Improved Multiplication Triple Generation over Rings via RLWE-based AHE (Full Version)*

Deevashwer Rathee[1], Thomas Schneider[2], and K. K. Shukla[1]

[1] Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi, India
{deevashwer.student.cse15,kkshukla.cse}@iitbhu.ac.in
[2] Department of Computer Science,
Technische Universität Darmstadt, Germany
schneider@encrypto.cs.tu-darmstadt.de

Abstract. An important characteristic of recent MPC protocols is an input-independent setup phase in which most computations are offloaded, which greatly reduces the execution overhead of the online phase where parties provide their inputs. For a very efficient evaluation of arithmetic circuits in an information-theoretic online phase, the MPC protocols consume Beaver multiplication triples generated in the setup phase. Triple generation is generally the most expensive part of the protocol, and improving its efficiency is the aim of our work.
We specifically focus on computation over rings of the form $\mathbb{Z}_{2^\ell}$ in the semi-honest model and the two-party setting, for which an Oblivious Transfer (OT)-based protocol is currently the best solution. To improve upon this method, we propose a protocol based on RLWE-based Additively Homomorphic Encryption. Our experiments show that our protocol is more scalable, and it outperforms the OT-based protocol in most cases. For example, we improve communication by up to 6.9x and runtime by up to 3.6x for 64-bit triple generation.

Keywords: Secure Two-party Computation · Beaver Multiplication Triples · Ring-LWE · Additively Homomorphic Encryption

1 Introduction

Secure multi-party computation (MPC) allows a set of distrusting parties to jointly compute a function on their inputs while keeping them private from one another. There is a multitude of MPC protocols such as [DPSZ12,KOS16,DSZ15] that allow secure evaluation of arithmetic circuits, which form the basis of many privacy-preserving applications. An important characteristic of many of the recent MPC protocols is an input-independent setup phase in which most computations are offloaded, which greatly reduces the execution overhead of the online phase where parties provide their inputs. The idea is to compute Beaver multiplication triples [Bea91] in the setup phase, and then use them to evaluate arithmetic circuits very efficiently in an information-theoretic online phase, without using any cryptographic operations. In light of their significance on the overall runtime of the protocol, the main focus of this work is efficient generation of such triples in the semi-honest setting.

In the malicious model and the multi-party setting, the first to employ RLWE-based Somewhat Homomorphic Encryption (SHE) for triple generation were [DPSZ12] in 2012. Their major source of efficiency was the packing method from [SV14]. In 2016, this method was replaced by an Oblivious Transfer (OT)-based method by Keller et al. [KOS16]. Later in 2017, SHE emerged again with the Overdrive methodology [KPR18]. These protocols were designed to generate triples over a finite field which can only be used to support finite field arithmetic in the online phase. In 2018, Cramer et al. [CDE+18] proposed an OT-based protocol that generates triples over rings of the form $\mathbb{Z}_{2^\ell}$. Designing protocols over rings is useful in a lot of applications since it greatly simplifies implementation of comparisons and bitwise operations, which are inefficient to realize with finite field arithmetic. Apart from this, using ring-based protocols also implies that we

* Please cite the conference version of this paper published at CANS'19 [RSS19].

Functionality $\mathcal{F}^n_{\mathsf{Triple}}$: Sample values $\mathbf{a}_0, \mathbf{a}_1, \mathbf{b}_0, \mathbf{b}_1, \mathbf{r} \leftarrow\!\!\$\, (\mathbb{Z}_{2^\ell})^n$. Output tuples $(\mathbf{a}_0, \mathbf{b}_0, (\mathbf{a}_0 + \mathbf{a}_1) \cdot (\mathbf{b}_0 + \mathbf{b}_1) + \mathbf{r})$ and $(\mathbf{a}_1, \mathbf{b}_1, -\mathbf{r})$ to P_0 and P_1, respectively, where arithmetic is performed modulo 2^ℓ.

Fig. 1: Functionality for generating Beaver multiplication triples

can leverage some special tricks that computers already implement to make integer arithmetic very efficient. In 2019, Orsini et. al [OSV19] presented a more compact solution based on SHE and argued that it is more efficient than the OT-based protocol of [CDE+18]. Concurrently, Catalano et. al. [CRFG19] used the Joye-Libert homomorphic cryptosystem [JL13] to improve upon the communication of [CDE+18] particularly for larger choices of ℓ.

Our Contributions. In this paper, we consider the semi-honest model and the two-party setting, for which the current best method for generating triples over rings is the OT-based approach of [DSZ15]. Taking inspiration from the changing trend in the malicious model, we propose a protocol based on RLWE-based Additively Homomorphic Encryption (RLWE-AHE) that improves upon the OT-based solution. In the process, we analyze the popular approaches for triple generation using AHE and adapt them to using state-of-the-art RLWE-AHE and our scenario. We also argue why the approach taken in [OSV19] does not provide the most efficient solution in our semi-honest setting. Our experiments show that our protocol is more scalable, and it outperforms the OT-based protocol in most cases. For example, we improve communication over [DSZ15] by up to 6.9x and runtime by up to 3.6x for 64-bit triple generation.

2 Preliminaries

2.1 Notation

We denote the players as P_0 and P_1. κ denotes the symmetric security parameter, σ the statistical security parameter, and λ the computational security parameter. $\langle x \rangle$ is a shared value of $x \in \{0,1\}^\ell$, which is a pair of ℓ-bit shares $(\langle x \rangle_0, \langle x \rangle_1)$, where the subscript represents the party that holds the share. A vector of shares is represented in bold face e.g. $\langle \mathbf{x} \rangle$, and multiplication, denoted by $\cdot$, is performed component-wise on it. To represent an element x being sampled uniformly at random from G, we use the notation $x \leftarrow\!\!\$\, G$. Assignment modulo 2^ℓ is denoted by $\leftarrow_\ell$.

2.2 Problem Statement

A Beaver multiplication triple [Bea91] is defined as the tuple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ satisfying:

$$(\langle a \rangle_0 + \langle a \rangle_1) \cdot (\langle b \rangle_0 + \langle b \rangle_1) \equiv (\langle c \rangle_0 + \langle c \rangle_1) \bmod 2^\ell.$$

Our aim is to construct a two-party protocol that securely realizes the $\mathcal{F}^n_{\mathsf{Triple}}$ functionality which is defined in Fig. 1.

