Constant-Rounds, Almost-Linear Bit-Decomposition of Secret Shared Values

Tomas Toft[1,2,⋆]

[1] CWI Amsterdam, The Netherlands
[2] TU Eindhoven, The Netherlands

Abstract. Bit-decomposition of secret shared values – securely computing sharings of the binary representation – is an important primitive in multi-party computation. The problem of performing this task in a constant number of rounds has only recently been solved.

This work presents a novel approach at constant-rounds bit-decomposition. The basic idea provides a solution matching the big-$\mathcal{O}$-bound of the original while decreasing the hidden constants. More importantly, further solutions *improve* asymptotic complexity with only a small increase in constants, reducing it from $\mathcal{O}(\ell \log(\ell))$ to $\mathcal{O}(\ell \log^*(\ell))$ and even lower. Like previous solutions, the present one is unconditionally secure against both active and adaptive adversaries.

Keywords: Secret Sharing, Constant-rounds Multi-party Computation, Bit-decomposition.

1 Introduction

Secure multi-party computation (MPC) allows a number of parties to perform a computation based on private inputs, learning the result but revealing no other information than what this implies. The general solutions providing unconditional security phrase the computation as a circuit over secret shared inputs, however, the gates must be evaluated in an iterative fashion, implying that round complexity is equivalent to circuit depth.[1]

A clever choice of representation of the inputs can have great influence on possible constant-rounds solutions. Consider determining the sum of a number of private integer values. Their binary representation could be taken as input, but constant-rounds bitwise addition is expensive. A much simpler solution would be to choose a prime, p, greater than the maximal sum, and view the inputs as elements of $\mathbb{Z}_p$ – the $\mathbb{Z}_p$ computation simulates the task to be performed. However, if we wish to determine some property of the sum, say a particular bit, then we are stuck. This would be trivial given the binary representation.

⋆ Supported by the research program Sentinels (`http://www.sentinels.nl`). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs.

[1] With a linear secret sharing scheme only multiplication gates are counted regarding depth.

M. Fischlin (Ed.): CT-RSA 2009, LNCS 5473, pp. 357–371, 2009.

Being able to switch representation gives us the best of both worlds. An efficient means of extracting all bits in a constant number of rounds is therefore an important primitive (the other direction is trivial). Though this problem has already been solved, it is worth improving the solutions, as more efficient primitives implies better complexity for any application.

1.1 Setting and Goal

Formalizing the above, our setting is the following: A value x is secret shared among n parties $P_1, \ldots, P_n$ using an unconditionally secure, linear secret sharing scheme over the prime-field $\mathbb{Z}_p$. In addition to this, the parties have access to a secure, constant-rounds multiplication protocol for shared values, i.e. given sharings of a and b, the parties can obtain a sharing of the product. All primitives are assumed to allow concurrent execution. The above can be instantiated with Shamir's secret sharing scheme [Sha79] and the protocols of Ben-Or et al. [BGW88].

The parties wish to securely compute sharings of the binary representation of x in a constant number of rounds of interaction. I.e. they should obtain sharings of $x_{\ell-1}, \ldots, x_0 \in \{0, 1\} \subseteq \mathbb{Z}_p$ – where ℓ is the bit-length of p – such that

$$x = \sum_{i=0}^{\ell-1} 2^i x_i$$

when viewed as occuring over the integers.

1.2 Related Work

Bit-decomposition has been considered in various settings. Algesheimer et al. presented a solution starting with an additive secret sharing, [ACS02]. This was not constant-rounds though, and only secure against semi-honest adversaries.

The first constant-rounds solution to the bit-decomposition problem is due to Damgård et al., [DFK+06]. Similar to here, unconditional security against active and adaptive adversaries was considered. The computation requires $\mathcal{O}(\ell \log(\ell))$ invocations of the multiplication protocol.

The [DFK+06] computation was later improved by a constant factor by Nishide and Ohta, [NO07] – the basic idea was the same, but it was observed that one of the two invocations of the most expensive primitive, bitwise addition, was unnecessary.

Toft later sketched a theoretically improved solution, [Tof07] also building on [DFK+06]. Complexity was reduced to $\mathcal{O}(\ell \log^{(c)}(\ell))$ for any constant integer, c, where $\log^{(c)}$ is the logarithm applied c times. The round complexity of that solution was $\mathcal{O}(c)$ and required multiple sequential applications of the bitwise addition protocol.

Concurrently and independently of Damgård et al., Schoenmakers and Tuyls considered bit-decomposition of Paillier encrypted values, i.e. the cryptographic setting, [ST06]. Though their main focus was not on constant round complexity, they noted that the techniques of [DFK+06] could be applied in that setting too.

1.3 Contribution

This work presents a practical, unconditionally secure,[2] constant-rounds bit-decomposition protocol with improved complexity compared to previous solutions. Security guarantees are equivalent to that of the primitives, i.e. when these are secure against active and adaptive adversaries, then so are the proposed protocols.

