Unconditionally Secure Constant-Rounds Multi-Party Computation for Equality, Comparison, Bits and Exponentiation

Ivan Damgård[1], Matthias Fitzi[1⋆], Eike Kiltz[2⋆⋆], Jesper Buus Nielsen[1⋆⋆⋆], and Tomas Toft[1†]

[1] University of Aarhus
Department of Computer Science
DK-8200 Aarhus N, Denmark
[2] CWI Amsterdam
The Netherlands

Abstract. We show that if a set of players hold shares of a value $a \in \mathbb{F}_p$ for some prime p (where the set of shares is written $[a]_p$), it is possible to compute, in constant rounds and with unconditional security, sharings of the bits of a, i.e., compute sharings $[a_0]_p, \ldots, [a_{\ell-1}]_p$ such that $\ell = \lceil \log_2 p \rceil$, $a_0, \ldots, a_{\ell-1} \in \{0, 1\}$ and $a = \sum_{i=0}^{\ell-1} a_i 2^i$. Our protocol is secure against active adversaries and works for any linear secret sharing scheme with a multiplication protocol. The complexity of our protocol is $\mathcal{O}(\ell \log \ell)$ invocations of the multiplication protocol for the underlying secret sharing scheme, carried out in $\mathcal{O}(1)$ rounds.
This result immediately implies solutions to other long-standing open problems such as constant-rounds and unconditionally secure protocols for deciding whether a shared number is zero, comparing shared numbers, raising a shared number to a shared exponent and reducing a shared number modulo a shared modulus.

1 Introduction

Assume that n parties have shared values $a_1, \ldots, a_\ell$ from some field $\mathbb{F}$ using some linear secret sharing scheme, such as Shamir's. Let $f : \mathbb{F}^\ell \to \mathbb{F}^m$. By

⋆ Supported by SECOQC, Secure Communication based on Quantum Cryptography, under the Information Societies Technology Programme of the European Commission, IST-2003-506813.

⋆⋆ The paper was written while the author was a visitor at University of California, San Diego, supported by a DAAD postdoc fellowship.

⋆⋆⋆ Supported by FICS, Foundations In Cryptology and Security, centre of the Danish National Science Research Council and ECRYPT, European Network of Excellence in Cryptology, under the Information Societies Technology Programme of the European Commission, IST-2002-507932.

† Supported by SCET, Secure Computing, Economy, and Trust, Alexandra Instituttet A/S.

computing f with unconditional security on the sharings we mean that the parties run among themselves a protocol using a network with perfectly secure point-to-point channels. The protocol results in the parties obtaining sharings of $(b_1, \ldots, b_m) = f(a_1, \ldots, a_\ell)$, while leaking no information on the values $a_1, \ldots, a_\ell$ or $b_1, \ldots, b_m$. The question *which functions can be computed with unconditional security on sharings, using a constant rounds protocol* is a long-standing open problem [BB89].

However, a number of functions are known to have unconditionally secure, constant-rounds protocols. The most general class with known solutions are functions with a constant-depth arithmetic circuit (counting unbounded fan-in addition and unbounded fan-in multiplication as one gate towards the depth).

The only non-trivial part needed in these solutions is unbounded fan-in multiplication $b = \prod_{i=1}^{\ell} a_i$. This can be done in constant rounds using the techniques by Bar-Ilan and Beaver [BB89], assuming a single multiplication can be done in constant rounds, which is indeed the case for standard linear (verifiable) secret-sharing schemes.

However, a number of functions do not have small constant-depth arithmetic solutions. Consider, e.g., the function $\stackrel{?}{<}: \mathbb{F}_p \times \mathbb{F}_p \to \mathbb{F}_p$, where $(a \stackrel{?}{<} b) \in \{0, 1\}$ and $(a \stackrel{?}{<} b) = 1$ iff $a < b$ (where a and b are considered as residues $a, b \in \{0, 1, \ldots, p-1\}$). This function has a huge number of zeros and is not constant zero. Therefore we cannot hope for an efficient arithmetic solution to computing $\stackrel{?}{<}$ (the function can of course be expressed as a polynomial over the field, and thus a constant-depth circuit, but the circuit would have a number of gates proportional to the size of the field).

On the other hand a number of results are known where if the inputs are given in a particular form, then any function which can be expressed by a binary Boolean circuit with g gates and depth d, can be computed unconditionally securely in constant rounds, by evaluating a constant-depth arithmetic circuit with $O(2^d g)$ gates (see e.g [BB89,IK00,IK02]).

If, in particular, the input a is delivered as bitwise sharings $[a_0]_p, \ldots, [a_{\ell-1}]_p$ and $b = f(a)$ can be computed using a binary Boolean circuit with depth d and g gates, then sharings of the bits of $b = f(a)$ can be computed with complexity[1] $O(2^d g)$, unconditionally secure in constant rounds. This can e.g. be done using Yao's circuit scrambling technique with an unconditionally secure encryption scheme — an observation first made by [IK02]. This would e.g. allow to compute the function $\stackrel{?}{<}: (\mathbb{F}_p)^\ell \times (\mathbb{F}_p)^\ell \to \mathbb{F}_p, ((a_0, \ldots, a_{\ell-1}), (b_0, \ldots, b_{\ell-1})) \mapsto \sum_{i=0}^{\ell-1} a_i 2^i \stackrel{?}{<} \sum_{i=0}^{\ell-1} b_i 2^i$ unconditionally securely in constant rounds.

So, different representations of the inputs allow different classes of functions to be computed unconditionally securely in constant rounds — at least

[1] For the rest of the paper we measure the complexity of protocols by the maximal number of invocations of the multiplication protocol, which is typically the dominating term in the communication complexity. The exact communication complexity then depends on the communication complexity of the multiplication protocol used.

with the current knowledge of the area. It would therefore be very useful to be able to change representations efficiently. Previously it was not known how to do this. For instance, this was the reason why the protocols of Cramer and Damgård [CD01] for linear algebra in constant rounds could not handle fields with large characteristic without assuming that the input was shared bitwise to begin with, which limits the applicability of those protocols. In this paper, we therefore investigate the problem of changing between sharings modulo a prime p and bitwise sharings.

1.1 Our results

Given a prime p, let $\ell = \lceil \log_2 p \rceil$. We will show how to compute, unconditionally secure and in constant rounds, $[a_0]_p, \dots, [a_{\ell-1}]_p$ from $[a]_p$ such that $a_0, \dots, a_{\ell-1} \in \{0,1\} \subseteq \mathbb{Z}_p$ and such that $a = \sum_{i=0}^{\ell-1} a_i 2^i$. The complexity is bounded by $\mathcal{O}(1)$ rounds and $\mathcal{O}(\ell \log_2 \ell)$ invocations of the multiplication protocol.

The only assumptions we need about the underlying secret sharing scheme are the following: 1) the secret sharing scheme is linear (i.e., given sharings $[a]_p$ and $[b]_p$ and public constants $c, d \in \mathbb{Z}_p$, the parties can securely compute a sharing $[ac + bd \bmod p]_p$ without interaction) and 2) there exists a constant-round multiplication protocol for the secret sharing scheme (i.e., given sharings $[a]_p$ and $[b]_p$, the parties can securely compute a sharing $[ab \bmod p]_p$ by interacting). If the multiplication protocol (and the secret sharing scheme) is secure against active adversaries, our protocols will be actively secure too. Likewise, if secret sharing scheme and multiplication protocol are adaptively secure, our protocols inherit this property. The assumption on multiplication implies that the adversary structure must be $Q2$ which, in the standard threshold case, means that we need honest majority.

