R$

where $\lfloor \cdot \rfloor$ denotes rounding to the next lower integer, i.e., $\lfloor \frac{\alpha}{2^{m-i}} \rfloor$ refers to the integer with binary representation $\alpha_{m-1} \dots \alpha_{m-i}$. To maintain the tree invariant, the parties securely compute correction bits $T_\sigma^{i,j}$, one for each tree node, that follow a similar tree invariant: $T_0^{i,j} = T_1^{i,j} \Leftrightarrow j \neq \lfloor \frac{\alpha}{2^{m-i}} \rfloor$.

3.2 Individual Steps of $\Pi_{\mathbf{SUV}}^{\mathbf{tree}}$

We now describe $\Pi_{\mathrm{SUV}}^{\mathrm{tree}}$ as depicted in Protocol 2. The input is a bit-wise secret-shared integer position $[\alpha]_m$. In the SETUP, step 1, the parties sample uniformly random private tree roots $s_\sigma^{0,0} \in \{0,1\}^\lambda$ and initialize the correction word CW^0 and bit $T_\sigma^{1,0}, T_\sigma^{1,1}$. Then they start step 2, the LOOP, where for $i = 1$, they apply the length-doubling PRG to compute preliminary node values $\bar{s}_\sigma^{1,0}, \bar{s}_\sigma^{1,1} \in \{0,1\}^\lambda$ of the first tree level. However, these preliminary values do not fulfill the tree invariant (3): Since $s_0^{0,0} \neq s_1^{0,0}$ it holds $\bar{s}_0^{1,0} \neq \bar{s}_1^{1,0}$ and $\bar{s}_0^{1,1} \neq \bar{s}_1^{1,1}$. Instead, it should hold $\bar{s}_0^{1,\alpha_{m-1}\oplus 1} = \bar{s}_1^{1,\alpha_{m-1}\oplus 1}$. To achieve this, the parties securely compute a correction word CW^1 and correction bits $T_\sigma^{1,0}, T_\sigma^{1,1}$ to "correct" the node values, under the premise to keep α_{m-1} secret. For $i = 1$, this correction comprises of the following:

- The length doubling PRG provides the parties with additional random intermediate bits $t_\sigma^{1,0}, t_\sigma^{1,1}$
- For $i = 1$, the parties set $u_\sigma^{i,0} = t_\sigma^{i,0}$ and $u_\sigma^{i,1} = t_\sigma^{i,1}$.
- The parties interactively compute public masked bits $\tau^{1,0}, \tau^{1,1}$ that mask the bit-shares $\alpha_{m-1,\sigma}$ with $u_\sigma^{1,0}, u_\sigma^{1,1}$.
- The public masked bits are used locally to compute private correction bits $T_\sigma^{1,0}, T_\sigma^{1,1}$, based on $T_\sigma^{0,0}$. By construction, these bits fulfill the tree invariant for the T-values (see the proof Section C.2).
- The parties use **MixedMult** to compute a correction word CW^1.
- The parties compute corrected node values $s_\sigma^{1,0} \leftarrow \bar{s}_\sigma^{1,0} \oplus T_\sigma^{1,1} \cdot \mathrm{CW}^1$ and $s_\sigma^{1,1} \leftarrow \bar{s}_\sigma^{1,1} \oplus T_\sigma^{1,0} \cdot \mathrm{CW}^1$. These values fulfill the tree invariant 3, see Section C.2 for a detailed proof.

The rest of the second step is a loop running through $i = 1, \cdots, m$, where the difference to $i = 1$ is that the interactive functions operate on sums $u_\sigma^{i,b}, z_\sigma^{i,b}$, which is necessary to hide information about α.

In the third and last step, CONVERSION the last round of node values $s_\sigma^{m,j}$ is locally transformed into private vectors $t_\sigma^L, t_\sigma^R \in \mathbb{F}^D$. Hereby, the property that t^L, t^R are unit vectors, with respect to α, is induced by the tree invariant of the last tree level.

3.3 Differences to Initial Presentation

The adaption and bug fixing compared to the initial presentation of Π_{SUV} [10] are as follows:

Restructuring to Fix the Definition of T-values: In the initial presentation, the definition of the correction bits is $T_\sigma^{i+1,2j+b} = t_\sigma^{i+1,2j+b} \oplus T_\sigma^{i,j} \cdot \tau^{i,\mathrm{Parity(j)}}$, where $\mathrm{Parity}(j)$ denotes the least significant bit of j. We point out that this definition results in a bug. The issue is that $T_\sigma^{i+1,2j}$ on the one hand depend on the correction bits $T_\sigma^{i,j}$ and public bits $\tau^{i,\mathrm{Parity}(j)}$ of the previous level i and, on the other hand depend on the private bits $t_\sigma^{i+1,2j+b}$ of the level i. However, due to the PRG calls in between the two levels, for each j, the values $T_\sigma^{i,j}, \tau^{i,\mathrm{Parity}(j)}$ are independent of the uniformly random values $t_\sigma^{i+1,2j+b}$. Hence, with this definition $T_\sigma^{i+1,2j+b}$ must fail to update the previous correction bits $T_\sigma^{i,j}$ as required to maintain the tree invariant.

To fix the bug, the t-values and τ-values have to refer to the same tree level, for which we restructure some operations, and shift some indices in the initial version of Π_{SUV} [10]. Hereby, we do not modify the definition of the other variables. For the technical correctness of our modifications we refer to the proof in Section C.2.

Explicit Treatment of Conv as PRG: In the initial presentation of Π_{SUV} [10], Conv is not explicitly specified to be a PRG. Instead of Conv, the initial presentation employs one more round of the length-doubling PRG and then applies a conversion from $\{0,1\}^\lambda$ to $\mathbb{F}$. We point out that the initial formulation is too weak to get uniformly pseudo-random vectors $s_\sigma^L, s_\sigma^R \in \mathbb{F}^D$, as required to get a pseudo-random SUV output $[x]$. More precisely, if the security parameter λ is smaller then the bit size ν, than, an embedding is not sufficient to fill the additional $\nu - \lambda$ bits uniformly random. This is in fact an issue, for recommended values $\lambda = 80, \nu = 128$ [9], and becomes more important for our Beaver triple expansion, where we fix $\lambda = 80$, independently of ν.

Instead, our Silentium implementation follows a previous DPF construction [9] and uses two PRGs, i.e., the length-doubling PRG and $\mathrm{Conv}\{0,1\}^\lambda \to \mathbb{F} \times \mathbb{F}$. With this, it is guaranteed that the outputs s_σ^L, s_σ^R are pseudrandom. In general, using two PRGs removes the condition $\lambda < \nu$. We stress, that our modification is only of technical nature and does not change Π_{SUV} conceptually.

