Thijs Veugen* and Mark Abspoel

Secure integer division with a private divisor

Abstract: We consider secure integer division within a secret-sharing based secure multi-party computation framework, where the dividend is secret-shared, but the divisor is privately known to a single party. We mention various applications where this situation arises. We give a solution within the passive security model, and extend this to the active model, achieving a complexity linear in the input bit length. We benchmark both solutions using the well-known MP-SPDZ framework in a cloud environment. Our integer division protocol with a private divisor clearly outperforms the secret divisor solution, both in runtime and communication complexity.

Keywords: secure multi-party computation, secret sharing, integer division, truncation

DOI 10.2478/popets-2021-0073
Received 2021-02-28; revised 2021-06-15; accepted 2021-06-16.

1 Introduction

Secure multi-party computation (MPC) is a well-known and increasingly popular technology that enables privacy-friendly computations on distributed data. Multiple parties, each having private inputs, jointly compute a function of these inputs using a cryptographic protocol, without revealing their inputs to each other. There exist various MPC frameworks that are capable of securely computing arbitrary functions of the parties' inputs, using tools like homomorphic encryption, garbled circuits, or secret sharing. Besides the two essential primitives of secure linear combinations and secure multiplication, such generic frameworks typically also implement optimized protocols for several core building blocks, such as integer comparison, integer division, exponentiation, and so on.

We consider the integer division building block, i.e. given a secret-shared dividend x, and a divisor d, compute a secret-sharing of $x \div d := \lfloor x/d \rfloor$. This is a relatively expensive operation within MPC, and it is required in various applications, such as secure clustering [5, 13], secure personal recommendation [14], secure face recognition [12], secure statistical analysis [17], secure auctions [4, 24], but also for computing the Levenshtein distance [25]. In these applications, the divisor is often either secret, like the dividend, or a publicly known constant. In the first case, this leads to secure, but computationally intensive protocols. In the second case, more efficient solutions are known [31].

We consider the less frequently studied case where one party privately holds the value of the divisor, and show solutions whose computational complexity resembles the efficient solutions in case the divisor is publicly known. In this setting, where the divisor can be seen as a direct input of one of the parties, a common solution is to upload the divisor to the secure computing platform [32] and perform the integer division with a secret divisor, but a dedicated protocol for a private divisor will, as illustrated in this paper, yield a more efficient solution.

In many of the applications mentioned above, the situation arises where the divisor is held privately by one of the parties. For example, in one secure recommendation system [14], each user privately learns the number of users that are similar to them, while the summed item ratings remain secret. To obtain the actual recommendation, the user needs to divide these two numbers, without learning sensitive rating information, i.e. the dividend.

Similarly, in each iteration of secure clustering, the sum of user values of one cluster needs to be divided by the number of users in each cluster [13]. Instead of letting the server know both values, and performing a cleartext division, which is how this division is currently computed, it would be more secure to keep the cluster sum secret, and give the number of users for each cluster to a specific user.

Also, when computing with a fixed-point representation [7], a truncation protocol is needed to divide intermediate results by some scaling factor. Although this scaling factor is usually public, its exact value might reveal some information on the input values, so keeping the value privately known to the holder of the inputs values makes the solution more secure.

Another good example of such a setting is a secure graph algorithm, where each party knows a subgraph, so

***Corresponding Author: Thijs Veugen:** TNO, The Netherlands, E-mail: thijs.veugen@tno.nl
Mark Abspoel: CWI, The Netherlands, E-mail: m.a.abspoel@cwi.nl

the in- and out-degree of nodes are privately held. For example, in the PageRank protocol, which has proven useful for reducing financial fraud [26], intermediate values have to be divided by the out-degree of the corresponding node.

1.1 Preliminaries

Our model of secure computation is very general. We assume a set of parties $P_1, \ldots, P_n$ have access to an *arithmetic black box* functionality, denoted $[\cdot]$, on integers modulo some modulus N. The arithmetic black box allows a party to encrypt a locally known input value, and it allows the parties to jointly perform linear operations and multiplications on encrypted values, as well as jointly decrypt a value. This arithmetic black box is typically realized using multiparty computation based on secret sharing, and we shall use this corresponding language throughout the paper.[1]

Security is defined against an adversary that corrupts some of the parties, where the precise set of parties that can be corrupted depends on the realization of the arithmetic black box. We distinguish between passive and active security (with abort), where in the former the parties are guaranteed to faithfully execute the protocol.

Furthermore, we assume access to a secure comparison functionality $[b] \leftarrow \mathsf{Compare}([x] < [y], \ell)$ that compares two integers of at most ℓ bits. Throughout the work, secrets are calculated modulo N, but we identify residue classes in $\mathbb{Z}_N$ with their representatives in the integer interval $[0, N)$, hence in particular integer comparison makes sense. The functionality takes secret integers x, y with $0 \leq x, y < N$, and outputs a secret-sharing of a bit b that equals one, if $x < y$, and zero, otherwise. Also, we require a functionality $\mathsf{RandInt}(\ell)$ that securely generates a secret-sharing of a uniformly random integer in the interval $[0, 2^\ell)$. Both of these functionalities are usually also implemented for MPC protocols that realize the arithmetic black box (e.g. see [10] based on $\text{SPD}\mathbb{Z}_{2^k}$), and their realizations usually require a linear number (in ℓ) of multiplications.

The private division functionality that we want to achieve in this work takes inputs $[x]$ and $[d]$, such that P_1 (without loss of generality) knows the value of d, and produces an output $[x \div d]$, such that $x = d \cdot (x \div d) + (x \bmod d)$, where $x \div d \geq 0$, and $0 \leq x \bmod d < d$.

We use $\log_2 x$ to denote the base two logarithm of a positive integer x as a measure for the number of bits of x. For convenience we ignore the fact that the result should be rounded upwards to the closest integer, as this logarithm is usually not integer valued. Let σ be the statistical security parameter, whose value is usually chosen around 40.