2.3 Security Model

Our protocol is secure against a semi-honest and computationally bounded adversary. This adversary tries to learn information from the messages it sees during the protocol execution, without deviating from the protocol. It is not allowed to deviate from the protocol execution.

2.4 Ring-LWE-based Additively Homomorphic Encryption (RLWE-AHE)

We use an IND-CPA secure AHE scheme with the following 5 algorithms:

- $\mathsf{KeyGen}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$: Key Generation is a randomized algorithm that outputs the key pair $(\mathsf{pk}, \mathsf{sk})$, with public key pk and secret key sk. We consider a single key pair $(\mathsf{pk}, \mathsf{sk})$ throughout the entire paper.
- $\mathsf{Enc}(\mathsf{pk}, \mathbf{m}) \rightarrow \mathsf{ct}$: Encryption is a randomized algorithm that takes a vector $\mathbf{m} \in (\mathbb{Z}_p)^n$ as input, where n depends on scheme parameters m and p (cf. §4.1), along with pk, and outputs a ciphertext ct. We assume that all ciphertexts in the following description of the scheme are encrypted with public key pk.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \rightarrow \mathbf{m}$: Decryption takes the secret key sk and a ciphertext ct, and outputs the plaintext $\mathbf{m} \in (\mathbb{Z}_p)^n$.
- $\mathsf{Add}(\mathsf{pk}; \mathsf{ct}_1, \mathsf{ct}_2) \rightarrow \mathsf{ct}'$: Addition takes as input two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$ and the public key pk, and outputs a ciphertext ct' such that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') = \mathbf{m}_1 + \mathbf{m}_2 \in (\mathbb{Z}_p)^n$, where addition is performed component-wise. This algorithm is also denoted by the $\oplus_{\mathsf{pk}}$ operator.
- $\mathsf{ScalarMult}(\mathsf{pk}; \mathsf{ct}, \mathbf{s}) \rightarrow \mathsf{ct}'$: Given inputs ciphertext ct and scalar $\mathbf{s}$, and the public key pk, scalar-multiplication outputs a ciphertext ct' such that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \cdot \mathbf{s} \in (\mathbb{Z}_p)^n$, where multiplication is performed component-wise. This algorithm is also denoted by the $\odot_{\mathsf{pk}}$ operator.

Possible instantiations of RLWE-based schemes that satisfy the description above are [FV12,BGV12]. These schemes are IND-CPA secure, and their security relies on the Decision RLWE assumption [LPR10]. We assume that the parameters of the scheme have been chosen to be large enough to allow evaluation of the circuit for our triple generation protocol and accommodate the extra noise added to prevent leakage through ciphertext noise (cf. §4.3).

3 Previous Works

The previous approaches for generating multiplication triples in the semi-honest model are based on AHE and OT. Initially, Beaver triples were generated using AHE schemes such as Paillier [Pai99] and DGK [DGK08]. However, the authors in [DSZ15] showed that the OT-based generation method greatly outperforms the AHE-based generation, and is currently the best method. In this section, we summarize both approaches. Although the protocols based on AHE are much slower, they are the basis for our proposed protocol.

3.1 AHE-based Generation

Case I - $2^\ell | p$. Fig. 2 describes a well-known protocol for generating triples using AHE [PBS12]. This protocol generates multiplication triples in $\mathbb{Z}_{2^\ell}$, using an AHE scheme with plaintext modulus p, and it works if and only if $2^\ell | p$. This is due to the fact that the AHE scheme implicitly reduces the underlying plaintext modulo p. We can use the DGK cryptosystem [DGK08] since it uses a 2-power modulus.

Case II - $2^\ell \nmid p$. We start by choosing r from an interval such that $d = \langle a \rangle_0 \cdot \langle b \rangle_1 + \langle b \rangle_0 \cdot \langle a \rangle_1 + r$ does not overflow the bound p. This affects the security of the protocol as we no longer have information theoretic security provided by uniform random masking by r. To get around this issue, we resort to "smudging" [AJL+12], where we get statistical security of σ-bits by sampling r from an interval that is by factor 2^σ larger than the upper bound on magnitude of the expression $v = \langle a \rangle_0 \cdot \langle b \rangle_1 + \langle b \rangle_0 \cdot \langle a \rangle_1$. Since the upper bound on v is $2^{2\ell+1}$, we sample r from $\mathbb{Z}_{2^{2\ell+\sigma+1}}$. Consequently, the plaintext modulus p has to be of bitlength $2\ell + \sigma + 2$. This prevents the overflow and provides statistical security of σ-bits [PBS12]. We can instantiate this case with the Paillier cryptosystem [Pai99], whose plaintext modulus is the product of two distinct primes.

3.2 OT-based Generation

The feasibility result for triple generation over $\mathbb{Z}_{2^\ell}$ using Oblivious Transfer was given in [Gil99], and it was shown in [DSZ15] that it is the currently best method for triple generation in the semi-honest setting.

