Linear, Constant-Rounds Bit-Decomposition

Tord Reistad[1] and Tomas Toft[2,⋆]

[1] NTNU
Trondheim, Norway
tordr@item.ntnu.no
[2] Dept. of CS Aarhus University
IT-parken, Aabogade 34
DK-8200 Aarhus N, Denmark
ttoft@cs.au.dk

Abstract. When performing secure multiparty computation, tasks may often be simple or difficult depending on the representation chosen. Hence, being able to switch representation efficiently may allow more efficient protocols.

We present a new protocol for bit-decomposition: converting a ring element $x \in \mathbb{Z}_M$ to its binary representation, $x_{(\log M)-1}, \ldots, x_0$. The protocol can be based on arbitrary secure arithmetic in $\mathbb{Z}_M$; this is achievable for Shamir shared values as well as (threshold) Paillier encrypted ones, implying solutions for both these popular MPC primitives. For additively homomorphic primitives (which is typical, and the case for both examples) the solution is constant-rounds and requires only $O(\log M)$ secure ring multiplications.

The solution is secure against active adversaries assuming the existence of additional primitives. These exist for both the Shamir sharing based approach as well as the Paillier based one.

Keywords: Constant-rounds secure multiparty computation, bit-decomposition.

1 Introduction

Since Yao introduced the concept of secure multiparty computation (MPC) [Yao82] – evaluate a function on distributed, private inputs without revealing additional information on those inputs – it has been rigorously studied. Different approaches have been suggested, including garbled circuits [Yao86], secret sharing based approaches [BGW88, CCD88], and techniques relying on homomorphic, public-key cryptography, e.g. [CDN01].

It has been demonstrated that any function can be evaluated by describing it as a Boolean circuit and providing the inputs in binary, e.g. stored as 0 and 1 in some field or ring over which the secure computation is performed. However, alternative representations may provide greater efficiency. If, for example, the

⋆ Work partially performed while at CWI Amsterdam and TU Eindhoven.

D. Lee and S. Hong (Eds.): ICISC 2009, LNCS 5984, pp. 245–257, 2010.

the function consists entirely of integer addition and multiplication, this can be simulated with arithmetic over $\mathbb{Z}_M$, where M is chosen larger than the maximal result possible. With primitives providing secure computation in $\mathbb{Z}_M$ – e.g. based on Shamir sharing if M is chosen prime [Sha79, BGW88] – this is much simpler than using ring arithmetic to emulate the Boolean gates needed to "simulate" the integer computation.

Unfortunately, other operations become difficult when integers are stored as ring elements, e.g. division, modulo reduction, and exponentiation. To get the best of both worlds, a way of changing representation is needed. To go from binary to ring element is easy: it is a linear combination in the ring. The other direction is more difficult, in particular when adding the requirement that the solution must be constant-rounds, i.e. that only a constant number of messages may be sent overall.

Related work. The problem of constant-rounds bit-decomposition was first solved by Damgård *et al.* in the context of secret shared inputs, [DFK+06]; this was later improved by a constant factor by Nishide and Ohta [NO07]. Both solutions require $O(\ell \log \ell)$ secure multiplications, where ℓ is the bit-length of the modulus defining the field. These solutions provided the same security guarantees as the primitives, i.e. to ensure active or adaptive security, it was sufficient to utilize secure arithmetic providing this.

Independently, Schoenmakers and Tuyls considered *practical* bit-decomposition of Paillier encrypted values, i.e. the cryptographic setting, where they obtained linear – *but non-constant-rounds* – solutions [ST06, Pai99]. (They also noted the applicability of [DFK+06] for the Paillier based setting.) The solution of [ST06] was also secure against active adversaries, however, this needed additional "proofs of correctness" added to the basic protocol – in difference to the above solutions, secure arithmetic was not sufficient by itself.

A constant-rounds, *almost* linear solution was proposed by Toft [Tof09]. The problem of bit-decomposition was first reduced to that of postfix comparison (PFC) (using $O(\ell)$ secure multiplications), which was then solved using $O(\ell \cdot \log^{*(c)} \ell)$ multiplications. Similarly to [DFK+06] and [NO07], security was inherited from the primitives implying security against both active and adaptive adversaries immediately, both based on secret sharing as well on (threshold) Paillier encryption.