1.2 Related work in secure integer division

The related work is subdivided into two categories. The first category contains secure integer division with a secret (either secret-shared, or encrypted) divisor, and for the second category, the divisor is publicly known. Integer division with public divisor is also known as truncation, and is strongly related to secure computation of the modular remainder with a public modulus.

Kiltz, Leander and Malone-Lee [21] presented an oracle-aided protocol for computing the mean of several database entries, which is described as a way of securely approximating the division of two additively shared values, namely the dividend and the secret divisor. Their main primitive is an oblivious polynomial evaluation, which requires an amount of exponentiations and communications linear in the size of the divisor.

Bunn and Ostrovsky [5] described a protocol for securely clustering two databases. They propose a subprotocol that securely computes the division of two secret-shared integers, using ideas of long division.

Jakobsen [19] described several methods for dividing two secret-shared integers. Also approximations are considered, but all methods use some kind of (expensive) binary search. A similar, exact approach for encrypted integers was found by Dahl, Ning and Toft [8], who currently have the theoretically most efficient solution for secret division. They use threshold homomorphic encryption for integer division by multiplying with the reciprocal, and truncating the result. However, in practice, for bit lengths up to a few hundred, their approach is less efficient than Catrina and Saxena [7], who worked out a way of securely computing with rational numbers using a fixed-point representation. A linear secret sharing scheme is used to protect the inputs and the output. Part of this framework is a truncation protocol that is capable of secure division with a known power of two. The work of Catrina and Saxena is improved further in the semi-honest model by Catrina and De Hoogh [6].

1 Although we use the language of secret sharing, our results also apply to secure multi-party computation based on homomorphic encryption (with threshold decryption).

In 2018, Ohata and Morita [27] described a privacy-preserving division protocol, with a trade-off between accuracy and efficiency, by approximating the reciprocal. At the same conference, together with other authors, they describe an exact division protocol with secret divisor, based on Goldschmidt's division algorithm, and apply it to Chi-squared tests [23].

Ugwuoke, Erkin and Lagendijk [30] present a secure division for encrypted dividend and divisor, by iterating over the digits. Hiwatashi, Ohata and Nuida [18] start from secret-shared dividend and divisor, and modify Goldschmidt's method with a building block for approximating large products. Ding et al. [11] develop a secure division for encrypted divisor in a setting for cloud data processing with multiple parties.

Damgård et al. [9] presented secure protocols for modulo reduction of secretly shared integers within constant rounds. The modulus is either public, or secretly shared. This secure modulo reduction could consequently be used to compute the division.

Schoenmakers and Tuyls [28] presented a method, based on threshold homomorphic crypto, to convert an encrypted integer to its encrypted bits. One of their results is a gate that securely computes the least significant bits. It can be used to compute an encryption of $x \bmod 2^m$ for some known integer m, and from that an encryption of $x \div 2^m$, thus it enables the division of an encrypted integer by a known power of 2. Toft [29] showed how to reduce a secretly shared integer with respect to a known modulus, thereby generalizing the results of Schoenmakers and Tuyls to arbitrary, but publicly known divisors. He also extended this result to a secretly shared modulus, using a secure comparison protocol among others.

Atallah et al. [3] presented secure protocols for integer division, either with an additively shared, or a public divisor. The specific setting where one party holds a private divisor has been introduced by Veugen [31].

1.3 Contribution

Our main contribution is an efficient protocol for secure integer division of $[x]$ by a private divisor $[d]$ that is known to one of the parties. Since we use the fact that P_1 knows d, we require its cooperation. An active adversary that corrupts P_1 can always refuse to cooperate, therefore the best security we can achieve[2] is active security with abort.

We build on earlier work by Veugen [31], who introduced two division protocols in the passive security model. The first protocol is for exact division with a *public* divisor d, and the second is secure *approximate* division with a private divisor d for which the length $\log_2 d$ is public. We show the second protocol as stated in [31] is in fact insecure, since it leaks some information about $x \bmod d$, and remedy this. In addition, we improve the protocol from approximate to exact division, and instead of requiring the bit length of the divisor to be public, we only use an upper bound ℓ on its bit size.

The intuition behind our protocol is as follows. We mask x by adding a (much larger) random integer R, so that $x + R$ statistically reveals no information about x. We reveal the masked value $z = x + R$ to P_1, who then performs cleartext division by d, and secret-shares the result. This resulting value equals $(x \div d) + (R \div d) + b$, for some $b \in \{0, 1\}$. By choosing R of the form $dR' + R''$, where R'' is small, we can subtract R' from the result and obtain the desired $x \div d$ plus a small error.

Some components of this error, like the value of b, can be computed using calls to Compare. The remaining error component is relatively small, but it is not necessarily at most one (as erroneously stated in [31]). We show a bound on this remaining error, and show how to reduce it to zero, by first shifting x a number of bits to the left before performing division, and then shifting back the result of the division. The same technique allows us to avoid leaking the exact bit-length of d, as was required in [31].

We make the key observation that we can obtain active security against an adversary corrupting P_1 by multiplying the secret-shared division result $[z'] = [z \div d]$ by $[d]$, and checking whether the result is "close enough" to z. More precisely, we check whether $[d] \cdot [w] \leq [x] < [d] \cdot ([w] + 1)$. These checks also leak some information to the adversary, but we provide a detailed analysis to show this is negligible in the security parameter.

2 Passive security

We present in Protocol 1 a passively secure version of our private division protocol $\mathsf{PrivateDivision}_{\mathsf{P}}$ (where P

2 Without degrading performance to that of division by a secret divisor.

stands for passive security). In the next section, we will provide the modifications needed to make the protocol actively secure.

We now explain Protocol 1 in more detail. P_1 receives z, which is the input x multiplied by $2^{\ell+\sigma}$ and then blinded with two random numbers: r'' and h, where h is a multiple of the divisor d. The randomness of r'' is needed because P_1 has knowledge of d, so if $r'' = 0$, P_1 can compute $z \bmod d = (2^{\ell+\sigma}x) \bmod d$, which leaks information about x. Since d is of bit-length at most ℓ, we choose r'' to be of bit-length $\ell+\sigma$, where σ is a statistical security parameter. The randomness of h is needed because $2^{\ell+\sigma}x + r''$ easily leaks the entire contents of x in the high order bits. The random number h should have σ more bits than $2^{\ell+\sigma}x$, so r' should have at least $m+\sigma$ bits. The input restriction ensures that z does not overflow.