4 MixedMult for Beaver Triple Expansion

For the purpose of a Beaver triple expansion scheme, we now modify Π_{Bt} such that the only form of consumed correlated randomness are Beaver triples over $\mathbb{F}$ (target field) and $\mathbb{F}_{2^\lambda}$ (internal field), referring to the secure computations over field elements with $[\cdot]$ and bit values with $[\cdot]_1$, respectively. Note that we only have to address the secret-sharing of bits, since the secure computations over $\mathbb{F}$ in the field phase $\Pi_{\mathrm{SUV}}^{\mathrm{field}}$ depends only on Beaver triples by a previous optimization [34]. Further note that we only have to address Π_{SUV} and Π_{SUV}^* since the computation of the tensor sum $\boldsymbol{\gamma}$, step 3 in Protocol 1, relies only on secure AND operations. With the instantiation of SPDZ over $\mathbb{F}_2$, these correspond to secure multiplications, consuming Beaver triples.

4.1 Secret Sharing of Bits with $[\cdot]$ over $\mathbb{F}_{2^\lambda}$

The initial proposal for Π_{Bt} [10] is to instantiate $[\cdot]_1$ with TinyOT, Section 2.1. In this case, Π_{Bt} requires preprocessing in the form of AND triples [19,24] and correlated OT. Instead, we instantiate with $[\cdot]$ over $\mathbb{F}_{2^\lambda}$, for which we write $[\cdot]_1^\lambda$. Concretely, since $\{0,1\} \cong \mathbb{F}_2 \subset \mathbb{F}_{2^\lambda}$ is a subfield (with respect to And and Xor operations), $[\cdot]$ over $\mathbb{F}_{2^\lambda}$ can be used to operate on bits in the sense of a restriction. With other words, once all secret-shared input values are verified to be in the subset $\mathbb{F}_2$, secure arithmetic over $[\cdot]_1^\lambda$ works identically to secure arithmetic over $[\cdot]$. Note that hereby, the MACs are still full values in $\mathbb{F}_{2^\lambda}$ to guarantee active security with respect to λ, while for the plain values it is possible to operate on single bit values to reduce communication costs.

With the restriction of $[\cdot]$ over $\mathbb{F}_{2^\lambda}$ to $\mathbb{F}_2$ comes the question, how to guarantee that the new inputted values are in fact bits, especially since a malicious party might provide arbitrary values in $\mathbb{F}_{2^\lambda}$ during input steps. Hence, security requires a mechanism to verify the bit-property of shares $[\cdot]_1^\lambda$. For this we remark that in $\mathbb{F}_{2^\lambda}$ it holds $x \cdot (x+1) = 0 \Leftrightarrow x \in \mathbb{F}_2$ (since the roots of $X(X+1) \in \mathbb{F}_{2^\lambda}[X]$ are unique). Hence to check that $[\cdot]_1^\lambda$ shares a bit (is in $\mathbb{F}_2$), the parties can perform a secure computation $Z \leftarrow \textbf{Open}([r] \cdot ([r]+1))$. Then, $Z \neq 0$ indicates malicious behavior so that a honest party can abort the protocol. Note that the correctness of Z itself is ensured by the MAC check included in **Open** and the secure multiplication.

4.2 Realization of MixedMult

To actually get Π_{Bt} running on Beaver triples only, we take the following steps:

1. For Π_{SUV}, we need to realize **MixedMult** over $[\cdot]$ with respect to one factor in $\mathbb{F}_2$ and one factor in $\mathbb{F}_{2^\lambda}$.
2. The protocol Π_{SUV}^* requires to sample a new position $[\alpha]_n^\lambda$. We include this as part of **MixedMult** with little overhead.

A straightforward realization of $\textbf{MixedMult}(x_0, x_1, [\delta]_1)$ over $[\cdot]$ would be to first let party P_σ input their private x_σ values to secret-shared values $[x_\sigma]^\lambda$ and than to compute $[z]^\lambda = ([x_0] + [x_1]) \cdot [\delta]_1$. This is however not sufficient for our purpose, since the creation of $[x_\sigma]^\lambda$ is not for free in terms of correlated randomness (like a random shared values). Instead in Protocol 3 we provide a realization Π_{Mmult} of **MixedMult** that avoids any costs to create $[x_\sigma]$, in terms of correlated randomness and communication.

The Protocol Π_{Mmult}. Our concrete implementation of **MixedMult** is given in Π_{Mmult}, Protocol 3. Basically, the mixed-multiplication follows a secure multiplication over $[\cdot]$, where the local phase, i.e., 1, takes place in step 3. Hereby, the parties do not need to compute the MAC component, since **MixedMult** does not require to output authenticated secret-shared values. In advance of the local multiplication step, the parties interactively compute the masked values δ, ϵ

Algorithm 3 Π_{MMult}

Input: Secret-shared bit $[r]_1^\lambda$ and $z_\sigma \in \mathbb{F}_{2^\lambda}$ from P_σ.
Correlated Randomness: Beaver Triple $[a], [b], [c]$ with respect to $\mathbb{F}_{2^\lambda}$.

1. The parties exchange $a_\sigma + z_\sigma$ to compute $\epsilon = a + z$
2. $\delta \leftarrow \textbf{Open}([b] + [r]_1^\lambda)$
3. Locally, the parties compute $u_\sigma = \delta \cdot z_\sigma + \epsilon \cdot r_\sigma + c_\sigma - \epsilon \cdot \delta$
4. Party P_σ returns u_σ.

(steps 1., 2.). Hereby the difference to a standard secure multiplication (Section 2.1) is that the value δ is computed without a MAC check. While on the one hand this avoids to create an authenticated secret-sharing $[z]$ in advance, this on the other hand is in accordance to skip the MAC in the local multiplication step.

For a security discussion, we point out that the techniques we use is similarly used in the field phase $\Pi_{\text{SUV}}^{\text{field}}$, as introduced in previous work [34]: The computation of $[Z^L], [Z^R]$ (Protocol 5) follows a modified Beaver triple computation, where the factors $[u^L], [v^R], [u^R], [v^L]$ are not given in secret-shared form in advance but rather are computed as part of the masking step with the Beaver triples, e.g., as $[v^L] = [b^L] - \delta^L$, where δ^L masks $v_0^L + v_1^L$ as in a Beaver multiplication (1). This is proven to be secure [34], since the correctness of the inputs u^L, v^R, u^R, v^L is ensured by the condition $Z = 0$. Similarly, the same condition verifies the correctness of z_σ in the tree phase since otherwise the SUV property of the vectors t_σ^L, t_σ^R in Π_{SUV} is corrupted. For a more formal security treatment, we refer to the respective optimization of the field phase [34].

The Protocol Π_{Mmult}^*. For Π_{SUV}^* we provide an extension of Π_{Mmult}^* of Π_{Mmult} that includes to sample the secret-shared bit $[r]$ instead of taking it as input(Protocol 4). We construct Π_{Mmult}^* in such a way that the only additional cost are due to the bit-verification check described in Section 4.1. In general, a secret-shared bit $[r]_1^\lambda$ can be created as follows. The parties take a random secret-share $[s]$ over $\mathbb{F}_{2^\lambda}$ and locally sample $r_\sigma \xleftarrow{\$} \mathbb{F}_2$. They mask r with s, i.e. they exchange $r_\sigma + s_\sigma$ such that they locally can compute $[r]_1^\lambda = (r + s) + [s]$. However, this method does not work for our purpose of a Beaver triple expansion since the sampling of $[s]$ essentially states nothing else than to consumed correlated randomness in the form of random secret-shared values.