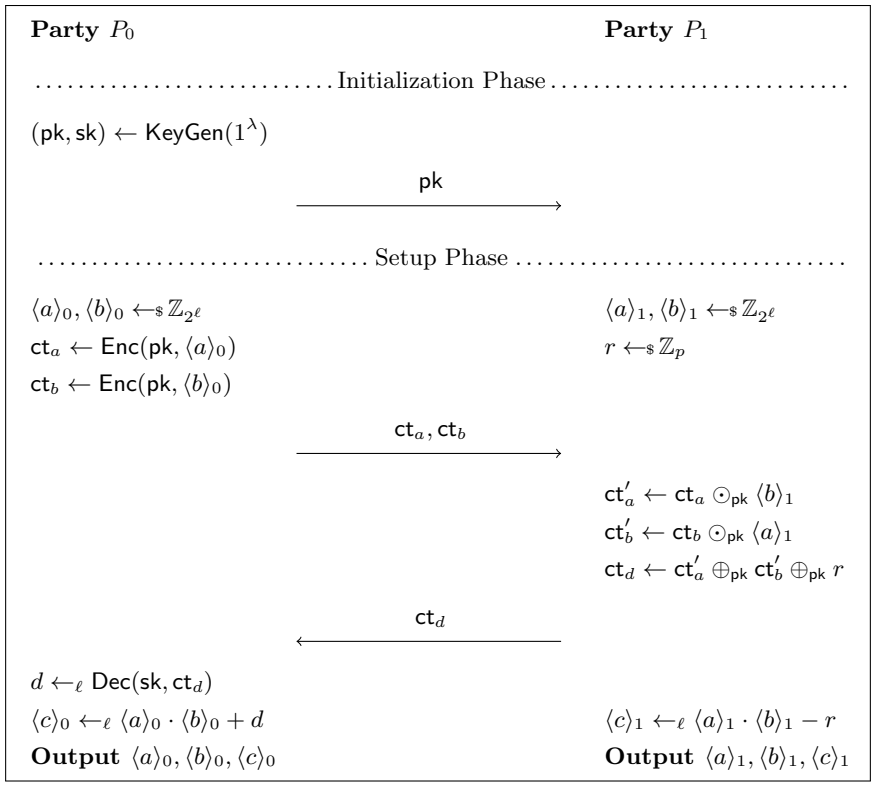


Fig. 2: $\Pi_{\mathsf{BasicTripleAHE}}$: Basic Beaver Triple Generation using AHE.

This protocol facilitates the triple generation by allowing secure computation of the product of two secret values. The amortized complexity of generating a triple in $\mathbb{Z}_{2^\ell}$ using OT-based generation is 2ℓ Correlated-OT (C-OT) over $(\ell+1)$/2-bit strings (cf. [DSZ15]). The protocol uses state-of-the-art C-OT extension (cf. [ALSZ13]) that requires $\kappa + \ell$-bit communication per C-OT on ℓ-bit strings.

4 RLWE-based Generation

In §3.1, we described two cases, namely $2^\ell | p$ and $2^\ell \nmid p$, and presented a protocol for both of them. While we can build a protocol based on our RLWE-AHE scheme that follows a similar design as in §3.1 for both cases, the two protocols are not equally efficient. In this section, we analyze these differences and show that the protocol for $2^\ell \nmid p$ is more efficient. Before comparing the cases, we detail two optimizations and a security consideration that are crucial for our analysis.

4.1 Batching Optimization

Using a RLWE-AHE scheme, we can generate many triples at the cost of generating one by leveraging the ciphertext packing technique described in [SV14]. For a prime p, we can encrypt a maximum of $n = \phi(m)/\mathsf{ord}_{\mathbb{Z}_m^*}(p)$ plaintexts $m_i \in \mathbb{Z}_p$ in a single ciphertext. The operations performed on a ciphertext are applied to all the slots of the underlying plaintext in parallel. As a result, in a single run of the protocol, we can generate n triples.

4.2 CRT Optimization

Using a very large plaintext modulus p results in inefficient instantiations since a larger p leads to a larger ciphertext modulus to contain the noise growth. Therefore, we use the CRT optimization to split the plaintext modulus p into e distinct primes p_i of equal bitlength such that $p = \prod_{i=1}^{i=e} p_i$ for some $e \in \mathbb{Z}$. We create e different instances of the cryptosystem for each p_i, and the whole protocol is performed for each instance. The plaintexts produced after decryption are combined using the Chinese Remainder Theorem (CRT) (with precomputed tables) to get the output in $\mathbb{Z}_p$. This technique also has the advantage that it can be parallelized in a straightforward manner.

4.3 Leakage through Ciphertext Noise

The ciphertexts of RLWE-based schemes have noise associated with them, whose distribution gets skewed on performing homomorphic operations on the ciphertext. This can lead to potential leakage through the noise, and reveal information in the case of scalar multiplication. A solution to this problem, called the noise flooding technique, was proposed in [Gen09]. This technique involves adding a statistically independent noise from an interval B' much larger than B, assuming that the ciphertext noise is bounded by B at the end of the computation. Specifically, this is done by publicly adding an encryption of zero with noise taken uniformly from $[-B', B']$ such that $B' > 2^{\sigma}B$, to provide statistical security of σ bits. We denote the encryption with noise from an interval $p \cdot 2^{\sigma}$ times larger than the normal encryption as Enc'.

4.4 Parameter Selection

The plaintext modulus p determines the protocol to be used as described in §3.1. After determining p, we can determine the other parameters to maximize efficiency as follows:

Case I - $2^{\ell} | p$: This approach was recently considered in [OSV19] for the malicious model. In order to generate authenticated triples in $\mathbb{Z}_{2^{\ell}}$, the authors required Zero Knowledge Proofs of Knowledge (ZKPoKs) and triples to be generated in $\mathbb{Z}_{2^{\ell+s}}$ to prevent a malicious adversary from modifying the triples with error probability $2^{-s+\log s}$. However in the semi-honest setting, the adversaries can not deviate from the protocol. Hence we do not require ZKPoKs, and computing triples in $\mathbb{Z}_{2^{\ell}}$ suffices. We start by choosing m to be a prime like in [OSV19] to ensure a better underlying geometry. Given that d is the order of 2 in $\mathbb{Z}_m^*$, we get $n = \phi(m)/d$ slots, each of which embeds a d-degree polynomial (cf. [SV14]). In case we naively utilize just the zero coefficient of the slot, even for small values of d, we are getting an order of magnitude fewer slots than the maximum possible value. In order to better utilize the higher coefficients of the polynomial embedded in each slot, we employ the packing method from [OSV19] to achieve a maximum utilization of $\phi(m)/5$ slots. Despite this significant optimization, most of the slots are wasted. Moreover, since p is a power of 2, we can not use the CRT optimization.