This result immediately implies efficient constant-rounds protocols for some interesting problems. In particular, we can also compute, in constant rounds, outputs from the following functions in shared form:

- The equality function asking whether a shared input value is zero or not. This function was exactly what was missing in [CD01] in order to handle fields with large characteristics.
- The less-than comparison function of two numbers from $\mathbb{F}_p$, when considered residues in $\{0, 1, \dots, p-1\}$.
- Modulo reduction, performing a discrete modulo reduction (with respect to a public/shared modulus).
- Discrete Exponentiation (with respect to a public/shared exponent and modulus).

We note that, while unconditional security is typically defined by requiring that the information leaked by the protocol is exponentially small in some security parameter κ, our protocols obtain a slightly stronger notion, which has also been considered in the literature. In particular, our protocols are perfectly secure except with probability $O(2^{-\kappa})$ — i.e. with probability $1 - O(2^{-\kappa})$ no information is leaked at all. Furthermore, the parties will be able to detect when a run of the protocol is in progress which would leak information if completed, and have the power to abort such a run. This yields a *perfectly secure protocol*, except that with probability $O(2^{-\kappa})$ it might terminate with some abort symbol $\perp$.[2]

1.2 Related work

There has been a considerable amount of previous work on unconditionally secure constant-rounds multi-party computation with honest majority (c.f. [BB89] and [FKN94,IK97,CD01,Bea00,IK00,IK02]). As mentioned, this work has shown that some functions can indeed be computed in constant rounds with unconditional security, but this has been limited to restricted classes of functions, such as NC_1 or non-deterministic log-space.

In [ACS02] Algesheimer, Camenisch and Shoup also present a protocol for securely computing the bit-decomposition $[a]_p \mapsto ([a_0]_p, \ldots, [a_{\ell-1}]_p)$. It however only provides correctness and privacy when a is guaranteed to be noticeably smaller than p. Furthermore, it is only passively secure and is not constant rounds.

1.3 Organization

In Section 2 we give some technical preliminaries. In Section 3 we give the high-level protocol for bit decomposition, assuming a number of results from subsequent sections, in particular that it is possible to add bitwise-shared numbers and compare bitwise-shared number within certain complexities. In Section 4 we show how to generate the sharing of a uniformly random bit. In Section 5 we give the protocol for comparing two bitwise-shared numbers and in Section 6 we give the protocol for adding two bitwise-shared numbers. Finally, in Section 7 we mention a couple of applications of the new bit-decomposition protocol.

[2] Choosing between unconditional (but imperfect) termination, correctness or privacy, we find that settling for imperfect termination but perfect correctness (on termination) and perfect privacy is the better choice. Simply because the other unconditional notions can be obtained from such a solution. To get perfect termination and perfect correctness but only unconditional privacy: when the protocol aborts, reconstruct the inputs and compute the results. This yields a protocol which is perfect except that it leaks information with small probability. To get perfect termination, perfect privacy but only unconditional correctness: when the protocol aborts, simply return with some dummy guess at the results. This yields a protocol which is perfect except that it is incorrect with small probability. Finally, to get a perfectly secure protocol: rerun the protocol when it aborts. This gives a perfectly secure protocol. It, however, only runs in *expected* constant rounds.

2 Preliminaries

In this section we introduce some notation and some known techniques.

We assume that n parties are connected by perfectly secure channels in a synchronous network. Let $\mathbb{F}_p$ denote the finite field with p elements where p is a prime, and let $\ell = \lceil \log_2 p \rceil$. We will assume throughout that $p > 2^\kappa$, and so whenever one of our protocols abort with a probability that is $O(1/p)$, this will be considered negligible and will be ignored. If one needs to execute our (sub)protocol(s) with a given (small) prime p, one can always execute in parallel a sufficiently large number of instances to make the failure probability small enough.

By $[a]_p$ we denote a secret sharing of $a \in \mathbb{F}_p$ over $\mathbb{F}_p$. We assume that the secret-sharing scheme allows to compute a sharing $[a + b \bmod p]_p$ from $[a]_p$ and $[b]_p$ without communication, and that it allows to compute $[ab \bmod p]_p$ from $a \in \mathbb{F}_p$ and $[b]_p$ without communication; We write

$$[a + b \bmod p]_p \leftarrow [a]_p + [b]_p$$

and

$$[ab \bmod p]_p \leftarrow a[b]_p$$

for these operations. The secret-sharing scheme should of course also allow to take a sharing $[c]_p$ and reveal the value $c \in \mathbb{F}_p$ to all parties; We write

$$c \leftarrow \textsc{reveal}([c]_p) \ .$$

We also assume that the secret sharing scheme allows to compute a sharing $[ab \bmod p]_p$ from $[a]_p$ and $[b]_p$ with unconditional security. We denote the multiplication protocol by MULT, and write

$$[ab \bmod p]_p \leftarrow \textsc{mult}([a]_p, [b]_p) \ .$$

Sometimes we will also write

$$[b \bmod p]_p \leftarrow \textsc{mult}([a_1]_p, \dots, [a_l]_p) \ ,$$

to avoid writing $b_2 \leftarrow \textsc{mult}([a_1]_p, [a_2]_p)$, $b_3 \leftarrow \textsc{mult}([b_2]_p, [a_3]_p)$, ..., $b \leftarrow \textsc{mult}([b_{l-1}]_p, [a_l]_p)$. This costs $l - 1$ rounds and $l - 1$ invocations of MULT.

We will express the protocols' round complexities as the number of sequential rounds of MULT invocations — and their communication complexities as the overall number of MULT invocations. I.e., if we first run a copies of MULT in parallel and then run b copies of MULT in parallel, then we say that we have round complexity 2 and communication complexity $a + b$. Note that standard linear (verifiable) secret-sharing schemes have efficient constant-rounds protocols for multiplication.

For our protocols to be actively secure, the secret sharing scheme and the multiplication protocol should be actively secure. This in particular means that the adversary structure must be $Q2$. By the adversary structure we mean the set $\mathcal{A}$ of subsets $C \subset \{1, \dots, n\}$ which the adversary might corrupt; It is $Q2$ if it holds for all $C \in \mathcal{A}$ that $\{1, \dots, n\} \setminus C \notin \mathcal{A}$.

All our protocols can be proven secure in the UC model [Can01]. In the UC model our protocols can be expressed in a hybrid model with an ideal functionality F allowing the parties to privately load values in $\mathbb{F}_p$ into F and allowing the parties to add, multiply and output loaded and/or computed values. For an approach to formulate such an ideal functionality, see e.g., the Arithmetic Black-Box (ABB) from [DN03]. It can then be shown that an information theoretic VSS with a multiplication protocol implements this ideal functionality (as an example the VSS schemes from [CDM00] will do). The full version of this paper will contain more details on how our protocols can be proven secure in the UC model.