Contribution. We present a novel, constant-rounds, linear solution to the PFC problem, and hence also to that of bit-decomposition. The solution is applicable for arbitrary secure arithmetic in $\mathbb{Z}_M$,[1] i.e. it is applicable for both secret sharing as well as Paillier based primitives. The protocol not only improves the theoretic complexity, it is also practical in the sense that the constants involved are similar to previous constant-rounds solutions.

[1] Our only restrictions are that M is odd and that it is possible to generate random, invertible elements efficiently. Further, the transformation to the PFC problem must also be constant-rounds. Focus will be on the cases where M is either prime (Shamir sharing) or an RSA modulus (Paillier encryption) were all requirements are satisfied.

In difference to [DFK+06, NO07, Tof09], perfect security cannot be provided, even if this is guaranteed by the primitives. Security is at most statistical, which still allows it to be unconditional. This further implies that we require M to be sufficiently large – $M > 2^{2(\kappa+\log n)}$ where κ is the security parameter and n the number of parties.

Similarly to [ST06], active security is *not* directly obtained from active security of the primitives. As there, active security is achievable when the parties can demonstrate that a provided input is less than some public bound. For both the Shamir based setting and the Paillier based setting, a constant-complexity protocol exists implying actively secure, constant-rounds, $O(\ell)$ bit-decomposition protocols in these settings, where $\ell = \log M$.

An overview of this paper. Section 2 presents the model of secure computation used along with additional high-level constructs. Then in Sect. 3 the postfix comparison problem is introduced. The basic solution is presented in Sect. 4. Finally, the steps needed to achieve security against active adversaries are then discussed in Sect. 5, while Sect. 6 contains concluding remarks.

2 Secure Arithmetic – Notation and Primitives

We present our result based on abstract protocols. The model of secure computation is simply the arithmetic black-box (ABB) of Damgård and Nielsen [DN03]. It is described as an ideal functionality in the UC framework of Canetti, [Can00], and the present work can be used together with any realizing protocols. Naturally, the Paillier based protocols of [DN03] realize this functionality, but it can equally well be realized with perfect, active, and adaptive security with Shamir sharing over primes field $\mathbb{F}_M$ [Sha79] and the protocols of Ben-Or *et al.* [BGW88].

2.1 The Arithmetic Black-Box

The arithmetic black-box allows a number of parties to securely store and reveal secret values of a ring, $\mathbb{Z}_M$, as well as perform arithmetic operations. Borrowing notation from secret sharing, a value, v, stored within the functionality will be written in square brackets, $[v]$. The notation, $[v]_B$ will be used to refer to a bit-decomposed value, i.e. it is shorthand for $\left[v_{\hat{\ell}-1}\right], \ldots, [v_0]$ of some bit-length $\hat{\ell}$. The ABB provides the following operations; we assume that it is realized using additively homomorphic primitives.

- **Input:** Party P may input a value $v \in \mathbb{Z}_M$. Depending on the primitives, this can mean secret share, encrypt and broadcast, etc.
- **Output:** The parties may output a stored $[v]$; following this, all parties know the now public v. This refers to reconstruction, decryption, etc.
- **Linear combination:** The parties may decide to compute a linear combination of stored values, $[\sum_i \alpha_i v_i] \leftarrow \sum_i \alpha_i [v_i]$. This follows immediately from the homomorphic property assumed above.
- **Multiplication:** The parties may compute products, $[v \cdot u] \leftarrow [v] \cdot [u]$ – this requires interaction, at least for the examples considered.

Note that secure computation is written using infix notation. Moreover, linear combinations and multiplications may be written together in larger expressions. Though further from the primitives, it improves readability as it emphasizes the intuition behind the secure computations performed.

Regarding complexity, we will only consider rounds and communication size. Note that this implies that linear combinations are considered costless. For rounds, it is assumed that the other primitives all require $O(1)$ rounds, and that an arbitrary amount may be performed in parallel. Note that with abstract primitives, we can only count the number of sequential executions, not the actual number of rounds of a concrete realization. Under big-O, the two are equivalent, though.

For communication complexity, the number of invocations of the primitives are simply counted. Moreover, rather than counting them individually, similarly to previous work they will simply be referred to collectively as *secure multiplications*. (An input from every party will be considered equivalent to a single multiplication – multiplication protocols typically require each party to provide at least one input).

We stress that security is only shown in the ABB model of computation, i.e. we are only concerned with primitives that securely realizes this ideal functionality. This reduces security to a question of ensuring that inputs are proper, e.g. taken from some subset of $\mathbb{Z}_M$, as well as ensuring that no information leaks when values are output.