Line 5 compares two numbers of $\ell + \sigma$ bits. In fact, the reason we have split the randomness of $h/d = r + 2^{\ell+\sigma}r'$ into two numbers r and r', is because we need the value of $(h/d) \bmod 2^{\ell+\sigma}$, which now by construction equals r.

We shall now prove correctness and privacy of our protocol. First, the following properties of integer division, which are readily verified, shall be helpful.

Proposition 1. *For integers a, b, and positive integers n, m, we have that:*
(1) $(a + b) \div n = (a \div n) + (b \div n) + ((a \bmod n) + (b \bmod n)) \div n$
(2) $(a + bn) \div n = (a \div n) + b$
(3) $(a \div n) \div m = a \div (nm) = (a \div m) \div n$
(4) $0 \leq ((a \bmod n) + (b \bmod n)) \div n \leq 1$
Furthermore, $((a \bmod n) + (b \bmod n)) \div n = 1$*, if and only if,* $(a + b) \bmod n < a \bmod n$.

Theorem 1. *Protocol 1 is correct in the passive security model. Concretely, if all parties faithfully follow the protocol, then the correct division result $x \div d$ is returned.*

Proof. In the protocol, P_1 obtains $z = 2^{\ell+\sigma}x + (r + 2^{\ell+\sigma}r')d + r''$. Let $\zeta_1 := (2^{\ell+\sigma}x + r'') \div d$ and $\zeta_2 := r + 2^{\ell+\sigma}r'$, such that $z \div d = \zeta_1 + \zeta_2$. By Proposition 1(1) we have

$$y = (z \div d) \div 2^{\ell+\sigma} = (\zeta_1 \div 2^{\ell+\sigma}) + (\zeta_2 \div 2^{\ell+\sigma}) + b,$$

where b is the result of the comparison $(z \div d) \bmod 2^{\ell+\sigma} < \zeta_2 \bmod 2^{\ell+\sigma}$.

Since $\zeta_2 \bmod 2^{\ell+\sigma} = (r + 2^{\ell+\sigma}r') \bmod 2^{\ell+\sigma} = r$, the bit b is correctly computed in Protocol 1. By Proposition 1(2), and the fact that $r < 2^{\ell+\sigma}$ we have that $\zeta_2 \div 2^{\ell+\sigma} = (r + 2^{\ell+\sigma}r') \div 2^{\ell+\sigma} = r'$. Similarly, since $r'' < 2^{\ell+\sigma}$, and additionally using Proposition 1(3), we see $\zeta_1 \div 2^{\ell+\sigma} = ((2^{\ell+\sigma}x + r'') \div d) \div 2^{\ell+\sigma} = ((2^{\ell+\sigma}x + r'') \div 2^{\ell+\sigma}) \div d = x \div d$. We conclude the output $w = y - b - r'$ equals $x \div d$.

To be complete, we have to make sure that there is no overflow modulo N. Clearly, the largest value computed in the protocol is $z = 2^{\ell+\sigma}x + h + r''$. For $h = (r + 2^{\ell+\sigma}r') \cdot d$ we have $h < (2^{\ell+\sigma} + 2^{\ell+\sigma} \cdot (2^{m+\sigma} - 1)) \cdot (2^{\ell} - 1) = 2^{2\ell+m+2\sigma} - 2^{\ell+m+2\sigma}$, which is upper bounded by $N - 2^{\ell+m+2\sigma}$. The remainder $2^{\ell+\sigma}x + r''$ is upper bounded by $2^{\ell+\sigma+m}$, such that $z < N$. □

Theorem 2. *Assuming the arithmetic black box,* RandInt*, and* Compare *are passively secure functionalities, Protocol 1 is private, i.e. the parties do not learn anything beyond what they can efficiently compute from their respective inputs and outputs.*

Proof. Given an adversary $\mathcal{A}$, we show there exists a polynomial-time simulator $\mathcal{S}$ that simulates the real-world view of $\mathcal{A}$. By using results on composition (e.g., [16, Theorem 7.5.7]), we may assume oracle access to our secure comparison functionality Compare, and the secure random generator RandInt. Let us distinguish two cases.

If the adversary $\mathcal{A}$ does not corrupt P_1, it only receives up to t shares of $[x]$, $[d]$, $[r]$, $[r']$, $[r'']$, $[h]$, $[z]$, $[y]$, $[y']$, $[b]$, and $[x \div d]$. By privacy of the secret-sharing scheme, these are identically distributed to random shares.

If the adversary does corrupt P_1, it additionally learns d and z. Since d is an input for P_1, $\mathcal{S}$ also learns this value. We now complete the proof by showing the simulator $\mathcal{S}$ can generate a uniformly random value that is statistically indistinguishable from z.

Observe that the simulator can perfectly simulate $z_R = h + r'' = (r + 2^{\ell+\sigma}r')d + r''$, since it knows d. We will show the statistical distance between $z = 2^{\ell+\sigma}x + z_R$ and z_R is negligible. First, we describe the probability distribution of z_R.