2.1 Some known techniques

The following known techniques will be of importance later on.

Random Elements. The parties can share a uniformly random, unknown field element. We write

$$[a]_p \leftarrow \text{RAN}_p() \ .$$

This is done by letting each party P_i deal a sharing $[a_i]_p$ of a uniformly random $a_i \in \mathbb{F}_p$. Then the parties compute the sharing $[a]_p = \sum_{i=1}^{n} [a_i]_p$. The communication complexity of this is given by n dealings, which we assume is upper bounded by the complexity of one invocation of the multiplication protocol.

If passive security is considered, this is trivially secure. If active security is considered and some party refuses to contribute with a dealing, the sum is just taken over the contributing parties. This means that the sum is at least taken over a_i for $i \in H$, where $H = \{1, \dots, n\} \setminus C$ for some $C \in \mathcal{A}$. Since $\mathcal{A}$ is Q2 it follows that $H \notin \mathcal{A}$. So, at least one honest party will contribute to the sum, implying randomness and privacy of the sum.

Random Invertible Elements. Using [BB89] the parties can share a uniformly random, unknown, invertible field element along with a sharing of its inverse. We write

$$([a]_p, [a^{-1}]_p) \leftarrow \text{RAN}_p^*() \ ,$$

and it proceeds as follows: $[a]_p \leftarrow \text{RAN}_p()$ and $[b]_p \leftarrow \text{RAN}_p()$. $[c]_p = \text{MULT}([a]_p, [b]_p)$. $c \leftarrow \text{REVEAL}([c]_p)$. If $c \notin \mathbb{F}_p^*$, then abort. Otherwise, proceed as follows: $[a^{-1} \bmod p]_p \leftarrow (c^{-1} \bmod p)[a]_p$. Output $([a]_p, [a^{-1}]_p)$.

The correctness is straightforward. As for privacy, if $c \in \mathbb{F}^*$, then (a, b) is a uniformly random element from $\mathbb{F}^* \times \mathbb{F}^*$ for which $ab \bmod p = c$, and thus a is a uniformly random element in $\mathbb{F}_p^*$. If $c \notin \mathbb{F}^*$, then the algorithm aborts. This happens with probability less than $2/p$. The complexity is (at most) 2 rounds and 3 invocations of MULT.

Unbounded Fan-In Multiplication. Using the technique from [BB89] it is possible to do unbounded fan-in multiplication in constant rounds. For the special case where we compute all "prefix products" $\prod_{i=1}^{m} a_i$ ($m = 1, \ldots, \ell$), we write

$$([a_1]_p, \ldots, [(a_1 a_2 \cdots a_\ell) \bmod p]_p) \leftarrow \textsc{mult}^*([a_1]_p, \ldots, [a_\ell]_p) \ .$$

In the following, we only need the case where we have inputs $[a_1]_p, \ldots, [a_\ell]_p$, where $a_i \in \mathbb{F}_p^*$. For $1 \le i_0 \le i_1 \le \ell$, let $a_{i_0,i_1} = \left(\prod_{i=i_0}^{i_1} a_i\right) \bmod p$. We are often only interested in computing $a_{1,\ell}$, but the method allows to compute any other a_{i_0,i_1} at the cost of one extra multiplication. For the complexity analysis, let A denote the number of a_{i_0,i_1}'s which we want to compute.

First run $\textsc{ran}_p^*$ $\ell + 1$ times in parallel, to generate $[b_0 \in_R \mathbb{F}^*]_p, [b_1 \in_R \mathbb{F}^*]_p, \ldots, [b_\ell \in_R \mathbb{F}^*]_p$, along with $[b_0^{-1}]_p, [b_1^{-1}]_p, \ldots, [b_\ell^{-1}]_p$, using 2 rounds and $3(\ell + 1)$ invocations of $\textsc{mult}$. For simplicity we will use the estimate of 3ℓ invocations.

Then for $i = 1, \ldots, \ell$ compute and reveal $[d_i]_p = \textsc{mult}([b_{i-1}]_p, [a_i]_p, [b_i^{-1}]_p)$, using 2 rounds and 2ℓ invocations of $\textsc{mult}$.

Now we have that $d_{i_0,i_1} = \prod_{i=i_0}^{i_1} d_i = b_{i_0-1}(\prod_{i=i_0}^{i_1} a_i) b_{i_1}^{-1} = b_{i_0-1} a_{i_0,i_1} b_{i_1}^{-1}$ (mod p), so we can compute $[a_{i_0,i_1}]_p = d_{i_0,i_1} \textsc{mult}([b_{i_0-1}^{-1}]_p, [b_{i_1}]_p)$, using 1 round and A invocations of $\textsc{mult}$.

The overall complexity is 5 rounds and $5\ell + A$ invocations of $\textsc{mult}$.

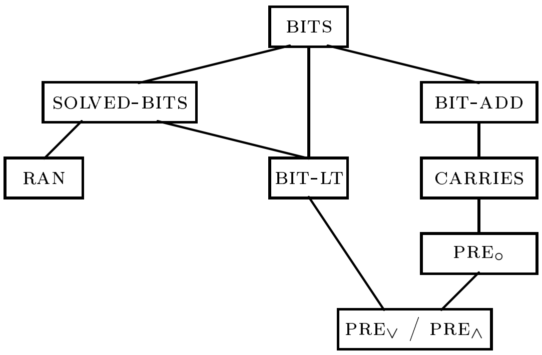

Fig. 1. *Protocol hierarchy.*

3 Bit-Decomposition

Let p be a prime $p \in [2^{\ell-1}, 2^\ell]$. We look at the bit-decomposition function $\textsc{bits} : \mathbb{F}_p \to (\mathbb{F}_p)^\ell, a \mapsto (a_0, \ldots, a_{\ell-1})$ given by $a_0, \ldots, a_{\ell-1} \in \{0, 1\} \subseteq \mathbb{F}_p$ and

$a = \sum_{i=0}^{\ell-1} a_i 2^i$, where $a \in \mathbb{F}_p$ is considered a residue $a \in \{0, 1, \ldots, p-1\}$. We denote a run of this protocol by

$$([a_0]_p, \ldots, [a_{\ell-1}]_p) \leftarrow \text{BITS}([a]_p) \ .$$

The protocol for bit decomposition makes use of various sub-protocols which in turn draw on further sub-protocols. The dependency between the building blocks can be seen in Fig. 1. We now describe the highest level sub-protocols:

- Random solved BITS. This protocol has no inputs, and has outputs

  $$([b_0]_p, \ldots, [b_{\ell-1}]_p, [b]_p) \leftarrow \text{SOLVED-BITS}() \ ,$$

  where b is a uniformly random element $b \in \mathbb{F}_p$ and $(b_0, \ldots, b_{\ell-1}) = \text{BITS}(b)$. As shown in the next subsection, this can be done using 21 rounds and 96ℓ invocations of the multiplication protocol.
- Bitwise sum. Let $[x]_{\text{B}} = [x_0]_p, \ldots, [x_{l-1}]_p$ denote a bitwise sharing of an integer x. We use

  $$[z]_{\text{B}} \leftarrow \text{BIT-ADD}([x]_{\text{B}}, [y]_{\text{B}})$$

  to denote the computation of a bitwise sharing $[z]_{\text{B}} = [z_0]_p, \ldots, [z_l]_p$ of $x+y$ from bitwise sharings, $[x]_{\text{B}} = [x_0]_p, \ldots, [x_{l-1}]_p$ and $[y]_{\text{B}} = [y_0]_p, \ldots, [y_{l-1}]_p$, of integers x and y. The length l need not be the length ℓ of the prime p. In Section 6 it is shown how to implement BIT-ADD unconditionally securely in constant rounds. When $x, y \in \{0, \ldots, 2^l - 1\}$ the complexity is 37 rounds and $55l \log_2 l$ invocations of the multiplication protocol.
- Bitwise less-than. Finally we use

  $$[x \overset{?}{<} y]_p \leftarrow \text{BIT-LT}([x]_{\text{B}}, [y]_{\text{B}})$$

  to denote the computation of a sharing of the bit $(x \overset{?}{<} y) \in \{0, 1\}$, where $(x \overset{?}{<} y) = 1$ iff $x < y$, starting from bitwise sharings, $[x]_{\text{B}} = [x_0]_p, \ldots, [x_{l-1}]_p$ and $[y]_{\text{B}} = [y_0]_p, \ldots, [y_{l-1}]_p$, of integers x and y; Again l need not be ℓ. In Section 5 it is shown how to implement BIT-LT unconditionally securely in constant rounds. The complexity is 19 rounds and $22l$ invocations of the multiplication protocol.

We sometimes run the above protocols on non-shared inputs. If e.g. x is an integer known by all parties, then we let

$$[z]_{\text{B}} \leftarrow \text{BIT-ADD}(x, [y]_{\text{B}}) \ ,$$

mean the following: first compute the bitwise representation $(x_0, x_1, \ldots, x_{l-1})$ of x, then let $[x]_{\text{B}} = ([x_0]_p, \ldots, [x_{l-1}]_p)$ be some dummy bitwise sharing of x, and then run $[z]_{\text{B}} \leftarrow \text{BIT-ADD}([x]_{\text{B}}, [y]_{\text{B}})$.

The bit decomposition of $[a]_p$ now proceeds as follows.

Protocol $[a]_{\mathrm{B}} \leftarrow \textbf{BITS}([a]_p)$

1. The input is $[a]_p$, where $a \in \mathbb{F}_p$.
2. $([b_0]_p, \ldots, [b_{\ell-1}]_p, [b]_p) \leftarrow \textsc{solved-bits}()$.
3. $[a-b]_p \leftarrow [a]_p - [b]_p$.
4. $c \leftarrow \textsc{reveal}([a-b]_p)$, where $c \in \mathbb{F}_p$.
5. $[d]_{\mathrm{B}} \leftarrow \textsc{bit-add}(c, [b]_{\mathrm{B}})$, where $[d]_{\mathrm{B}} = ([d_0]_p, \ldots, [d_\ell]_p)$.
6. $[q]_p \leftarrow \textsc{bit-lt}(p, [d]_{\mathrm{B}})$.
7. $(f_0, \ldots, f_{\ell-1}) = \textsc{bits}(2^\ell - p)$, the bitwise representation of the positive integer $2^\ell - p$.
8. For $i = 0, \ldots, \ell-1$ in parallel: $[g_i]_p = f_i[q]_p$.
9. $[g]_{\mathrm{B}} = ([g_0]_p, \ldots, [g_{\ell-1}]_p)$.
10. $[h]_{\mathrm{B}} \leftarrow \textsc{bit-add}([d]_{\mathrm{B}}, [g]_{\mathrm{B}})$, where $[h]_{\mathrm{B}} = ([h_0]_p, \ldots, [h_{\ell+1}]_p)$.
11. $[a]_{\mathrm{B}} = ([h_0]_p, \ldots, [h_{\ell-1}]_p)$.
12. Output $[a]_{\mathrm{B}}$.

As for the privacy, notice that assuming that the sub-protocols leak no information, the only place where information is potentially leaked is in Step 4, where c is leaked. Since b is assumed to be a uniformly random, unknown value from $\mathbb{F}_p$, independent of a, it however follows that c is uniformly random in $\mathbb{F}_p$ and leaks no information about a.

As for the correctness, notice that $c = a - b \bmod p$ and $d = c + b$ (in the integers). Therefore $d = a + qp$ for some $q \in \{0, 1\}$. Since $a \in \{0, 1, \ldots, p-1\}$ it follows that $q = 1$ iff $p < d$. A sharing of this q is computed in Step 6. Then let $f = 2^\ell - p$, let $g \in \mathbb{Z}$ be the integer which is bitwise shared in Step 9, and let $h \in \mathbb{Z}$ be the integer which is bitwise shared in Step 10. Clearly, $g = qf = q2^\ell - qp$ (in the integers). Therefore $h = d + g = (a + qp) + (q2^\ell - qp) = a + q2^\ell$. In Step 11 we then compute a as $h \bmod 2^\ell$ by dropping the two most significant bits of h.

As for the complexity, we generated one solved BITS, had two applications of BIT-ADD and one application of BIT-LT. This yields a total complexity of 114 rounds and $110\ell \log_2 \ell + 118\ell$ invocations.

3.1 Generating random solved BITS

We now describe the protocol SOLVED-BITS. As a sub-protocol we use a protocol $\textsc{ran}_2$ for generating uniformly random shared bits. This protocol has no inputs, and outputs a sharing $[a]_p$, where $a \in \{0, 1\} \subseteq \mathbb{F}_p$ is uniformly random. We write

$$[a]_p \leftarrow \textsc{ran}_2()\ .$$

In Section 4, we show how to implement $\textsc{ran}_2$ in 2 rounds and 2 invocations of the multiplication protocol.

The generation of a random input/output pair for BITS proceeds as follows.

Protocol $([b]_{\mathrm{B}}, [b]_p) \leftarrow$ **SOLVED-BITS**$()$

1. For $i = 0, \ldots, \ell - 1$ in parallel: $[b_i]_p \leftarrow \text{RAN}_2()$.
2. $[b]_{\mathrm{B}} = ([b_0]_p, \ldots, [b_{\ell-1}]_p)$.
3. $[c]_p \leftarrow \text{BIT-LT}([b]_{\mathrm{B}}, p)$.
4. $c \leftarrow \text{REVEAL}([c]_p)$.
5. If $c = 0$, then abort. Otherwise proceed as below.
6. $[b]_p \leftarrow \sum_{i=0}^{\ell-1} 2^i [b_i]_p$.
7. Output $([b]_{\mathrm{B}}, [b]_p)$.

As for the correctness, notice that $[b]_{\mathrm{B}}$ is by construction the bit-wise sharing of $[b]_p$. Furthermore, b is uniformly random from $\{0, 1, \ldots, 2^\ell - 1\}$. So under the condition that SOLVED-BITS does not abort, b is uniformly random from $\{0, 1, \ldots, p - 1\}$, as desired.

As for the privacy, when SOLVED-BITS does not abort, the only information leaked is that $b < p$. This is however an *a priory* fact by the output requirement on SOLVED-BITS.

