On Secure Two-party Integer Division

Morten Dahl[1], Chao Ning[2,3⋆], and Tomas Toft[1]

[1] Aarhus University, Denmark ⋆⋆ ⋆⋆⋆ †
{mdahl,ttoft}@cs.au.dk
[2] Institute for Interdisciplinary Information Sciences (IIIS)
Tsinghua University, Beijing, P.R. China
ncnfl@mail.tsinghua.edu.cn
[3] School of Computer Science and Technology
Shandong University, Jinan, China

Abstract. We consider the problem of *secure integer division*: given two Paillier encryptions of ℓ-bit values n and d, determine an encryption of $\lfloor \frac{n}{d} \rfloor$ without leaking any information about n or d. We propose two new protocols solving this problem.
The first requires $\mathcal{O}(\ell)$ arithmetic operation on encrypted values (secure addition and multiplication) in $\mathcal{O}(1)$ rounds. This is the most efficient constant-rounds solution to date. The second protocol requires only $\mathcal{O}\left((\log^2 \ell)(\kappa + \log\log \ell)\right)$ arithmetic operations in $\mathcal{O}(\log^2 \ell)$ rounds, where κ is a correctness parameter. Theoretically, this is the most efficient solution to date as all previous solutions have required $\Omega(\ell)$ operations. Indeed, the fact that an $o(\ell)$ solution is possible at all is highly surprising.

Keywords: Secure two-party computation, Secure integer division, Constant-rounds, Bit-Length

1 Introduction

Secure multiparty computation (MPC) allows two or more mutually mistrusting parties to evaluate a function on private data without revealing additional information. Many potential appliations are motivated by business needs, e.g. running

⋆ This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174 and 61173139.
⋆⋆ The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which part of this work was performed.
⋆⋆⋆ The authors acknowledge support from the Center for research in the Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council.
† The authors acknowledge support from Confidential Benchmarking (COBE), supported by The Danish Research Council for Technology and Production.

auctions where no-one gains any information on non-winning bids, or basing decisions on aggregate data from multiple sources. Examples include benchmarking and supply chain management [NT07,CK08].

Classic results show that any function can be computed with polynomial overhead [Yao86,GMW87,BGW88,CCD88] but specialised protocols are often used to improve efficiency: integer arithmetic can for instance be *simulated* using $\mathbb{Z}_M$ arithmetic. On the other hand, this makes non-arithmetic operations difficult, including determining which of two sums is the larger (as needed in the double auction of Bogetoft et al. [BCD+09]), or performing an integer division of sums (essentially the *computation of the mean* problem of Kiltz et al. [KLM05]). Surprisingly many applications become possible given only a small number of primitives – equality, greater-than, and integer division – and improving any primitives immediately allows more efficient high-level protocols.

When the number of parties providing data is large, executing a full MPC protocol may be infeasible. In this case, an alternative is to have the parties (clients) provide their input in, say, encrypted form to two or more servers, $S_1, \ldots, S_N$, who then execute the secure computation. In this case privacy is maintained when we assume that some subset of the servers remain honest. Indeed, such a setting has seen practical use in the Danish "sugar beet auction" [BCD+09] which had more than one thousand bidders and three servers. Further, even if all servers collude it can still be possible to guarantee correctness of the result by publishing a transcript of the MPC protocol and using non-interactive zero-knowledge (NIZK) to demonstrate that the execution was correct, and even if privacy was breached.

In this paper we consider the problem of secure integer division – computing $\lfloor n/d \rfloor$ given n and d – in the two-party setting. Immediate applications include statistics on data from companies in the same business, as well as data-mining tasks, e.g. the k-means clustering protocol of Jagannathan and Wright [JW05]. Further, since the problem of secure integer division is equivalent to that of secure modulo reduction – $n \bmod m = n - m \cdot \lfloor n/m \rfloor$ – any such protocol may be utilized in joint key-generation protocols as e.g. done by Algesheimer et al. [ACS02].

Related work. Algesheimer et al. introduced the problem of secure integer division in the context of passively secure RSA-modulus generation with honest majority [ACS02]; they also noted that standard techniques may provide active security. Their solution was based on Newton iteration and required $\mathcal{O}(\ell)$ work and communication (using the notation of the present paper) in $\mathcal{O}(\log \ell)$ rounds, where ℓ is the bit-length of the inputs. The protocols were later implemented by Jakobsen and From in the passively secure three-party setting [JF05]. Recently, Catrina and Dragulin have applied similar ideas to constructing secure fixed-point arithmetic [CD09].

Regarding constant-rounds solutions to secure integer division, Kiltz et al. proposed specialised protocols based on Taylor series for the related, but simpler, problem of computing the means in a two-party setting [KLM05]. Damgård et al. [DFK+06] later noted that combining the ideas of [ACS02] and [KLM05] with the bit-decompositon (BD) of [DFK+06] implied a general, constant-rounds

modulo reduction (and hence also integer division). The construction was not presented in any detail and no explicit complexity measure was presented, though naturally it at least equalled that of BD, $\mathcal{O}(\ell \log \ell)$. We remark that BD has later been improved in [Tof09,RT10] to $\mathcal{O}(\ell)$ work.

The simpler problem where d is known to all parties (a single party) has been studied by Guajardo et al. [GMS10] and Ning and Xu [NX10] (Veugen [Veu10]).

Finally, we remark that it is possible to "switch technique" mid-protocol and use homomorphic encryption for arithmetic and (small) Yao circuits for primitives such as integer division as done by Henecka et al. [HKS+10]. However, achieving active security in this setting typically requires the use of cut-and-choose techniques. And, while it is possible to use generic NIZK proofs to demonstrate correct protocol execution to the clients, this will be much more expensive than making the non-generic zero-knowledge proofs of our solution non-interactive, e.g., using the Fiat-Shamir heuristic [FS86].

Contribution. We present two two-party protocols for the problem of secure integer division: given Paillier encryptions of ℓ-bit values n and d, compute an encryption of $\lfloor n/d \rfloor$ without leaking any information. Both are based on Taylor series. The first protocol requires $\mathcal{O}(\ell)$ encryptions to be exchanged between the parties in a constant number of rounds; this is quite practical for small inputs, e.g., up to 40 bits. The second protocol communicates $\mathcal{O}\left((\log^2 \ell)(\kappa + \text{loglog}\, \ell)\right)$ encryptions in $\mathcal{O}(\log^2 \ell)$ rounds. Moreover, we are able to avoid bit-decomposition; indeed, as the latter complexity is *sub-linear* in the bit-length, it precludes the use of bit-decompositon entirely. That a sub-linear solution is possible at all is quite surprising, but is of theoretical rather than practical interest. Moreover, in the appendix, we present a protocol for computing the exact bit-length of an encrypted value, which is constructed without replying on bit-decomposition and which may be of independent interest.

We remark that though our protocols are presented in the two-party Pailier-based setting, they are easily converted to both the multiparty setting as well as to other forms of secure arithmetic, e.g. unconditionally secure multiparty computation based on Shamir's secret sharing scheme and the protocols of Ben-Or et al. [Sha79,BGW88]. Note that due to the underlying primitives, security of the sub-linear solution requires the presence of two mutually incorruptible parties, at least with current state-of-the-art knowledge.

The structure of this paper. Section 2 briefly introduces Paillier's encryption scheme, as well as the basic protocols for secure computation. Then Section 3 presents the main idea, i.e. the Taylor series which defines the desired computation, and the overall protocol is presented in Section 4. Sections 5 and 6 then present the two solutions. Concluding, we elaborate on variations in Section 7.

Acknowledgements. The authors would like to thank the anonymous reviewers for their comments and suggestions.

2 Preliminaries

After presenting Paillier encryption and secure two-party computations we introduce a set of protocols used in our constructions. All sub-protocols are secure against malicious (i.e., potentially deviating) attackers. Regarding complexity, we shall use R_π and C_π to denote respectively the number of rounds used and the number of ring elements communicated during a single run of protocol π.

Paillier encryption. Paillier's encryption scheme [Pai99] is an additively homomorphic, sematically secure public key encryption scheme based on the decisional composite residuosity assumption (DCRA) of RSA-moduli, i.e. ultimately based on the hardness of factoring. The construction is based on the observation that for $M = p \cdot q$ being the product of two large primes we have $\mathbb{Z}^*_{M^2} \cong \mathbb{Z}^*_M \times \mathbb{Z}_M$. A ciphertext $c \in \mathbb{Z}^*_{M^2}$ thus contains a random element $R \in \mathbb{Z}^*_M$ and message $m \in \mathbb{Z}_M$. Encryption is defined as $E_M(r, m) = r^M \cdot g^m \bmod M^2$, where r is uniformly random in $\mathbb{Z}^*_{M^2}$, and g generates the subgroup of $\mathbb{Z}^*_{M^2}$ of order M. For integer d, decryption then consists of computing c^d where

$$d \equiv 1 \bmod M \ \wedge \ d \equiv 0 \bmod \phi(M)$$

and solving a discrete logarithm base g (which can be done efficiently in this setting). Suppressing the randomness used, we write $[m]$ to denote an encryption of $m \in \mathbb{Z}_M$ below.