Complexity is $\mathcal{O}(\ell \log^{(*^{(c)})}(\ell))$ invocations of the multiplication protocol for any constant integer c, where $\log^{(*^{(1)})} = \log^*$, the iterated logarithm, while for $c > 1$, $\log^{(*^{(c)})}$ is the iteration of $\log^{(*^{(c-1)})}$.

$$\log^{(*^{(c)})}(\ell) = \begin{cases} 0 & \ell \leq 1 \\ 1 + \log^{(*^{(c)})}(\log^{(*^{(c-1)})}(\ell)) & \ell > 1 \end{cases}$$

Round complexity is $\mathcal{O}(c)$ implying a tradeoff between communication complexity and the number of rounds: decreasing big-$\mathcal{O}$-complexity comes at the cost of increased constants. However, even small c imply an essentially linear solution – $\log^*$ is in practice at most 5. Thus, the constants remain competitive for a solution which is "linear in practice." A comparison of this and previous work on constant-rounds bit-decomposition is seen in Table 1.

Table 1. Complexities of constant-rounds bit-decomposition protocols

	Rounds	**Multiplications**
[DFK+06]	$\mathcal{O}(1)$	$\mathcal{O}(\ell \log(\ell))$
[NO07]	$\mathcal{O}(1)$	$\mathcal{O}(\ell \log(\ell))$
[Tof07]	$\mathcal{O}(c)$	$\mathcal{O}(\ell \log^{(c)}(\ell) + c\ell)$
This paper	$\mathcal{O}(c)$	$\mathcal{O}(c\ell \log^{(*^{(c)})}(\ell))$

It is noted that the techniques presented here are also applicable in other settings, e.g. to MPC protocols based on Paillier encryption. The ideas may also be used to construct novel, non-constant-rounds variations, which may be preferable in practice.

1.4 An Overview of This Paper

Section 2 introduces preliminaries, including known sub-protocols. Section 3 presents the overall intuition and translation of the problem to post-fix comparison, which is introduced fully in Sect. 4. The following two sections both provide solutions to post-fix comparison, with Sect. 6 building on Sect. 5. Finally, Sect. 7 provides analysis and concluding remarks.

[2] Technically, security is perfect; there is, however, a negligible probability of "faults." The effect can be chosen, e.g. only expected constant-rounds, only unconditional security, imperfect correctness.

2 Preliminaries

This section elaborates on the setting introduced in Sect. 1.1. Moreover, additional primitives needed below are also listed.

2.1 Setting and Complexity

The basic setting of a linear secret sharing scheme with a multiplication protocol can be modelled as an incorruptible third party. This party allows the parties of the protocol to input values (share), perform arithmetic (when sufficiently many agree), and to output values (reconstruct). More formally, this trusted party is instantiated as an ideal functionality, e.g. similar to the arithmetic black-box of Damgård and Nielsen [DN03]. The present protocols then apply to any schemes shown equivalent to this ideal functionality. This also implies that in order to show security, it must simply be argued that any values revealed, do not reveal any information; sub-protocols are secure by assumption.

Similar to other work, interaction is considered the most important resource. As the primitives considered are abstract, the measures are phrased as invocations of the sub-protocols. For simplicity, as the multiplication protocol is both the most expensive and the most used primitive, only its invocations are counted. Complexity is therefore measured in *secure multiplications*. Note that secure computation of linear combinations follows directly from the linearity of the underlying scheme and is therefore considered costless.

Complexity is divided into two measures: Rounds (the overall number of times that messages are exchanged between all parties) and the communication size (the overall size of all messages). The former is represented by the number of sequential secure multiplications, while the latter consists of the overall number. Note that round complexity counts sequential multiplications, and *not* the actual number of messages. However, with abstract primitives it is not possible to be more precise. Moreover, with a constant-rounds multiplication protocol, the rounds-measures are equivalent under big-$\mathcal{O}$.

2.2 Basic Notation

Though protocols are considered, the notation used will be algorithmic in nature. $[a]$ will denote a sharing of a, with $a \leftarrow \texttt{output}([a])$ indicating its reconstruction. Computation will be written in an infix notation, which eases readability. For example,

$$[y] \leftarrow c_1 \cdot [x_1] \cdot [x_2] + c_2 \cdot [x_3] \cdot [x_4]$$

denotes two invocations of the multiplication protocol followed by a linear computation. As the protocol invocations may be performed in parallel, this requires only a single round.

Certain intermediate values of the protocol will be bit-decomposed. To distinguish these, they will be written $[x]_B$. The notation is simply shorthand for a list bits, $[x_i]$. Any arithmetic operation taking such an input implicitly converts it to a field element. This is a linear computation and therefore costless.

Abusing notation, we sometimes write multiple shared values – such as a list of shared values – with the same notation as a single sharing, $[x]$. It will be clear from context when this occurs.

2.3 Known Primitives

Regarding the primitives needed, all are described in detail in both [DFK+06] and [NO07]. The most basic is that of random bit generation – generating a random sharing of a uniformly random bit unknown to all parties. This is considered equivalent to two sequential multiplications.

The second primitive consists of comparison of secret shared values. An execution results in a sharing of $[b] \in \{0, 1\}$ which is 1 exactly when the first input is greater than the second. This work only needs a protocol for the case when the inputs are already bit-decomposed. Similar to arithmetic, this is also written with an infix notation, $\overset{?}{\geq}$. For efficiency, the present work considers a variation of the one used in [DFK+06] and [NO07]; the basic idea is the same, hence the difference is only sketched.