Let us examine the probability that SOLVED-BITS aborts. In case one is able to control the choice of the prime p, an optimal choice would be to let p be a Mersenne prime $p = 2^\ell - 1$ for some $\ell > \kappa$. In that case the probability that $b \geq p$ is less than $2^{-\kappa}$. Although the Mersenne primes soon become sparse, this would at least work for small values of ℓ. At the time of writing $p = 2^{25964951} - 1$ is the largest p for which we know that this works [NWKo]. Other primes close to powers of two work almost as nicely.

In the worst-case, where we have no control over p, our only guarantee is that $p \in [2^{\ell-1}, 2^\ell]$ for some ℓ. In that case the probability that $b \leq p$ when $b \in_R \{0, 1, \ldots, 2^\ell - 1\}$ can be as large as $1/2$. Using a Chernoff bound it can be seen that if one generates $n = 12\kappa$ candidates, then the probability that less than $n/4$ of them satisfy $b < p$ is upper bounded by $2^{-\kappa}$.

As for the complexity, one run of the basic SOLVED-BITS requires ℓ calls of RAN_2 and one call of BIT-LT, neglecting the cost of the one call to REVEAL. This gives a complexity of 21 rounds and 24ℓ invocations of the multiplication protocol. If the basic protocol has to be repeated in parallel to get a lower abort probability, the round complexity is still 21, and the amortized communication complexity goes up to 96ℓ.

4 Random Bits

We now describe a protocol RAN_2 for securely generating a sharing of a uniformly random bit. The protocol has no inputs, and the output is a sharing $[a]_p$ of a uniformly random $a \in \{0, 1\} \subseteq \mathbb{F}_p$. We assume that $p > 2$ such that $\mathbb{F}_p$ does not have characteristic 2.

First some notation. Let $\mathbb{F}_p^*$ be the set of non-zero elements of $\mathbb{F}_p$ and let $Q_p \subset \mathbb{F}_p^*$ be the subset of squares. For $a \in Q_p$, let $\sqrt{a}$ be the unique $b \in \{1, \ldots, (p-1)/2\}$ where $b^2 \bmod p = a$. We define $S : \mathbb{F}_p^* \to \mathbb{F}_p$ by $S(x) = 1$ if

$0 < x < p/2$ and $S(x) = -1$ if $p/2 < x < p$. Note that it holds for all $x \in \mathbb{F}_p^*$ that $x = S(x)\sqrt{x^2} \bmod p$. Clearly, if $a \in_R \mathbb{F}_p^*$ is a uniformly random non-zero element, then $S(a)$ is uniformly random in $\{1, -1\}$ and, furthermore, $S(a) = a(\sqrt{a^2})^{-1}$. This suggests the following protocol.

Protocol $[d]_p \leftarrow$ **RAN**$_2()$

1. $[a]_p \leftarrow \mathrm{RAN}_p()$.
2. $[a^2 \bmod p]_p = \mathrm{MULT}([a]_p, [a]_p)$.
3. $a^2 \bmod p \leftarrow \mathrm{REVEAL}([a^2 \bmod p]_p)$.
4. If $a^2 \bmod p = 0$, then abort. Otherwise, proceed as below.
5. $b = \sqrt{a^2} \bmod p$.
6. $[c]_p \leftarrow (b^{-1} \bmod p)[a]_p$.
7. $[d]_p \leftarrow 2^{-1}([c]_p + 1)$.
8. Output $[d]_p$.

As for correctness, notice that when RAN_2 does not abort, then $c = S(a)$, where c is the value shared in Step 6. Therefore c is uniformly random in $\{1, -1\}$. It then easily follows that d is uniformly random in $\{0, 1\}$.

As for privacy, note that when the protocol does not abort, then a is uniformly random from $\mathbb{F}_p^*$, and we are essentially using $S(a)$ as output. The only information leaked about a is $a^2 \bmod p$, which is independent of $S(a)$ when a is uniformly random in $\mathbb{F}_p^*$.

If $a = 0$, then the protocol aborts. This happens with probability $1/p$.

The complexity of generating $[a]_p$ is bounded by the complexity of one multiplication. Then one multiplication is needed to compute $[a^2 \bmod p]_p$. The rest is essentially for free. This gives a complexity of 2 rounds and 2 invocations.

5 Bitwise Less-Than

We show how to compare two bitwise-shared numbers in constant rounds. We first present two sub-protocols.

5.1 Symmetric functions

Assume that we have inputs $[a_1]_p, \ldots, [a_\ell]_p$, where $a_1, \ldots, a_\ell \in \{0, 1\} \subseteq \mathbb{F}_p$, and want to compute a symmetric Boolean function f on these. We also need to assume that $\mathbb{F}_p$ has characteristic larger than $\ell + 1$, which here just means that we need that $\ell < p - 1$.

A symmetric Boolean function only depends on the number of 1's in its input, it can therefore be written as $f(x_1, \ldots, x_\ell) = \phi(1 + \sum_{i=1}^{\ell} x_i)$ for some function $\phi : \{1, 2, \ldots, \ell + 1\} \to \{0, 1\}$. By Lagrange interpolation, we can construct a polynomial with coefficients $\alpha_0, \ldots, \alpha_\ell$ such that $\phi(X) = \sum_{i=0}^{\ell} \alpha_i X^i \bmod p$ for $X \in \{1, 2, \ldots, \ell + 1\}$. This allows a particularly efficient secure computation, as follows.

Protocol $[f(a_1,\ldots,a_\ell) \bmod p]_p \leftarrow f([a_1]_p,\ldots,[a_\ell]_p)$ 1. $[a]_p \leftarrow 1+\sum_{i=1}^{\ell}[a_i]_p$. 2. $([a]_p,[a^2 \bmod p]_p,\ldots,[a^{\ell+1} \bmod p]_p) \leftarrow \textsc{mult}^*([a]_p,\ldots,[a]_p)$. 3. $[f(a) \bmod p]_p \leftarrow \sum_{i=0}^{\ell}\alpha_i[a^i \bmod p]_p$.

In Step 2 we have that $a \in \mathbb{F}_p^*$, so we can apply the protocol $\textsc{mult}^*$ securely. The protocol is clearly private and correct. The complexity is 5 rounds and 6ℓ invocations of $\textsc{mult}$.

5.2 Prefix-Or

Assume that we have inputs $[a_1]_p,\ldots,[a_\ell]_p$, where $a_1,\ldots,a_\ell \in \{0,1\} \subseteq \mathbb{F}_p$, and want to compute the prefix-or $[b_1]_p,\ldots,[b_\ell]_p$, where $b_i = \vee_{j=1}^{i} a_j$.

To obtain complexity linear in ℓ, we use the method by Chandra, Fortune and Lipton [CFL83a]. For notational convenience, assume that $\ell=\lambda^2$ for an integer λ. We will split a into λ blocks of λ bits each. For this purpose we rename each bit a_k as $a_{i,j}$ where $k=\lambda(i-1)+j$, and $i,j=1,\ldots,\lambda$. Thus, $a=(a_{1,1},a_{1,2},\ldots,a_{1,\lambda},a_{2,1},\ldots,a_{2,\lambda},\ldots,a_{\lambda,\lambda})$, and for $i=1,\ldots,\lambda$, we call $a_{i,1},\ldots,a_{i,\lambda}$ a block of a. The desired output will be split in blocks using the same notation. Note that we can compute an Or with unbounded fan-in, $[x]_p \leftarrow \vee_{j=1}^{\lambda}[x_j]_p$, using Section 5.1, as this is a symmetric function.