2.2 Complex Primitives

The protocols proposed will not be presented directly in the ABB model. A number of high-level primitives are simply sketched – these are obtained from previous work. We assume five rounds of preprocessing, where all random values needed are generated and prepared in parallel. Hence, round complexity of the primitives below are "online only." The number of multiplications is still the overall count.

Random, unknown elements. The parties can generate uniformly random, unknown elements. Each party inputs a random value, the sum of these is uniformly random and unknown assuming even a single honest party. This is considered equivalent to one multiplication. Random, invertible elements can be generated by verifying that the element indeed is invertible: generate two elements and publish the product. If this is invertible, then so was the first element, while the second acts as a one-time-pad masking it within the group. This consists of three multiplications.

Element inversion, constant-rounds multiplication, and prefix-products. Element inversion is possible using four secure multiplications with one round online: generate a random, multiplicative mask; mask and reveal the input; invert the (public) masked value; and securely unmask. This may be further used to obtain constant-rounds, unbounded fan-in multiplication of invertible elements in

one round using five multiplications per input. Both are due to Bar-Ilan and Beaver [BB89]. This may then be used to compute prefix-products: given an array of invertible values, $([v_0], \ldots, [v_{m-1}])$, compute $([p_0], \ldots, [p_{m-1}])$, where $[p_i] = \prod_{j=0}^{i} [v_j]$. Complexity is $5m$ multiplications in one round; see [DFK+06, Tof09] for details.

Random bit generation. We require a protocol for generating a uniformly random bit which is unknown to all parties. If M is prime, this is achievable with two secure multiplications, [DFK+06]. For non-prime M it can be achieved by letting each party share a uniformly random value in $\{1, -1\}$, computing the product of these, and mapping the result (which is still ± 1) to $\{0, 1\}$.

Note that when M is not prime, complexity is $O(n)$ multiplications, where n is the number of parties. For simplicity, we consider M prime in the analysis. In general, by assuming only a constant number of parties this factor disappears under big-O. This problem is not exclusive to our work; the factor n occurs in *all* comparable solutions known to the authors.

Comparison. A protocol for comparing bit-decomposed values is required, i.e. we allow expressions of the form

$$[a > b] \leftarrow [a]_B \overset{?}{>} [b]_B .$$

The comparison protocol of Reistad [Rei09] solves this problem using $7\hat{\ell} + 3$ multiplications in three rounds, where $\hat{\ell}$ is the bit-length. See Sections 4.1 and 4.3 for more information.

Random, bit-decomposed element generation. Another requirement is the ability to generate a uniformly random, unknown element along with its bit-decomposition. This can be achieved by generating ℓ random bits and viewing these as the binary representation. Computing the value itself is a simple linear combination, while a comparison is used to verify that the value is indeed less than M. Overall, this requires $6\ell+3$ multiplications; the comparison is slightly cheaper than above as one input is public. In the event of failure, the protocol must be rerun; as in previous work on bit-decomposition, we generate multiple candidates in parallel and assume a factor of four efficiency loss, i.e. $24\ell + 12$ multiplications. If *expected constant-rounds* suffices, then only a factor of two is required, i.e. $12\ell + 6$ multiplications. See [DFK+06] for justification.

This primitive is the most expensive with regard to preprocessing, i.e. this is where the five rounds originate. Note that the failure probability depends on M. If this can be chosen freely – e.g. as a Mersenne prime – then the issue can be eliminated and complexity essentially reduced to 2ℓ multiplications.

Least significant bit (LSB) gate. Finally, the ability to extract the least significant bit of an $\hat{\ell}$-bit value, $[x]$, of bounded size will be needed. [ST06] describes a way to do this for Paillier encrypted values when there is sufficient "headroom" in the ring, $2^{\hat{\ell}+\kappa+\log n} < M$, where κ is a security parameter and n is the number of parties. The result is not limited to the case of Paillier encryption, but can be utilized with arbitrary realizing protocols.