Secure computation. Secure multi-party computation can be based on Paillier encryption with a shared threshold key using the protocols of Cramer et al. [CDN01]. The threshold sharing can be constructed using the ideas of Damgård and Jurik [DJ01]. Though not explicitly stated, apart from guaranteed termination, the protocols of [CDN01] are still valid even if all but a single party are corrupt. In particular this allows the two-party setting. We assume the following setting:

- Alice and Bob know a public Paillier key and share the decryption key.
- Inputs and intermediary values are held in encrypted form by *both* parties.

For secure computation, note that given $[m]$ and $[m']$ both parties may compute an encryption $[m + m']$ by multiplying the ciphertexts. For readability we denote this operation by infix operations in the plaintext space and hence write $[m + m'] \leftarrow [m] + [m']$ instead of $[m + m'] \leftarrow [m] \cdot [m']$.

To perform a multiplication the parties first convert $[m]$ to an additive sharing of m in $\mathbb{Z}_M$: Alice picks her share m_A uniformly at random, encrypts it, and sends $[m_A]$ to Bob. The parties then compute $[m_B] \leftarrow [m] - [m_A]$ and decrypt this towards Bob. Then Alice uses the homomorphic property to compute an encryption of $[m_A \cdot m'] \leftarrow m_A \cdot [m']$, and Bob does likewise to compute $[m_B \cdot m']$. The parties send these to each other. As the final step, both parties compute

$$[m \cdot m'] \leftarrow [m_A \cdot m'] + [m_B \cdot m'] = [(m_A + m_B) \cdot m'] \,.$$

Assuming through-out the entire paper that decryption to a party is a constant round and communication procedure, the complexity of π_{mult} is also constant in both parameters. Finally, note that zero-knowledge (ZK) proofs may be used to ensure correct behavior, e.g. that Alice knows the plaintext of $[m_A]$ and that $[m_A \cdot m']$ and $[m_B \cdot m']$ have been computed correctly. These can be made non-interactive – and hence globally verifiable – using the Fiat-Shamir heuristic [FS86]: The challenge is computed as a hash of the initial message, which can be proven secure in the random oracle model.

Zero-knowledge proof of boundedness. In addition to secure arithmetic in $\mathbb{Z}_M$ we require a zero-knowledge proof of boundedness, i.e. that Alice and Bob may demonstrate to each other that the plaintext of an encryption $[m]$ sent to the other party (where the sender knows m) is smaller than some public bound B. For Paillier encryption this can be achieved with $\mathcal{O}(1)$ communication (of ring elements) using integer commitments and the fact that any non-negative integer can be written as a sum of four squares. See [Bou00,Lip03] for further discussion.

Computing the greater-than relation. Given encryptions $[m]$ and $[m']$ of ℓ-bit values, obtain an encryption $[b]$ of a bit b such that $b = 1$ iff $m > m'$. A constant-rounds protocol $\pi^c_{>?}$ for this can be based off of the comparison protocol of Nishide and Ohta [NO07]; communication complexity is $C_{\pi^c_{>?}} = \mathcal{O}(\ell)$ ring elements. We use $\pi^c_{\leq ?}$ as syntactic sugar for running $\pi^c_{>?}$ with inputs swapped.

A sub-linear protocol, denoted $\pi^s_{>?}$ and $\pi^s_{\leq ?}$, is possible due to Toft [Tof11]. Its complexity is $C_{\pi^s_{>?}} = \mathcal{O}\left((\log \ell)(\kappa + \text{loglog}\, \ell)\right)$ ring elements in $R_{\pi^s_{>?}} = \mathcal{O}(\log \ell)$ rounds, where κ is a correctness parameter.

Computing the inverse of an element. Given an encryption $[x]$ of $x \in \mathbb{Z}^*_M$, compute an encryption $\left[x^{-1}\right]$ of its inverse. We use the protocol from [BB89] which performs this task in a constant number of rounds and communicating a constant number of field elements. We shall use this protocol in both the constant-rounds and the sub-linear protocol and hence simply denote it by π_{inv}.

Bit-decompositon. Decomposing an encrypted ℓ-bit value $[m]$ into binary form, i.e. determining encryptions $[m_{\ell-1}], \ldots, [m_0]$ such that $m_i \in \{0, 1\}$ and $m = \sum_{i=0}^{\ell-1} 2^i \cdot m_i$, is not strictly required as shown in Appendix A. However, for clarity we will use bit-decomposition below. By π^c_{BD} we denote the $\mathcal{O}(1)$ rounds protocol of Reistad and Toft [RT10]; this protocol communicates $C_{\pi^c_{\text{BD}}} = \mathcal{O}(\ell)$ encryptions (and ring elements).

Prefix-or of a sequence of bits. Given encrypted bits $[x_{\ell-1}], \ldots, [x_0]$, compute encrypted bits $[y_{\ell-1}], \ldots, [y_0]$ such that $y_i = \bigvee_{j=i}^{\ell-1} x_j$. An $\mathcal{O}(1)$-rounds protocol communicating $C_{\pi^c_{\text{pre-}\vee}} = \mathcal{O}(\ell)$ elements, $\pi^c_{\text{pre-}\vee}$, is provided in [DFK+06].

Powers of a number. Given an encrypted number $[x]$ and public $\omega \in \mathbb{Z}$, compute $\left[x^1\right], \left[x^2\right], \ldots, [x^\omega]$. This can be accomplished using a prefix-product protocol [BB89,DFK+06] by giving $[x]$ for all ω inputs. The protocol $\pi^c_{\text{pre-}\Pi}$ runs in $\mathcal{O}(1)$ rounds and uses $C_{\pi^c_{\text{pre-}\Pi}} = \mathcal{O}(\omega)$ communication.

3 The Intuition Behind the Constructions

In this section we take a high-level view and present the ideas behind the desired computation. The following sections then explain how to do this securely in the stated complexity. Assume in the following that n and d are ℓ-bit integers, and let k be a suitable large, public integer. Our solutions then consist of two steps:

I. Compute an encrypted approximation $[\tilde{a}]$ of $a = \lfloor 2^k/d \rfloor$
II. Compute $[\lfloor n/d \rfloor]$ as $\lfloor ([\tilde{a}] \cdot [n])/2^k \rfloor$

Step I is explained over the reals in Section 3.1. This is then converted to integer computation in Section 3.2 and finally realised using $\mathbb{Z}_M$ arithmetic in Section 3.3. Note that the integer division in step II is simpler as 2^k is public.

3.1 The Taylor Series

Similarly to [KLM05] or the constant depth division circuit of Hesse et al. [HAB02], we start with a geometric series to compute a "k-shifted" approximation of $1/d$:

$$\frac{1}{\alpha} = \sum_{i=0}^{\infty}(1-\alpha)^i = \sum_{i=0}^{\omega}(1-\alpha)^i + \epsilon_\omega \tag{1}$$

where $\epsilon_\omega = \sum_{i=\omega+1}^{\infty}(1-\alpha)^i$. This is easily verified for any real $0 < \alpha < 1$. Further, approximating $1/\alpha$ by keeping only the first $\omega + 1$ terms of the summation introduces an additive error of ϵ_ω. If $0 < 1 - \alpha \leq 1/2$ then this error is at most

$$\epsilon_\omega = \sum_{i=\omega+1}^{\infty}(1-\alpha)^i \;=\; (1-\alpha)^{\omega+1} \cdot \sum_{i=0}^{\infty}(1-\alpha)^i \;\leq\; 2^{-\omega-1} \cdot \frac{1}{\alpha} \;\leq\; 2^{-\omega}. \tag{2}$$

By picking ω sufficiently large this ensures an appropriately small error below.

3.2 Converting the Taylor Series to an Integer Computation

Multiplying $1/\alpha$ by a power of two "shifts" the value; this ensures that each of the $\omega + 1$ terms of the finite sum of Eq. (1) are integer. The non-integer part of the shifted value is entirely contained in ϵ_ω, which will be discarded.