Protocol $([b_1]_p,\ldots,[b_\ell]_p) \leftarrow \mathbf{PRE}_\vee([a_1]_p,\ldots,[a_\ell]_p)$ 1. For $i=1,\ldots,\lambda$, in parallel: $[x_i]_p = \vee_{j=1}^{\lambda}[a_{i,j}]_p$. 2. For $i=1,\ldots,\lambda$, in parallel: $[y_i]_p = \vee_{k=1}^{i}[x_k]_p$. 3. $[f_1]_p=[x_1]_p$. 4. For $i=2,\ldots,\lambda$, let $[f_i]_p=[y_i]_p-[y_{i-1}]_p$. 5. For $i=1,\ldots,\lambda$, $j=1,\ldots,\lambda$, in parallel: $[g_{i,j}]_p = \textsc{mult}([f_i]_p,[a_{i,j}]_p)$. 6. For $j=1,\ldots,\lambda$: $[c_j]_p=\sum_{i=1}^{\lambda}[g_{i,j}]_p$. 7. For $j=1,\ldots,\lambda$, in parallel: $[b_{\cdot,j}]_p = \vee_{k=1}^{j}[c_k]_p$. 8. For $i=1,\ldots,\lambda$, $j=1,\ldots,\lambda$, in parallel: $[s_{i,j}]_p = \textsc{mult}([f_i]_p,[b_{\cdot,j}]_p)$. 9. For $i=1,\ldots,\lambda$, $j=1,\ldots,\lambda$: $[b_{i,j}]_p \leftarrow [s_{i,j}]_p+[y_i]_p-[f_i]_p$.

The privacy follows from the fact that we only call private sub-protocols. As for the correctness, the variables have the following interpretation. We have that $x_i=1$ iff the the i'th block contains a 1. Therefore $y_i=1$ iff there is a 1 in one of the i first blocks, and $f_i=1$ iff the i'th block is the first block to contain a 1. Hence the sequence of f_i values has form $f=(0,\ldots,0,1,0,\ldots,0)$, and we let i_0 be the position of the single 1-bit. Now, for $i<i_0$, the i'th block of the output should be all 0's. For $i>i_0$, the i'th block of the output should be all 1's. Finally, the i_0'th block of the output should the prefix-or of the i_0'th input block. The block $c=(c_1,\ldots,c_\lambda)$ is formed by taking the "inner product" of f

and a and therefore, by the special form of f, equals the i_0'th block of a. The values $(b_{\cdot,1}, \ldots, b_{\cdot,\lambda})$ are the prefix-or bits of c. This means that the bits $s_{i,j}$ form an all-0 vector, except that the i_0'th block equals c. It now follows directly from the form of the $s_{i,j}$'s, f_i's and y_i' s that the output bits $b_{i,j}$ get the correct value in the final step.

The protocol uses 3λ invocations of the protocol for symmetric functions, in three rounds and on problems of size λ. This gives a complexity of 15 rounds and 18ℓ invocations. Besides this there are two rounds of ℓ multiplications each, giving a total complexity of 17 rounds and 20ℓ invocations.

5.3 Bitwise less-than

We now describe the protocol BIT-LT. Note that given sharings of two bits $[a]_p$ and $[b]_p$ we can compute their Xor in one round by first computing $[d]_p \leftarrow [a_i]_p - [b_i]_p$ and then computing $[e]_p \leftarrow \text{MULT}([d]_p, [d]_p)$. Below we write this as $[e]_p \leftarrow \text{XOR}([a]_p, [b]_p)$.

Protocol $[c]_p \leftarrow$ **BIT-LT**$([a]_\mathrm{B}, [b]_\mathrm{B})$

1. For $i = 0, \ldots, \ell - 1$: $[e_i]_p \leftarrow \text{XOR}([a_i]_p, [b_i]_p)$.
2. $([f_{\ell-1}]_p, \ldots, [f_0]_p) = \text{PRE}_\vee([e_{\ell-1}]_p, \ldots, [e_0]_p)$.
3. $[g_{\ell-1}]_p = [f_{\ell-1}]_p$.
4. For $i = 0, \ldots, \ell - 2$: $[g_i]_p \leftarrow [f_i]_p - [f_{i+1}]_p$.
5. For $i = 0, \ldots, \ell - 1$: $[h_i]_p \leftarrow \text{MULT}([g_i]_p, [b_i]_p)$.
6. $[h]_p \leftarrow \sum_{i=0}^{\ell-1} [h_i]_p$.
7. Output $[h]_p$.

Privacy follows from the fact that we only call private sub-protocols. As for the correctness, assume that $a \neq b$, and let i_0 denote the largest index i, where $a_i \neq b_i$. Then $a < b$ iff $b_{i_0} = 1$. Note that i_0 is the largest i for which $f_i = 1$, and thus $g_i = 1$ iff $i = i_0$. Therefore $h = b_{i_0}$. In the special case $a = b$, clearly $h = 0$, as it should be.

The protocol uses one invocation of $\text{PRE}_\vee$ on an instance of size ℓ, costing 17 rounds and 20ℓ invocations of MULT. Then there are two rounds more, each of ℓ invocations of MULT, giving a total of 19 rounds and 22ℓ invocations of MULT.

6 Bitwise Sum

We show how to add two bitwise-shared numbers in constant rounds. We first present a sub-protocol.

6.1 Generic prefix computations

Assume that we have some alphabet $\Sigma \subseteq \{0,1\}^n$ and bitwise-shared inputs $[a_1]_B, \dots, [a_\ell]_B$, where $a_i \in \Sigma$. That is, $[a_i]_B = [a_{i,1}]_p, \dots, [a_{i,n}]_p$ consists of n sharings of bits, and $(a_{i,1}, \dots, a_{i,n}) \in \Sigma$. Assume furthermore that an associative binary operator $\circ : \Sigma \times \Sigma \to \Sigma$ is given and that we want to compute sharings

$$([b_1]_B, \dots, [b_\ell]_B) = \text{PRE}_\circ([a_1]_B, \dots, [a_\ell]_B) \ ,$$

where $b_i = \circ_{j=1}^{i} a_j$. Assume that it is possible to securely compute a sharing $[b_\ell] = \circ_{j=1}^{\ell}[a_j]$ with complexity R rounds and $C(\ell)$ invocations of MULT. For short, we will refer to $\circ_{j=1}^{\ell}[a_j]$ as the "sum" of $a_1, \dots, a_\ell$. We assume for notational convenience that $\ell = 2^k$ for some k.