The idea is that the parties initially generate a random, unknown bit m_0, and that each party P_k inputs a uniformly random $(\kappa + \hat{\ell} - 1)$-bit value, $\left[m^{(k)}\right]$ from which a random mask is computed,

$$[m] \leftarrow 2(\sum_{k=1}^{n} m^{(k)}) + [m_0]\,.$$

Then $d = [x] + [m]$ is computed and output. By the assumption on the size of M, we have $d_0 = x_0 \oplus m_0$, where d_0 and x_0 are the least significant bits of d and x respectively. Thus, $[x_0]$ is easily obtained, $[x_0] \leftarrow d_0 + [m_0] - 2d_0\,[m_0]$. Overall, this is considered equivalent to three multiplications.

As the ABB is secure by definition, only one potential leak exists: d. However, m_0 is unknown and uniformly random, so for any honest party, P_k, $2 \cdot \left[m^{(k)}\right] + [m_0]$ statistically hides any information, and no adversary can learn anything, even when all but one parties are corrupt. Note that this is where perfect security is lost – the LSB gate is only statistically secure.

3 The Postfix Comparison Problem

The postfix comparison problem was introduced by Toft in [Tof09].

Problem 1 (Postfix Comparison [Tof09]). *Given two secret, $\hat{\ell}$-bit values $[a]_B = \left(\left[a_{\hat{\ell}-1}\right], \left[a_{\hat{\ell}-2}\right], \ldots, [a_0]\right)$ and $[b]_B = \left(\left[b_{\hat{\ell}-1}\right], \left[b_{\hat{\ell}-2}\right], \ldots, [b_0]\right)$, compute*

$$[c_i] = \left[a \bmod 2^i\right]_B \overset{?}{>} \left[b \bmod 2^i\right]_B$$

for all $i \in \{1, 2, \ldots, \hat{\ell}\}$.

¿From that paper, we get the following lemma, which states that a protocol for solving the problem of bit-decomposition can be obtained from any protocol solving the PFC problem.

Lemma 1 ([Tof09]). *Given a constant-rounds solution to Problem 1 using $O(f(\ell))$ secure multiplications, constant-rounds bit-decomposition is achievable using $O(\ell + f(\ell))$ secure multiplications.*

The proof is by construction, which we sketch, see [Tof09] for the full explanation.

To bit-decompose $[x]$, first compute $\left[x \bmod 2^i\right]$ for all i. Now, a bit, $[x_i]$, of $[x]$ can be computed using only arithmetic, $2^{-i}(\left[x \bmod 2^{i+1}\right] - \left[x \bmod 2^i\right])$. To reduce $[x]$ modulo all powers of 2, first add a random, bit-decomposed mask, $[r]_B$, *over the integers*:

$$[c]_B = [x] + [r]_B\,.$$

Then reduce both c and r modulo 2^i (easy, as they are already decomposed) and simulate computation modulo 2^i using secure $\mathbb{Z}_M$ arithmetic:

$$\left[x \bmod 2^i\right] \leftarrow \left[c \bmod 2^i\right] - \left[r \bmod 2^i\right] + 2^i \cdot \left(\left[r \bmod 2^i\right]_B \overset{?}{>} \left[c \bmod 2^i\right]_B\right).$$

The integer addition of $[x]$ and $[r]_B$ is achieved by computing and revealing $\tilde{c} = [x] + [r] \bmod M$ using ring arithmetic. This reveals no information, as $\tilde{c}$ is uniformly random. We now have $c \in \{\tilde{c}, \tilde{c} + M\}$; both values are known, so it is merely a matter of securely choosing the bits of the relevant candidate: compare $[r]_B$ and $\tilde{c}$ to determine if an overflow has occurred, and use the outcome to select the right candidate using arithmetic. The comparisons of $r \bmod 2^i$ and $c \bmod 2^i$ for all i is a postfix comparison problem. This transformation requires $24\ell + 12 + 6\ell + 3 = 30\ell + 15$ multiplications and two online rounds (the public $\tilde{c}$ improves efficiency slightly).

4 The New Constant-Rounds Solution

The proposed solution is based on (a variation of) a comparison protocol due to Reistad, [Rei09]. The parts relevant for this paper – including the minor alterations – are presented in Sect. 4.1. The improved postfix comparison protocol is then presented and analyzed in Sect. 4.2. The solution has a restriction, which must be eliminated in order to obtain the final bit-decomposition protocol, this is described in Sect. 4.3. For the purpose of this section, assume that the parties are honest-but-curious.