Given two ℓ-bit inputs, $[a]_B$ and $[b]_B$, their bitwise XOR is computed in the standard way, $[a_i \oplus b_i] = [a_i] + [b_i] - 2[a_i b_i]$. Then, rather than performing a full prefix-OR on this, we stop after having determined and run brute-force prefix-OR on the first interesting block of size $\sqrt{\ell}$ (i.e. the first block containing a one). The associated block of $[a]_B$ is determined (in parallel with that of the XOR-list) and the answer to $[a]_B > [b]_B$ may be extracted as the dot product between these two list of length $\sqrt{\ell}$ – this is the bit of $[a]_B$ at the most significant differing bit-position, which is also the result.

Overall this requires six rounds (plus two rounds of preprocessing) in which the multiplication protocol is invoked $\ell + 5\ell + (5/2)(\ell + \sqrt{\ell}) + 2\ell + (5/2)(\ell + \sqrt{\ell}) + \sqrt{\ell} = 13\ell + 6\sqrt{\ell}$ times.[3]

In this work, we also need a comparison which does not only provide single bit output, but considers three cases – smaller, equal, or larger – encoded in two bits:

$$\mathrm{comp}\left([a]_B, [b]_B\right) = \begin{cases} \binom{[1]}{[0]} & \text{if } [a]_B > [b]_B \\ \binom{[0]}{[0]} & \text{if } [a]_B = [b]_B \\ \binom{[0]}{[1]} & \text{if } [a]_B < [b]_B \end{cases}$$

This is essentially just a comparison that goes both ways – comparing both $[a]_B$ to $[b]_B$ and $[b]_B$ to $[a]_B$. This adds only $\ell + \sqrt{\ell}$ extra multiplications and require no additional rounds. The bulk of the work consists of determining the most significant bit-position where the inputs differ.

The final primitive is the generation of uniformly random, bitwise shared values of $\mathbb{Z}_p$, written $[r]_B \in_R \mathbb{Z}_p$. For ℓ-bit p, this is done by securely generating ℓ random bits, $[x]_B = ([x_{\ell-1}], \ldots, [x_0])$, and verifying that $[x]_B < p$. This may of

[3] In difference to [DFK+06] and [NO07], it is noted that the quadratic, brute force prefix-OR requires $5/2(k + \sqrt{k})$ multiplications on lists of size $\sqrt{k}$ rather than $5k$.

course fail; as in previous work, it is assumed that four attempts (run in parallel) are needed. This implies a complexity of seven rounds and $4(2\ell+(11\ell+6\sqrt{\ell}))=52\ell+24\sqrt{\ell}$ multiplications; note the slightly reduced complexity as p is public.

3 Overall Intuition – Simplifying the Problem

The overall solution transforms the problem of bit-decomposition to one of postfix comparison described fully in Sect. 4 below. For clarity, we first introduce the overall idea through a naïve solution. Complexity is worse than previous ones, but the solution – seen as Protocol 1 – provides intuition needed for the following sections.

Protocol 1. The overall bit-decomposition

Input: $[x]$.
Output: The bit-decomposition, $[x]_B=([x_{\ell-1}],[x_{\ell-2}]\ldots,[x_0])$

$[r]_B \in_R \mathbb{Z}_p$
$c \leftarrow \texttt{output}([x]+[r]_B)$
$c' \leftarrow c+p$ ▷ Addition over the integers
$[f] \leftarrow [r]_B \overset{?}{>} c$
5: **for** $i=0,\ldots,\ell$ **do**
$[\tilde{c}_i] \leftarrow [f]\cdot(c'_i-c_i)+c_i$
end for
$[\tilde{c}]_B \leftarrow ([\tilde{c}_\ell],\ldots,[\tilde{c}_0])$
$\left[x \bmod 2^\ell\right] \leftarrow [x]$
10: **for** $i=\ell-1,\ldots,1$ **do**
$[u_i] \leftarrow \left[r \bmod 2^i\right]_B \overset{?}{>} \left(\left[\tilde{c} \bmod 2^i\right]_B\right)$
$\left[x \bmod 2^i\right] \leftarrow \left(\left[\tilde{c} \bmod 2^i\right]_B\right) - \left[r \bmod 2^i\right]_B + 2^i[u_i]$
$[x_i] \leftarrow 2^{-i}\left(\left[x \bmod 2^{i+1}\right] - \left[x \bmod 2^i\right]\right)$
end for
15: $[x_0] \leftarrow [x \bmod 2]$

Correctness: The protocol starts similarly to the one from [NO07], lines 1 through 8. $[x]$ is additively masked by a random $[r]_B$, and the binary representation of $\tilde{c}=x+r$ (over the integers) is computed securely. It is clear that $\tilde{c}\in\{c,c+p\}$, thus, if it can be determined which is the case, then for every bit-position, the relevant bit can be selected. This selection occurs in line 6, where $[f]$ is used to obliviously choose between the two bits. $[f]$ is 1 when $[r]_B>c$, i.e. exactly when an overflow has occurred in the computation of c. This is the case where $c+p$ should be chosen.