Instead, Π_{Mmult}^* uses the factors of the Beaver triples to mask the private bits r_σ, similarly to how we handle the input values x_σ in Π_{Mmult}. Hereby, compared to Π_{Mmult} the parties explicitly compute $[r]$. For active security, Π_{SUV}^* than requires to check that r is indeed a bit, for which the parties perform one Beaver multiplication. In general, the security of Π_{Mmult}^* follows from the security of Π_{Mmult} and the fact that computing the output $[r]$ in secret-shared form is secure (as securely done in $\Pi_{\text{SUV}}^{\text{field}}$ for u^L, u^R [34]).

Algorithm 4 Π^*_{Mmult}

Input: $z_\sigma \in \mathbb{F}_{2^\lambda}$ from P_σ
Correlated Randomness: Two Beaver triple with respect to $\mathbb{F}_{2^\lambda}$, e.g., $[a], [b], [c]$

1. The parties exchange $a_\sigma + z_\sigma$ to compute $\epsilon = a + z$
2. Each party samples $r_\sigma \xleftarrow{\$} \mathbb{F}_2$
3. The parties exchange $b_\sigma + r_\sigma$ to compute $\delta = b + r$
4. $[r] = [b] + (b + r)$
5. $Z \leftarrow \mathbf{Open}([r] \cdot ([r] + 1))$ (taking the second Beaver triple). If $Z \neq 0$, abort.
6. Locally, the parties compute $u_\sigma = \delta \cdot z_\sigma + \epsilon \cdot r_\sigma + c_\sigma - \epsilon \cdot \delta$
7. Return $[r]$ and P_σ returns u_σ.

5 Number Theoretic Transform for $\mathbb{F}_{2^\lambda}$-Beaver Triples

To enable our Beaver triple expansion scheme, the Bt-PCG needs to be instantiated for the generation of $\mathbb{F}_{2^\lambda}$-Beaver triples (left side of Figure 1). We now show how to efficiently run Π_{Bt} over the field $\mathbb{F}_{2^\lambda}$, based on a special purpose setting for the NTT. In theory, running Π_{Bt} over a binary field is simple: one just has to specify $\mathbb{F} = \mathbb{F}_{2^\lambda}$. However, the practice comes with the question how to setup the ring $\mathcal{R}$ for the ring-LPN assumption, which turns out to be challenging with respect to an efficient ring arithmetic.

5.1 General Arithmetic Setting

In the Bt-PCG, the NTT takes the role to map a degree N polynomial from $\mathcal{R} := \mathbb{F}[X]/F$ to $\mathbb{F}^N$, that is, the NTT can be seen as isomorphism $\psi : \mathbb{F}[X]/F \cong \mathbb{F}$. For ψ to be an isomorphism F must be fully reducible over $\mathbb{F}$ with N many distinctive roots. In general the construction with such isomorphisms ψ ensures, that all N Beaver triples are over $\mathbb{F}$ (fully reducible) and distinct (different roots). The Bt-PCG works with any such F (as long as it is considered to be secure), the most attractive variant are F of the form $F = X^N - 1$: For prime fields this allows to compute ψ with well established NTT algorithms efficiently.

Consider F to be fully reducible with distinctive roots. Then we conjecture the instantiation of the Bt-PCG over a binary field, i.e. the respective instantiation of the ring-LPN assumption for, as secure, since we do not expect binary field specific options for attackers [10]. While we believe that most of the security analysis [10] is independent of the field characteristic (prime or binary), a formal treatment is out of the scope of this work. The point is that a formal proof requires deep knowledge about coding theory and assumptions based on lattice based cryptography. So, we take the analysis on the prime field case as [10] as orientation, and believe that if issues arise, they can be address with larger parameter choice for c and t. We are not aware of any further work on the ring-LPN assumption than the initial one [10].

5.2 Challenges Compared to the Prime Field Case

Over a prime field $\mathbb{F}$ the most convenient choice is to set $\mathcal{R} = \mathbb{F}[X]/(X^N - 1)$, where $N = 2^n$, since in this setup ψ can be setup efficiently with the Number Theoretic Transform (NTT). Over a binary field $\mathbb{F}_{2^\lambda}$ the situation is however different. The point is that the condition $2^\lambda \cong 1 \mod N$ can not be fulfilled if N is a power of two (for any even $m = X^m + 1 = (X+1)^m$ so the only root of unity is 1 and ψ is not an isomorphism). To overcome this issue, we see two options:

- Keep $N = 2^{20}$ as a power of two, but change the format of F. Note that hereby, the condition is that F fully splits into linear factors. Then the NTT is replaced by a more general isomorphism induced by F. Transformations for such ψ are in general harder to compute than NTT algorithms.
- Use another value for N, such that the NTT can be used for $\mathbb{F}[X]/(X^N+1)$. Note however, that this NTT can not benefit from standard radix-two methods. Furthermore, the new NTT degree $\tilde{N}$ should be only slightly smaller than the SUV degree $N = 2^n$ (which is fixed to be a power of two), since this gives a gap with resepct to the polynomial reduction.

We decide to take the second option, since we believe that concrete efficiency for the NTT is better compared to generic algorithmic solutions in the first case. For the concrete choice see Section 5.3.

5.3 Specific Arithmetic Setting

The instantiation of $\mathcal{R} = \mathbb{F}_{2^\lambda}[X]/(X^N + 1)$ can be selected by λ and N. Altogether, this is under the following constraints:

- $X^N + 1$ must be fully reducible with distinct roots, as algebraic requirement: $N \equiv 1 \mod 2^\lambda$ (as division: $N | 2^\lambda - 1$). Especially, any even N is not allowed (where indeed it would hold $X^N + 1 = (X+1)^N$).
- λ has to be large enough since it takes the role of a security parameter.
- N should be in the same range as for the prime field case, to be close to the respective ring-LPN security guarantees. On the other hand, N should not be too large, to avoid epxensive computation overhead.
- N should be slightly below of a power of two. The point is, that Π_{SUV} runs on powers of two, which introduces a gap. We try to keep that gap small: For optimal efficiency and to be closer to the initial ring-LPN parameter regime.
- We require N to have only small prime factors, since this gives better reduction for the NTT algorithms.

Our solution is to set $\lambda = 80$ and $N = 2^{20} - 1$ with a minimal gap of 1 to the prime power in Silentium [35,9]. For this choice it holds $2^{20} - 1 \mid 2^{80} - 1$, due to a general algebraic lemma (and this is how we actually found our setting)[29]. Furthermore, it holds $2^{20} - 1 = 3 \cdot 5^2 \cdot 11 \cdot 31 \cdot 41$, i.e., all prime factors are small. This choices allows furthermore an efficient reduction algorithm for the NTT by combinations of various NTT techniques (radix-algorithm, multi-dimensional NTT, Raders algorithm), see Section 5.4.