4.1 The Comparison of [Rei09]

Let $[r]_B$ and $[c]_B$ be two $\hat{\ell}$-bit, bit-decomposed numbers to be compared. Further, let κ be a security parameter, let n be the number of parties, and assume that $2^{\hat{\ell}+\kappa+\log n} < M$. The overall idea for computing $[r > c]$ is to first compute a value, $[e_i]$, for each bit-position, i. The expression is written with intuition in mind; details on how to perform the actual computation follow below.

$$[e_i] \leftarrow [r_i]\,(1 - [c_i])2^{\sum_{j=i+1}^{\hat{\ell}-1}[r_j] \oplus [c_j]}. \tag{1}$$

Note that $[e_i]$ is either 0 (when $[r_i]$ is not set, or when both $[r_i]$ and $[c_i]$ are set) or a distinct power of two strictly less than $2^{\hat{\ell}} - 2^{\hat{\ell}} < M$ by assumption so the computation can be viewed as occurring over the integers. Note also that $[e_i] = 1$ can only occur when i is the most significant differing bit-position. Thus, all values except at most one are even. And an odd value, 1, occurs only if r_i is set at the most significant differing bit-position, i.e. if $[r]_B$ is bigger than $[c]_B$.

Since at most one value is odd and this exists exactly when $[r]_B > [c]_B$, computing the least significant bit, $[E_0]$, of

$$[E] \leftarrow \sum_{i=0}^{\hat{\ell}-1} [e_i]$$

provides the desired result. This bit is determined with a LSB gate.

Security of this protocol is trivial: the arithmetic black-box can only leak information when something is deliberately output, and this only occurs in sub-protocols, which have already been considered.

For the computation of the $[e_i]$ above, $2^{\sum_{j=i+1}^{\hat{\ell}}[r_j]\oplus[c_j]}$ must be computed for every bit-position, i. This is done by first computing $[r_j \oplus c_j] \leftarrow [r_j] + [c_j] - 2\,[r_j]\,[c_j]$ for each bit-position, j. Rewriting the exponentiation of Eq. (1) as

$$\prod_{j=i+1}^{\hat{\ell}-1} (1 + [r_j \oplus c_j]),$$

illustrates not only how to compute it for a single bit-position, it also allows it to be computed efficiently for *every* such position: it is simply a prefix-product with terms $1 + [r_j \oplus c_j]$ and the most significant bit-position first. This is computable in $O(1)$ rounds since all terms are invertible – they are either 1 or 2, and M is odd.

Analyzing the protocol in more detail, it can be seen that $7\hat{\ell}+3$ multiplications are needed, and these can be performed in three rounds (plus preprocessing). First $[c_i r_i]$ is computed for every position. Then $5\hat{\ell}$ multiplications are needed for the prefix-product for the exponentiations, while an additional multiplication is needed for each of the $\hat{\ell}$ instantiations of Eq. (1). (Note that the multiplications from the $\oplus$ operations may be reused.) The LSB gate is then applied to conclude the computation.

4.2 Solving the PFCP with [Rei09]

Recall the PFC problem: we are given two $\hat{\ell}$-bit values, $[r]_B$ and $[c]_B$, and must compare all postfixes, i.e. all reduction modulo 2-powers. The above comparison cannot be applied naively at every bit-position, that would be too costly. Our goal is therefore to compute a value, $\left[E^{(k)}\right]$, for every bit-length, $k \in \{1, \ldots, \hat{\ell}\}$, equivalent to $[E]$ above. This suffices as the goal are the least significant bits of these values, and the LSB-gate requires only constant work.

Values similar to the $[e_i]$ above cannot be computed; there is a quadratic number of them. Instead the $[e_i]$ are computed as before. These are equivalent to the desired $\left[e_i^{(\hat{\ell}-1)}\right]$, and will be used in *all* the ensuing computation,

$$\left[\tilde{E}^{(k)}\right] \leftarrow \sum_{i=0}^{k-1} [e_i].$$

The computed values quite likely differ from the desired $\left[E^{(k)}\right]$. However, $[e_i]$ is only off from $\left[e_i^{(k)}\right]$ – which should have been used – by a factor of some two-power, $2^{\sum_{j=k}^{\hat{\ell}-1}[r_j]\oplus[c_j]}$. For any fixed k, this value is also fixed. Therefore $\left[\tilde{E}^{(k)}\right]$ is also simply "wrong" by a factor of this.

