Improved Multiplication Triple Generation over Rings via RLWE-based AHE

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 preprocessing 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 preprocessing 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 the semi-honest model and the two-party setting, for which an Oblivious Transfer (OT)-based protocol is the currently best solution. To improve upon this method, we propose a protocol based on RLWE-based Additive 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 · Additive 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 preprocessing 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 preprocessing phase, and then use them to evaluate arithmetic circuits very efficiently in an information-theoretic online phase, free from 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 these triples.

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 $\mathbb{Z}_{2^\ell}$, which was later implemented in [DEF$^+$19]. Designing protocols over rings is useful in a lot of applications since it greatly simplifies implementation of comparisons and bitwise operations, which are inefficient to realize with finite field arithmetic (cf. [DEF$^+$19]). Apart from this, using ring-based protocols also implies that we can leverage some special tricks that computers already implement to make integer arithmetic very efficient. In 2019, Orsini et. al [OSV19] presented a solution based on SHE and argued that it is more efficient than the OT-based protocol of [CDE$^+$18].

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 Additive Homomorphic Encryption (RLWE-AHE) that improves upon the widespread 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 by up to 6.9x and runtime by up to 3.6x for 64-bit triple generation.

2 Preliminaries

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 respectively. $\langle x \rangle$ 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. The key pair of party P_i is denoted by $(\mathsf{pk}_i, \mathsf{sk}_i)$.

Problem Statement. A Beaver multiplication triple [Bea91] is defined as the tuple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ with $(\langle a \rangle_0 + \langle a \rangle_1) \cdot (\langle b \rangle_0 + \langle b \rangle_1) \equiv (\langle c \rangle_0 + \langle c \rangle_1) \bmod 2^\ell$. Our aim is to construct a two-party protocol that securely realizes a functionality which generates a Beaver multiplication triple in the ring $\mathbb{Z}_{2^\ell}$ for two parties P_0

and P_1, and sends the respective shares of the triple to each party.

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

Ring-LWE-based Additive Homomorphic Encryption (RLWE-AHE). We use an AHE scheme with the following 5 algorithms:

- $\mathsf{KeyGen}(1^\lambda)$: Key Generation is a randomized algorithm that outputs the key pair $(\mathsf{pk}, \mathsf{sk})$, with public key pk and secret key sk.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{m})$: Encryption takes a vector $\mathsf{m} \in \mathbb{Z}_p^n$ as input, where n depends on scheme parameters m and p, along with pk, and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: Decryption takes the secret key sk and a ciphertext ct, and outputs the underlying plaintext $\mathsf{m} \in \mathbb{Z}_p^n$.
- $\mathsf{Add}(\mathsf{ct}_1, \mathsf{ct}_2)$: Addition takes as input two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$, and outputs a ciphertext ct_a such that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_a) = \mathsf{m}_1 + \mathsf{m}_2 \in \mathbb{Z}_p^n$, where addition is performed component-wise. This algorithm is also denoted by the $+$ operator.
- $\mathsf{ScalarMult}(\mathsf{ct}_1, \mathsf{m}_2)$: Given inputs ct_1 and m_2, scalar-multiplication outputs a ciphertext $\mathsf{ct}_s \in \mathcal{C}$ such that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_s) = \mathsf{m}_1 \cdot \mathsf{m}_2 \in \mathbb{Z}_p^n$, where multiplication is performed component-wise. This algorithm is also denoted by the $\cdot$ operator.

Possible instantiations of RLWE-based schemes that satisfy the description above are [FV12,BGV12]. 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 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, their design is similar to our proposed protocol.