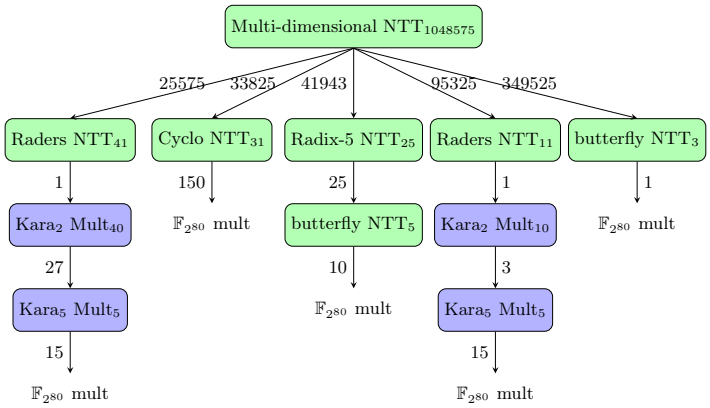


Fig. 2. Workflow of our NTT decomposition. The numbers on the arrows count the number of calls.

5.4 Decomposing the Number Theoretic Transform

In the general setting of arbitrary degree N, there are two major approaches for the NTT algorithms: the Additive Fast Fourier Transform (AFFT [20]) and the standard Fast Fourier Transform algorithm. Usually, when we are free to choose both the set of evaluating points and the size of the target NTT, the first approach is preferable. In fact, it enjoys the usual butterfly structure and can be easily generalized to different sizes and prime fields as necessary in our use case. Furthermore, with the fixed parameters above, we can tailoer the NTT algorithm to be efficient, there is no need to be generic with respect to parameters. So the AFFT approach is more attractive.

Figure 2 visualizes how we actually compute the NTT of degree $\tilde{N} = 2^{20} - 1$ with the AFFT. The general approach is a suitable reduction to smaller degree NNTs, using the algorithms from Section 2.3. In a first step, we apply the multi-dimensional NTT, following the factorization of $\tilde{N}$, that is for each factor prime-power factor $q \in \{3, 11, 25, 31, 41\}$, we need to apply $\frac{\tilde{N}}{q}$ many degree q-NTTs. Note that the choice of multi-dimensional NTT (rather than Cooley-Tukey) avoids costly multiplications with twiddle factors.

In a next step, we distinguish the NTT methods between $q = 3$, which is already maximally reduced, $q = 25$, which is a prime square, and the three small prime numbers $q \in \{11, 31, 41\}$. For all these NTT degrees, we focus on reducing the number of field multiplications, since additions are almost for free (bit-operations).

- For $q = 3$, the NTT needs no further decomposition, that is we apply a direct evaluation of the NTT. Hereby, we replace ω_3^2 by $1 + \omega_3$ (butterfly NTT), which actually means that we compute the degree-3 NTT with one field multiplication.

- For $q = 25$ we apply a radix-5 NTT (one step) to reduce the task to degree 5 NTTs. These are small enough to efficiently apply a direct evaluation. Hereby, we replace the largest power $\omega_5^4 = \omega_5^3 + \omega_5^2 + \omega_5 + 1$, reducing the amount of field multiplications to 10.
- For $q = 31$, we apply the cyclotomic NTT, in fact $q = 2^5 - 1$. It can be checked that there are $l = 7$ cyclotomic classes modulo 31, having the same cardinality $m = 5$. Thus, this algorithm takes $(l-1)m^2 = 150$ multiplications.
- For $q \in \{11, 41\}$ we apply Raders algorithm, which turns us to convolutions of polynomials with degree $q - 1$. We reduce these multiplications to $\mathbb{F}_{2^\lambda}$ multiplications by applying nested Karatsuba multiplications:
  - For $q = 10 = 2 \cdot 5$, we apply nested Karatsuba steps, taking $\frac{2^2+2}{2} \cdot \frac{5^2+5}{2} = 45$ multiplications of $\mathbb{F}_{2^\lambda}$ elements.
  - For $q = 40 = 8 \cdot 5$ we apply three rounds of Karatsuba-2 steps and a Karatsuba-5 step, taking $(\frac{2^2+2}{2})^3 \cdot \frac{5^2+5}{2} = 405$ multiplications.

6 Evaluation

For an evaluation, our focus is on communication cost and running times. Since a more comprehensive analysis requires further knowledge about the parameter selection and design of Silentium [35], a more detailed evaluation of our work, with absolute numbers, can be found in the Appendix, Section D (1). Note as well, that since the security assumption on the binary ring-LPN assumption are only preliminary, exact numbers on the evaluation can be as well only preliminary.

Fixing the bug in Π_{SUV} does not affect the costs, since it is about local and simple bit operations. For the other contributions, the performance is as follows.

6.1 Evaluation of Replacing TinyOT by SPDZ

The choice of the binary MPC instantiation (SPDZ or TinyOT) does not affect the running time, since it is about fast binary operations, and the respective lines of code are only a small fraction of the PCG.

The choice of the binary MPC instantiation with SPDZ compared to TinyOT increases the communication costs of the BT-PCG by a factor of 1.5. Hereby, the communiction with respect to the binary MPC is increased by a factor 2, while the effect on the whole Bt-PCG is only 1.5, since it also takes into account the MPC operations over $\mathbb{F}$ (and these remain unchanged). For a formal analysis, depending on the variables $\log(N), \lambda, \nu$, see Section D.2.

6.2 Evaluation of the Bt-PCG for $\mathbb{F}_{2^{80}}$ Triples

For our evaluation of the Bt-PCG, we run the Bt-PCG with binary SPDZ, instead of TinyOT, since this is the setting of the Beaver triple expansion use case.

The increase in communication is by a factor 1.3. This is a little smaller than above, since here we consider a smaller target field (80 bits instead of 128).

The running time is, in average over different parameter choices, increased by a factor 1.1. Since switching from binary SPDZ MPC to TinyOT does not change the running time significantly, this can be explained by switch from a 128 bit field to an 80 bit field. The positive effect of reducing the field size (which is minor sigificant), is compensated by the more complex structure of the NTT (which is major significant). In fact, we can state our NTT instantiation and methodology over $F_{2^{80}}$ as efficient, since it only results in about 1.1 increase of running time of the whole protocol.

6.3 Conclusion: Evaluation of the Beaver Triple Expansion

The most complex question is, if the Beaver triple expansion scheme is efficient. We do not have exact numbers for the running time, since the expansion is not deployed in Silentium. For now, one can only run the PCG instances individually. Instead as preprocessing, for now, Silentium contains only a fake preprocessing, where one party samples all correlated randomness and sends it to the other party. While this can be deployed automatically for benchmarking, it is total insecure and inefficient. Our proposal of a Beaver triple expansion is compact and more secure alternative.

Altogether, our modifications on Silentium come only with little overhead, which is acceptable for improving the deployment of the Bt-PCG in the future. Our notion of Beaver triple expansion is a large step forward with respect to an practical solution for the preprocessing phase of the Bt-PCG. This in total might simplify the deployment of Silentium as preprocessing phase (offline phase) for large-scale MPC with the SPDZ protocol.

Acknowledgements: A thanks goes to the Robert Bosch GmbH, Research Campus Renningen, where this project was located, and the University of Stuttgart, where this project was additionally supervised.[6]

References

1. Abram, D., Scholl, P.: Low-communication multiparty triple generation for spdz from ring-lpn. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography – PKC 2022. Springer International Publishing, Cham (2022)
2. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) Advances in Cryptology — CRYPTO '91. Springer Berlin Heidelberg (1992)

[6] This work has been done as part of the CRYPTECS project that received funding from the German Federal Ministry of Education and Research under Grant Agreement No. 16KIS1441 and from the French National Research Agency under Agreement Grant No. ANR-20-CYAL-0006. The authors have no competing interest to declare that are relevant to the content of this article.