Let $\ell_d = \lfloor \log_2(d) + 1 \rfloor$ be the bit-length of d such that $2^{\ell_d - 1} \leq d < 2^{\ell_d}$. Define ℓ_n similarly. For accuracy it is sufficient to use any $\omega \geq max\{\ell_n - \ell_d, 0\}$. However, letting ω depend on ℓ_n or ℓ_d leaks information as ω must be public. Instead we let $\omega = \ell$, which clearly satisfies the accuracy condition. For $\alpha = d/2^{\ell_d}$ and

$k = \ell^2 + \ell$ the following expression gives us $1/d$ shifted up by k bits:

$$\begin{aligned}
\frac{2^k}{d} &= 2^{k-\ell_d} \cdot \frac{1}{d/2^{\ell_d}} \\
&= 2^{k-\ell_d} \cdot \left(\sum_{i=0}^{\omega} \left(1 - \frac{d}{2^{\ell_d}}\right)^i + \epsilon_\omega \right) \\
&= 2^{k-\ell_d(\omega+1)} \cdot \sum_{i=0}^{\omega} \left(1 - \frac{d}{2^{\ell_d}}\right)^i \cdot 2^{\ell_d \omega} + 2^{k-\ell_d} \cdot \epsilon_\omega \\
&= 2^{k-\ell_d(\omega+1)} \sum_{i=0}^{\omega} \left(1 - \frac{d}{2^{\ell_d}}\right)^i \cdot \left(2^{\ell_d}\right)^i \cdot 2^{\ell_d(\omega-i)} + 2^{k-\ell_d} \cdot \epsilon_\omega \\
&= 2^{k-\ell_d(\omega+1)} \sum_{i=0}^{\omega} \left(2^{\ell_d} - d\right)^i \cdot 2^{\ell_d(\omega-i)} + 2^{k-\ell_d} \cdot \epsilon_\omega.
\end{aligned}$$

We define the desired approximation of $2^k/d$ as

$$\tilde{a} = 2^{k-\ell_d(\omega+1)} \cdot \sum_{i=0}^{\omega} \left(2^{\ell_d} - d\right)^i \cdot 2^{\ell_d(\omega-i)}. \tag{3}$$

Note that not only is this an integer since $k \geq \ell_d(\omega+1)$ and $2^{\ell_d} > d$, it may also be computed as the product of $2^{k-\ell_d(\omega+1)}$ and the evaluation of the integer polynomial with coefficients $2^{\ell_d(\omega-i)}$ for $0 \leq i \leq \omega$ at point $2^{\ell_d} - d$. Furthermore, since $0 < 1 - d/2^{\ell_d} \leq 1/2$ we have a bound on the additive error by Eq. (2):

$$2^{k-\ell_d} \cdot \epsilon_\omega \leq 2^{k-\ell_d-\omega}.$$

This ensures that the result computed in step II is off by at most 1: we have

$$\left\lfloor \frac{n}{d} \right\rfloor = \left\lfloor \frac{n \cdot \left(\tilde{a} + 2^{k-\ell_d} \cdot \epsilon_\omega\right)}{2^k} \right\rfloor = \left\lfloor \frac{n \cdot \tilde{a}}{2^k} + \frac{n \cdot 2^{k-\ell_d} \cdot \epsilon_\omega}{2^k} \right\rfloor \tag{4}$$

and see that the second summand is bound by

$$\frac{n \cdot 2^{k-\ell_d} \cdot \epsilon_\omega}{2^k} \leq \frac{n \cdot 2^{k-\ell_d-\omega}}{2^k} < \frac{2^k}{2^k} = 1$$

since $\ell_n \leq \omega$. It is clear that $\lfloor \frac{n \cdot \tilde{a}}{2^k} \rfloor$ is the desired result *except* that the sum of the error, $n \cdot 2^{k-\ell_d} \cdot \epsilon_\omega$, and the discarded bits of the approximation, $n \cdot \tilde{a} \bmod 2^k$, may be greater than 2^k; i.e. the result may have an additive error of -1 due to a lost carry-bit.

To recap: Given integers $2^{k-\ell_d(\omega+1)}$, $2^{\ell_d} - d$ and $2^{\ell_d(\omega-i)}$ for $0 \leq i \leq \ell$, performing step I yields an approximation $\tilde{a}$ of $2^k/d$ using Eq. (3). Down-shifting this almost gives the desired result, namely $\tilde{q} \in \{q, q-1\}$, where $q = \lfloor n/d \rfloor$.

3.3 Performing the Integer Computation Using $\mathbb{Z}_M$ Arithmetic

The underlying primitives provide secure $\mathbb{Z}_M$ arithmetic, with $M = pq$ being the Paillier key whose secret key is held jointly by the parties. We assume[4] that

$$M \gg 2^{\ell^2+\ell+\kappa_s},$$

where κ_s is a statistical security parameter, e.g. $\kappa_s = 100$. This implies that no "overflow" modulo M occurs in Eq. (3), hence it can be seen as occurring in $\mathbb{Z}_M$. However, for efficiency reasons we rephrase the expression as

$$\begin{aligned}
\tilde{a} &= 2^{k-\ell_d(\omega+1)} \cdot \sum_{i=0}^{\omega} \left(2^{\ell_d} - d\right)^i \cdot 2^{\ell_d(\omega-i)} \\
&= 2^{k-\ell_d(\omega+1)} \cdot \sum_{i=0}^{\omega} \left(2^{\ell_d} - d\right)^i \cdot \left(2^{-\ell_d}\right)^i \cdot 2^{\ell_d\omega} \\
&= 2^{k-\ell_d} \cdot \sum_{i=0}^{\omega} \left(\left(2^{\ell_d} - d\right) \cdot 2^{-\ell_d}\right)^i
\end{aligned} \tag{5}$$

where addition and multiplication occur in $\mathbb{Z}_M$. This should no longer be seen as an integer computation, however, the key observation is that it is irrelevant *how* the encryption $[\tilde{a}]$ is obtained; what matters is that the plaintext is correct. Essentially this altered calculation can be viewed as using the encoding of rational values suggested in [FSW02]. Note that this simplifies the desired calculation: we now only need the values $2^{k-\ell_d}$, $2^{\ell_d} - d$, and $2^{-\ell_d}$ as well as the evaluation of a $\mathbb{Z}_M$-polynomial with known coefficients (all equal to 1).

4 The Overall Division Protocol

Having presented the desired $\mathbb{Z}_M$-expression for computing the approximation $\tilde{a} \approx 2^k/d$ in Section 3.3 above, the goal now is to give a high-level view of the actual protocol. We first formalise the required sub-tasks, and then present the overall protocol based on assumed protocols for these. Instantiating these protocols with either the constant-rounds (Section 5) or the sub-linear (Section 6) versions of the sub-protocols we obtain our two division protocols.

4.1 Sub-tasks and Sub-protocols

In addition to the basic primitives of Section 2 we require the following sub-protocols:

- π_{BL}: Given an encryption $[d]$ of an ℓ-bit value d, determine an encryption $\left[2^{\ell_d}\right]$ for $\ell_d = \lfloor log_2(d) + 1 \rfloor$

[4] M needs to be at least a thousand bits long to ensure security of the Paillier scheme and hence this assumption is not as bad as it may appear at first glance.

– π_{poly}: Given an encryption $[p]$ of $p \in \mathbb{Z}_M^*$, evaluate the known polynomial $A(x) = \sum_{i=0}^{\omega} x^i$ over $\mathbb{Z}_M$ securely at point p, i.e. compute encryption $[A(p)]$
– π_{trunc}: Given an encryption $[\hat{q}]$ of an $(\ell + k)$-bit value $\hat{q} \in \mathbb{Z}_M$, compute an encryption $[\tilde{q}]$ of an approximation of $\lfloor \hat{q}/2^k \rfloor$ s.t. $\tilde{q} = \lfloor \hat{q}/2^k \rfloor + \epsilon$ for $\epsilon \in \{0, 1\}$.

4.2 The High-level View

The full division protocol is seen in Figure 1 and proceeds by the following steps:

I. Compute an encryption $[\tilde{a}]$ of the approximation
   (a) Determine $\left[2^{\ell_d}\right]$ and in turn compute $\left[2^{k-\ell_d}\right]$ and $[p] = \left[(2^{\ell_d} - d) \cdot 2^{-\ell_d}\right]$
   (b) Evaluate the polynomial of Eq. (5) in $[p]$ and securely multiply by $\left[2^{k-\ell_d}\right]$
II. Compute $[\lfloor n/d \rfloor]$
   (a) Obtain encryption $[\tilde{q}]$ of $\tilde{q} \approx \lfloor n/d \rfloor$ by computing and truncating $[n \cdot \tilde{a}]$
   (b) Eliminate errors introduced by approximations, i.e., compute $[q]$ from $[\tilde{q}]$

where the elimination of errors are performed by two secure comparions.