Case II - $2^{\ell} \nmid p$: Here, we choose m to be a power of 2 for efficiency reasons described in [CLP17], and big enough to provide security greater than 128-bits. Accordingly, we choose a prime plaintext modulus p of $2\ell + \sigma + 2$ bits that satisfies $p \equiv 1 \bmod m$, thereby maximizing the number of slots to $\phi(m)$. A concern of inefficiency here is that now our plaintext modulus is much larger than it was in the previous case. However, using the CRT optimization, we can split the plaintext modulus into e distinct primes p_i and get e instances of the cryptosystem with similar parameter lengths as in the previous case. A run of the protocol will require e times more computation and communication, but we can use the maximum number of slots. An important consideration here is that while we will have similar plaintext modulus and ciphertext modulus bitlengths, taking a 2-power m might result in an *at most* twice as large n than is required for 128-bit security. However with increasing n, the communication and computation increase only linearly and quasi-linearly respectively, and the number of triples generated increase linearly as well. Therefore, the amortized communication remains the same and the amortized computation increases *at most* by a factor of $\Delta = (\log(n) + 1)/\log(n)$, which is small for the minimum value of n typically required to maintain security

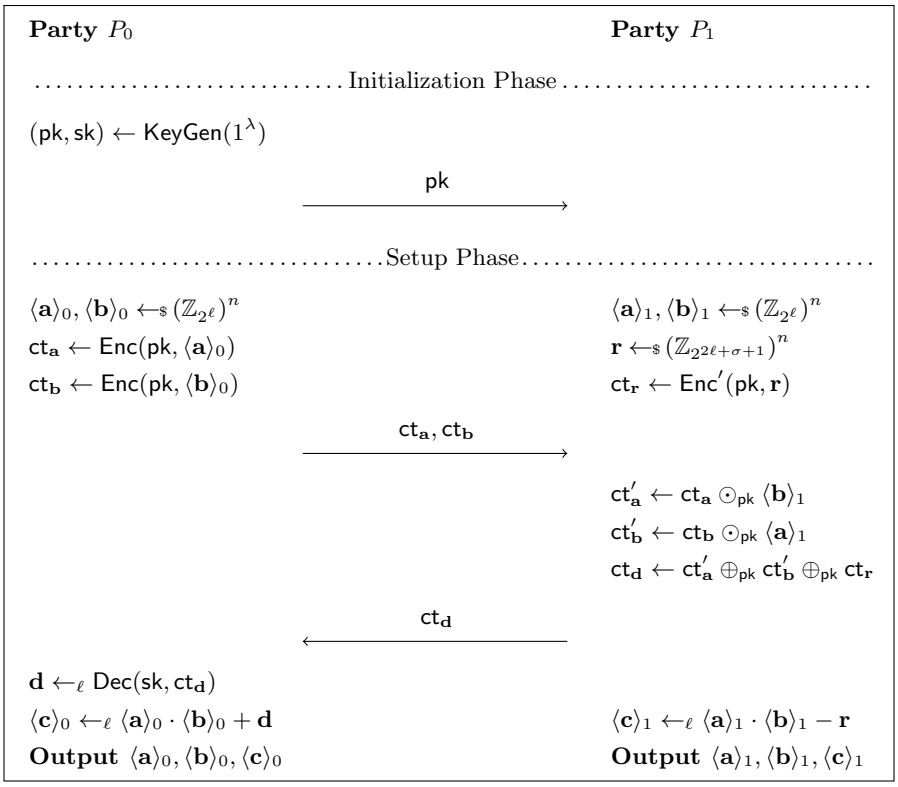


Fig. 3: $\Pi_{\mathsf{TripleRLWE}}$: Beaver Triple Generation using RLWE-AHE. Enc' denotes encryption with extra noise (cf. §4.3) and n denotes the number of plaintext slots (cf. §4.1).

(for $n = 4096, \Delta = 1.08$). Choosing $n = 8192$ instead of $n = 4096$, while keeping the other parameters unchanged, we experimentally observed a maximum slowdown in amortized runtime by 1.035x.

Conclusion: A single run of the protocol for Case I requires $e = (2\ell + \sigma + 2)/\ell$ times more computation and communication than Case II. However, the protocol for Case II requires *at least* 5 runs of the protocol to generate the same number of triples. Hence, considering $\sigma = 40$-bits and with the exception of small values of ℓ ($\ell \leq 15$), Case II is more efficient. Although we conclude that Case I could be better for smaller ℓ, we have implemented the protocol just for Case II because SEAL [CLP17], currently the most efficient publicly available library that satisfies the description of our RLWE-AHE scheme, only supports 2-power cyclotomics.

4.5 Our Final Protocol

Our final protocol is given in Fig. 3. In the protocol, we have shown an initialization phase for the generation of n triples. However, arbitrary many triples can be generated following a single initialization phase (involving a single key-pair). As discussed above, we have used the parameters for Case II with $2^\ell \nmid p$. Rather than drowning the ciphertext noise with a fresh encryption of zero with extra noise, we combine it with the step of adding r, and simply add a fresh encryption of r with extra noise. The advantage of using RLWE-AHE for generating triples is not only efficiency (cf. §5); we also get post-quantum security, unlike the OT-based approach which heavily relies on OT extension for efficiency.

Theorem 1. *The $\Pi_{\mathsf{TripleRLWE}}$ protocol (cf. Fig. 3) securely computes the $\mathcal{F}^n_{\mathsf{Triple}}$ functionality (cf. Fig. 1) in the presence of semi-honest adversaries, providing statistical security against a corrupted P_0 and computational security against a corrupted P_1.*

Table 1: Concrete scheme paramters for generating ℓ-bit triples using RLWE-AHE. n and q denote the number of plaintext slots and the ciphertext modulus respectively. The plaintext modulus is split into e co-prime p_i's using the CRT optimization (cf. §4.2).

ℓ	$\lfloor \log_2(q) \rfloor$	$\lfloor \log_2(p_i) \rfloor$	e	n
8	150	30	2	8192
16	170	38		
32	218	54		
64	190	44	4	

Proof. We first show that the output of the functionality $\mathcal{F}^n_{\mathsf{Triple}}$ and the output of the protocol $\Pi_{\mathsf{TripleRLWE}}$ are identically distributed. Then we construct a simulator for each corrupted party that outputs a view consistent with the output of the functionality.