The remaining computation – lines 9 to 15 – extracts the actual bits. The main idea is to reduce $[x]$ securely modulo all powers of 2 and compute differences between neighbors. Clearly

$$(x \bmod 2^{i+1}) - (x \bmod 2^i) = 2^i x_i$$

which is easily mapped to a binary value.

p is ℓ-bit, so $\left[x \bmod 2^{\ell}\right]$ is computed correctly. Moreover, as we have $\tilde{c} - r = x$ over the integers, reducing both $[\tilde{c}]_B$ and $[r]_B$ modulo 2^i and subtracting provides the right result, at least if the computation had been modulo 2^i. Since it occurs modulo p, an underflow occurs when $\left[r \bmod 2^i\right]_B > \left[\tilde{c} \bmod 2^i\right]_B$. Line 12 compensates by adding 2^i in this case. Thus, the $[x_i]$ are correct. The needed reductions of $[\tilde{c}]_B$ and $[r]_B$ modulo powers of two are easily done. Both are bit-decomposed, and it is therefore merely a question of ignoring the top bits.

Complexity: Until the start of the first loop, a random element is generated, added to $[x]$, and then compared to the (public) outcome. Following this are the costless linear combinations of the loop as well as a bit of renaming, lines 8 and 9. All of this is linear and requires only a constant number of rounds.

In the actual bit extraction, $\ell - 1$ comparisons of length $1, \ldots, \ell - 1$ must be performed. This implies a quadratic number of multiplications, however, note that though the bits are extracted in a linear manner, this does not imply a linear number of rounds. Line 11 does not depend on previous iterations of the loop, and all other computation is costless.

Thus, overall complexity is $\mathcal{O}(\ell^2)$ multiplications and $\mathcal{O}(1)$ rounds. However, we have an immediate improvement, if the comparisons in the final loop can be performed more efficiently.

Security: Security follows directly from the security of the sub-protocols and the masking of $[x]$. $[r]$ is chosen uniformly at random from $\mathbb{Z}_p$, thus c is also uniformly random over $\mathbb{Z}_p$ and therefore reveals nothing.

4 The Post-fix Comparison Problem

Section 3 transformed the problem of bit-decomposition to that of performing $\ell - 1$ comparisons (line 11). In general this is quadratic, however, here the numbers are highly interdependent: they are reductions modulo all powers of two of the same initial values, $[\tilde{c}]_B$ and $[r]_B$, which leads to the formulation of the post-fix comparison problem.

Problem 1 (Post-fix Comparison). *Given two bitwise shared $\hat{\ell}$-bit values $[a]_B = \sum_{i=0}^{\hat{\ell}-1} 2^i [a_i]$ and $[b]_B = \sum_{i=0}^{\hat{\ell}-1} 2^i [b_i]$, compute*

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

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

Recall from Sect. 3 that in addition to the post-fix comparison, Protocol 1 required only $\mathcal{O}(\ell)$ secure multiplications. Presenting a protocol which solves post-fix comparison more efficiently than $\mathcal{O}(\ell \log(\ell))$ ($\mathcal{O}(\ell \log^{(c)}(\ell))$) therefore implies an improvement over [DFK+06, NO07] ([Tof07]). The naïve solution above is of course much worse.

5 Solving Post-fix Comparison, $\mathcal{O}(\ell \log(\ell))$ Work

The $\mathcal{O}(\ell \log(\ell))$ bound is not beaten in one go. Rather, we start with a solution equalling it, which is used as a sub-protocol below. This not only introduces needed primitives, but also provides a gradual presentation of the ideas.

For this section, assume that we are given a post-fix comparison problem of size ℓ.[4] (Assume for simplicity that ℓ is a power of 2, this can always be achieved by padding with zeros.) Rather than considering the input as two separate numbers to be compared, we may write it as a string of bit-pairs,

$$\left(\binom{[a_{\ell-1}]}{[b_{\ell-1}]}, \ldots, \binom{[a_0]}{[b_0]}\right) \ .$$

Abusing notation, we write

$$\mathrm{comp}\left(\binom{[a_{i+j-1}]}{[b_{i+j-1}]}, \ldots, \binom{[a_j]}{[b_j]}\right)$$

meaning the comparison of the numbers represented by the (sub-string) of bit-pairs.

The naïve solution to the post-fix comparison problem divided the inputs into all post-fixes, and processed these individually. Here, we split into blocks of length powers of two. The final result can then be constructed from these. Rather than simply solving Problem 1, we consider a slightly more difficult variation and require the output from the extended comparison, comp$(\cdot)$. This results in slightly more (shared) information which will be needed below.

The division into blocks is best viewed as a balanced, binary tree. Every node represents a comparison of some sub-block of bit-positions. The root node represents the full comparison, while the leaves represent (the comparison of) single bit positions. Internal nodes represent the comparison of either the first half (left child) or second half (right child) of its parent. I.e. if a node represents the comparison of $[a_{i_{2^k-1}}], \ldots, [a_{i_0}]$ and $[b_{i_{2^k-1}}], \ldots, [b_{i_0}]$, then its left (right) child represents $[a_{i_{2^k-1}}], \ldots, [a_{i_{2^{k-1}}}]$ and $[b_{i_{2^k-1}}], \ldots, [b_{i_{2^{k-1}}}]$ ($[a_{i_{2^{k-1}-1}}], \ldots, [a_{i_0}]$ and $[b_{i_{2^{k-1}-1}}], \ldots, [b_{i_0}]$). Such a tree is constructed by running Protocol 2 on the full input.