3.1 AHE-based Generation

Case I - $2^\ell | p$. Protocol 1 describes a well-known protocol for generating triples using AHE. 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

Protocol 1 Basic Beaver Triple Generation using AHE

```
$\Pi_{\mathsf{BasicTripleAHE}}$:
    $P_0$: Sample uniformly random $\langle a\rangle_0, \langle b\rangle_0 \in \mathbb{Z}_{2^\ell}$
    $P_1$: Sample uniformly random $\langle a\rangle_1, \langle b\rangle_1 \in \mathbb{Z}_{2^\ell}$ and $r \in \mathbb{Z}_p$
        $\langle c\rangle_1 = \langle a\rangle_1 \cdot \langle b\rangle_1 - r \bmod 2^\ell$
    $P_0 \to P_1$: $\mathsf{ct}_a = \mathsf{Enc}(\mathsf{pk}_0, \langle a\rangle_0), \mathsf{ct}_b = \mathsf{Enc}(\mathsf{pk}_0, \langle b\rangle_1)$
    $P_1 \to P_0$: $\mathsf{ct}_d = (\mathsf{ct}_a \cdot \langle b\rangle_1) + (\mathsf{ct}_b \cdot \langle a\rangle_1) + r$
    $P_0$: $\langle c\rangle_0 = \langle a\rangle_0 \cdot \langle b\rangle_0 + \mathsf{Dec}(\mathsf{sk}_0, \mathsf{ct}_d) \bmod 2^\ell$
```

p. We can use the DGK cryptosystem [DGK08] since it uses a 2-power modulus, whereas for using the Paillier cryptosystem [Pai99] whose plaintext modulus is the product of two distinct primes, we require some changes to $\Pi_{\mathsf{BasicTripleHE}}$.

Case II - $2^\ell \nmid p$. Firstly, we choose 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.

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. 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 two cases, we detail some optimizations and a security consideration that are crucial for our analysis.

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.

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.

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'.

Parameter Selection. The plaintext modulus p determines the protocol to be used as mentioned in §3.1. After determining p and essentially the approach, 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

Protocol 2 Beaver Triple Generation using RLWE-AHE

```
$\Pi_{\mathsf{TripleRLWE}}$:
    $P_0$: Sample uniformly random $\langle a\rangle_0, \langle b\rangle_0 \in \mathbb{Z}^n_{2^\ell}$
    $P_1$: Sample uniformly random $\langle a\rangle_1, \langle b\rangle_1 \in \mathbb{Z}^n_{2^\ell}$ and $r \in \mathbb{Z}^n_{2^{2\ell+\sigma+1}}$
        $\langle c\rangle_1 = \langle a\rangle_1 \cdot \langle b\rangle_1 - r \bmod 2^\ell$
        $\mathsf{ct}_r = \mathsf{Enc}'(\mathsf{pk}_0, r)$        ▷ $\mathsf{Enc}'$: Encryption with noise in $[-B', B']$
    $P_0 \to P_1$: $\mathsf{ct}_a = \mathsf{Enc}(\mathsf{pk}_0, \langle a\rangle_0), \mathsf{ct}_b = \mathsf{Enc}(\mathsf{pk}_0, \langle b\rangle_1)$
    $P_1 \to P_0$: $\mathsf{ct}_d = (\mathsf{ct}_a \cdot \langle b\rangle_1) + (\mathsf{ct}_b \cdot \langle a\rangle_1) + \mathsf{ct}_r$
    $P_0$: $\langle c\rangle_0 = \langle a\rangle_0 \cdot \langle b\rangle_0 + \mathsf{Dec}(\mathsf{sk}_0, \mathsf{ct}_d) \bmod 2^\ell$
```

embedded in each slot, we employ the packing method from [OSV19] to essentially achieve a maximum utilization of $\phi(m)/5$ slots, each packing a message $\in \mathbb{Z}_p$. Despite this significant optimization, most of the slots are being 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 $e = (2\ell + \sigma + 2)/\ell$ 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 here 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 *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 (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 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.

Table 1: 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 2: Amortized communication (in Bytes) for generating one ℓ-bit Beaver multiplication triple. Smallest values are marked in bold.

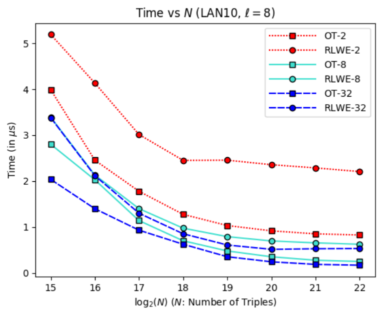
ℓ	OT	RLWE	Impr.
8	272	**224**	1.21x
16	576	**224**	2.57x
32	1280	**256**	5.00x
64	3072	**448**	6.85x

Our Final Protocol. Our final protocol that we have used to obtain the benchmarks is given in Protocol 2. 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 with previous OT and AHE-based schemes.