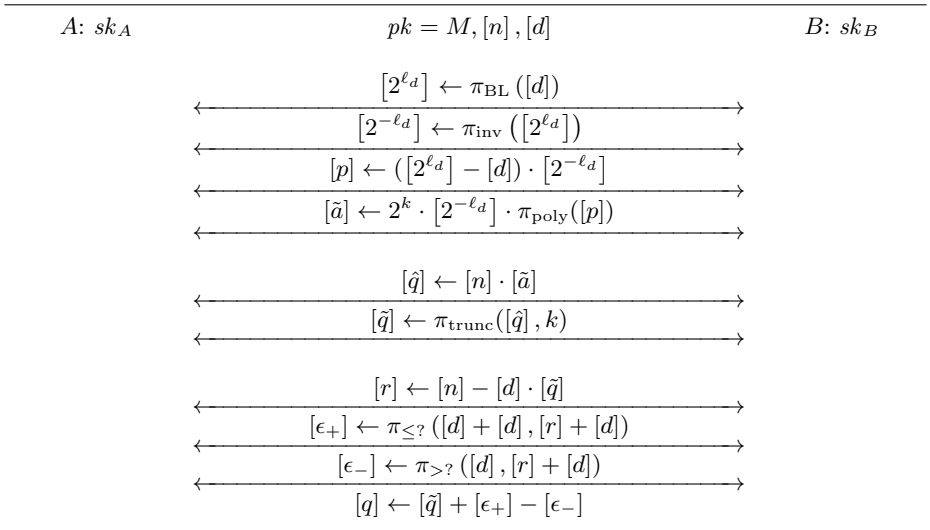

Fig. 1. The full division protocol, $\pi_{\text{div}}([n], [d]) \mapsto [\lfloor n/d \rfloor]$

Correctness. Correctness follows almost entirely from the previous section. For the plaintext of $[\hat{q}]$, the most significant bits are off by at most 1:

$$\lfloor \hat{q}/2^k \rfloor \in \{\lfloor n/d \rfloor, \lfloor n/d \rfloor - 1\}.$$

The execution of π_{trunc} may introduce an additional additive error, i.e. we have

$$\tilde{q} \in \{\lfloor n/d \rfloor - 1, \lfloor n/d \rfloor, \lfloor n/d \rfloor + 1\}.$$

Using $r = n - d \cdot \tilde{q} \in [-d; 2d[$ we can securely determine which case we are in. Namely, $\tilde{q} + 1 = \lfloor n/d \rfloor$ when $d \leq r$ and $\tilde{q} - 1 = \lfloor n/d \rfloor$ when $0 > r$. In order to deal only with positive integers we scale these tests to respectively $2d \leq r + d$ and $d > r + d$. Letting ϵ_+ and ϵ_- denote the Boolean outcome of these tests, it follows that $q = \tilde{q} + \epsilon_+ - \epsilon_- = \lfloor n/d \rfloor$.

Privacy. The protocol reveals no information about the inputs (other than the desired encryption of the result). This follows from the fact that no value is ever decrypted and that we only invoke secure sub-protocols which do not leak information. We note that π_{inv} and π_{poly} require the input to be invertible – this is indeed the case as M is the product of two odd primes, $p, q \approx \sqrt{M}$, while $2^{\ell_d}, 2^{\ell_d} - d \leq 2^{\ell} \ll \sqrt{M}$. Further, the input $[n \cdot \tilde{a}]$ for the truncation is $\ell + k$-bit long as $n < 2^{\ell}$ and $\tilde{a} \leq 2^k/d \leq 2^k$, and hence the input is of the correct size.

A formal security proof using the real/ideal paradigm requires the construction of a simulator for each party. These are straightforward to construct from the simulators of the sub-protocols; as our protocol consists of the sequential evaluation of sub-protocols, the overall simulator simply consists of the sequential execution of the simulators of these.

Complexity. The complexity depends on the details of the sub-protocols π_{BL}, π_{poly}, π_{trunc}, and $\pi_{>?}$. Formally we have

$$\begin{aligned} R_{\pi_{\text{div}}} &= R_{\pi_{\text{BL}}} + R_{\pi_{\text{inv}}} + R_{\pi_{\text{poly}}} + R_{\pi_{\text{trunc}}} + 2 \cdot R_{\pi_{>?}} + 3 \cdot R_{\pi_{\text{mult}}} \\ &= R_{\pi_{\text{BL}}} + R_{\pi_{\text{poly}}} + R_{\pi_{\text{trunc}}} + \mathcal{O}(R_{\pi_{>?}}) + \mathcal{O}(1) \\ C_{\pi_{\text{div}}} &= C_{\pi_{\text{BL}}} + C_{\pi_{\text{inv}}} + C_{\pi_{\text{poly}}} + C_{\pi_{\text{trunc}}} + 2 \cdot C_{\pi_{>?}} + 3 \cdot C_{\pi_{\text{mult}}} \\ &= C_{\pi_{\text{BL}}} + C_{\pi_{\text{poly}}} + C_{\pi_{\text{trunc}}} + \mathcal{O}(C_{\pi_{>?}}) + \mathcal{O}(1) \end{aligned} \tag{6}$$

such that for the constant-rounds instantiation we get $R_{\pi^c_{\text{div}}} = R_{\pi^c_{\text{BL}}} + R_{\pi^c_{\text{poly}}} + R_{\pi^c_{\text{trunc}}} + \mathcal{O}(1)$ and $C_{\pi^c_{\text{div}}} = C_{\pi^c_{\text{BL}}} + C_{\pi^c_{\text{poly}}} + C_{\pi^c_{\text{trunc}}} + \mathcal{O}(\ell)$. Likewise, for the sub-linear instantiation we get $R_{\pi^s_{\text{div}}} = R_{\pi^s_{\text{BL}}} + R_{\pi^s_{\text{poly}}} + R_{\pi^s_{\text{trunc}}} + \mathcal{O}(log\ \ell)$ and $C_{\pi^s_{\text{div}}} = C_{\pi^s_{\text{BL}}} + C_{\pi^s_{\text{poly}}} + C_{\pi^s_{\text{trunc}}} + \mathcal{O}\left((\log \ell)(\kappa + \text{loglog}\, \ell)\right)$. Finally, a slight optimisation regarding rounds is possible by invoking $\pi_{>?}$ and $\pi_{\leq ?}$ in parallel.

Active Security. The protocol in Figure 1 is only passively secure. However, obtaining active security is straightforward by executing appropriate ZK proofs. This increases the communication complexity by a constant factor.

5 The Constant-rounds Protocol

In this section we plug in protocols for the three sub-tasks. All protocols use a constant number of rounds and linear communication. Combined with the previous section this provides a constant-rounds protocol for division.

5.1 The constant-rounds π_{BL} protocol

In Appendix A we give a π^c_{BL} protocol that, somewhat surprising, does not rely on bit-decomposition. Here, for clarity the π^c_{BL} protocol presented in Figure 2 is composed of two protocols introduced in Section 2: π^c_{BD} and $\pi^c_{\text{pre-}\vee}$. To recap, the former takes an encryption $[d]$ as input and returns a set of encrypted bits $[x_{\ell-1}], \ldots, [x_0]$ for which it holds that $\sum_{i=0}^{\ell-1} x_i \cdot 2^i = d$. The latter takes such a set of encrypted bits and returns another set with the property that $y_i = \bigvee_{j=i}^{\ell-1} x_j$.

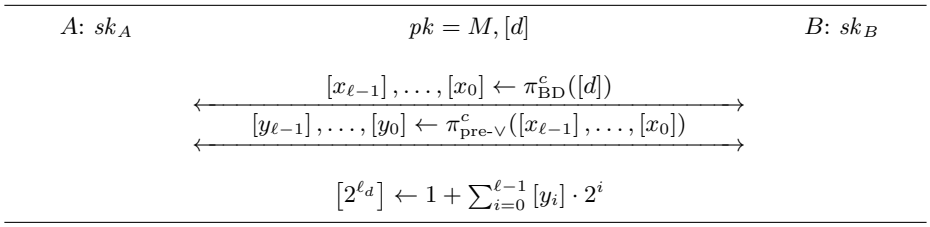

Fig. 2. Constant-rounds bit-length protocol, $\pi^c_{\text{BL}}([d]) \mapsto \left[2^{\ell_d}\right]$

Correctness. By the correctness of the two sub-protocols we only have to argue the correctness of the final step. Note that the result of $\pi^c_{\text{pre-}\vee}$ is a set such that $y_i = 1$ if and only if $d \geq 2^i$. This means that $1 + \sum_{i=0}^{\ell-1} y_i \cdot 2^i$ is the desired 2^{ℓ_d}.

Privacy and Active Security. Follows immediately by the privacy and security guarantees of the two sub-protocols.

Complexity. Since the final step of π^c_{BL} is a local computation we simply have that $R_{\pi^c_{\text{BL}}} = R_{\pi^c_{\text{BD}}} + R_{\pi^c_{\text{pre-}\vee}} = \mathcal{O}(1)$ and $C_{\pi^c_{\text{BL}}} = C_{\pi^c_{\text{BD}}} + C_{\pi^c_{\text{pre-}\vee}} = \mathcal{O}(\ell)$.