Output Distribution. The functionality chooses the shares $\langle \mathbf{a} \rangle_i, \langle \mathbf{b} \rangle_i$ uniformly at random for $i \in \{0, 1\}$, as do the parties P_0 and P_1 in the protocol, which makes them identically distributed in both cases. Let $\mathbf{u} = (\langle \mathbf{a} \rangle_0 + \langle \mathbf{a} \rangle_1) \cdot (\langle \mathbf{b} \rangle_0 + \langle \mathbf{b} \rangle_1) \mod 2^\ell$ and $\mathbf{v} = \langle \mathbf{a} \rangle_1 \cdot \langle \mathbf{b} \rangle_1 \mod 2^\ell$. The functionality sets $\langle \mathbf{c} \rangle_0 = \mathbf{u} + \mathbf{r} \mod 2^\ell$ and $\langle \mathbf{c} \rangle_1 = -\mathbf{r} \mod 2^\ell$ for some $\mathbf{r} \leftarrow_\$ (\mathbb{Z}_{2^\ell})^n$, while the parties compute $\langle \mathbf{c} \rangle_0 = \mathbf{u} + (\mathbf{r}^* - \mathbf{v}) \mod 2^\ell$ and $\langle \mathbf{c} \rangle_1 = -(\mathbf{r}^* - \mathbf{v}) \mod 2^\ell$ for some $\mathbf{r}^* \leftarrow_\$ (\mathbb{Z}_{2^{2\ell+\sigma+1}})^n$. Since $2^\ell \mid 2^{2\ell+\sigma+1}$, $\mathbf{t} = \mathbf{r}^* - \mathbf{v} \mod 2^\ell$ is uniformly distributed in $(\mathbb{Z}_{2^\ell})^n$ and the joint distribution of $\langle \mathbf{c} \rangle_0$ and $\langle \mathbf{c} \rangle_1$ is identically distributed in the ideal functionality and the protocol. Hence, the output is identically distributed in both scenarios.

Corrupted P_0. The Simulator S_0 receives $(\langle \mathbf{a} \rangle_0, \langle \mathbf{b} \rangle_0, \langle \mathbf{c} \rangle_0)$ as input. It chooses a uniformly random tape ρ for P_0, and uses this tape to run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. It then uses independent randomness to sample a uniformly random $\mathbf{d}^* \in (\mathbb{Z}_{2^{2\ell+\sigma+1}})^n$ such that $\mathbf{d}^* \equiv \langle \mathbf{c} \rangle_0 - \langle \mathbf{a} \rangle_0 \cdot \langle \mathbf{b} \rangle_0 \bmod 2^\ell$, and to encrypt $\mathbf{d}^*$ with extra noise. Its output is $(\rho, \mathsf{ct}_{\mathbf{d}^*} = \mathsf{Enc}'(\mathsf{pk}, \mathbf{d}^*))$.
$\mathsf{ct}_{\mathbf{d}^*}$ is statistically indistinguishable from $\mathsf{ct}_{\mathbf{d}}$ received by P_0 in the protocol. This follows from the fact that $t = v + r$ and r^*, where $v \in \mathbb{Z}_{2^{2\ell+1}}$ and $r, r^* \leftarrow_\$ \mathbb{Z}_{2^{2\ell+\sigma+1}}$, are statistically $2^{-\sigma}$ indistinguishable [PBS12]. Therefore, the underlying plaintexts are statistically indistinguishable. From a similar argument, the ciphertexts are also statistically indistinguishable (cf. §4.3). Hence, the output distributions are identical and the corresponding views are statistically indistinguishable, implying that the joint distribution of party P_0's view $(\rho, \mathsf{ct}_{\mathbf{d}})$ and the protocol output is statistically indistinguishable in the ideal and the real execution.

Corrupted P_1. The Simulator S_1 receives $(\langle \mathbf{a} \rangle_1, \langle \mathbf{b} \rangle_1, \langle \mathbf{c} \rangle_1)$ as input. It chooses a uniformly random tape ρ for P_1. It then uses independent randomness to run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and to perform encryptions on a vector of zeros (denoted by $\mathbf{0}^n$) using pk. Its output is $(\rho, \mathsf{pk}, \mathsf{ct}_{\mathbf{a}} = \mathsf{Enc}(\mathsf{pk}, \mathbf{0}^n), \mathsf{ct}_{\mathbf{b}} = \mathsf{Enc}(\mathsf{pk}, \mathbf{0}^n))$.
The computational indistinguishability of the view follows from the IND-CPA security of the AHE scheme (cf. §2.4), because the distinguisher doesn't have access to the randomness used to generate the key-pair $(\mathsf{pk}, \mathsf{sk})$ with which $\mathsf{ct}_{\mathbf{a}}$ and $\mathsf{ct}_{\mathbf{b}}$ were encrypted. Hence the joint distribution of party P_1's view $(\rho, \mathsf{pk}, \mathsf{ct}_{\mathbf{a}}, \mathsf{ct}_{\mathbf{b}})$ and the protocol output is computationally indistinguishable in the ideal and the real execution. □

5 Implementation Results

In this section, we compare the performance of our RLWE-based method (cf. §4) with the OT-based method (cf. §3.2) for generating Beaver multiplication triples.

5.1 Experimental Setup

Our benchmarks were performed on two servers, each equipped with an Intel Core i9-7960X @ 2.8 GHz CPU with 16 physical cores and 128 GB RAM. We consider triple generation for bitlenghts $\ell \in \{8, 16, 32, 64\}$.

Table 2: Amortized runtime (in μs) for generating one ℓ-bit Beaver multiplication triple with T threads in the LAN10, LAN1, and WAN setting. A total of $N = 2^{20}$ triples are generated. Smallest values are marked in bold.

Setting	ℓ	$T = 2$			$T = 8$			$T = 32$		
		OT	RLWE	Impr.	OT	RLWE	Impr.	OT	RLWE	Impr.
LAN10	8	**0.92**	2.36	0.39x	**0.35**	0.70	0.51x	**0.24**	0.51	0.47x
	16	**1.74**	2.38	0.73x	**0.56**	0.69	0.81x	**0.39**	0.50	0.77x
	32	3.35	**2.37**	1.41x	0.99	**0.68**	1.46x	0.75	**0.49**	1.51x
	64	6.53	**4.61**	1.41x	1.89	**1.30**	1.46x	1.61	**0.80**	2.01x
LAN1	8	**1.30**	3.07	0.42x	**1.27**	2.07	0.61x	**1.28**	2.02	0.64x
	16	**2.64**	3.08	0.85x	2.56	**2.09**	1.22x	2.58	**1.99**	1.29x
	32	5.55	**3.07**	1.81x	5.53	**2.34**	2.36x	5.49	**2.24**	2.45x
	64	13.14	**5.85**	2.25x	13.09	**4.06**	3.23x	13.03	**3.88**	3.35x
WAN	8	20.48	**20.02**	1.02x	**19.33**	25.11	0.77x	**20.14**	22.90	0.88x
	16	31.10	**20.39**	1.53x	32.66	**26.11**	1.25x	28.98	**23.83**	1.22x
	32	60.81	**23.85**	2.55x	60.22	**26.42**	2.28x	61.25	**26.44**	2.32x
	64	140.48	**39.34**	3.57x	138.54	**45.20**	3.07x	140.79	**41.57**	3.39x

Table 3: Amortized communication (in Bytes) for generating one ℓ-bit Beaver multiplication triple. Smallest values are marked in bold.