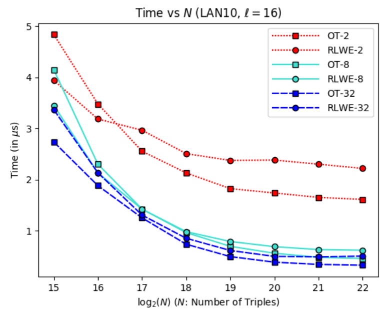
5 Implementation Results

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.

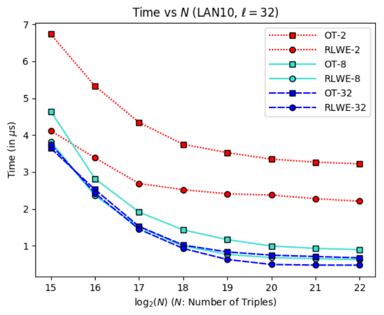
Experimental Setup. Our benchmarks were performed on two servers, each equipped with an Intel Core i9-7960X @ 2.8 GHz CPU with 16 physical cores and 128 GB RAM. We consider triple generation for bitlenghts $\ell \in \{8, 16, 32, 64\}$. We



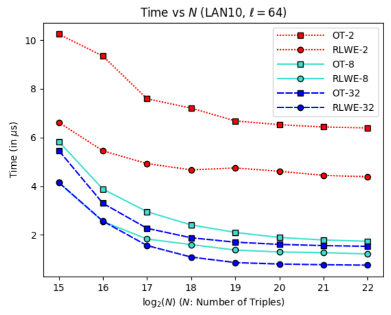
(a) $\ell = 8$



(b) $\ell = 16$



(c) $\ell = 32$

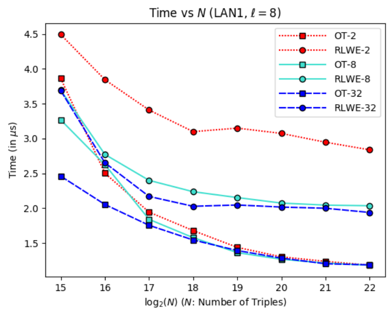


(d) $\ell = 64$

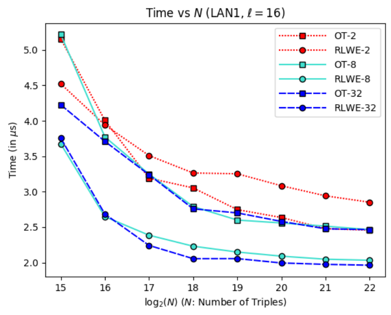
Fig. 1: Performance plots showing amortized runtime (over generating N triples) to compute one ℓ-bit Beaver multiplication triple in the LAN10 scenario. The legend entries represent the method and the number of threads T used.

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 128-bits.
We run the benchmarks for three network settings (bandwidth, latency): LAN10 (10 Gbps, 0.5ms), LAN1 (1 Gbps, 0.5ms), and WAN (100 Mbps, 50ms). In each setting, we performed experiments for $N \in \{2^{15}, 2^{16}, \ldots, 2^{22}\}$ triples and $T \in \{2, 8, 32\}$ threads.

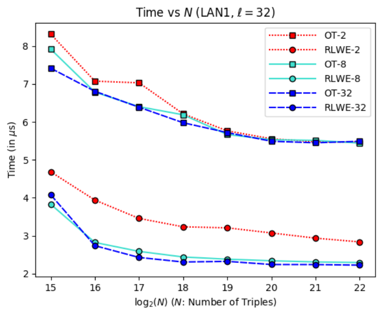
Results and Analysis. We give the amortized (over generating $N = 2^{20}$ triples) runtimes in Tab. 1 and the communication in Tab. 2 to compute one Beaver multiplication triple using RLWE-AHE and OT for bitlengths $\ell \in \{8, 16, 32, 64\}$ and 128-bit security parameters. For a more detailed analysis, we also varied the