3. Bendlin, R., Damgård, Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) Advances in Cryptology – EUROCRYPT 2011. Springer Berlin Heidelberg (2011)
4. Bombar, M., Couteau, G., Couvreur, A., Ducros, C.: Correlated pseudorandomness from the hardness of quasi-abelian decoding. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023. Springer Nature Switzerland, Cham (2023)
5. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector ole. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA (2018)
6. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Resch, N., Scholl, P.: Correlated pseudorandomness from expand-accumulate codes. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022. Springer Nature Switzerland, Cham (2022)
7. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Resch, N., Scholl, P.: Correlated pseudorandomness from expand-accumulate codes. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022. Springer Nature Switzerland, Cham (2022)
8. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Rindal, P., Scholl, P.: Efficient two-round ot extension and silent non-interactive secure computation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA (2019)
9. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent ot extension and more. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. Springer International Publishing, Cham (2019)
10. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-lpn. In: Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II. Springer-Verlag, Berlin, Heidelberg (2020)
11. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015. Springer Berlin Heidelberg (2015)
12. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA (2016)
13. Chu, E.: Discrete and Continuous Fourier Transforms: Analysis, Applications and Fast Algorithms. CRC Press (2008), https://books.google.it/books?id=6jyR-dJUQbAC
14. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex fourier series. Mathematics of Computation **19**, 297–301 (1965), https://api.semanticscholar.org/CorpusID:121744946
15. Coxon, N.: Fast hermite interpolation and evaluation over finite fields of characteristic two. Journal of Symbolic Computation **98**, 270–283 (2020). https://doi.org/https://doi.org/10.1016/j.jsc.2019.07.014, https://www.sciencedirect.com/science/article/pii/S0747717119300793, special Issue on Symbolic and Algebraic Computation: ISSAC 2017

16. Coxon, N.: Fast transforms over finite fields of characteristic two. Journal of Symbolic Computation **104**, 824–854 (2021). https://doi.org/https://doi.org/10.1016/j.jsc.2020.10.002, https://www.sciencedirect.com/science/article/pii/S0747717120301127
17. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits. In: ESORICS. LNCS, vol. 8134. Springer (2013)
18. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Springer Berlin Heidelberg (2012)
19. Frederiksen, T.K., Keller, M., Orsini, E., Scholl, P.: A unified approach to mpc with preprocessing using ot. In: Proceedings, Part I, of the 21st International Conference on Advances in Cryptology – ASIACRYPT 2015 - Volume 9452. Springer-Verlag, Berlin, Heidelberg (2015)
20. Gao, S., Mateer, T.: Additive fast fourier transforms over finite fields. IEEE Transactions on Information Theory (Volume: 56, Issue: 12, December 2010) **56**(12) (2010). https://doi.org/10.1109/TIT.2010.2079016, https://doi.org/10.1109/TIT.2010.2079016
21. Gentleman, W.M., van der Sande, G.: Fast fourier transforms: for fun and profit. In: AFIPS '66 (Fall) (1966), https://api.semanticscholar.org/CorpusID:207170956
22. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology – EUROCRYPT 2014. Springer Berlin Heidelberg (2014)
23. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM (2019)
24. Hazay, C., Scholl, P., Soria-Vazquez, E.: Low cost constant round mpc combining bmr and oblivious transfer. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. Springer International Publishing, Cham (2017)
25. Heyse, S., Kiltz, E., Lyubashevsky, V., Paar, C., Pietrzak, K.: Lapin: An efficient authentication protocol based on ring-lpn. In: Canteaut, A. (ed.) Fast Software Encryption. Springer Berlin Heidelberg (2012)
26. Keller, M.: Mp-spdz: A versatile framework for multi-party computation. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA (2020)
27. Keller, M., Orsini, E., Scholl, P.: Mascot: Faster malicious arithmetic secure computation with oblivious transfer. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. p. 830–842. Association for Computing Machinery, New York, NY, USA (2016)
28. Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Improved garbled circuit building blocks and applications to auctions and computing minima. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) Cryptology and Network Security. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
29. Lidl, R., Niederreiter, H.: Introduction to finite fields and their applications. Cambridge University Press (1986)
30. Longa, P., Naehrig, M.: Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In: Foresti, S., Persiano, G. (eds.) Cryptology and Network Security. Springer International Publishing, Cham (2016)
31. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Springer Berlin Heidelberg (2012)

32. Rader, C.: Discrete fourier transforms when the number of data samples is prime. Proceedings of the IEEE **56**(6), 1107–1108 (1968)
33. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM (2009)
34. Rieder, V.: Generation of authenticated secret-shared scaled unit vectors for beaver triples. In: Selected Areas in Cryptography – SAC 2024. Springer-Verlag, Berlin, Heidelberg (2025)
35. Rieder, V.: Silentium: Implementation of a pseudorandom correlation generator for beaver triples. In: Applied Cryptography and Network Security Workshops. Springer-Verlag, Berlin, Heidelberg (2025)
36. Trifonov, P.V., Fedorenko, S.V.: A method for fast computation of the fourier transform over a finite field. Problems of Information Transmission 39 (2003)
37. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated ot with small communication. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA (2020)

A The ring-LPN Assumption

For the formulation of the ring-LPN assumption over the ring $\mathcal{R} = \mathbb{F}[X]/F$, let $\mathrm{negl}(\lambda)$ denote any negligible function (that grows slower than any inverse polynomial).

We formulate the ring-LPN assumption in its general case. For Π_{Bt} we from now on assume that $\mathbb{F}$ is fully reducible with distinctive roots over $\mathbb{F}$, i.e. $\mathcal{R} \cong \mathbb{F}^N$, which is a necessary requirement to map the SUVs into N distinct Beaver triples over $\mathbb{F}$. For the details how the ring-LPN assumption applies to Π_{Bt} in Section Protocol Protocol 1, and for a security analysis we refer to previous work [10].

Lemma 1 (ring-LPN assumption [10]). *Let λ be a security parameter and $c \geq 2$ be an integer. Let $\mathcal{R}(\lambda) = \mathcal{R} = \mathbb{F}[X]/F$, where F has degree N and let $t(\lambda) = t$ be another integer (noise rate). Let $\mathcal{HW}_t$ be the distribution of sparse polynomials over $\mathcal{R}$ that samples t noise positions $\alpha_i \xleftarrow{\$} [0, N)$, t payloads $A_i \xleftarrow{\$} \mathbb{F}$ and outputs the polynomial $\sum_{i=0}^{n-1} A_i \cdot X^{\alpha_i}$. Then, we say that the $\mathcal{R}^c - LPN_t$ problem is hard if for every nonuniform polynomial-time distinguisher $\mathcal{A}$, it holds that*

$$|\Pr\big(\mathcal{A}(a, \langle a, e\rangle + f) = 1\big) - \Pr\big(\mathcal{A}(a, u) = 1\big)| \leq \mathrm{negl}(\lambda)$$

where the probabilities $\Pr(\cdot)$ are taken over $a \leftarrow \mathcal{R}^{c-1}, u \leftarrow \mathcal{R}, e \leftarrow \mathcal{HW}_t^{c-1}, f \leftarrow \mathcal{HW}_t$.