ℓ	OT	RLWE	Impr.
8	272	**224**	1.21x
16	576	**224**	2.57x
32	1280	**256**	5.00x
64	3072	**448**	6.85x

We have used the Microsoft SEAL library v3.1 [CLP17] to implement the RLWE-based method $\Pi_{\mathsf{TripleRLWE}}$, and the OT-based method Π_{TripleOT} is implemented in ABY library [DSZ15]. In all experiments, we have set the symmetric security parameter to $\kappa = 128$, and the statistical security parameter to $\sigma = 40$. The computational security parameter λ for the RLWE-AHE scheme has been chosen to get security of at least 128-bits. The concrete parameters for the RLWE-AHE scheme are given in Tab. 1. .
We run the benchmarks for three network settings (bandwidth, latency): LAN10 (10 Gbps, 0.5ms RTT), LAN1 (1 Gbps, 0.5ms RTT), and WAN (100 Mbps, 50ms RTT). In each setting, we performed experiments for $N \in \{2^{15}, 2^{16}, \ldots, 2^{22}\}$ triples and $T \in \{2, 8, 32\}$ threads.

5.2 Results and Analysis

We give the amortized (over generating $N = 2^{20}$ triples) runtimes in Tab. 2 and the communication in Tab. 3 to compute one Beaver multiplication triple using RLWE-AHE and OT for bitlengths $\ell \in \{8, 16, 32, 64\}$. The initialization costs are given in Tab. 4. For a more detailed analysis, we also varied the number of triples generated N, and the corresponding plots are given for the LAN10 (Fig. 4), LAN1 (Fig. 5), and WAN (Fig. 6) scenario. The breakeven-points, in terms of N, beyond which RLWE-AHE performs better than OT (including initialization costs) are given in Tab. 5. The results of our experiments can be summarized as follows:

1. RLWE-AHE requires less communication than OT, and the difference grows with increasing ℓ. For $\ell = 64$, the improvement factor over OT is 6.9x.
2. RLWE-AHE requires more computation than OT for smaller bitlengths, since OT has a smaller runtime than RLWE-AHE in the LAN10 setting where communication is not a bottleneck.
3. RLWE-AHE is faster than OT for larger bitlengths due to lower computation and communication requirements, achieving speedup of 3.6x for $\ell = 64$ in the WAN setting.

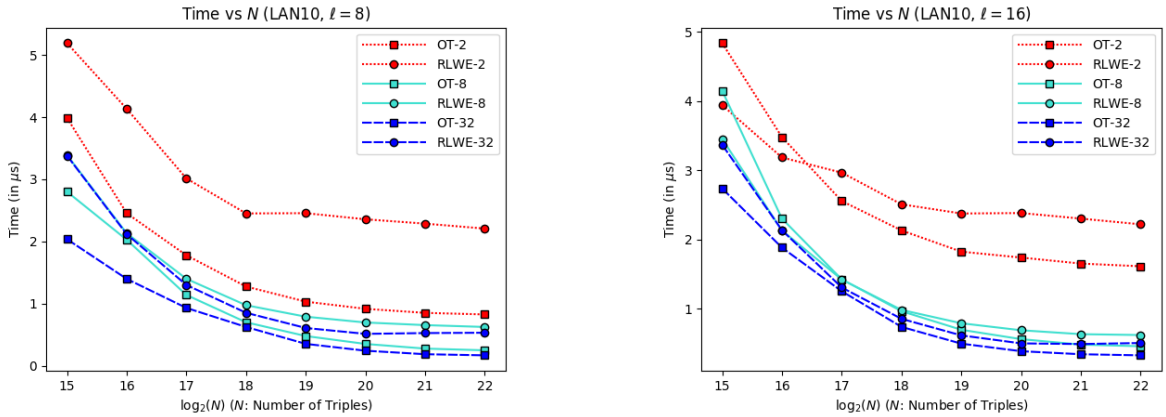
Table 4: (One-time) Initialization Costs for ℓ-bit Beaver multiplication triple generation with $T = 2$ threads in the LAN10, LAN1, and WAN setting. Smallest values are marked in bold.

Setting	ℓ	Time [in ms]			Comm. [in KB]		
		OT	RLWE	Impr.	OT	RLWE	Impr.
LAN10	8	**246.51**	509.16	0.48x	**97.57**	1024.28	0.09x
	16	**246.51**	558.99	0.44x	**97.57**	1024.28	0.09x
	32	**246.51**	576.31	0.43x	**97.57**	1024.28	0.09x
	64	**246.51**	1031.02	0.24x	**97.57**	2048.57	0.05x
LAN1	8	**253.12**	514.50	0.49x	**97.57**	1024.28	0.09x
	16	**253.12**	562.38	0.45x	**97.57**	1024.28	0.09x
	32	**253.12**	590.37	0.43x	**97.57**	1024.28	0.09x
	64	**253.12**	1053.20	0.24x	**97.57**	2048.57	0.05x
WAN	8	**547.87**	564.11	0.97x	**97.57**	1024.28	0.09x
	16	**547.87**	584.82	0.93x	**97.57**	1024.28	0.09x
	32	**547.87**	611.59	0.90x	**97.57**	1024.28	0.09x
	64	**547.87**	1125.41	0.49x	**97.57**	2048.57	0.05x

Table 5: Break-even point (in $\lceil \log_2(N) \rceil$, where N is the number of triples) beyond which RLWE-AHE performs better than OT in the worst-case scenario, where the initialization costs are also included in the cost of triple generation. The cases where the break-even point does not exist are denoted by '-'.