To "correct" $\left[\tilde{E}^{(k)}\right]$, first note that $\left[2^{\sum_{j=k}^{\hat{\ell}-1} r_j \oplus c_j}\right]$ has already been computed by the prefix-product. Further, the factor can be eliminated as it is invertible. I.e. the desired $\left[E^{(k)}\right]$ is securely computable.

$$\left[E^{(k)}\right] \leftarrow \left[\tilde{E}^{(k)}\right] \cdot \left[2^{\sum_{j=k}^{\hat{\ell}-1} r_j \oplus c_j}\right]^{-1}$$

At this point, invoking an LSB-gate on every $\left[E^{(k)}\right]$ provides the final result.

Correctness follows from the above discussion along with that of Sect. 4.1. Regarding security, the protocol clearly does not leak information. More values are output from the ABB, but this still occurs only in sub-protocols. Thus, no information is leaked.

We conclude with a complexity analysis of the protocol. Securely computing both the factors, $\left[2^{\sum_{j=k}^{\hat{\ell}-1}[r_j]\oplus[c_j]}\right]$, and the $[e_i]$ for all bit-positions, i, and bit-lengths, k, requires only $7\hat{\ell}$ secure multiplications in 3 rounds. All that was required was the computation of $[r_j \oplus c_j]$ for every bit-position, the prefix-product, and the concluding computation for each $[e_i]$.

Computing the $\left[\tilde{E}^{(k)}\right]$ is costless at this point, while correcting them – computing the $\left[E^{(k)}\right]$ – requires $\hat{\ell}$ additional multiplications. The element inversions are costless as they can reuse the masking from the prefix-product. Thus, only one multiplication is needed per bit-length, and these may all be processed in parallel. Similarly, the concluding LSB-gates are are equivalent to three multiplications each and may also be executed concurrently.

Combining all of the above, the requirement is $(7+1+3)\hat{\ell} = 11\hat{\ell}$ multiplications in four rounds plus preprocessing. Thus, the following theorem is obtained.

Theorem 1. *There exists a protocol which solves postfix comparison problems of size $\hat{\ell}$ in $O(1)$ rounds using $O(\hat{\ell})$ secure multiplications of elements of $\mathbb{Z}_M$, for $M > 2^{\hat{\ell}+\kappa+\log n}$.*

4.3 Performing Bit-Decomposition

It remains to apply Lemma 1 and Theorem 1 to obtain the main result of this paper. There is, however, still one problem to be solved. The PFC problem to solve is of size $\ell = \lceil \log M \rceil$, but to apply Theorem 1 we must have $M > 2^{\ell+\kappa+\log n}$; this is of course contradictory.

Assuming that $M > 2^{2(\kappa+\log n)}$, then the following variation of the above solution fixes the problem. The trick, taken from [Rei09], consists of considering pairs of bit-positions rather than single bit-positions when computing the $[e_i]$. This results in half as many $[e_i]$, thereby halving the bit-length needed. I.e. the resulting modified $\left[E^{(k)}\right]$ have at least $\kappa+\log n$ bits of headroom in $\mathbb{Z}_M$, allowing the LSB-gate to be applied. This was the sole reason for the restriction on M. For simplicity it is assumed that ℓ is even in the following, where Eq. (1) is replaced by Eq. (2) which is computed only for the *odd* bit-positions, i.

First values, $[u_i]$, are computed,

$$[u_i] \leftarrow [r_i] \wedge (\neg [c_i]) \vee (\neg([r_i] \oplus [c_i])) \wedge [r_{i-1}] \wedge (\neg [c_{i-1}]).$$

Note that this is simply a comparison circuit for 2-bit numbers. Though somewhat complex, the expression translates readily to arithmetic.

$$[r_i]\,(1-[c_i]) + (1 + 2\,[r_i]\cdot[c_i] - [r_i] - [c_i])\,[r_{i-1}]\,(1-[c_{i-1}])$$

The $[u_i]$ are then used in the computation replacing Eq. (1). $[e'_i]$ is set to a 2-power exactly when the 2-bit position of $[r]_B$ is greater than that position of $[c]_B$, and the powers are smaller, as only the number of differing 2-bit blocks are "counted."

$$[e'_i] \leftarrow [u_i] \cdot 2^{\sum_{j=((i-1)/2)+1}^{\ell/2-1}([r_{2j}]\oplus[c_{2j}])\vee([r_{2j+1}]\oplus[c_{2j+1}])} \tag{2}$$

Again, the expression is slightly more involved than before, but it can also be translated to a prefix-product,

$$\prod_{j=i+1}^{\hat{\ell}-1} (1 + ([r_{2j} \oplus c_{2j}] + [r_{2j+1} \oplus c_{2j+1}] - [r_{2j} \oplus c_{2j}] \cdot [r_{2j+1} \oplus c_{2j+1}]))\,,$$

where the $\oplus$ is computed as above. Overall, Eq. (2) requires only $10\ell/2$ multiplications in four rounds.