5.2 The constant-rounds π_{poly} protocol

As shown in the protocol in Figure 3, we simply evaluate polynomial $A(x) = \sum_{i=0}^{\omega} x^i$ in point $p = \left(2^{\ell_d} - d\right) \cdot 2^{-\ell_d}$ using the prefix-product protocol $\pi^c_{\text{pre-}\Pi}$. This gives encryptions of $p^1, p^2, \ldots, p^\omega$ – and knowing these all there is left to do is to sum these together with $p^0 = 1$ to form $A(p)$.

Correctness, Privacy, Complexity, and Active Security. Noting that the second step of π^c_{poly} is a local computation, all properties directly reflect those of the $\pi^c_{\text{pre-}\Pi}$ subprotocol. Formally, $R_{\pi^c_{\text{poly}}} = \mathcal{O}(1)$ and $C_{\pi^c_{\text{poly}}} = \mathcal{O}(\omega)$.

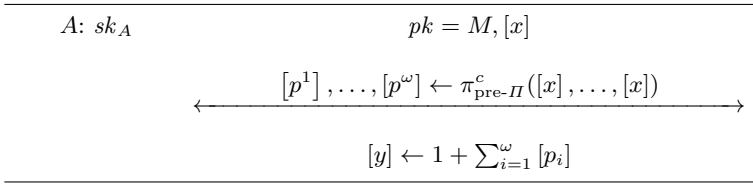

Fig. 3. Constant-rounds polynomial evaluation protocol, $\pi^c_{\text{poly}}([x]) \mapsto [A(x)]$

5.3 The constant-rounds π_{trunc} protocol

Our constant-rounds protocol for truncation (shown in Figure 4) takes encryption $[\hat{q}]$ and public k as input and returns $[\tilde{q}]$ such that $\tilde{q} \approx \lfloor q/2^k \rfloor$. The result may have an additive error $c \leq 1$. It is possible to eliminate this error with a comparison $[c] \leftarrow ([\tilde{q}] \cdot 2^k >_? [\hat{q}])$, and computing the correct result as $[q] \leftarrow [\tilde{q}] - [c]$. However, instead of comparing two ℓ^2-bit numbers here, we handle the error in the main protocol with a comparison of two ℓ-bit numbers instead.

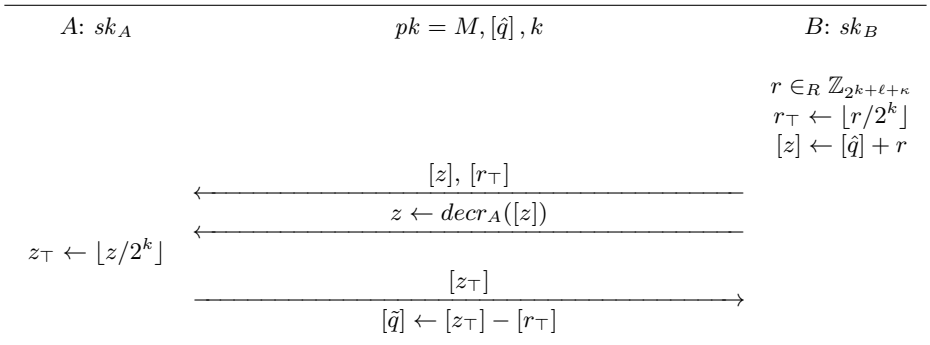

Fig. 4. Constant-rounds truncation protocol, $\pi^c_{\text{trunc}}([\hat{q}], k) \mapsto \left[\lfloor \hat{q}/2^k \rfloor + c\right]$

To perform the truncation, party B first picks a random integer of a bit-length sufficient for using it as a mask for $\hat{q}$. He also stores the $\ell + \kappa$ most significent bits of r as $r_\top$ and computes an encryption of it. Upon receiving $[z]$, the masked value of $\hat{q}$, A and B now decrypt $[z]$ for A to see. After learning this value z, A can locally perform the truncation to form $z_\top$. She sends an encryption of this value to B and both can finally compute the output locally by $[z_\top] - [r_\top]$.

Correctness. When computing z it may happen that r causes a carry bit c from the k least significant bits to spill over into the $\ell + \kappa$ most significant bits. In this case the truncation of z will maintain this carry bit, causing the result of $z_\top - r_\top$ to be $\lfloor \hat{q}/2^k \rfloor + 1$ instead of $\lfloor \hat{q}/2^k \rfloor$. For efficiency we allow this error.

Privacy. The only point where information could potentially be leaked is through A seeing z. However, since r is chosen uniformly at random and κ bit longer than $\hat{q}$, z leaks information about $\hat{q}$ with probability negligible in κ.

Complexity. We see that the complexity of π^c_{trunc} is $R_{\pi^c_{\text{trunc}}} = 2 + R_{decr} = \mathcal{O}(1)$ where R_{decr} is the round complexity of a decryption, assumed to be constant. Likewise the communication complexity is $C_{\pi^c_{\text{trunc}}} = 3 + C_{decr} = \mathcal{O}(1)$.

Active Security. To obtain active security B must also send $\left[r_\perp = r \bmod 2^k\right]$ to A, who in turn must also send $\left[z_\perp = z \bmod 2^k\right]$. B can now append a zero-knowledge proof that $z = (r_\top \cdot 2^k + r_\perp) + \hat{q}$ as well as proofs that both $r_\top$ and $r_\perp$ are within the correct bounds. Similary, A also appends a proof of $z = z_\top \cdot 2^k + z_\perp$ and that $z_\top$ and $z_\perp$ are within bounds.

5.4 Combined Protocol and Analysis

By plugging the protocols introduced in this section into the π_{div} protocol of Section 4 we obtain our constant-rounds division protocol π^c_{div}. Correctness, privacy, and active security follow from the discussions above. Using the complexity expressions in Eq. 6 from Section 4 and the fact that $\omega = \ell$ we get:

$$R_{\pi^c_{\text{div}}} = R_{\pi^c_{\text{BL}}} + R_{\pi^c_{\text{poly}}} + R_{\pi^c_{\text{trunc}}} + \mathcal{O}(1) = \mathcal{O}(1)$$

$$C_{\pi^c_{\text{div}}} = C_{\pi^c_{\text{BL}}} + C_{\pi^c_{\text{poly}}} + C_{\pi^c_{\text{trunc}}} + \mathcal{O}(\ell) = \mathcal{O}(\omega) + \mathcal{O}(\ell) = \mathcal{O}(\ell).$$

6 The Sub-linear Protocol

In this section we give the protocols needed for giving the division protocol of Section 3 a sub-linear communication complexity. We can reuse the truncation protocol π^c_{trunc} from Section 5 and hence only present two new π_{BL} and π_{poly} protocols.

6.1 The sub-linear π_{BL} protocol

To compute $\left[2^{\ell_d}\right]$ from $[d]$ in sub-linear communication complexity we take inspiration from [Tof11] and perform, in a sense, a binary search. Assuming we have a protocol $\pi^s_{\leq ?}$ for performing comparison of two encrypted numbers, we give the protocol in Figure 5. For simplicity we assume that $\ell = 2^\gamma$ for some integer γ.

Intuitively, our construction recursively computes a pointer p into the binary representation of d. Initially p points to the first bit position ($p_0 = 2^0$). In the first round we then ask in which half of the binary representation of d the most significant 1 occurs and store the result in bit c_1. Next we update p to point to position $\ell/2^1$ if $c = 1$ (i.e. $p_1 = p_0 \cdot 2^{\ell/2^1}$) and to the same position as before if $c = 0$ (i.e. $p_1 = p_0 \cdot 1$). Iterating in this way p will eventually point to the position of the most significant bit of d. Shifting the position by one will give us integer 2^{ℓ_d}.

Correctness and Privacy. Correctness follows from the above description of the protocol, and privacy follows immediately from the sub-protocols as we only compute on encrypted values.