Let $M = 2^{\ell+m+2\sigma}$, hence $h \in \{i \cdot d \mid 0 \leq i < M\}$. Let α be an integer with $0 \leq \alpha < \alpha_m$, where $\alpha_m = (M - 1)d + 2^{\ell+\sigma}$ is a strict upper bound for z_R. We define the set $S(\alpha) := \{i \in \mathbb{Z} \mid 0 \leq i < M \text{ and } 0 \leq \alpha - di < 2^{\ell+\sigma}\}$. Since h is d multiplied by a uniformly random number

Protocol 1. $\mathsf{PrivateDivision_P}([x],[d],\ell)$.
Constraints: $0 < d < 2^\ell$, $0 \le x < 2^m$, $m + 2(\ell+\sigma) < \log_2 N$, and P_1 knows d.

```
1: $[r] \leftarrow \mathsf{RandInt}(\ell+\sigma)$
   $[r'] \leftarrow \mathsf{RandInt}(m+\sigma)$
   $[r''] \leftarrow \mathsf{RandInt}(\ell+\sigma)$
2: $[h] \leftarrow ([r] + 2^{\ell+\sigma}[r']) \cdot [d]$
3: $[z] \leftarrow 2^{\ell+\sigma}[x] + [h] + [r'']$
   The parties reveal $[z]$ to $P_1$.
4: $P_1$ computes $y \leftarrow z \div (2^{\ell+\sigma}d)$ and $y' \leftarrow (z \div d) \bmod 2^{\ell+\sigma}$
   $P_1$ secret-shares $y$ and $y'$.
5: $[b] \leftarrow \mathsf{Compare}([r] > [y'], \ell+\sigma)$          ▷ $\ell+\sigma$ bits
6: $[w] \leftarrow [y] - ([b] + [r'])$                               ▷ $w = x \div d$
7: return $[w]$
```

in $[0, M)$, we have

$$\Pr(z_R = \alpha) = \sum_i \Pr(h = di) \cdot \Pr(r'' = \alpha - di) = \sum_{i \in S(\alpha)} \frac{1}{M} \cdot \frac{1}{2^{\ell+\sigma}} = \frac{|S(\alpha)|}{M2^{\ell+\sigma}}.$$

For *most* values of α, the cardinality $|S(\alpha)|$ equals $2^{\ell+\sigma} \div d + e$, where $e \in \{0,1\}$. Concretely, this holds (at least) for all α with $2^{\ell+\sigma} \le \alpha \le \alpha_m$. For the remaining values of α ($0 \le \alpha < 2^{\ell+\sigma}$), the upper bound $|S(\alpha)| \le 2^{\ell+\sigma}$ suffices.

Adding $2^{\ell+\sigma}x$ to the random variable z_R 'shifts' the probability distribution to the right, with at most $2^{\ell+m+\sigma}$. Using the above bounds, we can bound the statistical distance between the original and shifted distributions:

$$\Delta[z, z_R] = \frac{1}{2}\sum_\alpha |\Pr(z = \alpha) - \Pr(z_R = \alpha)|.$$

For most values of α, i.e. the values with $2^{\ell+m+\sigma} \le \alpha \le \alpha_m$, the difference $|\Pr(z = \alpha) - \Pr(h + r'' = \alpha)|$ is bounded by $1/(M2^{\ell+\sigma})$. In the two tails, which are both of length $2^{\ell+m+\sigma}$, the probability of either distribution is bounded by $1/M$, such that

$$\Delta[z, z_R] \le \frac{1}{2}\left(\frac{\alpha_m - 2^{\ell+m+\sigma} + 1}{M2^{\ell+\sigma}} + 2 \cdot \frac{2^{\ell+m+\sigma}}{M}\right) < \frac{1}{2M}\left(2^{\ell+m+\sigma} + 2^{\ell+m+\sigma+1}\right) = \frac{3}{2} \cdot 2^{-\sigma},$$

which is negligible in the security parameter σ. □

Although we reveal z in the protocol to P_1 only, the proof shows that it could also be revealed to all parties, and considered as a public value.

By leaving out the secure comparison that computes b, we can get a faster protocol that approximates the division result (has error at most 1), just like in [31, Protocol 3]. An important difference with that protocol is that we choose r to be of length $\ell+\sigma$ instead of ℓ. This is necessary for guaranteeing statistical security. Furthermore, Protocols 4 and 5 from [31], where the comparison result of two integers is approximated by comparing only the most significant bits, are not suitable when the two inputs are close. Note that [31, Protocol 6], which computes an approximate minimum of two encrypted numbers, implicitly overcomes this problem.

In Section 4 we implement an obvious variation of Protocol 1, where the divisor is public. In this variation, all parties learn z and d, and they individually compute y and y'. For the secure comparison, they (non-interactively) fix a secret-sharing of y'. Assuming an actively secure arithmetic black box, this gives an actively secure division protocol in the case of a public divisor.

3 Active security

We now show how to modify Protocol 1 to turn it into an actively secure protocol. First, we make the additional assumption that the underlying primitives, i.e. the arithmetic black box and the functionalities Compare and RandInt, are actively secure. Then, the thing that remains is to ensure that P_1 properly calculates y and y', as in line 4 of Protocol 1.

Since both y and y' can be derived from (the bits of) $z \div d$, the goal is to have P_1 compute $z' = z \div d$, and verify its correctness. This is done by verifying that the remainder $z'' = z - z' \cdot d$ of z after division by d is between 0 and d: $0 \leq z'' < d$.

An efficient and well-known way for P_1 to prove that the value of z' is less than $2^{m+\ell+2\sigma+1}$, is to generate its bits z'_i, $0 \leq i \leq m+\ell+2\sigma$, securely upload them to the arithmetic black box, and let the parties compute $[z'] = \sum_{i=0}^{m+\ell+2\sigma}[z'_i]2^i$. To show that the z'_i are indeed bits, the parties can compute and reveal $z'_i(1-z'_i)$, and check whether the outcome is zero. Since $z'_i(1-z'_i) = z'_i - {z'_i}^2$, a dedicated (and more efficient) squaring could be used instead of a regular secure multiplication. We denote this subprotocol, which takes $\mathcal{O}(m+\sigma)$ secure multiplications, by $\mathsf{RangeProof}(P_1, z', m+\ell+2\sigma+1)$. We work this out in more detail in Protocol 2 ($\mathsf{PrivateDivision_A}$, A stands for active security).

An alternative, and possibly more efficient way would be to replace the range proofs by edaBits [15], which could also be used for the RandBits functionality.

In case the modulus of the secret-sharing scheme is a power of two, a secret-sharing of y' could be obtained from $[z']$ directly by reducing the shares modulo $2^{\ell+\sigma}$. Similarly, the RangeProof of z'' could be avoided, by reducing $[z'']$ modulo 2^{ℓ} directly.