We use the method by Chandra, Fortune and Lipton [CFL83b]. For each $i = 1, \dots, k$ we will split the sequence $a_1, \dots, a_\ell$ into consecutive blocks of size 2^i items each. We let $b_{i,j}$ be the "sum" of the j'th such block, i.e., $b_{i,j} = \circ_{m=j\cdot 2^i+1}^{j\cdot 2^i+2^i} a_m$. There are $\ell - 1$ of the "sums" $b_{i,j}$, namely one of length $\ell = 2^k$, two of length 2^{k-1}, up to $\ell/2$ of length two. The complexity for computing all of them in parallel is thus R rounds and $\sum_{i=1}^{k} 2^i C(\ell \cdot 2^{-i})$ invocations of MULT.

It is easy to see that each of the ℓ values b_i can be computed as a "sum" of at most k of the $b_{i,j}$'s. Doing this in parallel for all b_i's costs another R rounds and at most $\ell C(k)$ invocations. Therefore the total complexity is upper bounded by $2R$ rounds and $\sum_{i=1}^{\log_2 \ell} 2^i C(\ell \cdot 2^{-i}) + \ell C(\log_2 \ell) \leq \log_2 \ell \cdot C(\ell) + \ell C(\log_2 \ell)$ invocations of MULT.

6.2 Bitwise sum

We now describe the protocol $[d]_B \leftarrow \text{BIT-ADD}([a]_B, [b]_B)$.

For $i = 1, \dots, \ell$, define the carry $c_i \in \{0,1\}$ by $c_i = 1$ iff $\sum_{j=0}^{i-1} 2^j(a_j + b_j) > 2^i$. It is straightforward to verify that given a bitwise sharing of the carries we can compute a bitwise sharing of the sum as follows.

Protocol $[d]_B \leftarrow$ BIT-ADD$([a]_B, [b]_B)$

1. $([c_1]_p, \dots, [c_\ell]_p) \leftarrow \text{CARRIES}([a]_B, [b]_B)$.
2. $[d_0]_p = [a_0]_p + [b_0]_p - 2[c_1]_p$.
3. $[d_\ell]_p = [c_\ell]_p$.
4. For $i = 1, \dots, \ell - 1$: $[d_i]_p = [a_i]_p + [b_i]_p + [c_i]_p - 2[c_{i+1}]_p$.
5. Output $[d]_B = ([d_0]_p, \dots, [d_\ell]_p)$.

Evidently, the complexities of this protocol are the same as those of subprotocol CARRIES as presented below. We therefore get 37 rounds and $55\ell \log_2 \ell$ invocations of MULT.

6.3 Computing the carry bits

In order to compute the carries, we use the well-known *carry set/propagate/kill* algorithm. Let $\Sigma = \{S, P, K\}$. The algorithm uses an operator $\circ : \Sigma \times \Sigma \to \Sigma$, defined by $x \circ S = S$ for all $x \in \Sigma$, $x \circ K = K$ for all $x \in \Sigma$, and $x \circ P = x$ for all $x \in \Sigma$. This is the carry-propagation operator, and it can be verified to be associative[3].

For two bitwise-represented numbers $a = (a_0, \ldots, a_{\ell-1})$ and $b = (b_0, \ldots, b_{\ell-1})$, for $i = 0, \ldots, \ell - 1$, let $e_i = S$ iff a carry is set at position i (i.e., $a_i + b_i = 2$); $e_i = P$ iff a carry would be propagated at position i (i.e. $a_i + b_i = 1$); and $e_i = K$ iff a carry would be killed at position i, (i.e. $a_i + b_i = 0$). It is straightforward to verify that $c_i = 1$ (the i'th carry bit is set) if and only if $e_0 \circ \cdots \circ e_{i-1} = S$.

We represent S, P, and K with bit vectors $(1, 0, 0), (0, 1, 0)$ and $(0, 0, 1) \in \{0, 1\}^3$. The values of the e_i's in this representation can be easily computed from the a_i's and b_i's as shown below. Hence, given a protocol for unbounded fan-in computation of the carry-propagation operator $\circ$ on this representation, we compute carries as follows.

Protocol $[c]_B \leftarrow$ **CARRIES**$([a]_B, [b]_B)$

1. For $i = 0, \ldots, \ell - 1$, in parallel: $[s_i]_p = \text{MULT}([a_i]_p, [b_i]_p)$.
2. For $i = 0, \ldots, \ell - 1$: $[p_i]_p = [a_i]_p + [b_i]_p - 2[s_i]_p$, $[k_i]_p = 1 - [s_i]_p - [p_i]_p$ and set $[e_i]_B = ([s_i]_p, [p_i]_p, [k_i]_p)$, i.e., interpret the sharings $[s_i]_p, [p_i]_p, [k_i]_p$ as a bit-wise sharing of a 3-bit string $e_i \in \Sigma$.
3. $([f_0]_B, \ldots, [f_{\ell-1}]_B) \leftarrow \text{PRE}_\circ([e_0]_B, \ldots, [e_{\ell-1}]_B)$.
4. For $i = 0, \ldots, \ell - 1$, set $([s_i]_p, [p_i]_p, [k_i]_p) = [f_i]_B$, i.e., each $[f_i]_B$ consists of shares of 3 bits which we now name s_i, p_i and k_i .
5. Output $[c]_B = ([s_0]_p, [s_1]_p, \ldots, [s_{\ell-1}]_p)$.

The privacy follows from only using private sub-protocols, and correctness follows readily from the above arguments.

Section 6.1 describes how to compute $\text{PRE}_\circ([e_0]_B, \ldots, [e_{\ell-1}]_B)$ assuming a protocol for computing the $\circ$-operator with unbounded fan-in. The next section shows how to do this unbounded fan-in with complexity 18 rounds and 27ℓ invocations of MULT. This and the analysis of the protocol from Section 6.1 shows that we can compute all $f_0, \ldots, f_{\ell-1}$ with complexity 36 rounds and $54\ell \log_2 \ell$ invocations. Besides this, the CARRIES protocol has only one round containing a total of ℓ invocations of MULT, giving a total complexity of 37 rounds and $55\ell \log_2 \ell$ invocations of MULT.

6.4 Unbounded fan-in carry propagation

We describe a protocol for computing $\circ_{i=1}^{\ell} e_i$, where we again represent e_i as (s_i, p_i, k_i). The protocol uses an unbounded fan-in And in Step 1 and a prefix-And in Step 2. These protocols are defined equivalently to a unbounded fan-in

[3] Note that this definition is changed from the standard one to be consistent with the fact that we write numbers with the least significant bit first

Or and prefix-Or, and implemented in the same complexity using DeMorgan's Rule.

Protocol $([a]_p, [b]_p, [c]_p) \leftarrow \circ_{i=1}^{\ell}([s_i]_p, [p_i]_p, [k_i]_p)$

1. $[b]_p \leftarrow \wedge_{i=1}^{\ell}[p_i]_p$.
2. $([q_\ell]_p, \ldots, [q_1]_p) \leftarrow \text{PRE}_\wedge([p_\ell]_p, \ldots, [p_1]_p)$.
3. $[c_\ell] = [k_\ell]$.
4. For $i = 1, \ldots, \ell - 1$, in parallel: $[c_i]_p \leftarrow [k_i] \wedge [q_{i+1}]_p$.
5. $[c]_p = \sum_{i=1}^{\ell}[c_i]_p$.
6. $[a]_p \leftarrow 1 - [b]_p - [c]_p$.