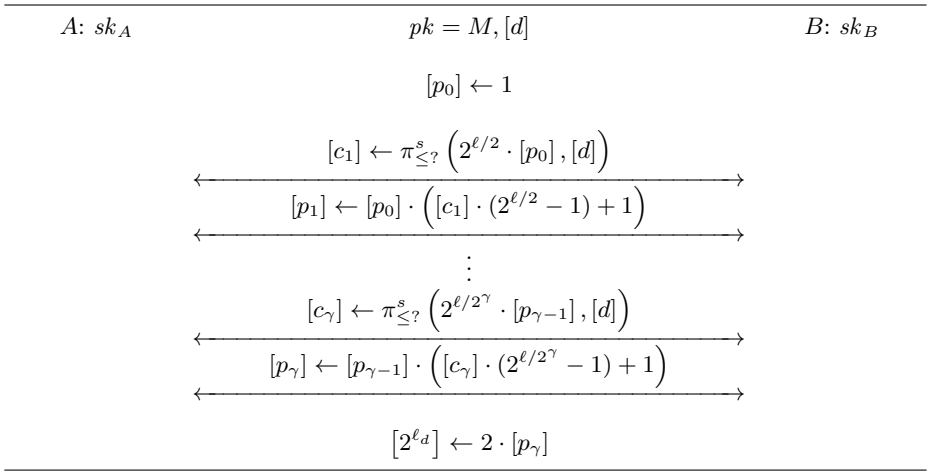

Fig. 5. Sub-linear bit-length protocol, $\pi^s_{\mathrm{BL}}([d]) \mapsto \left[2^{\ell_d}\right]$

Complexity. The protocol requires $\gamma = \log_2 \ell$ iterations, each requiring one comparison and one multiplication (not counting multiplication by public values). Hence we get round complexity $R_{\pi^s_{\mathrm{BL}}} = \gamma \cdot (R_{\pi^s_{\leq ?}} + R_{\pi_{\mathrm{mult}}}) = \mathcal{O}(\log^2 \ell)$ and communication complexity $C_{\pi^s_{\mathrm{BL}}} = \gamma \cdot (C_{\pi^s_{\leq ?}} + C_{\pi_{\mathrm{mult}}}) = \mathcal{O}\left((\log^2 \ell)(\kappa + \mathrm{loglog}\, \ell)\right)$.

Active Security. Since the sub-protocol is actively secure, we only have to append zero-knowledge proofs of correctness to every multiplication in order to make the protocol resistant against active attackers. This increases the number of messages communicated but only by a constant factor.

6.2 The sub-linear $\pi_{\mathbf{poly}}$ protocol

Evaluating the $A(x) = \sum_{i=0}^{\omega} x^i$ polynomial at a point p can be done by a method similar to "square and multiply". We give the protocol in Figure 6 where for simplicity we have assumed that $\omega = 2^\gamma$ for some integer γ. The intuition behind the notation is that $\sigma_j = \sum_{i=1}^{2^j} x^i$ and $x_j = x^{2^j}$ – it is not hard to see that this is indeed the case. Specifically this gives us that $\sigma_\gamma = \sum_{i=1}^{2^\gamma} x^i$ and hence $\sigma_\gamma + 1 = \left(\sum_{i=1}^{\omega} x^i\right) + 1 = \sum_{i=0}^{\omega} x^i$ as required.

Correctness, Privacy, and Complexity. The first two follow respectively from the description above and from that fact that only arithmetical operations on

encryptions are performed. For complexity we have that the protocol requires $\gamma = \log_2 \omega$ iterations with two multiplications in each. Hence the round complexity is $R_{\pi^s_{\text{poly}}} = \gamma \cdot (2 \cdot R_{\pi_{\text{mult}}}) = \mathcal{O}(\log \omega)$, and likewise for the communication complexity $C_{\pi^s_{\text{poly}}} = \gamma \cdot (2 \cdot C_{\pi_{\text{mult}}}) = \mathcal{O}(\log \omega)$.

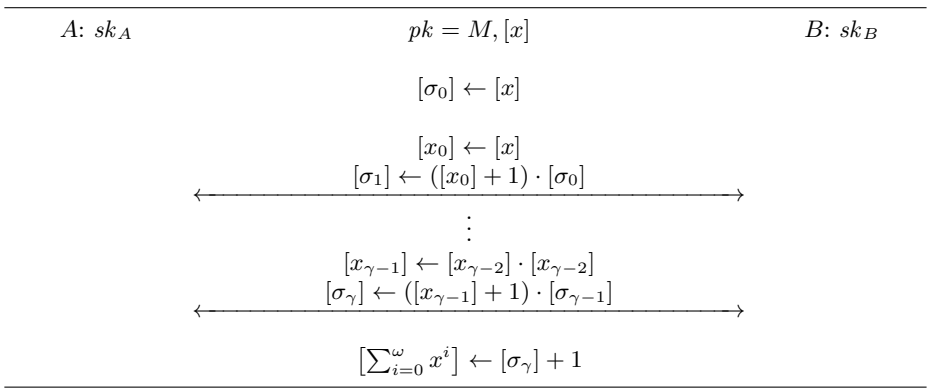

Fig. 6. Sub-linear polynomial evaluation protocol, $\pi^s_{\text{poly}}([x]) \mapsto [A(x)]$

Active Security. By appending zero-knowledge proofs of correctness to every multiplication we make the protocol resistant against active attackers. This increases the number of messages communicated but only by a constant factor.

6.3 The sub-linear π_{trunc} protocol

The truncation protocol π^c_{trunc} of Section 5 is efficient enought to be reused for the sub-linear protocol π^s_{trunc}: only a single operation is performed, namely the decryption of $[z]$. The remaining operations can be carried out locally.

6.4 Combined Protocol and Analysis

Our sub-linear division protocol π^s_{div} is obtained from the π_{div} protocol of Section 4. Correctness, privacy, and active security follow from the discussions in the previous sections and in this section. As for complexity, since $\omega = \ell$, we get:

$$R_{\pi^s_{\text{div}}} = R_{\pi^s_{\text{BL}}} + R_{\pi^s_{\text{poly}}} + R_{\pi^s_{\text{trunc}}} + \mathcal{O}(\log \ell) = \mathcal{O}(\log^2 \ell) + \mathcal{O}(\log \omega) + \mathcal{O}(\log \ell) = \mathcal{O}(\log^2 \ell)$$

$$C_{\pi^s_{\text{div}}} = C_{\pi^s_{\text{BL}}} + C_{\pi^s_{\text{poly}}} + C_{\pi^s_{\text{trunc}}} + \mathcal{O}\left((\log \ell)(\kappa + \text{loglog}\, \ell)\right) = \mathcal{O}\left((\log^2 \ell)(\kappa + \text{loglog}\, \ell)\right).$$

7 Variations and Extensions

The multiparty case. Though we have presented our protocols in the two-party setting, the ideas are also applicable to the multiparty case, based e.g. on the protocols of [CDN01]. Arithmetic operations on encrypted values are immediate, hence we must only consider π_{BL}, π_{trunc}, and the sublinear comparison $\pi_{>?}$.

For the constant-rounds protocol we may use the arithmetic-based comparison of [NO07] while π_{BL} is essentially the bit-decomposition of [RT10]. Thus, these immediately work in the multiparty setting. The π_{trunc} protocol in Figure 4 can be jointly played by the parties. Part A is played publicly and part B is played using the protocols of [CDN01]. First each party P_i $(1 \leq i \leq n)$ supplies an encryption of a random value $\left[r^{(i)}\right]$ as well as $\left[r_{\top}^{(i)}\right]$ with plaintext $\left\lfloor r^{(i)}/2^k \right\rfloor$. The parties then compute and decrypt $[z] \leftarrow [\hat{q}] + \sum_{i=1}^{n} \left[r^{(i)}\right]$ and in turn $[\hat{q}] \leftarrow \left\lfloor z/2^k \right\rfloor - \sum_{i=1}^{n} \left[r_{\top}^{(i)}\right]$. This is the right result plus an additive error originating from a carry in the addition of r. Since r is a sum itself, the possible error grows linearly in the number of parties. However, as in the main protocol (Figure 1) this may be corrected using a number of secure comparisons.

With the additional requirement of two named and mutually incorruptible parties, the sub-linear case follows analogously by the protocols of [Tof11]. Since π_{BL} is based on comparison and arithmetic, and π_{trunc} is the same as the constant-rounds case, a sub-linear multiparty protocol is possible too.

Unconditionally secure integer division. Unconditionally secure variations of our protocols are possible, based e.g. on Shamir's secret sharing scheme and the protocols of Ben-Or et al. [Sha79,BGW88]. The construction is straightforward as all sub-protocols are applicable in this setting as well.

Improving the complexity of the sub-linear protocol. Using the other comparison protocol given in [Tof11] we may obtain slightly better bounds on our division protocol, namely $\mathcal{O}(\log \ell)$ rounds and $\mathcal{O}\left((\log \ell)\sqrt{\ell}(\kappa + \log \ell)\right)$ communications.

References

[ACS02] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432. Springer, 2002.

[BB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In Piotr Rudnicki, editor, *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 201–209, New York, 1989. ACM Press.

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and

Philippe Golle, editors, *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2009.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM Press, 1988.

[Bou00] F. Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807, pages 431–444, 2000.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th Annual ACM Symposium on Theory of Computing*, pages 11–19. ACM Press, 1988.

[CD09] Octavian Catrina and Claudiu Dragulin. Multiparty computation of fixed-point multiplication and reciprocal. *Database and Expert Systems Applications, International Workshop on*, 0:107–111, 2009.