The upper bounds on the comparison and range proof inputs determine the total (order of) number of secure multiplications, such that Protocol 2 has a total complexity of $\mathcal{O}(m+\ell+2\sigma)$ secure multiplications, which is somewhat larger than, but comparable to, the $\mathcal{O}(\ell+\sigma)$ secure multiplications of Protocol 1.

If one of the verifications (of β_1, or β_2), or one of the range proofs fail, the parties know that P_1 has cheated, and the protocol aborts. We formally prove the correctness and security of Protocol 2.

We now argue the security of Protocol 2 in the ideal/real stand-alone model. We show that for each adversary participating in an execution of the real protocol, there exists a simulator that produces indistinguishable output while interacting with the following ideal functionality $\mathcal{F}$:

1. The parties send their shares of $[x]$ and $[d]$ to $\mathcal{F}$.
2. P_1 sends the value of d to $\mathcal{F}$.
3. $\mathcal{F}$ secret-shares the division result $x \div d$ to the parties.

At any step, the adversary is given the option to abort the functionality. Our proof constructs a straight-line black box simulator that only uses oracle access to the adversary; hence, security under general composition, or equivalently in the universal composability (UC) model, follows from existing results [22].

Theorem 3. *Assuming the arithmetic black box, including* RandInt *and* Compare*, are actively secure functionalities, Protocol 2 is actively secure.*

Proof. First we show Protocol 2 computes the correct result, i.e. $w = x \div d$, in case the protocol does not abort. The input bounds for the calls to Compare are guaranteed by the output bounds of the functionalities RangeProof, Compare and RandInt. We only need to show z' indeed equals $z \div d$. Since the values of y and y' are computed from z', using only the functionality Compare, and basic operations from the arithmetic black box, we can use the proof of Theorem 1 to conclude correctness.

Assume the protocol does not abort. Then, since $\beta_2 = 0$, we know the equivalence $z \equiv z' \cdot d + z'' \pmod N$ holds. Furthermore, $\beta_1 = 1$ guarantees $0 \leq z'' < d$, such that $z' = (z+kN) \div d$ for a certain k, $0 \leq k < d$. Using the bounds on x, d, and N, and the bounds obtained from the RangeProofs, we see that firstly $N \div d > N \cdot 2^{-\ell} \geq 2^{m+\ell+2\sigma+1}$, and secondly, since $z = 2^{\ell+\sigma}x + (r + 2^{\ell+\sigma}r') \cdot d + r''$, $z \div d = r + 2^{\ell+\sigma}r' + (2^{\ell+\sigma}x + r'') \div d \leq 2^{m+\ell+2\sigma} - 1 + (2^{m+\ell+\sigma} - 1) \div d < 2^{m+\ell+2\sigma+1}$. It follows that $k = 0$, and $z' = z \div d$.

Second, we show that the protocol is private, given an arithmetic black box with security with abort and t-threshold secret sharing. More precisely, given an adversary $\mathcal{A}$, we show there exists a polynomial-time simulator $\mathcal{S}$ that simulates the real-world view of $\mathcal{A}$. We assume oracle access to our secure comparison functionality Compare, and the secure random generator RandInt. Let us distinguish two cases.

If the adversary $\mathcal{A}$ does not corrupt P_1, it receives up to t shares of the values $[x]$, $[d]$, $[r]$, $[r']$, $[r'']$, $[z'_i]_{0 \leq i \leq m+\ell+2\sigma}$, $[z''_i]_{0 \leq i < \ell}$, $[b]$; depending on whether and when an abort occurred. Since P_1 is not corrupted, the simulator will also receive secret-sharings of known values $\beta_1 = 1$, and $\beta_2 = 0$; it can also generate those. By privacy of the secret-sharing scheme, the simulator can generate shares that are indistinguishable from the shares in the real execution. The remaining secret-sharings $[y]$, $[y']$, $[z']$, $[z'']$, and $[w]$ can be obtained by local computation from the above shares.

If the adversary does corrupt P_1, it additionally learns d and z (if the adversary chooses not to abort before step 3). Since d is an input for P_1, $\mathcal{S}$ also learns this value. As shown in Theorem 2, the simulator $\mathcal{S}$ can generate a uniformly random value z_R that is statistically indistinguishable from z. The adversary can deviate from the protocol by choosing different values for z' and z'' and possibly learn something from the values β_1 and β_2. However, since β_1 and β_2 only depend on val-

Protocol 2. $\mathsf{PrivateDivision}_\mathsf{A}([x], [d], \ell)$.
Parameters: $0 < d < 2^\ell$, $0 \le x < 2^m$, $m + 2(\ell + \sigma) + 1 < \log_2 N$, and only P_1 knows d.

```
1: $[r] \leftarrow \mathsf{RandInt}(\ell + \sigma)$
   $[r'] \leftarrow \mathsf{RandInt}(m + \sigma)$
   $[r''] \leftarrow \mathsf{RandInt}(\ell + \sigma)$
2: $[h] \leftarrow ([r] + 2^{\ell+\sigma}[r']) \cdot [d]$
3: $[z] \leftarrow 2^{\ell+\sigma}[x] + [h] + [r'']$
   The parties reveal $[z]$ towards $P_1$.
4: $P_1$ computes $z' = z \div d$ and its bits $z'_0, \ldots, z'_{m+\ell+2\sigma}$.
5: $P_1$ secret-shares $z'$ via its bit decomposition and shows that it was correctly computed:
   $[z'] \leftarrow \mathsf{RangeProof}(P_1, z', m + \ell + 2\sigma + 1)$        ▷ $[z'] \leftarrow \sum_{i=0}^{m+\ell+2\sigma} 2^i [z_i]$
   $P_1$ computes $z'' = z \bmod d$ and its bits $z''_0, \ldots, z''_{\ell-1}$.
   $[z''] \leftarrow \mathsf{RangeProof}(P_1, z'', \ell)$        ▷ $[z''] \leftarrow \sum_{i=0}^{\ell-1} 2^i [z''_i]$
6: $[\beta_1] \leftarrow \mathsf{Compare}([z''] < [d], \ell)$        ▷ $\ell$ bits
   $[\beta_2] \leftarrow [z] - [z'] \cdot [d] - [z'']$
   Reveal $\beta_1$ and $\beta_2$, and verify $\beta_1 = 1$ and $\beta_2 = 0$.
7: $[y'] \leftarrow \sum_{i=0}^{\ell+\sigma-1} 2^i [z'_i]$ and $[y] \leftarrow \sum_{i=\ell+\sigma}^{m+\ell+2\sigma} 2^{i-\ell-\sigma} [z'_i]$        ▷ $y' = z' \bmod 2^{\ell+\sigma}$ and $y = z' \div 2^{\ell+\sigma}$
   $[b] \leftarrow \mathsf{Compare}([r] > [y'], \ell + \sigma)$        ▷ $\ell + \sigma$ bits
8: $[w] \leftarrow [y] - ([b] + [r'])$        ▷ $w = x \div d$
   **return** $[w]$
```