B NTT methods

The following provides a description of how we apply the Karatsuba method. Let $f, g \in \mathcal{R}$, with degrees $\deg(f) = \deg(g) = n = ks$ for some integers k and s. Then we can uniquely write

$$f(x) = f_0(x) + f_1(x)x^s + \cdots + f_{k-1}(x)x^{(k-1)s} \tag{4}$$

$$g(x) = g_0(x) + g_1(x)x^s + \cdots + g_{k-1}(x)x^{(k-1)s} \tag{5}$$

for some $f_i, g_j \in \mathcal{R}$, with $\deg(f_i), deg(g_j) < k$. When computing fg by multiplying $f_i g_j$ by $0 \leq i, j \leq k-1$, terms of the form $f_i g_j + f_j g_i$ occur. For such terms, we can observe that

$$f_i g_j + f_j g_i = (g_i + g_j)(f_i + f_j) - g_i f_i - g_j f_j. \tag{6}$$

Since the terms $g_i f_i - g_j f_j$ naturally appear in the computations of other coefficients of the product, we can compute $g_i f_j + g_j f_i$ at the cost of computing $(g_i + g_j)(f_i + f_j)$, thus avoiding a multiplication. We are basically converting a degree n polynomial multiplication into C degree k multiplications, where $C = p^2 - \binom{p}{2} = \frac{p^2+p}{2}$. This can be easily checked, since with the Karatsuba method we avoid a multiplication for each crossed product with respect to the quadratic approach.

C Proof and details on Π_{SUV}

C.1 Tree Phase of Π_{SUV}

For completeness we present the tree phase of Π_{SUV}, which is identical to previous work, in $\Pi^{\text{field}}_{\text{SUV}}$, see Protocol Protocol 2. In short, the field phase [35], The differences for $\Pi_{\text{SUV}}, \Pi_{\text{Gen}}, \Sigma_{\text{Eval}}$ are in the payload correction step, where:

- $\Pi^{\text{field}}_{\text{SUV}}$ interactively computes correction words CW^L, CW^R given the random payloads $[u^L], [u^R]$ of t^L, t^R and given $[A], [\text{m}]$. These correction words are such that the parties can locally correct the payloads of t^L, t^R by scalar-vector-multiplications. This finally defines the output SUV $[x]$
- $\Pi^{\text{field}}_{\text{Gen}}$ computes CW^L, CW^R as $\Pi^{\text{field}}_{\text{SUV}}$ and outputs these values instead of an SUV
- Σ_{Eval} takes CW^L, CW^R as input (provided by $\Pi^{\text{field}}_{\text{Gen}}$) and computes $[x]$ as $\Pi^{\text{field}}_{\text{SUV}}$.

We stress that Π^*_{SUV} and Σ^*_{Eval} differ from $\Pi_{\text{SUV}}, \Sigma_{\text{Eval}}$ only in the last payload correction step, where CW^L is identified with u^L which saves some interactive computations [35].

C.2 Correctness Proof of the Tree Phase

We stress that the bug we fix in this work only affects the correctness and not the security. We only address the correctness, since the bug in the initial description of Π_{SUV} is continues to be a bug in the UC-security proof for Π_{SUV}[10]. This proof is in the UC-model and states that $\Pi_{\text{c-SUV}}$ implements a certain ideal functionality. It is not the functionality itself that is broken but rather the proof. Hereby, the respective step to proof correctness is skipped. We now fill this gap of the security proof, while not taking care about the other, unaffected parts. Especially, we only need to cover passive security for the tree phase, since the detection of malicious behavior is covered by field phase, which we do not modify. We only need to show that Π_{SUV} is correct with respect to the properties of the generated vectors t^L_σ, t^R_σ. Concretely, to ensure that they are secret-shared unit vectors, it is sufficient to prove the following:

Algorithm 5 Field phases $\Pi_{\text{SUV}}^{\text{field}}$

INPUT: The outputs t_σ^L, t_σ^R from the tree phase and the secret-shared payload $[A]$ and MAC $[\mathrm{m}]$. Furthermore let $[a^L], [b^L], [c^L]$ and $[a^R], [b^R], [c^R]$ be two Beaver triples, e.g., $c^L = a^L \cdot b^L$.

4. VERIFICATION

   Let $u_\sigma^L = \sum_{j=0}^{M-1} t_\sigma^{L,j}$ and $u_\sigma^R = \sum_{j=0}^{M-1} t_\sigma^{R,j}$

   The parties exchange $u_\sigma^L - a_\sigma^L$ to compute $\epsilon^L = (u_0^L - a_0^L) + (u_1^L - a_1^L)$
   The parties exchange $u_\sigma^R - a_\sigma^R$ to compute $\epsilon^R = (u_0^R - a_0^R) + (u_1^R - a_1^R)$

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

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

   Let $v_\sigma^L = \sum_{j=0}^{M-1} r^j \cdot t_\sigma^{L,j}$ and $v_\sigma^R = \sum_{j=0}^{M-1} r^j \cdot t_\sigma^{R,j}$

   The parties exchange $v_\sigma^L - b_\sigma^L$ to compute $\delta^L = (v_0^L - b_0^L) + (v_1^L - b_1^L)$
   The parties exchange $v_\sigma^R - b_\sigma^R$ to compute $\delta^R = (v_0^R - b_0^R) + (v_1^R - b_1^R)$

   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

5. 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 $[x]$ consisting of an additive share $\mathrm{CW}^L \cdot [t^L]$ and MAC share $\mathrm{CW}^R \cdot [t^R]$

Lemma 2. *If the parties follow the protocol description honestly the values* $s_\sigma^{L,j}, s_\sigma^{R,j} \in \mathbb{F}$ *are indistinguishable from uniformly random under the condition that*

$$t_0^{L,j} + t_1^{L,j} = 0 \text{ and } t_0^{R,j} + t_1^{R,j} = 0 \quad \Leftrightarrow \quad j \neq \alpha \tag{7}$$

Proof. We start to prove the following claims about the output of the loop through the tree levels (step 2). For all $0 \leq k < 2^i$ it holds:

1. $T_0^{m,k} = T_1^{m,k}$ if and only if $k \neq \alpha$.
2. If $k \neq \alpha$, then $s_0^{m,k} = s_1^{m,k}$.

We prove the claims by induction, for general $m \in \mathbb{N}$.

Induction start (initial round, $m = 0$): For $m = 0$ the loop in step 2 is empty and it holds $\alpha = 0, j = 0$. Hence the first claim reduces to the statement that $T_0^{0,0} \neq T_1^{0,0}$ and $s_0^{0,0} \neq s_1^{0,0}$ except negligible probability. Both properties are satisfied according to the setup step 1.