[CDN01] R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045, pages 280–300, 2001.

[CK08] Octavian Catrina and Florian Kerschbaum. Fostering the uptake of secure multiparty computation in e-commerce. In *ARES*, pages 693–700. IEEE Computer Society, 2008.

[DFK+06] Ivan Damgaard, Matthias Fitzi, Eike Kiltz, Jesper Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer Berlin / Heidelberg, 2006.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2001.

[FS86] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In A. Odlyzko, editor, *Advances in Cryptology - Crypto '86*, pages 186–194, Berlin, 1986. Springer-Verlag. Lecture Notes in Computer Science Volume 263.

[FSW02] Pierre-Alain Fouque, Jacques Stern, and Jan-Geert Wackers. Cryptocomputing with rationals. In Matt Blaze, editor, *Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 136–146. Springer, 2002.

[GMS10] Jorge Guajardo, Bart Mennink, and Berry Schoenmakers. Modulo reduction for paillier encryptions and application to secure statistical analysis. In Radu Sion, editor, *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 375–382. Springer, 2010.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM.

[HAB02] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695 – 716, 2002.

[HKS+10] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In *CCS '10: Proceedings of the 17th ACM conference on Computer and communications security*, pages 451–462, New York, NY, USA, 2010. ACM.

[JF05] T. Jakobsen and S. From. Secure multi-party computation on integers. Master's thesis, Aarhus University, 2005. Available at `http://users-cs.au.dk/tpj/uni/thesis/`.

[JW05] Geetha Jagannathan and Rebecca N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett, editors, *KDD*, pages 593–599. ACM, 2005.

[KLM05] Eike Kiltz, Gregor Leander, and John Malone-Lee. Secure computation of the mean and related statistics. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 283–302. Springer, 2005.

[Lip03] H. Lipmaa. On diophantine complexity and statistical zero-knowledge arguments. In Chi-Sung Laih, editor, *Advances in Cryptology – ASIACRYPT 2003*, volume 2894, pages 398–415, 2003.

[NO07] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Workshop on Theory and Practice in Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 343–360, 2007.

[NT07] Kurt Nielsen and Tomas Toft. Secure relative performance scheme. In Xiaotie Deng and Fan Chung Graham, editors, *WINE*, volume 4858 of *Lecture Notes in Computer Science*, pages 396–403. Springer, 2007.

[NX10] C. Ning and Q. Xu. Multiparty computation for modulo reduction without bit-decomposition and a generalization to bit-decomposition. In *ASIACRYPT 2010*, pages 483–500, 2010.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, 1999.

[RT10] Tord Reistad and Tomas Toft. Linear, constant-rounds bit-decomposition. In Donghoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology - ICISC 2009*, volume 5984 of *Lecture Notes in Computer Science*, pages 245–257. Springer Berlin / Heidelberg, 2010.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Tof09] Tomas Toft. Constant-rounds, almost-linear bit-decomposition of secret shared values. In Marc Fischlin, editor, *Topics in Cryptology - CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 357–371. Springer Berlin / Heidelberg, 2009.

[Tof11] T. Toft. Sub-linear, secure comparison with two non-colluding parties. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 174–191. Springer, 2011.

[Veu10] Thijs Veugen. Encrypted integer division. In *IEEE Workshop on Information Forensics and Security (WIFS'10)*, Seattle, 2010. IEEE, IEEE.

[Yao86] A. Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society Press, 1986.

A Computing the Bit-length without Bit-decomposition

In this section we present an alternative way of computing an encryption of the exact bit-length of $[d]$, which is denoted as $[\ell_d]$ in the text. We name this alternative protocol Bit-Length(); that is to say, we have

$$[\ell_d] \leftarrow \text{Bit-Length}([d])$$

The protocol performs this in a constant number of rounds using a linear number of secure multiplications. We stress that most of the variables mentioned in this section are encrypted or bit-wise encrypted, denoted $[\cdot]_B$, but for simplicity we often do not refer to this explicitly.

Initially, we generate a random, ℓ-bit, bitwise encrypted value, $[r]_B$. This can be done by having each party supply ℓ random bits and computing the XOR: $b_0 \oplus b_1 = b_0 + b_1 - 2b_0b_1$. This will be used as a random mask for $[d]$; to mask the final carry each party supplies an additional κ-bit, random value, $\left[r_\top^{(i)}\right]$ which is multiplied by 2^ℓ and added to the masked value.[5]

We then decrypt

$$[e] \leftarrow [d] + [r] + \sum_{i=1}^{n} \left([r_\top^{(i)}] \cdot 2^\ell\right)$$

and reduce e modulo 2^ℓ to get $\bar{e} = (e \bmod 2^\ell)$. Since both d and r are at most ℓ-bit long, then it is easy to see that the *integer* sum of d and r (denoted as $\tilde{e} = d + r$) has two possible values: $\bar{e}$ and $\bar{e} + 2^\ell$; by comparing $[r]_B$ and $\bar{e}$ (which is public), we can decide which case it is and then select the correct (bitwise encrypted) value for the integer sum $\tilde{e} = d + r$. Note that the bit-length of $\tilde{e}$ may reach $\ell + 1$.

The addition process (over the integers) $\tilde{e} = d + r$ can be seen in Figure 7.

For this addition process, we will use c_i to denote the carry-bit generated in bit-position i, i.e. $c_i = 1$ means the $(i+1)$'th bit-position will get a "1" from the i'th bit-position. After computing $[\tilde{e}]_B$, we perform $[e']_B \leftarrow [\tilde{e}]_B \oplus [r]_B$ and set $[E]_B \leftarrow \pi^c_{\text{pre-}\vee}([e']_B)$.

Then, obviously, E must be of form

$$E = 000 \cdots 000111 \cdots 111$$

We use *cur* (which is a shorthand for "*current*") to denote the *index* of the left-most "1" in E. Obviously, we can get the encryption of *cur* easily from $[E]_B$ (by adding up the bits of $[E]_B$). Additionally, we use *aim* to denote the *index* of the left-most "1" in d. Obviously, *aim* is essentially the desired index, as $\ell_d = aim + 1$. Moreover, we have $cur \geq aim$.

[5] In this section, as in the text, we assume that d is of bounded bit-length ℓ. In fact, for d of unbounded size (or we can say "when d can be any value in $\mathbb{Z}_M$"), the discussions in this section can still hold; all that we need are some minor modifications.

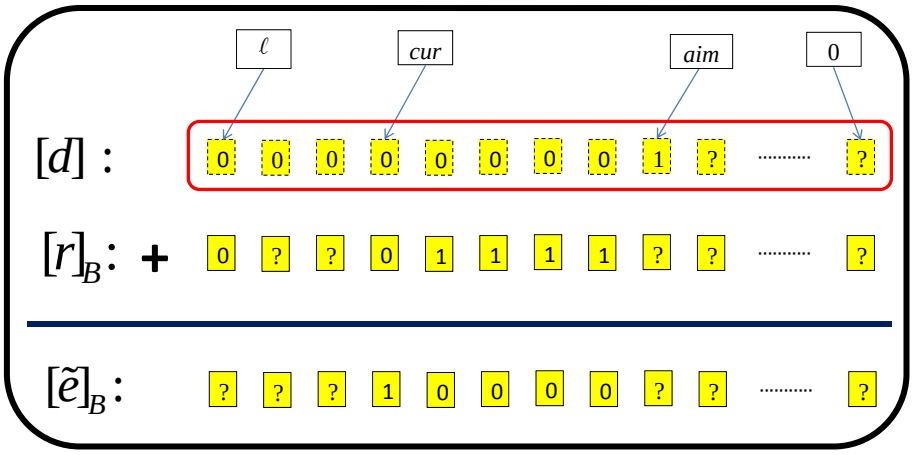

Fig. 7. The Addition Process

Below we define some necessary notations.

– Denote by $a_{i \leftrightarrow j}$ for $i > j$ the sub-string of bit-vector a from i to j, i.e.

$$a_{i \leftrightarrow j} = (a_i, a_{i-1}, ..., a_{j+1}, a_j)\,.$$

– For any bitwise encrypted variable $[x]_B$, we defined an operation "shift". Specifically, when we say "right-shift $[x]_B$ 1 bit", we mean "moving every bit of $[x]_B$ to the right by 1 bit-position; dropping the right-most bit; setting the left-most bit to be 0". Although all the bits of x are encrypted, this "shift" operation can be carried out. We use $[\overrightarrow{x}^{1}]_B$ to denote the obtained value by "right-shifting $[x]_B$ 1 bit". Other "shift" operations can be denoted similarly; for example, we can use $[\overrightarrow{x}^{3}]_B$ and $[\overleftarrow{x}^{1}]_B$ to denote the obtained values by "right-shifting $[x]_B$ 3 bits" and "left-shifting $[x]_B$ 1 bit", respectively.