From this tree,

$$\binom{\left[c_i^{\top}\right]}{\left[c_i^{\perp}\right]} = \mathrm{comp}\left(\left[a \bmod 2^i\right], \left[b \bmod 2^i\right]\right)$$

can be computed using Protocol 3. Note that $||$ denotes the concatenation of lists; line 5 sets $[s]$ to an empty list, while line 8 appends the outcomes of the comp$(\cdot)$'s, i.e. pairs of bits. Executing the protocol on all i, $1 \leq i < \ell$, results in the desired output.

[4] Though $[\tilde{c}]_B$ is $(\ell+1)$-bit, the extra bit added by the conversion to the post-fix comparison problem can be ignored.

Protocol 2. The tree construction protocol, treecomp$(\cdot)$, for solving post-fix comparison using $\mathcal{O}(\ell \log(\ell))$ multiplications.

Input: Two $\hat{\ell}$-bit, bitwise secret shared numbers, $[a_{\hat{\ell}-1}], \ldots, [a_0]$ and $[b_{\hat{\ell}-1}], \ldots, [b_0]$.
Output: A binary tree of results of sub-comparisons; each node represents a block of bit-positions.

if $\hat{\ell} = 1$ **then**
  return $(\text{comp}([a_0], [b_0]), \perp, \perp)$
end if
$[d] \leftarrow \text{comp}(([a_{\hat{\ell}-1}], \ldots, [a_0]), ([b_{\hat{\ell}-1}], \ldots, [b_0]))$
$[c_{\text{left}}] \leftarrow \text{treecomp}(([a_{\hat{\ell}-1}], \ldots, [a_{\hat{\ell}/2}]), ([b_{\hat{\ell}-1}], \ldots, [b_{\hat{\ell}/2}]))$
$[c_{\text{right}}] \leftarrow \text{treecomp}(([a_{\hat{\ell}/2-1}], \ldots, [a_0]), ([b_{\hat{\ell}/2-1}], \ldots, [b_0]))$
return $([d], [c_{\text{left}}], [c_{\text{right}}])$

Protocol 3. Extracting one result for post-fix comparison from a treecomp$(\cdot)$-tree.

Input: A bit-position, $1 \leq i < \ell$ and the output of treecomp$(\cdot)$ on two ℓ-bit, bitwise secret shared numbers, $[a_{\ell-1}], \ldots, [a_0]$ and $[b_{\ell-1}], \ldots, [b_0]$.
Output: Output equivalent to that of $\text{comp}([a \bmod 2^i], [b \bmod 2^i])$.

if $i = \ell$ **then**
  return root.$[d]$
end if
node $\leftarrow$ leaf(i)
$[s] \leftarrow$ EmptyList
while node is not root **do**
  if node is left child **then**
    $[s] \leftarrow [s]||(\text{sibling(node)}.[d])$
  end if
  node $\leftarrow$ parent(node)
end while
return comp $([s])$

Correctness: The intuition behind this section is that the division into blocks creates halves, quarters, eighths, etc. Any postfix of bit-positions, $i-1, \ldots, 0$, can be "covered" by only a logarithmic number of disjoint blocks – at most one from each level of the tree is needed. Replacing the bits of a block with the output of comp$(\cdot)$ changes the numbers involved, but comparing these new numbers provides the same result. For example, for blocks $[a_\top]$ and $[a_\perp]$ ($[b_\top]$ and $[b_\perp]$) covering $[a]$ ($[b]$), we have

$$\text{comp}\left(\begin{pmatrix}[a_\top]\\[b_\top]\end{pmatrix}||\begin{pmatrix}[a_\perp]\\[b_\perp]\end{pmatrix}\right) = \text{comp}\left(\text{comp}\begin{pmatrix}[a_\top]\\[b_\top]\end{pmatrix}||\,\text{comp}\begin{pmatrix}[a_\perp]\\[b_\perp]\end{pmatrix}\right)$$

To see this, note that comparison simply extracts the most significant, differing bit-position. Replacing blocks with their output of comp essentially compresses

strings to bit-pairs containing the relevant information, i.e. which was bigger (or whether they were equal). Thus, replacing the bits of the blocks of the cover with the outcome of their comparisons, results in a much smaller comparison problem with the same result.

Protocol 2 constructs all the covers needed for all positions, while Protocol 3 selects the covers needed for the i'th position. These are exactly the siblings of the left children on the path to the root. Finally, the log-length comparisons provide the right result.

Complexity: It is easily verified that Protocol 2 requires $\mathcal{O}(\ell \log(\ell))$ multiplications. Each of the $\log(\ell)$ levels of the tree is a subdivision of the original problem into disjoint blocks, which are then compared, i.e. each level requires a linear number of secure multiplications, which matches the stated bound. Round complexity is $\mathcal{O}(1)$; the only secure computation is the invocations of $\text{comp}(\cdot)$. As these are all independent of each other, they may be performed in parallel.

Protocol 3 traverses a path from a leaf to the root and gathers data. This is purely bookkeeping, the only multi-party computation is the comparison at the very end. The path from leaf to root is of length $\log(\ell)$, thus the bit-length of the values of $[s]$ is at most $\log(\ell)$, implying $\mathcal{O}(\log(\ell))$ multiplications.