Induction step: For the induction step, we always parse $k = 2j + b$ where $b \in \{0, 1\}$ and $0 \leq j < 2^i$ as used in the last two statements of the loop in step 2. Assume that all claims are true for $m - 1$. To prove the claims for m, note that $m - 1$ refers to a truncated protocol call with $m - 1$ calls of the tree loop and with respect to the position $\lfloor \frac{\alpha}{2} \rfloor$. We split the induction step into two parts. The first shows claim 1 for m, given all previous claims for $m-1$. The seconds proofs claims 2 for m, given claims 2 for $m-1$ and the previous result on claim 1 for m.

Induction step for claim 1: To prove claim 1 we prove the equivalent statement that

$$T_0^{m,2j+b} \oplus T_1^{m,2j+b} = 0 \quad \Leftrightarrow \quad 2j + b \neq \alpha.$$

By definition of $T_\sigma^{m,2j+b}$ this becomes

$$t_0^{m,2j+b} \oplus t_1^{m,2j+b} \oplus (T_0^{m-1,j} \oplus T_1^{m-1,j}) \cdot \tau^{m,b} = 0 \quad \Leftrightarrow \quad 2j + b \neq \alpha.$$

Furthermore, by the definition of $\tau^{m,b}$ following $\mathbf{PassiveOutput}_1$, where we can assume honest behavior, we need to consider the following term:

$$t_0^{m,2j+b} \oplus t_1^{m,2j+b} \oplus (T_0^{m-1,j} \oplus T_1^{m-1,j}) \cdot (u_0^{m,b} \oplus u_1^{m,b} \oplus \alpha_0 \oplus 1 \oplus b)$$

By definition of $u_\sigma^{m,b}$ this becomes

$$t_0^{m,2j+b} \oplus t_1^{m,2j+b} \oplus (T_0^{m-1,j} \oplus T_1^{m-1,j}) \cdot \Big(\bigoplus_{j=0}^{2^{m-1}-1} (t_0^{m,2j+b} \oplus t_1^{m,2j+b}) \oplus \alpha_0 \oplus 1 \oplus b \Big)$$

By induction on claim 2 it holds $s_0^{m-1,j} = s_1^{m-1,j}$ whenever $j \neq \lfloor \frac{\alpha}{2} \rfloor$. This implies that $t_0^{m,2j+b} = t_1^{m,2j+b}$ whenever $j = \lfloor \frac{\alpha}{2} \rfloor$ such most terms $t_0^{m,2j+b} \oplus t_1^{m,2j+b}$ cancel out. Hence, for the first induction step, we need to show that

$$t_0^{m,2j+b} \oplus t_1^{m,2j+b} \oplus (T_0^{m-1,j} \oplus T_1^{m-1,j}) \cdot (t_0^{m,2j+b} \oplus t_1^{m,2j+b} \oplus \alpha_0 \oplus 1 \oplus b) \tag{8}$$

is zero if and only if $k = 2j + b \neq \alpha$.

- If $k \neq \alpha$, than there are two cases:
  - If $j \neq \lfloor \frac{\alpha}{2} \rfloor$, than by induction for claim 1 it holds $T_0^{m-1,j} = T_1^{m-1,j}$ and hence Equation Equation Equation 8 reduces to $t_0^{m,2j+b} \oplus t_1^{m,2j+b}$. However by induction on claim 2 it holds $s_0^{m-1,j} = s_1^{m-1,j}$ which implies that $t_0^{m,2j+b} = t_1^{m,2j+b}$. Hence Equation 8 is zero as intended.
  - If $j = \lfloor \frac{\alpha}{2} \rfloor$, than by induction for claim 1 it holds $T_0^{m-1,j} \neq T_1^{m-1,j}$ and hence Equation 8 becomes $t_0^{m,2j+b} \oplus t_1^{m,2j+b} \oplus t_0^{m,2j+b} \oplus t_1^{m,2j+b} \oplus \alpha_0 \oplus 1 \oplus b) = \alpha_0 \oplus 1 \oplus b$. Note that $k \neq \alpha$ and $j = \lfloor \frac{\alpha}{2} \rfloor$ implies that $\alpha_0 \neq b$. Hence $\alpha_0 \oplus 1 \oplus b = 0$, i.e., 8 is zero as intended.
- If $k = \alpha$, than $j = \lfloor \frac{\alpha}{2} \rfloor$. Hence by induction on claim 1, we know that $T_0^{m-1,j} \neq T_1^{m-1,j}$ and 8 becomes $t_0^{m,2j+b} \oplus t_1^{m,2j+b} \oplus t_0^{m,2j+b} \oplus t_1^{m,2j+b} \oplus \alpha_0 \oplus 1 \oplus b) = \alpha_0 \oplus 1 \oplus b$. Note that $k = \alpha$ implies that $\alpha_0 = b$. Hence $\alpha_0 \oplus 1 \oplus b = 1$, i.e., 8 is 1 as intended.

Induction step for claim 2: For the claims 2, 3 we need consider the equivalent statement that $s_0^{i,2j+b} \oplus s_1^{i,2j+b}$ is zero if $k = 2j + b = \alpha$ and 1 else (except negligible probability in λ). More concretely, by definition of $s_\sigma^{m,2j+b}$ we need to consider the term $\bar{s}_0^{m,2j+b} \oplus \bar{s}_1^{m,2j+b} \oplus (T_0^{m,2j+1-b} \oplus T_1^{m,2j+1-b}) \cdot \mathrm{CW}^i$. Hereby, by definition of $\mathbf{PassiveOutput}_\lambda$, in the honest case, it holds $\mathrm{CW}^i = y_0 \oplus y_1 \oplus z_0^{m,1} \oplus z_1^{m,1}$ and furthermore by definition of **MixedMult**, in the honest case, it holds $\mathrm{CW}^i = \alpha_0 \cdot (z_0^{m,0} \oplus z_0^{m,1} \oplus z_1^{m,0} \oplus z_1^{m,1}) \oplus z_0^{m,1} \oplus z_1^{m,1} = z_0^{m,1-\alpha_0} \oplus z_1^{m,1-\alpha_0}$. By induction on claim 1 it holds $s_0^{m-1,j} = s_1^{m-1,j}$ whenever $j \neq \lfloor \frac{\alpha}{2} \rfloor$, which implies that $\bar{s}_0^{m-1,j} = \bar{s}_1^{m-1,j}$ whenever $j \neq \lfloor \frac{\alpha}{2} \rfloor$. Hence similar to the u-values in the induction step for claim 1, most terms in the summands for $z_0^{m,1-b}, z_1^{m,1-b}$ cancel out, such that we $\mathrm{CW}^i = \bar{s}_0^{m,2j+1-b} \oplus \bar{s}_1^{m,2j+1-b}$. Altogether, we need to show that the term

$$\bar{s}_0^{m,2j+b} \oplus \bar{s}_1^{m,2j+b} \oplus (T_0^{m,2j+1-b} \oplus T_1^{m,2j+1-b}) \cdot (\bar{s}_0^{m,2j+1-\alpha_0} \oplus \bar{s}_1^{m,2j+1-\alpha_0}) \quad (9)$$

is zero if and only if $k \neq \alpha$, except negligible probability for the direction $k = \alpha$. We distinguish two cases:

- If $j \neq \lfloor \frac{\alpha}{2} \rfloor$, than $2j+1-b \neq \alpha$ and by claim 1 it holds $T_0^{m,2j+1-b} = T_1^{m,2j+1-b}$ and hence 9 reduces to $\bar{s}_0^{m,2j+b} \oplus \bar{s}_1^{m,2j+b}$. However by induction on claim 2 it holds $s_0^{m,j} = s_1^{m-1,j}$ which implies that $\bar{s}_0^{m,2j+b} = \bar{s}_1^{m,2j+b}$. Hence 9 is zero as intended.
- If $j = \lfloor \frac{\alpha}{2} \rfloor$, than $2j+1-b = \alpha$ (since $k = 2j+b \neq \alpha$). Hence by claim 1 it holds $T_0^{m,j} \neq T_1^{m-1,j}$ and 9 becomes $\bar{s}_0^{m,2j+b} \oplus \bar{s}_1^{m,2j+b} \oplus \bar{s}_0^{m,2j+1-\alpha_0} \oplus \bar{s}_1^{m,2j+1-\alpha_0}$. Note that $k \neq \alpha$ and $j = \lfloor \frac{\alpha}{2} \rfloor$ implies that $\alpha_0 \neq b$, i.e., $1 - \alpha_0 = b$, such that 8 is zero as intended.

Correctness of the conversion to $\mathbb{F}^M$: To conclude the proof note that 7 is a direct consequence of claim 2 and the definition of t_σ^L, t_σ^R. For the pseudorandomness note that Conv is a PRG, such that it is sufficient to have pseudorandomness for the values $s_\sigma^{m,j}$. Thisis however ensured by the properties of the

Table 1. Running time in seconds (sec) and communication in KB per party for different instantiations and parameter regimes in Silentium. The parameter are as in [35] and the instantiation differ in the target field $\mathbb{F}_{128}, F_{2^{80}}$ and the intern binary MPC version.

Parameters						$\mathbb{F}_{128}$ triples (TinyOT)		$\mathbb{F}_{128}$ triple (SPDZ)		$\mathbb{F}_{2^{80}}$ triples (SPDZ)	
λ_0	d	c	b	t	mode	sec	KB	sec	KB	sec	KB
80	32	8	4	1	2b	372	791	392	1221	434	1099
128	40	8	1	5	2b	463	1224	460	1888	490	1699
80	40	4	2	5	2b	289	1189	284	1820	241	1632
80	32	8	2	2	2b	287	769	317	1177	381	1056
80	32	8	4	1	1	175	747	177	1134	246	1012
80	32	8	4	1	2b	252	747	281	1134	348	1012
128	64	4	8	2	2b	193	2826	198	4245	179	3768
128	64	4	16	1	2b	126	2740	134	4075	134	3598

PRG and definition of CW^i, for details, which are conceptually not affected by the bug we fix, we refer to the initial proof of Π_{SUV} [10].

C.3 Variations of Π_{SUV} in Silentium

The Silentium [35] also covers a separation of Π_{SUV} into an seed generation protocol Π_{Gen} (which outputs small seed instead of an SUV) and a Σ_{Gen} to locally exapnd the seed into the private SUV share. This can be used to improve the deployment of the Bt-PCG.

Our correction and enhancement can be directly shifted to Π_{Gen} and Σ_{Eval}. The point is, that Π_{Gen} only differns from Π_{SUV} in the format of the output and that Σ_{Eval} localizes the operations from Π_{SUV}, by taking seed values as replacement for the interactive inputs. Especially, in Silentium Π_{Gen} and Σ_{Eval} are correct in the sense of our proof Section C.2.

D Details on the Evaluation

This section extends the evualtion from Section 6 with insights, depending on previous work [35,34]

D.1 Absolute numbers for the Bt-PCG

Table 1 contains absolute numbers for running time and communication costs of the Bt-PCG. The numbers are on two powerful local Intel servers with Intel Xeon Gold 6342 processor. The number are given to back the claims in Sections 6.1, 6.2 with absolute values.

For the parameter regime, we refer to previous work [35]. For short, λ_0 is a security parameter (for internal PRG), $d = c \cdot b \cdot t$ are internal parameters setup by the ring-LPN assumption and modus is a compile time selection to parallelize operations. The number of Beaver triples $N = 2^n = 2^{20}$, i.e. the size of the SUV

vectors and NTT degree is fix, apart that for the binar field the NTT size is $\tilde{N} = 2^{20} - 1$. We stress that these choices for the binary version are not backed by a security analysis of the ring-LPN assumption, but are taken over form the recommendations for the 128-bit field [10]. We stress that future changes on the parameterts can be configured in Silentium with little effort, since the implementation takes care about thes uncertainty be beeing configurable [35].

The numbers refer to the direct PCG version Section Section C.3, but without the costs for the fake (npreprocessing in Silentium (insecure to simulate the generation of correlated randomness in a naive way). We exclude the running time for this, since with the Beaver tripe expansion we come up with a proper solution for preprocessing.

D.2 Analysis of the communication costs

To explain the factor of 1.5 (resp. 1.3 for the binary field triples in increased communication costs for when switching form internal MPC with TinyOT to internal MPC with SPDZ, we provide the formulas for the communication costs in bits. These values depend on λ, ν and m, where $m = n - b$ (for details on the regular variant see previous work [35])

For the version with TinyOT, the communication costs for generation a small SUVs (Π^*_{SUV}), as implemented in Silentium is $m(2 \cdot \lambda + 2) + 13\nu$. There a $2d$ invocations. For the large SUV, an invocation of Π_{SUV} takes $(m+1)\cdot(2\lambda+2)+17\nu$ bits. There are d^2 invocations. Finally, each of the d^2 correlation steps for the positions and payloads takes $2(m + \nu)$ bits. This is given by m AND operations (m mult. in the integer addition gate, with 2 bits per AND with TinyOT) and one Beaver multiplication each.

For the version with binary SPDZ, the communication costs for generation a small SUVs (Π^*_{SUV}), as implemented in Silentium is $m(4\lambda+2)+13\nu$. There a $2d$ invocations. For the large SUV, an invocation of Π_{SUV} takes $(m+1)\cdot(2\lambda+2)+17\nu$ bits. There are d^2 invocations. Finally, each of the d^2 correlation steps for the positions and payloads takes $2(m\lambda+\nu)$ bits. This is given by m AND operations (m mult. in the integer addition gate, with 2λ bits AND with SPDZ) and one Beaver multiplication.

Altogether

$$C_{\text{TinyOT}} = 2dm + 2dm\lambda + 13d\nu + 2d^2 + 4d^2m + 2d^2\lambda + 2d^2m\lambda + 19d^2\nu \quad (10)$$

$$C_{\text{SPDZ}} = 2dm + 4dm\lambda + 13d\nu + 2d^2 + 2d^2m + 2d^2\lambda + 4d^2m\lambda + 19d^2\nu \quad (11)$$

Since d^2 dominates d, the difference can be explained by the terms d^2m and $d^2m\lambda$. For our parameter regimes, this middles to the increase of 1.5 (resp. 1.3) for the version with SPDZ.