Setting	ℓ	Time			Comm.
		$T = 2$	$T = 8$	$T = 32$	
LAN10	8	-	-	-	15
	16	-	-	-	13
	32	19	21	21	12
	64	19	19	19	11
LAN1	8	-	-	-	15
	16	-	20	20	13
	32	17	17	17	12
	64	17	15	16	11
WAN	8	-	-	-	15
	16	13	15	15	13
	32	12	12	12	12
	64	13	13	13	11

4. OT is faster than RLWE-AHE for smaller bitlengths in most cases. For instance, OT is better than RLWE-AHE in all cases for $\ell = 8$ in the LAN1 setting (cf. Fig. 5a).
5. Due to less communication, the improvement factor in runtime of RLWE-AHE over OT increases with decreasing network performance.
6. RLWE-AHE benefits more from multi-threading than OT for faster networks. For $\ell = 64$ in the LAN1 setting, the improvement factor increases from 2.25x to 3.35x as we move from 2 to 32 threads. When communication is the bottleneck, multi-threading does not benefit either method.
7. OT benefits more from increasing N in general, and the gains are more prominent for smaller ℓ. For $\ell = 8$ (resp. 64) in the LAN10 setting and $T = 2$ threads, the performance of OT improves by 4.80x (resp. 1.60x) as we increase N from 2^{15} to 2^{22}, compared to a performance improvement by 2.19x (resp. 1.56x) for RLWE-AHE.
8. The performance of RLWE-AHE saturates for a smaller N as compared to OT. For instance, in Fig. 5, the performance of RLWE-AHE saturates at $N = 2^{18}$ for $\ell = 8$ (resp. $N = 2^{17}$ for $\ell = 64$), while the performance of OT saturates at $N = 2^{21}$ (resp. $N = 2^{19}$).



(a) $\ell = 8$

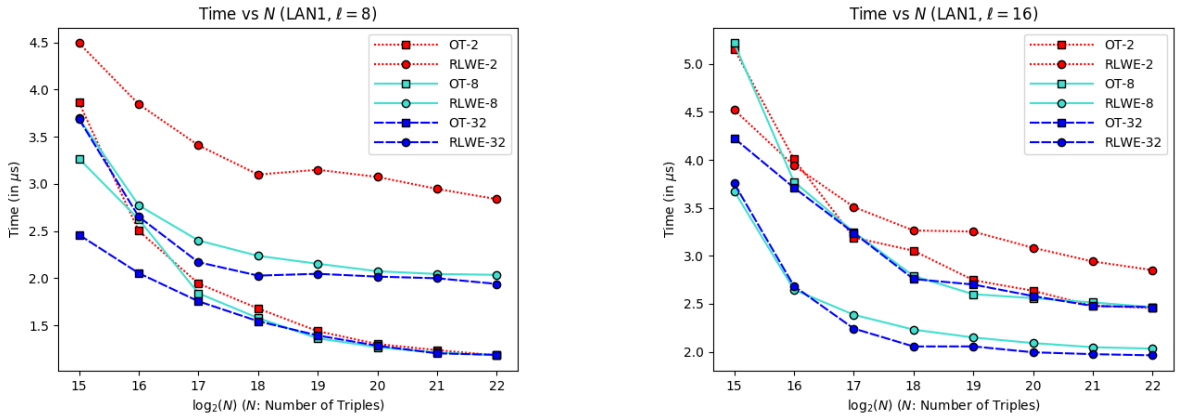
(b) $\ell = 16$

(c) $\ell = 32$

(d) $\ell = 64$

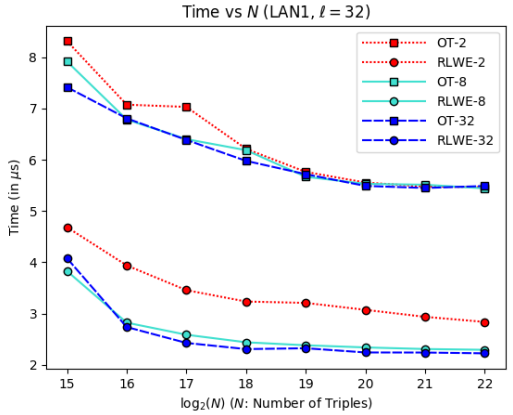
Fig. 4: Performance plots showing amortized runtime (over generating N triples) to compute one ℓ-bit Beaver multiplication triple in the LAN10 scenario. The legend entries represent the method and the number of threads T used.

9. Even though the initialization cost of RLWE-AHE is significantly larger than that of OT (cf. Tab. 4) (especially for the faster networks), this cost is incurred only once. Moreover, this cost can be further reduced for RLWE-AHE by reusing the same public key pk for multiple clients.
10. RLWE-AHE is not suitable for smaller values of N ($\leq 2^{12}$) even in the WAN setting (cf. Tab. 5), where the initialization costs are very close. This is owed to the fact that RLWE-AHE based solution derives its efficiency mainly from the batching optimization (cf. §4.1), that allows generation of n triples essentially at the cost of one. To maintain security, n needs to be greater than a certain threshold, which in our case is $8192 = 2^{13}$ (cf. Tab. 1). Therefore for smaller values of N, many plaintext slots are left unused, which leads to a poor amortized performance.
11. Overall, our RLWE-based method is a better option for most practical cases. It is faster in almost all scenarios for the WAN setting, while even in the LAN10 setting, the performance improvement is significant for larger bitlengths. However, for smaller bitlengths such as $\ell = 8$, the OT-based method is more suitable even in the WAN setting.