Applying Protocol 2 on the input and Protocol 3 on the tree ℓ times – once for every bit-position i – is overall $\mathcal{O}(\ell \log(\ell))$. Only $\mathcal{O}(1)$ rounds are needed as the bit-positions may be processed concurrently.

Security: Security is easily argued. Neither of Protocols 2 and 3 output any information. They are simply deterministic applications of the primitives, which are secure and composable by assumption.

6 Solving Post-fix Comparison, $\mathcal{O}(\ell \log^*(\ell))$ and Better

In order to use the present ideas, while also decreasing complexity, the height of the tree of sub-comparisons of Sect. 5 must be reduced. However, if fan-out is increased (the block of each node split into more than two sub-blocks), each of the results of the post-fix comparison, $[c_i]$, may depend on more than one block from each level. There is no longer a single node per level which handles all less significant bits of ancestors on the path to the root. The solution is to increase fan-out and then "combine" the blocks represented by siblings, such that each output again depends on only one node per level.

Analogously to Sect. 5, initially each node will simply represent a distinct sub-block of that of its parent. For each node, the parts of the inputs it represents are compared. As each level simply divides $[a]$ and $[b]$ into sub-blocks, overall complexity is $\mathcal{O}(h\ell)$, where h is the height of the tree. The construction is analogous to Protocol 2 and is seen as Protocol 4; the main difference is the addition of the function, F, specifying the fan-out which depends on the size of the parent node.

Protocol 4. Extending the tree construction protocol, treecomp(·) to non-binary fan-out.

Input: Two $\hat{\ell}$-bit, bitwise secret shared numbers, $\left[a_{\hat{\ell}-1}\right], \ldots, [a_0]$ and $\left[b_{\hat{\ell}-1}\right], \ldots, [b_0]$, as well as a $F : \mathbb{N} \mapsto \mathbb{N}$ specifying the fan-out.

Output: A tree of results of sub-comparisons with fan-out specified by F; each node represents a block of bit-positions.

if $\hat{\ell} = 1$ **then**
  return $(\text{comp}([a_0], [b_0]), \perp)$
end if
$[d] \leftarrow \text{comp}((\left[a_{\hat{\ell}-1}\right], \ldots, [a_0]), (\left[b_{\hat{\ell}-1}\right], \ldots, [b_0]))$
$f \leftarrow F(\hat{\ell})$
$[c] \leftarrow$ EmptyList
for $i = 0, \ldots, f-2$ **do**
  $[c] \leftarrow [c] || \text{ treecomp} \left((\begin{bmatrix} a_{\hat{\ell}-i(\lceil \hat{\ell}/f \rceil)-1} \\ b_{\hat{\ell}-i(\lceil \hat{\ell}/f \rceil)-1} \end{bmatrix}), \ldots, (\begin{bmatrix} a_{\hat{\ell}-(i+1)(\lceil \hat{\ell}/f \rceil)} \\ b_{\hat{\ell}-(i+1)(\lceil \hat{\ell}/f \rceil)} \end{bmatrix}) \right)$
end for
$[c] \leftarrow [c] || \text{treecomp} \left((\begin{bmatrix} a_{\hat{\ell}-(f-1)(\lceil \hat{\ell}/f \rceil)-1} \\ b_{\hat{\ell}-(f-1)(\lceil \hat{\ell}/f \rceil)-1} \end{bmatrix}), \ldots, (\begin{bmatrix} [a_0] \\ [b_0] \end{bmatrix}) \right)$
return $([d], [c])$

In order to ensure that only one node from each level is needed, each node must be "extended" to represent not only the comparison of its own block, but also those of its less significant siblings. This is best explained through an example. Consider node n_{32} on Fig. 1, this is the third child of n_3 and should therefore represent the comparison of the initial blocks of nodes n_{32}, n_{31}, and n_{30}. Once this is the case for all nodes, each post-fix will again depend on at most one node from every level. Continuing the example of Fig. 1: n_2, n_1, and n_0 are combined and stored in n_2 at the first level, while n_{32}, n_{31}, and n_{30} are combined and stored in n_{32} at the second. Together they represent everything up to and including the block of n_{32}.

Thus, once siblings are combined – $[d]$ is set to represent the comparison of not only their own block but that of all their less significant siblings – each of the ℓ output values can be computed. The task, which is a straight forward generalization of Protocol 3 of Sect. 5, is seen as Protocol 5. For all non-rightmost nodes on the path to the root, its right-neighbor sibling – which provides a cover for all the less significant bits of the parent – is included. Together, these provide the full cover.

The arguments of correctness and security are the same as in the previous section. Sub-strings are compressed resulting in a smaller comparison, and all interaction still occurs using only secure primitives. Regarding complexity, apart from the combining of the comparison results of siblings, this is $\mathcal{O}(h\ell)$. First each level is split into disjoint sub-blocks which are compared. Then ℓ comparisons of h-bit numbers are performed. It remains to explain how to choose F and combine siblings.