ues z, z', z'', and d that are known by the adversary, the simulator can also just generate β_1 and β_2 by itself, depending on the input distributions for z' and z'' that it copies from the adversary.

Formally, the ideal-world tuple (z_R, β_1, β_2) is statistically indistinguishable from the real-world (z, β_1, β_2). □

Although we reveal z in the protocol to P_1 only, the proof shows that it could also be revealed to all parties, and considered as a public value.

4 Performance

In this section we describe the way we implemented our protocols, explain a state-of-the-art secret divisor protocol, show the benchmark results, and end with a complexity analysis.

4.1 Implementation

We implemented our protocols in the MP-SPDZ framework [20] using primitives based on 3-party (allowing one corruption) replicated secret-sharing modulo a power of two, and using $\sigma = 40$ bits of statistical security. Concretely, we use the 3-party techniques by [2] for the arithmetic black box, with active security with abort using the protocol from [1].[3] The built-in functionalities RandInt and Compare of MP-SPDZ have been realized using protocols described in [10].

This framework allows the user to write secure algorithms in a Python-derived DSL, which are then compiled to MPC-specific byte-code, to be executed by an optimized virtual machine. We slightly modified the framework by adding a byte-code instruction that performs cleartext integer division.[4]

We ran our benchmarks on three Amazon EC2 virtual servers of type `m5dn.2xlarge` (8 vCPU, 32 GB memory) in the `us-east-2c` region, connected through a 25 Gbps LAN (average round-trip latency 0.075 ms). In our benchmarks, we ran each protocol in a batch of 100 divisions in parallel. We tested each batch three times consecutively and measured elapsed wall clock time for each of them, and then took the minimum runtime.

A complication is that the virtual machine in MP-SPDZ does not support instructions for partially opening values to a specific party P_1 mid-execution, and then have that party run local computations and secret-share

3 We use the `replicated-ring-party` and `sy-rep-ring-party` engines for passive and active security, respectively.

4 Our modification was forked off from commit `15d179a`, after version 0.2.0.

the result to the other parties. To this end, we benchmark a protocol that needs slightly more computation, and thus our benchmarks provide an upper bound for our secure protocol. Instead of partially opening a value to P_1, our benchmarking protocol does not maintain privacy, and instead opens the value to all parties (that are also given the value of d), each running the local computations that only P_1 should. To emulate P_1 secret-sharing the result of the local computations in Protocols 1 and 2, P_1 then inputs a sharing of 0, that is added to the publicly known output of the local computation.

4.2 Division protocol of MP-SPDZ

In the benchmarks, we compare our solutions with the built-in primitives of MP-SPDZ. These primitives are targeted towards fixed-point arithmetic, where each number consists of an integral and a fractional part, both having a fixed number of bits. As a result, to divide an m-bit dividend by an ℓ-bit divisor, they need two $\max\{\ell, m\}$-bit integers, and $\log_2 N \geq 4\max\{\ell, m\} + \sigma$ bits of space.

The division protocol of MP-SPDZ is based on the fixed-point solution from Catrina and Saxena [7]. They use Goldsmidt's method for iteratively approximating x/d: each iteration both dividend and divisor are multiplied with a suitably chosen fraction, such that the divisor converges to one, and hence the dividend to x/d. The desired accuracy determines the number of iterations, but five iterations are already sufficient for 112 bit precision. This makes the approach more efficient than competitors like [8]. Given two inputs consisting of k bits, the division protocol takes $\mathcal{O}(\log_2 k)$ communication rounds, and $\mathcal{O}(k \log_2 k)$ secure multiplications.

4.3 Improved public divisor protocol

Although the division protocol from MP-SPDZ was designed for a secret divisor, they also use it when the divisor is public, which is convenient, but not the most efficient approach for this setting. With a small modification to our *private* divisor protocol Protocol 1, we obtain a division protocol for a *public* divisor, which outperforms the method built into MP-SPDZ. In the benchmarks, we refer to the resulting protocol as Public*.

In the modification, we treat d as public, and let each party "act as P_1". Concretely, instead of opening $[z]$ to a single party, z is revealed to all parties, and each party computes y and y' locally. By running the protocol with actively secure underlying primitives, correctness is guaranteed, since once the value of z is revealed, each party computes the same publicly-known values y and y', and active security of the arithmetic black box ensures these values must be identical.

4.4 Benchmarks

We compare three different secure integer division protocols. The first one with a public divisor that is known to all parties. The second one with a private divisor, known to only one party, and the final one with a secret divisor, unknown to all parties. To show the merits of having a separate solution for private divisor, instead of treating it as a secret divisor, we compare the complexity of all three solutions.