The following computation then determines ℓ_d $(= aim + 1)$.

I. First decide whether $cur = aim$, i.e. determine $[cur \stackrel{?}{=} aim]$. This is done based on the following observation:
   – If $cur = aim$, then $c_{cur-1} = c_{aim-1} = 0$. This follows from the fact that $cur = aim$ implies $c_{aim} = 0$, and then by the fact that $d_{aim} = 1$, we can see $r_{aim} = c_{aim-1} = 0$.
   – Otherwise (i.e. $cur > aim$), we have $c_{cur-1} = 1$. In this case, $c_{aim} = 1$, and therefore all the bits in $r_{cur-1 \leftrightarrow aim+1}$ are "1".

   That is to say, "$cur = aim$" $\Leftrightarrow$ "$c_{cur-1} = 0$". Hence, we need only compute $[c_{cur-1}]$, which can be done by comparing the bit-decomposed $r_{cur-1 \leftrightarrow 0}$ and

$\tilde{e}_{cur-1 \leftrightarrow 0}$. Note the important fact that both $r_{cur-1 \leftrightarrow 0}$ and $\tilde{e}_{cur-1 \leftrightarrow 0}$ (padded with "0"s on the left such that they are ℓ bits long) can be obtained easily using a componentwise logical AND (multiplication) with the vector $[E]_B$.

After determining $[cur \stackrel{?}{=} aim]$, we can compute ℓ_d in both the two cases ($cur = aim$ and $cur > aim$) and then select the correct one (using $[cur \stackrel{?}{=} aim]$).

II. Below we will discuss the two cases in detail.

(a) $cur = aim$

In this case, we have $c_{aim} = 0$.

This case is very simple. We need only to sum up all the bits of $[E]_B$ to get ℓ_d.

(b) $cur > aim$

In this case, we have $c_{aim} = 1$.

This case is somewhat involved. We need to compute the gap between cur and aim (i.e. the number of the bits between bit-positions cur and aim). We proceed as follows

i. First we denote by t an $(\ell + 1)$-bit long, bit-decomposed integer satisfying "$t_i = 1$" $\Leftrightarrow$ "$r_i = 1$ and $\tilde{e}_i = 0$" for $i \in 0, 1, \cdots, l$. This can be obatined by setting $t_i = r_i - r_i \cdot \tilde{e}_i$. It is easy to see that all the bits in $t_{cur-1 \leftrightarrow aim+1}$ are "1" and that the bits of $t_{\ell \leftrightarrow cur}$ are "0".

ii. For this integer t, we set all the bits in $t_{\ell \leftrightarrow cur}$ to "1" by adding the all-1 vector and entrywise subtracting $[\overrightarrow{E}]_B$. The resulting vector will be denoted t'.

iii. We compute $T = \text{Prefix-}\wedge(t')$ (using a protocol analogous to $\pi^c_{\text{pre-}\vee}$, [DFK+06]). We have the following two facts:

A. **All the bits in $T_{\ell \leftrightarrow aim+1}$ are "1".**

This is obvious because all the bits in $t'_{\ell \leftrightarrow aim+1}$ are "1".

B. **All the bits in $T_{aim-1 \leftrightarrow 0}$ are "0".**

This can be proven based on two cases:

- If $t'_{aim} = 0$, then this is obvious.
- If $t'_{aim} = 1$, then $t_{aim} = 1$ and this means $r_{aim} = 1$ and $\tilde{e}_{aim} = 0$. Recall that $d_{aim} = 1$, so we can see that $c_{aim-1} = 0$. Then we can say that "$r_{aim-1} = 1$" and "$\tilde{e}_{aim-1} = 0$" can *not* hold at the same time (because if this happens, we have $c_{aim-1} = 1$). So, we have $t'_{aim-1} = 0$ and the claim is proved.

iv. Then we need to decide the value of the bit T_{aim}. We use a *comparison* to do this. We denote the *index* of the *left-most* "0" of $[T]_B$ (or we can say "the *first* '0' of $[T]_B$") as $fir0$. Then obviously, $fir0$ can be aim (when $T_{aim} = 0$) or $aim - 1$ (when $T_{aim} = 1$). It is easy to see that, no matter what $fir0$ is, $r_{fir0 \leftrightarrow 0}$ and $\tilde{e}_{fir0 \leftrightarrow 0}$ can be obtained with the help of $[T]_B$. Below we discuss what will happen when T_{aim} is 0 or 1.

- When $T_{aim} = 0$

  In this case, we have $fir0 = aim$. Recall that we are discussing

the case where $cur > aim$ which means $c_{aim} = 1$. From $c_{aim} = 1$, we can get

$$\tilde{e}_{fir0 \leftrightarrow 0} = \tilde{e}_{aim \leftrightarrow 0} < r_{aim \leftrightarrow 0} = r_{fir0 \leftrightarrow 0}.$$

– When $T_{aim} = 1$
In this case, $fir0 = aim - 1$. Obviously, $T_{aim} = 1$ means $t'_{aim} = 1$ (because $T = \text{Prefix-And}(t')$). Recall that

$$t'_{aim} = 1 \Rightarrow r_{aim} = 1 \ \wedge \ \tilde{e}_{aim} = 0 \Rightarrow c_{aim-1} = 0.$$

So we can see that

$$\tilde{e}_{fir0 \leftrightarrow 0} = \tilde{e}_{aim-1 \leftrightarrow 0} \geq r_{aim-1 \leftrightarrow 0} = r_{fir0 \leftrightarrow 0}.$$

Concluding the above two cases, we can say

$$\text{“}T_{aim} = 0\text{”} \Leftrightarrow \text{“}\tilde{e}_{fir0 \leftrightarrow 0} < r_{fir0 \leftrightarrow 0}\text{”}$$

i.e. we have $T_{aim} = 1 - (\tilde{e}_{fir0 \leftrightarrow 0} \overset{?}{<} r_{fir0 \leftrightarrow 0})$. That is to say, to compute $[T_{aim}]$ we need only to determine vectors, $\tilde{e}_{fir0 \leftrightarrow 0}$ and $r_{fir0 \leftrightarrow 0}$, (using a logical AND with $[T]_B$) and compare these.

v. Finally, we have

$$[\ell_d] = (\ell + 1) - \left(\sum [T]_B - [T_{aim}]\right)$$

in which $(\ell + 1)$ is the total length of d and $\sum [T]_B$ represents the encryption of the sum of all the bits of T.

Below are some further discussions.

1. In the above discussion, we introduce a protocol for computing $[\ell_d]$ given encryption $[d]$; we believe it a basic protocol and it will be useful for constructing complex protocols. However, in the text of this paper, we do not need $[\ell_d]$ but only need $[2^{\ell_d}]$ (See the π^{c}_{BL} protocol in Section 5.1). If we first get $[\ell_d]$ and then compute $[2^{\ell_d}]$ by using a private exponentiation protocol, then it will be much too complex. To solve this problem, we can first get the (bitwise-encrypted) prefix-or of d (denoted as $[D]_B$ hereafter) and then we have $[2^{\ell_d}] = [D]_B + 1$. For getting $[D]_B$, we can modify the length protocol as follows:
   (a) In the case where $cur = aim$, obviously, we have $[D]_B = [E]_B$.
   (b) In the case where $cur > aim$, after getting $[T]_B$, we left shift it 1 bit to get $[\overleftarrow{T}^{1}]_B$, then we perform the following selection to get $[\bar{D}]_B$:

$$[\bar{D}]_B \longleftarrow [T_{aim}] \ ? \ [\overleftarrow{T}^{1}]_B \ : \ [T]_B$$

Then, by using the all-1 vector to entrywise subtracting $[\bar{D}]_B$, we can get $[D]_B$.

Then, similar to the original length protocol, after getting both of the two possible values of $[D]_B$ in the two case ($cur = aim$ and $cur > aim$), we can select the correct value for $[D]_B$ using $[cur \stackrel{?}{=} aim]$.

2. A surprising point is that, even in the case where $cur = aim$, by performing the steps designed for the $cur > aim$ case (i.e. all the steps in Part IIb), we can also get the correct result! That is to say, Part I and IIa are both unnecessary at all. Similar conclusion can also be arrived at when we are computing $[D]_B$ in Part 1; that is to say, Part 1a is also unnecessary. We can simply ignore whether $cur > aim$ holds and perform the computations designed for the case where it holds, and this will give us the correct result (both the length of d, ℓ_d, and the prefix-or of d, $[D]_B$). However, despite this, we still separate the two cases in the protocol for readability and clarity.

3. All the discussions in this section can be carried over to the linear secret sharing setting in which all the operations take place over a prime field, e.g. $\mathbb{Z}_p$. All we need are some minor modifications and, in this setting, we can get a constant-rounds, linear Bit-Length protocol with perfect security.