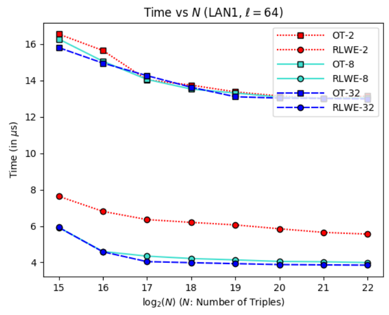
(a) $\ell = 8$



(b) $\ell = 16$



(c) $\ell = 32$



(d) $\ell = 64$

Fig. 2: 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.

number of triples generated N, and the corresponding plots are given for the LAN10 (Fig. 1), LAN1 (Fig. 2), and WAN (Fig. 3) scenario. 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.
3. RLWE-AHE is faster than OT for larger bitlengths due to lower computation and communication requirements, achieving speedup by 3.6x for $\ell = 64$ in the WAN setting.
4. OT is faster than RLWE-AHE for smaller bitlengths and faster networks.
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

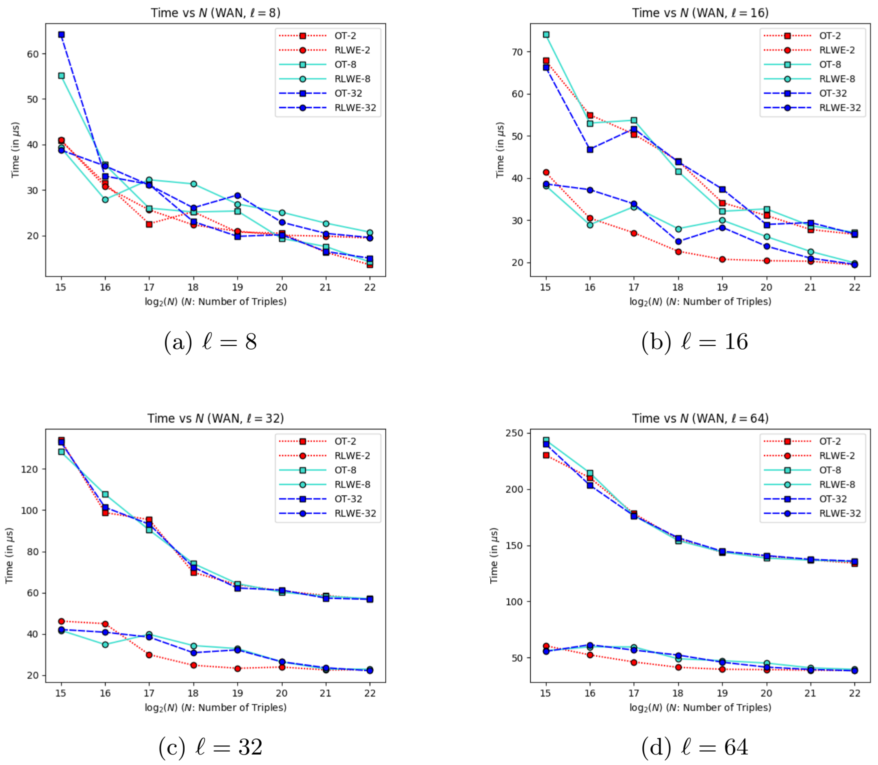


Fig. 3: Performance plots showing amortized runtime (over generating N triples) to compute one ℓ-bit Beaver multiplication triple in the WAN scenario. The legend entries represent the method and the number of threads T used.

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, and the gains are more prominent for smaller ℓ. For $\ell = 8$ (resp. 64) in the LAN10 setting, 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. As we get to $N = 2^{20}$, the performance of both the methods tends to converge in almost all cases.
8. Overall, 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.

Note. Table 2 shows that the communication complexity remains the same for $\ell = 8$ and $\ell = 16$. 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.

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`.

DEF$^+$19. I. Damgrd, D. Escudero, T. Frederiksen, M. Keller, P. Scholl, and N. Volgushev. New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning. In *IEEE Symposium on Security and Privacy (SP)*, 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.

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.

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.

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