The smaller $\left[\tilde{E}^{(k)}\right]$ can now be computed, however, there are two cases, as values $[e'_i]$ are also needed for the even bit-positions.

$$[e'_i] \leftarrow [r_i]\,(1-[c_i]) \cdot 2^{\sum_{j=(i/2)+1}^{\ell/2-1}([r_{2j}]\oplus[c_{2j}])\vee([r_{2j+1}]\oplus[c_{2j+1}])}$$

At this point we may compute

$$\left[\tilde{E}^{(k)}\right] \leftarrow \begin{cases} \sum_{i=0}^{k/2-1} \left[e'_{2i+1}\right] & \text{when } k \text{ is even} \\ \left[e'_{k-1}\right] + \sum_{i=0}^{(k-1)/2-1} \left[e'_{2i+1}\right] & \text{when } k \text{ is odd} \end{cases}$$

after which the incorrect powers of 2 can be eliminated and the LSB-gates applied as above. This solves the PFC problem such that $[x]_B$ can be determined. The final complexity is $(10/2 + 1/2 + 1 + 3)\ell = 9.5\ell$ multiplications in 5 rounds plus preprocessing and the conversion to the PFC problem, i.e. $9.5\ell + 30\ell + 15 = 39.5\ell + 15$ secure multiplications in $5 + 2 + 5 = 12$ rounds overall (including preprocessing). The result is summarized in the following theorem.

Theorem 2. *There exists a protocol which bit-decomposes a secret value, $[x]$ of $\mathbb{Z}_M$, to $[x]_B$ using $O(\ell)$ secure multiplications in $O(1)$ rounds. The protocol is statistically secure against passive adversaries when the arithmetic primitives are this. When they only provide computational security, then so does the present protocol.*

5 Active Security

As noted in the introduction, active security is not immediate. Even when actively secure protocols are used for the computation, problems occur when the

parties are asked to share a random value from a domain different from $\mathbb{Z}_M$. For example, when M is not a prime, the parties must verify that inputs are really ± 1 during the bit-generation protocol. This is easily achieved with a zero-knowledge proof in the case of Paillier values, [DJ01]. The problem does not affect the solution based on Shamir sharing. There $\mathbb{Z}_M$ must be a field which implies that M is a prime. A second problem occurs in the LSB gates. It must be verified that the masks, $\left[m^{(k)}\right]$, are indeed of the specified bit-length. But these are the only problems.

By the definition of the arithmetic black-box, no adversary can do other harm. Thus, given a *constant-work* means of proving that a value is of bounded size (an interval proof), the solution can be made secure against active adversaries. There exists such proofs for both our examples.

As noted in [ST06], it is possible to demonstrate that a Paillier encryption contains a value within a specified range using the results of [Bou00, Lip03, DJ02]. The solution reveals no other information than the fact that indeed the value was from the desired range. Hence, active security is quite easily obtained in a Paillier based setting.

Regarding protocols based on Shamir sharing, such a proof is not immediate. It is, however, possible to obtain efficient range proofs by taking a detour through linear integer secret sharing (LISS). The solution follows directly from LISS, hence we only sketch it; see [Tho09] for a full explanation.

First off, LISS not only provides secret sharing of integer values, it can also form the basis for unconditionally and actively secure MPC. Further, it is possible convert a linear integer secret sharing to a Shamir sharing over $\mathbb{Z}_M$ simply by reducing the individual share modulo M. The solution is therefore to first share the $m^{(k)}$ using LISS, and for each of them demonstrate that it is in the desired range using constant work, [Tho09]. Secondly, those secret sharings are then converted to Shamir sharings over $\mathbb{Z}_M$. This ensures that the Shamir shared value is of bounded size as required.

6 Conclusion

We have proposed a novel protocol for constant-rounds bit-decomposition based on secure arithmetic with improved theoretic complexity compared to previous solutions. The complexity reached – $O(\ell)$ – appears optimal, as this is also the number of outputs, however, that this is the case is not immediately clear. Proving that $\Omega(\ell)$ secure multiplications is indeed a lower bound is left as an open problem.