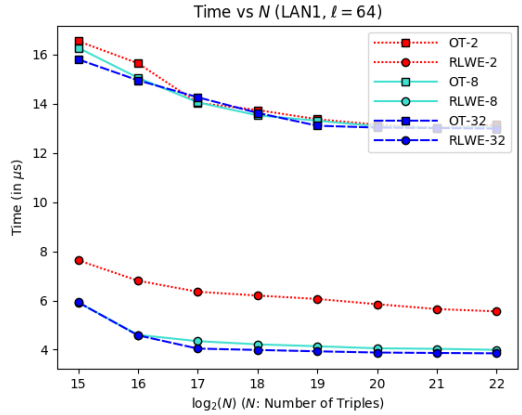


(a) $\ell = 8$

(b) $\ell = 16$



(c) $\ell = 32$

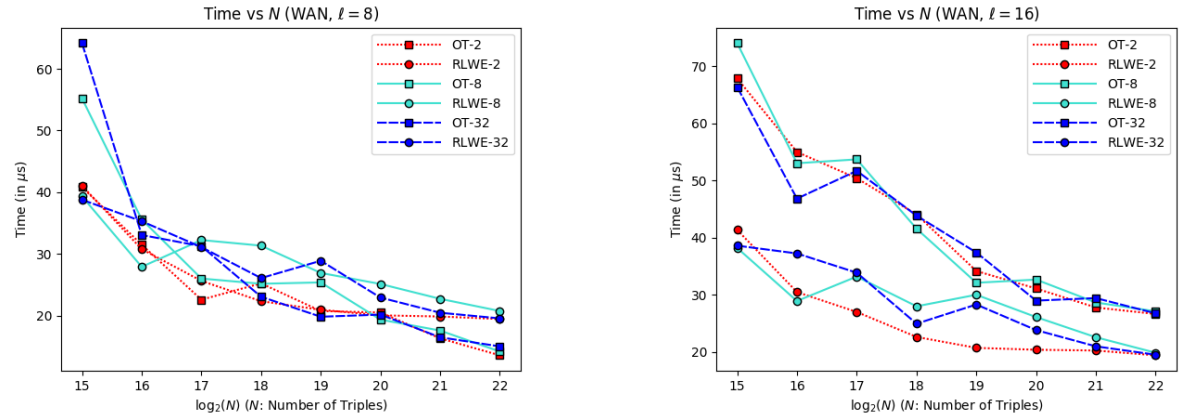


(d) $\ell = 64$

Fig. 5: Performance plots showing amortized runtime (over generating N triples) to compute one ℓ-bit Beaver multiplication triple in the LAN1 scenario. The legend entries represent the method and the number of threads T used.

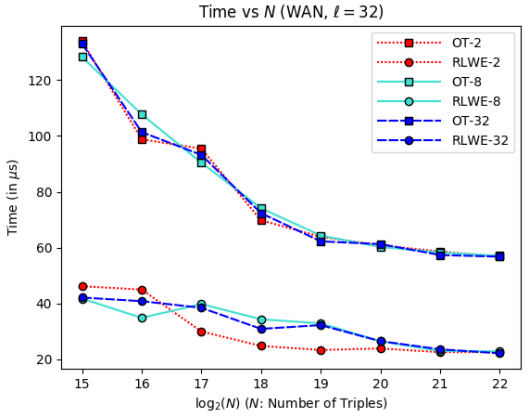
Note. Table 3 and Table 4 show that the communication complexity remains the same for different values of ℓ. This is due to the fact that SEAL serializes each prime in the ciphertext modulus as a 64-bit integer, irrespective of the size of the prime. Therefore, the communication for a ciphertext modulus consisting of 4 primes of 40 bits each and that of 4 primes of 50 bits each is the same and only the computation is increased for the latter case.

Acknowledgements. This work was co-funded by the DFG as part of project E4 within the CRC 1119 CROSSING and project A.1 within the RTG 2050 "Privacy and Trust for Mobile Users", and by the BMBF and the HMWK within CRISP.

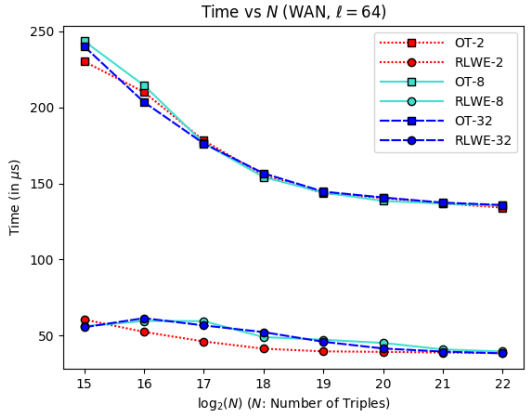


(a) $\ell = 8$

(b) $\ell = 16$



(c) $\ell = 32$



(d) $\ell = 64$

Fig. 6: Performance plots showing amortized runtime (over generating N triples) to compute one ℓ-bit Beaver multiplication triple in the WAN scenario. The legend entries represent the method and the number of threads T used.

References

AJL+12. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *EUROCRYPT*, 2012.

ALSZ13. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *ACM CCS*, 2013.

Bea91. Donald Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO*, 1991.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Innovations in Theoretical Computer Science Conference*, 2012.

CDE+18. Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority. In *CRYPTO*, 2018.

CLP17. Hao Chen, Kim Laine, and Rachel Player. Simple Encrypted Arithmetic Library - SEAL v2.1. In *WAHC at FC*, 2017. Code: `https://github.com/microsoft/SEAL`.

CRFG19. Dario Catalano, Mario Di Raimondo, Dario Fiore, and Irene Giacomelli. Mon$\mathbb{Z}_{2^k}$a: Fast Maliciously Secure Two Party Computation on $\mathbb{Z}_{2^k}$. Cryptology ePrint Archive, Report 2019/211, 2019.

DGK08. Ivan Damgård, Martin Geisler, and Mikkel Krøigård. Homomorphic Encryption and Secure Comparison. *International Journal of Applied Cryptography*, 1(1), 2008.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO*, 2012.

DSZ15. Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*, 2015. Code: `https://encrypto.de/code/ABY`.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

Gen09. Craig Gentry. Fully Homomorphic Encryption using Ideal Lattices. In *STOC*, 2009.

Gil99. Niv Gilboa. Two Party RSA Key Generation. In *CRYPTO*, 1999.

JL13. Marc Joye and Benoît Libert. Efficient Cryptosystems from 2^k-th Power Residue Symbols. In *EUROCRYPT*, 2013.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *ACM CCS*, 2016.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ Great Again. In *EUROCRYPT*, 2018.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. In *EUROCRYPT*, 2010.

OSV19. Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. Overdrive2k: Efficient Secure MPC over $\mathbb{Z}_{2^k}$ from Somewhat Homomorphic Encryption. Cryptology ePrint Archive, Report 2019/153, 2019.

Pai99. Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, 1999.

PBS12. Pille Pullonen, Dan Bogdanov, and Thomas Schneider. The Design and Implementation of a Two-party Protocol Suite for SHAREMIND 3. *CYBERNETICA Institute of Information Security, Tech*, 2012.

RSS19. Deevashwer Rathee, Thomas Schneider, and K. K. Shukla. Improved Multiplication Triple Generation over Rings via RLWE-based AHE. In *CANS*, 2019.

SV14. N. P. Smart and F. Vercauteren. Fully Homomorphic SIMD Operations. *Designs, Codes and Cryptography*, 71(1), 2014.