We ran our protocol on different bit-lengths of the dividend, and took the divisor to be half its bit-length. In this way, the outcome of the division, and the divisor, will have the same ℓ bit precision, just as in the benchmarked protocol with secret divisor that uses fixed-point arithmetic. The results can be found in Table 1, the coloured marks referring to the corresponding lines in Figure 1. The table consists of communication and runtime performance of all secure division protocols, in both the passive, and the active security model. The runtime results have also been depicted in Figure 1, where the red (public divisor; ●) and green (secret divisor; ▲) lines are for the protocols built into MP-SPDZ, and the blue (private divisor; ■) and purple (public divisor; ⬙) lines are for our solutions. As mentioned above in Section 4.3, we found that our public version of the private divisor protocol, as depicted in Protocol 1, outperforms the standard MP-SPDZ protocol for public divisor. Likewise, our protocol for a private divisor outperforms the public divisor one from MP-SPDZ, but here it does not make sense to benchmark Public*in the passive case, since it is basically the same protocol as the private divisor protocol (contrary to the actively secure case, where our private divisor protocol is Protocol 2 rather than Protocol 1).

Table 1 also shows the communication complexity, in particular the number of communicated shares. Since in the secret-sharing setting, each secure multiplication requires a fixed amount of communication (and additions can be performed locally), this complexity measure is also a fair indication of computational complexity. The required secret sharing modulus sizes are depicted in Table 2, which are the same for both security

	Divisor		Bit-length of the dividend							
			8	16	24	32	40	48	54	64
Passive security	● Public	Runtime	0.093	0.121	0.145	0.181	0.221	0.242	0.298	0.372
		Communication	4.0	10.8	18.1	33.6	49.7	64.5	86.1	127.5
	■ Private	Runtime	0.015	0.017	0.028	0.031	0.043	0.047	0.055	0.071
		Communication	8.2	10.8	16.5	20.0	27.6	32.1	41.6	47.0
	▲ Secret	Runtime	0.025	0.059	0.123	0.196	0.295	0.387	0.574	0.784
		Communication	8.6	32.6	62.3	121.0	179.6	255.6	339.7	492.7
Active security	● Public	Runtime	0.147	0.262	0.322	0.518	0.663	0.914	1.161	2.069
		Communication	16.6	40.3	63.2	115.0	166.1	211.2	279.6	416.8
	◇ Public*	Runtime	0.121	0.135	0.187	0.228	0.287	0.402	0.502	0.531
		Communication	30.9	38.4	57.0	66.9	91.1	104.4	134.3	149.8
	■ Private	Runtime	0.270	0.292	0.324	0.415	0.450	0.611	0.670	0.718
		Communication	34.9	44.1	54.3	78.1	91.8	122.3	139.7	158.9
	▲ Secret	Runtime	0.125	0.431	0.660	1.305	1.743	2.780	3.693	6.498
		Communication	33.1	113.2	201.9	377.6	545.2	762.1	1005.6	1446.6

Table 1. Runtime (in seconds) and communication (in MB) for our different protocols. Public* is the protocol discussed in Section 4.3. The colored marks correspond to Figure 1.

Table 2. Modulus size of division protocols

Divisor	Input restrictions $x \leq 2^m$ and $d \leq 2^\ell$
Private	$m + 2(\ell + \sigma) + 1 < \log_2 N$
Secret [7]	$m = \ell$ and $4\ell + \sigma < \log_2 N$

models. As explained before, the modulus space for the secret divisor protocol is due to the fixed-point arithmetic approach.

4.5 Complexity analysis

Both the passively secure protocol with private divisor (and its variant with public divisor), and the actively secure protocol with private divisor have a complexity depending on ℓ, m, and σ. Both solutions are linear in the input bit length (assuming linear complexity of RandBits and Compare).

However, the most intensive operation in the passively secure protocol seems to be the secure comparison with inputs of $\ell + \sigma$ bits, whereas the actively secure protocol requires an additional RangeProof of length $m + \ell + 2\sigma + 1$. Figure 1 shows that secure integer division with secret divisor is more complex than public, or private divisor.

5 Conclusion

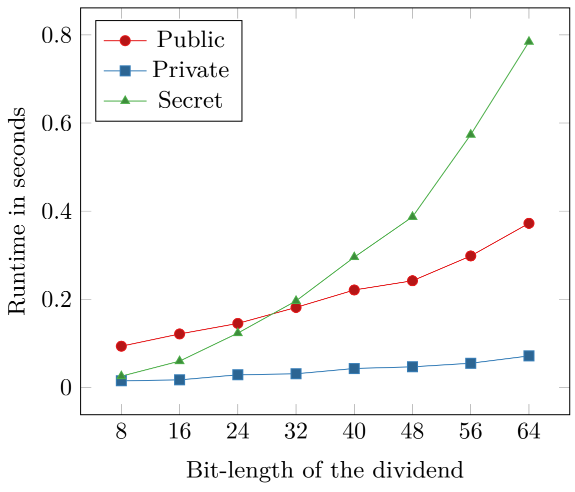
We considered securely computing the integer division with multiple parties, only one party holding the divisor. We showed to which applications this problem was relevant. We developed a new solution for the private divisor problem, and extended it from the passive security model to the actively secure one. This extension required some additional RangeProofs, but maintained a linear complexity in the bit length of the inputs.

We implemented both solutions within the well-known MP-SPDZ framework, and ran benchmarks in an independent cloud environment. The complexity of our private divisor protocols clearly outperformed the secret divisor protocol from MP-SPDZ. It even outperformed the public divisor solution from MP-SPDZ. Our public variant of the private divisor solution did the same in the active security model.

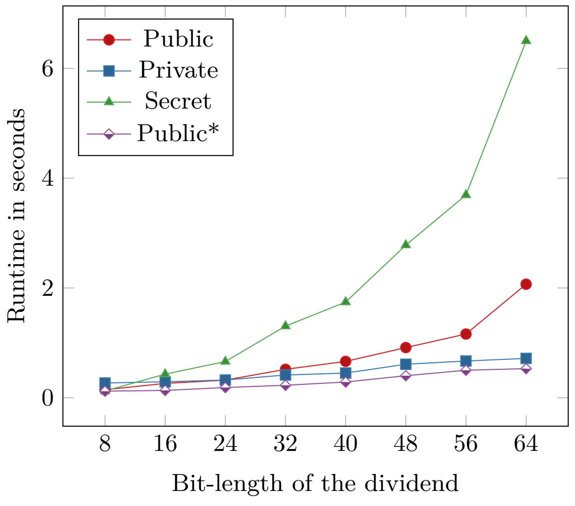
We conclude that we found a new solution for secure integer division with private division, worthwhile for many applications, with the potential of improving performance, even with respect to some public divisor solutions.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 731583 (SODA) and No. 74079 (ALGSTRONGCRYPTO). The authors



(a) Protocols with passive security.