Unfortunately, the present solution also has some minor "defects" compared to the previous ones. Firstly, "only" statistical security is guaranteed (at most), rather than perfect. This still allows linear, constant-rounds, unconditionally secure bit-decomposition, though. That security cannot be perfect also implies that the underlying ring must be sufficiently large to accommodate the large, random elements needed for statistical security. I.e. the protocol is only applicable for large moduli.

A second, worse "defect" is that – similarly to [ST06] – the basic solution does not provide out-of-the-box active security. This must be obtained through additional protocols, which of course increases complexity of the operations where these are needed. However as demonstrated, efficient, active security can be achieved quite readily for both Paillier based and Shamir sharing based settings.

We conclude by comparing the explicit complexity of our solution to that of previous ones, Table 1 taken from [Tof09]. Counting the exact number of secure multiplications provides a direct comparison for the case of passive security. It is noted that the proposed protocol not only improves theoretic complexity, it is also highly competitive with regard to the constants involved. In particular, if M can be chosen as a Mersenne prime, the overall number of secure multiplications can be reduced to essentially 17.5ℓ.

Table 1. Complexity of constant-rounds bit-decomposition

	Rounds	**Multiplications**
[DFK$^+$06]	38	$94\ell \log \ell + 63\ell + 30\sqrt{\ell}$
[NO07]	25	$47\ell \log \ell + 63\ell + 30\sqrt{\ell}$
[Tof09]	$23 + c$	$(31 + 26c)\ell \cdot \log^{*(c)}\ell + 71\ell + 14c\sqrt{\ell}\log^{*(c)}(\ell) + 30\sqrt{\ell}$
This paper	12	$39.5\ell + 15$

With regard to active security, the comparison is not fair. The present solution must of course also take into account the proofs that the masks shared by parties are well-formed. Their complexity depends on the realizing primitives, which are outside the arithmetic black-box; they are therefore not included in the overview. Complexity is reasonable, though. The main trick is that any positive integer can be written as the sum of four squares, thus only eight multiplications (over the integer scheme) are needed to show that an input is both upper and lower bounded. In addition to this, there is the cost of transferring the input to the scheme of the ABB.

Acknowledgements. The authors would like to thank Ivan Damgård for comments and suggestions.

References

[BB89] Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In: Rudnicki, P. (ed.) Proceedings of the eighth annual ACM Symposium on Principles of distributed computing, pp. 201–209. ACM Press, New York (1989)

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

[Bou00] Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg (2000)

[Can00] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000), http://eprint.iacr.org/

[CCD88] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: 20th Annual ACM Symposium on Theory of Computing, pp. 11–19. ACM Press, New York (1988)

[CDN01] Cramer, R., Damgård, I., Nielsen, J.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–300. Springer, Heidelberg (2001)

[DFK+06] Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (2006)

[DJ01] Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: Kim, K.-c. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001)

[DJ02] Damgård, I.B., Jurik, M.: Client/Server tradeoffs for online elections. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 125–140. Springer, Heidelberg (2002)

[DN03] Damgård, I., Nielsen, J.: Universally composable efficient multiparty computation from threshold homomorphic encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 247–264. Springer, Heidelberg (2003)

[Lip03] Lipmaa, H.: On diophantine complexity and statistical zero-knowledge arguments. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 398–415. Springer, Heidelberg (2003)

[NO07] Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 343–360. Springer, Heidelberg (2007)

[Pai99] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)

[Rei09] Reistad, T.: Multiparty comparison - an improved multiparty protocol for comparison of secret-shared values. In: Proceedings of SECRYPT 2009, pp. 325–330 (2009)

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

[ST06] Schoenmakers, B., Tuyls, P.: Efficient binary conversion for paillier encrypted values. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 522–537. Springer, Heidelberg (2006)

[Tho09] Thorbek, R.: Linear Integer Secret Sharing. PhD thesis, Aarhus University (2009)

[Tof09] Toft, T.: Constant-rounds, almost-linear bit-decomposition of secret shared values. In: Fischlin, M. (ed.) RSA Conference 2009. LNCS, vol. 5473, pp. 357–371. Springer, Heidelberg (2009)

[Yao82] Yao, A.: Protocols for secure computations (extended abstract). In: 23th Annual Symposium on Foundations of Computer Science (FOCS 1982), pp. 160–164. IEEE Computer Society Press, Los Alamitos (1982)

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