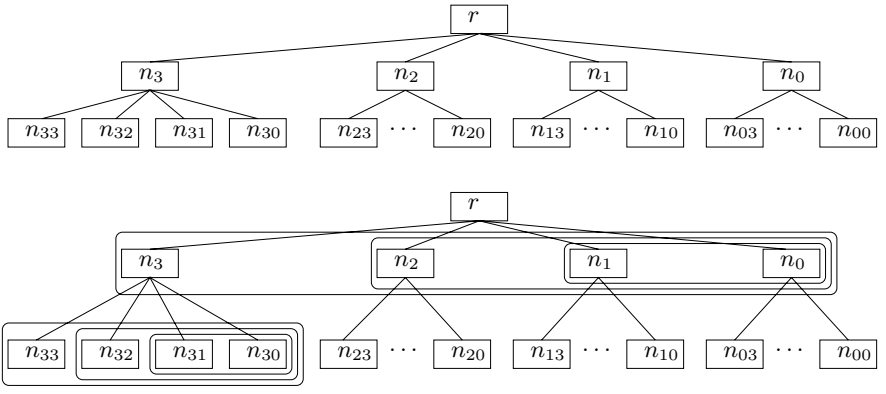

Fig. 1. Combining children with less significant siblings

Protocol 5. Extracting one result for post-fix comparison from a generalized treecomp(·)-tree.

Input: A bit-position, $1 \leq i < \ell$ and the output of the generalized treecomp(·) on two ℓ-bit, bitwise secret shared numbers, $[a_{\ell-1}], \ldots, [a_0]$ and $[b_{\ell-1}], \ldots, [b_0]$.
Output: Output equivalent to that of comp($\lceil a \bmod 2^i \rceil, \lceil b \bmod 2^i \rceil$).

```
    if i = ℓ then
        return root.[d]
    end if
    node ← leaf(i)
 5: [s] ← EmptyList
    while node is not root do
        if node is a non-rightmost child then
            [s] ← [s]||(right_sibling(node).[d])
        end if
10:     node ← parent(node)
    end while
    return comp([s])
```

6.1 $\mathcal{O}(\ell \log^*(\ell))$

In order to see how the sub-results of the children can be combined efficiently, note that *combine* means *compare up to and including this block*. This is a sub-problem of the post-fix comparison problem, where only every $\hat{\ell}/F(\hat{\ell})$'th result is needed. Viewed differently, it is the full post-fix comparison problem on the numbers represented by the outcome of the comparisons of the children – as before, replacing a block with the outcome of its comparison does not change the result.

While it seems as if we are back to square one, this is not the case, as the size of the post-fix comparison problem is reduced. For a (parent) problem with $\hat{\ell}$-bit inputs, letting the children be of logarithmic-size – i.e. setting the fan-out

$F(x) = \lceil x/\log(x) \rceil$ – implies that the new problem is of size $\lceil \hat{\ell}/\log(\hat{\ell}) \rceil$. This can be solved using $\mathcal{O}(\hat{\ell})$ secure multiplications – linear in the parent – using the solution of Sect. 5. Doing this for *all* nodes is linear at each level of the tree, as each node still represents a distinct block of the inputs. Thus, overall this is $\mathcal{O}(h\ell)$.

The choice of F reduces the size of children in log-steps resulting in $h = \log^*(\ell)$, i.e. this solution requires only $\mathcal{O}(\ell \log^*(\ell))$ multiplications. Security again follows from the fact that the primitives are secure by definition and no information is output.

6.2 Decreasing Complexity Further

Further improvements are trivial. It is merely a question of choosing a greater fan-out and solving the post-fix comparison problems linearly in the parents. The choice of fan-out above ensured linearity with the solution of Sect. 5. Increasing fan-out to $F(x) = \lceil x/\log^*(x) \rceil$ provides post-fix comparison problems of size $\hat{\ell}/\log^*(\hat{\ell})$ for nodes of size $\hat{\ell}$. These can be solved using $\mathcal{O}(\hat{\ell})$ secure multiplications by applying the improved solution of Sect. 6.1. This implies a $\mathcal{O}(\ell \log^{(*^{(2)})}(\ell))$ solution.

The process can be repeated indefinitely, however, each iteration increases round complexity. Each new version executes its predecessor as a sub-computation in addition to the initial comparison for every block and the concluding comparison for every output. Thus, to ensure a constant number of rounds, only a constant number of improvements are possible.

It is simple to verify that the complexity of the this secure computation for post-fix comparison is as stated – $\mathcal{O}(c)$ rounds and $\mathcal{O}(\ell \log^{(*^{(c)})}(\ell))$ where c is constant. From Sect. 4 it now follows that constant-rounds bit-decomposition is possible in the same complexity.

Theorem 1. *For any integer c, there exists a $\mathcal{O}(c)$-rounds bit-decomposition protocol for arbitrary $[x] \in \mathbb{Z}_p$, which requires only $\mathcal{O}(\ell \log^{(*^{(c)})}(\ell))$ invocations of the multiplication protocol.*

7 Analysis and Conclusion

This paper has presented a novel solution to the problem of constant-rounds bit-decomposition. Rather than simulating bitwise addition, a random mask and the masked value (transformed to simulate that the addition occurred over the integers) are reduced modulo all powers of two. Subtraction in $\mathbb{Z}_{2^i}$ for all i is then simulated efficiently with $\mathbb{Z}_p$ arithmetic. The resulting complexity of $\mathcal{O}(\ell \log^{(*^{(c)})}(\ell))$ is essentially linear – $\log^*$ is "bounded" by 5 on any reasonable input. Only when the *bit-length*, ℓ, of the prime defining the field exceeds 2^{65536} is it larger. A detailed analysis allows the protocol to be compared to the previous solutions.