(b) Protocols with active security. Note that Public* represents Protocol 1 with d made public. All protocols use an actively secure arithmetic black box.

Fig. 1. A comparison of the runtimes (in seconds) for division protocols for a public, private, and secret divisor.

would like to thank the anonymous reviewers for several helpful comments.

References

[1] M. Abspoel, A. Dalskov, D. Escudero, and A. Nof. An efficient passive-to-active compiler for honest-majority MPC over rings. Cryptology ePrint Archive, Report 2019/1298, 2019. https://eprint.iacr.org/2019/1298.

[2] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 805–817, Oct. 2016. 10.1145/2976749.2978331.

[3] M. Atallah, M. Bykova, J. Li, K. Frikken, and M. Topkara. Private collaborative forecasting and benchmarking. In *Proceedings of the ACM Workshop on Privacy in an Electronic Society*, pages 103–114, 2004.

[4] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In *Financial Cryptography and Data Security*, volume 5628, pages 325–343. Springer-Verlag, 2009.

[5] P. Bunn and R. Ostrovsky. Secure two-party k-means clustering. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 486–497, USA, 2007.

[6] O. Catrina and S. de Hoogh. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks (SCN)*, volume 6280 of *Lecture Notes in Computer Science*, pages 182–199, 2010.

[7] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 35–50, Berlin / Heidelberg, 2010. Springer.

[8] M. Dahl, C. Ning, and T. Toft. On secure two-party integer division. In *The 16th International Conference on Financial Cryptography and Data Security*, 2012.

[9] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Proceedings of the third Theory of Cryptography Conference, TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer-Verlag, 2006.

[10] I. Damgård, D. Escudero, T. K. Frederiksen, M. Keller, P. Scholl, and N. Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy*, pages 1102–1120. IEEE Computer Society Press, May 2019. 10.1109/SP.2019.00078.

[11] W. Ding, X. Qian, R. Hu, Z. Yan, and R. H. Deng. *Flexible Access Control over Privacy-Preserving Cloud Data Processing*, pages 231–255. Springer International Publishing, Cham, 2020.

[12] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, R. L. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Proceedings of the Privacy Enhancing Technologies Symposium*, pages 235–253, Seattle, USA, 2009.

[13] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk. Privacy-preserving user clustering in a social network. In *IEEE International Workshop on Information Forensics and Security*, 2009.

[14] Z. Erkin, M. Beye, T. Veugen, and I. Lagendijk. Privacy enhanced recommender system. In *Thirty-first Symposium on Information Theory in the Benelux*, pages 35–42, Rotterdam, 2010.

[15] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In *Advances in Cryptology - CRYPTO*, 2020.

[16] O. Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, 2001.

[17] J. Guajardo, B. Mennink, and B. Schoenmakers. Modulo reduction for Paillier encryptions and application to secure statistical analysis. In *SPEED'09*, Lausanne, Switzerland, sep 2009.

[18] K. Hiwatashi, S. Ohata, and K. Nuida. An efficient secure division protocol using approximate multi-bit product and new constant-round building blocks. In M. Conti, J. Zhou, E. Casalicchio, and A. Spognardi, editors, *Applied Cryptography and Network Security*, pages 357–376, Cham, 2020. Springer International Publishing. ISBN 978-3-030-57808-4.

[19] T. Jakobsen. Secure multi-party computation on integers. Master Thesis University of Aarhus, Denmark, 2006.

[20] M. Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1575–1590, New York, NY, USA, 2020. Association for Computing Machinery.

[21] E. Kiltz, G. Leander, and J. Malone-Lee. Secure computation of the mean and related statistics. In *Proceedings of Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 283–302, 2005.

[22] E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. *IACR Cryptol. ePrint Arch.*, 2009:630, 2009. URL http://eprint.iacr.org/2009/630.

[23] H. Morita, N. Attrapadung, S. Ohata, K. Nuida, S. Yamada, K. Shimizu, G. Hanaoka, and K. Asai. Secure division protocol and applications to privacy-preserving chi-squared tests. In *ISITA2018*, October 2018.

[24] M. Naor, B. Pinkas, and R. Summer. Privacy preserving auctions and mechanism design. *ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[25] S. Rane and W. Sun. Privacy preserving string comparisons based on Levenshtein distance. *IEEE Workshop on Information Forensics and Security*, December 2010.

[26] A. Sangers, M. van Heesch, T. Attema, T. Veugen, M. Wiggerman, J. Veldsink, O. Bloemen, and D. Worm. Secure multiparty pagerank algorithm for collaborative fraud detection. In *Financial Cryptography and Data Security*, volume 23, February 2019.

[27] G. H. Satsuya Ohata, Hiraku Morita. Accuracy/efficiency trade-off for privacy-preserving division protocol. In *ISITA2018*, pages 535–539, October 2018.

[28] B. Schoenmakers and P. Tuyls. Efficient binary conversion for Paillier encrypted values. In *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 522–537. Springer, 2006.

[29] T. Toft. *Primitives and Applications for Multi-party Computation*. PhD thesis, University of Aarhus, Aarhus, Denmark, 2007.

[30] C. Ugwuoke, Z. Erkin, and R. L. Lagendijk. Secure fixed-point division for homomorphically encrypted operands. In *Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018*, pages 33:1–33:10. ACM, 2018.

[31] T. Veugen. Encrypted integer division and secure comparison. *International Journal of Applied Cryptography*, 3, 06/2014 2014.

[32] T. Veugen, R. de Haan, R. Cramer, and F. Muller. A framework for secure computations with two non-colluding servers and multiple clients, applied to recommendations. *IEEE Transactions on Information Forensics and Security*, 10(3): 445–457, 2015.