The initial transformation requires the generation of a random element and a comparison. As the latter has a public input, the initial XOR and the selection of the associated block are costless. This implies $(52\ell + 24\sqrt{\ell}) + (11\ell + 6\sqrt{\ell}) = 63\ell + 30\sqrt{\ell}$ multiplications in $7 + 5 = 12$ rounds.

The basic $\mathcal{O}(\ell \log(\ell))$ solution to the post-fix comparison problem first compares at each level (equivalent to one full ℓ-bit comparison per level); second, it performs a comparison for each of the ℓ paths. With a few observations, namely that

- a single bitwise XOR of the inputs suffices for *all* levels
- for the output of comp$(\cdot)$, the XOR is free as $\binom{1}{1}$ does not occur
- for bit-decomposition, the final comparison of the post-fix comparison problem does not have to be comp$(\cdot)$

it can be seen that this requires $1 + 5 + 5 = 11$ rounds and

$$\ell + \log(\ell) \cdot \left(13\ell + 7\sqrt{\ell}\right) + \ell \cdot \left(12\log(\ell) + 6\sqrt{\log(\ell)}\right) < 31\ell\log(\ell) + 8\ell$$

multiplications, where the inequality holds for $\ell \geq 4$.

The analysis of the improved solution is similar. The "inner" post-fix comparison protocol is bounded by two sequential invocations of comp$(\cdot)$ (for each parent). This adds 10 rounds as well as

$$2(13\ell + 7\sqrt{\ell}) = 26\ell + 14\sqrt{\ell}$$

multiplications per level. Thus, complexity for the general solution is $11 + 10c$ rounds and $(31+26c)\ell\log^{(*^{(c)})}(\ell)+8\ell+14c\sqrt{\ell}\log^{(*^{(c)})}(\ell)$ secure multiplications. For the full bit-decomposition protocol, this implies $23 + 10c$ rounds and

$$(31 + 26c)\ell\log^{(*^{(c)})}(\ell) + 71\ell + 14c\sqrt{\ell}\log^{(*^{(c)})}(\ell) + 30\sqrt{\ell}$$

multiplications overall.

Concluding, we focus on the $\mathcal{O}(\ell\log(\ell))$ and $\mathcal{O}(\ell\log^*(\ell))$ solutions, as these are closest to the competition and there is little practical gain from the theoretically better options. For fairness, the optimized comparison (and thus also random element generation) is used in all cases. The details are summarized in Table 2.

The proposed $\mathcal{O}(\ell\log(\ell))$ solution requires fewer rounds and multiplications than any of the other solutions of that complexity. In the long run, the better complexity of [Tof07] will provide fewer multiplications, however, this comes at

Table 2. Explicit complexities of constant-rounds bit-decomposition protocols

	Rounds	**Multiplications**
[DFK+06]	38	$94\ell\log(\ell) + 63\ell + 30\sqrt{\ell}$
[NO07]	25	$47\ell\log(\ell) + 63\ell + 30\sqrt{\ell}$
[Tof07]	$38c$	$94\ell\log^{(c)}(\ell) + (130(c-1)+63)\cdot\ell + 30\sqrt{\ell}$
This paper, $\mathcal{O}(\ell\log(\ell))$	23	$31\ell\log(\ell) + 71\ell + 30\sqrt{\ell}$
This paper, $\mathcal{O}(\ell\log^*(\ell))$	33	$57\ell\log^*(\ell) + 71\ell + 14\sqrt{\ell}\log^*(\ell) + 30\sqrt{\ell}$

a cost of increased round complexity. Moreover, for the $\mathcal{O}(\ell \text{loglog}(\ell))$ solution, the turning point does not occur until $\ell \approx 2^{16}$, due to the large constants.

The proposed $\mathcal{O}(\ell \log^*(\ell))$ solution beats [Tof07] outright, however, both the solution of [NO07] and the present $\mathcal{O}(\ell \log(\ell))$ have better constants. When $\ell \approx 38$ this solution and the one of [NO07] use about the same number of secure multiplications, while the present $\mathcal{O}(\ell \log(\ell))$ solution is beaten around $\ell \approx 180$ bits. Thus, if additional rounds are acceptable, then the theoretically better solution may also win in practice.

Finally, it is noted that "hybrids" are also possible. The ideas of [Tof07] can be applied to [NO07], as well as be combined with the ideas of this paper. Though asymptotic complexity is not improved, such variations may be competitive in practice, as the c factor is on the linear rather than on the $\ell \log^{(c)}(\ell)$ term.

Acknowledgements

The author would like to thank Eike Kiltz and Berry Schoenmakers for discussions and suggestions. The anonymous referees are thanked for their input and comments.

References

[ACS02] Algesheimer, J., Camenisch, J.L., Shoup, V.: Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 417–432. Springer, Heidelberg (2002)

[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)

[DFK+06] Damgård, I.B., Fitzi, M., Kiltz, E., Nielsen, J.B., 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)

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

[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)

[Tof07] Toft, T.: Primitives and Applications for Multi-party Computation. PhD thesis, University of Aarhus (2007), `http://www.daimi.au.dk/~ttoft/publications/dissertation.pdf`