As for correctness, it should be clear that $b = 1$ (a propagate) iff $p_i = 1$ for $i = 1, \ldots, \ell$, making b correct. Furthermore, we have that $c = 1$ (a kill) iff there exists some i such that $k_i = 1$ and $p_{i+1} = 1, \ldots, p_\ell = 1$. I.e. $c = \vee_{i=1}^{\ell}(k_i \wedge q_{i+1})$. Since k_i and p_i are never 1 simultaneously it can be seen that at most one of the expressions $k_i \wedge q_{i+1}$ equals one. This implies that $c = \sum_{i=1}^{\ell}(k_i \wedge q_{i+1})$. By our representation it follows that $a = 1 - b - c$.

Since we can compute $[b]_p$ and $[c]_p$ in parallel, the overall complexity for an unbounded fan-in carry propagation can be verified to be 18 rounds and 27ℓ invocations of MULT.

7 Applications

In this section we mention some secure multi-party protocols for specific tasks that use our new constant-rounds protocol for computing shares of the bit decomposition as an atomic sub-protocol. All application protocols are unconditionally secure constant-rounds protocols. We want to stress that even though the number of invocations of the underlying multiplication protocol is always polynomial in $\ell = \lceil \log_2 p \rceil$ and the number of rounds is constant we did not put much effort in optimizing the running time and round complexity.

For the remaining part of this section let p be a prime, $p \in [2^{\ell-1}, 2^\ell]$.

7.1 Comparison and equality

In this subsection we look at the equality function $\stackrel{?}{=} : \mathbb{F}_p \to \mathbb{F}_p$, where $(x \stackrel{?}{=} y) \in \{0, 1\}$ and $(x \stackrel{?}{=} y) = 1$ iff $x = y$, and the comparison function $\stackrel{?}{<} : \mathbb{F}_p \times \mathbb{F}_p \to \mathbb{F}_p$, where $(x \stackrel{?}{<} y) \in \{0, 1\}$ and $(x \stackrel{?}{<} y) = 1$ iff $x < y$.

For equality, assume the shares $[x]_p, [y]_p$ are given and we want to compute shares $[x \stackrel{?}{=} y]_p$. Setting $z = x - y \in \mathbb{F}_p$ the problem clearly reduces to computing $[z \stackrel{?}{=} 0]_p$. The latter one can be done by first computing shares of the bits $[z]_B = [z_0]_p, \ldots, [z_{\ell-1}]_p$ and then $[z \stackrel{?}{=} 0]_p = \wedge_{i=0}^{\ell-1}[z_i]_p$, which can be computed in constant round using Section 5.1, as it is a symmetric function.

For the comparison function we are given shares $[x]_p, [y]_p$ and want to computes shares $[x \stackrel{?}{<} y]_p$. This can be done by first computing shares of the bits $[x]_B = [x_0]_p, \ldots, [x_{\ell-1}]_p$ and $[y]_B = [y_0]_p, \ldots, [y_{\ell-1}]_p$. Now shares of the comparison function can be computed using BIT-LT from Section 5.

7.2 Private exponentiation

The exponentiation function $\exp : \mathbb{F}_p \times \mathbb{Z}_p \to \mathbb{F}_p$ is given by $\exp(x, a) = (x^a \bmod p) \in \mathbb{F}_p$.

PUBLIC EXPONENT a. We first deal with the case where the exponent a is publicly known and the value x is shared, i.e., given $[x]_p$ and a we want to compute $[x^a]_p$. Assume there exists a protocol that outputs random shares $[r]_p$ of a random non-zero value $r \in \mathbb{F}_p^*$ together with shares of its a^{th} power $[r^a]_p$. We will show later how to implement such a protocol in constant rounds.

Assuming such a protocol exists, a protocol to securely compute the exponentiation function is straightforward (using the Bar-Ilan and Beaver [BB89] inversion trick): First the parties run the protocol to get shares of $[r]_p$ and $[r^a]_p$ for a random $r \in \mathbb{F}_p^*$. Then they compute $[xr]_p = \text{MULT}([x]_p, [r]_p)$, open $[xr]_p$ to get $xr \in \mathbb{F}_p$, and every player individually computes $y = (xr)^a = x^a r^a \in \mathbb{F}_p$. Now $[x^a]_p$ is obtained by computing $[x^a]_p = y[r^{-a}]_p$ where $[r^{-a}]_p = [(r^a)^{-1}]_p$ is obtained from $[r^a]_p$ using the Bar-Ilan and Beaver inversion protocol.

It is easy to see that this protocol is private as long as $x \neq 0$. We handle the case $x = 0$ as an "exception" using our protocol for evaluating the equality function from Section 7.1. The idea is to substitute x by $\tilde{x} = x + (x \stackrel{?}{=} 0)$. Note that this always assures $\tilde{x} \neq 0$. Then shares $[x^a]_p$ can be computed as

$$[x^a]_p = [\tilde{x}^a]_p - [(x \stackrel{?}{=} 0)]_p \ .$$

We note that this "exception trick" may also be used in some other places (like in the inversion protocol) to handle special shared inputs that may lead to information leakage. Any protocol that initially leaks information for m different shared input values can now be updated to a protocol providing perfect privacy by (roughly) the cost of additional m (parallel) executions of the equality protocol.

It remains to provide the protocol that, given a public a, outputs shares $[r]_p$ together with shares of its a^{th} power $[r^a]_p$ for a random non-zero value $r \in \mathbb{F}_p^*$. In the honest-but-curious model this is simply done by letting each player j locally select a random non-zero value $r_j \in \mathbb{F}_p^*$ together with its a^{th} power $r_j^a \in \mathbb{F}_p^*$. Each value r_j is shared among the players. Now define r as the product of all r_j such that r^a also equals to the product of all r_j^a. Shares of both products $r = \prod_{j=1}^n r_j$ and $r^a = \prod_{j=1}^n r_j^a$ can be computed using the unbounded fan-in multiplication protocol, MULT*. We now show how to make this protocol robust against active adversaries using a "cut-and-choose" technique: In addition to $[r_i]_p$, $[r_i^a]_p$, user i generates random sharings $[s_i]_p$, $[s_i^a]_p$. The players jointly form a random bit b. Then they compute and open (s_i, s_i^a) or $(s_i r_i, s_i^a r_i^a)$, according to the value of b and verify that the first number is non-zero and that the second number is the first raised to the public a. This can be repeated in parallel an appropriate number of times.

SHARED EXPONENT a. Now we consider the case where the exponent a is also given as a share, i.e., the users are given $[x]_p$ and $[a]_p$ and want to compute $[x^a]_p$. We show how this case can be reduced to the previous one.

First run the bit decomposition protocol to obtain shares of the bits $[a]_B = [a_0]_p, \dots [a_{\ell-1}]_p$ of the exponent a such that $a = \sum_{i=0}^{\ell-1} 2^i a_i$ with $a_i \in \{0, 1\}$. Then, using unbounded fan-in multiplication, shares $[x^a]_p$ may now be obtained via the equation

$$x^a = x^{\sum_{i=0}^{\ell-1} 2^i a_i} = \prod_{i=0}^{\ell-1} x^{2^i a_i} = \prod_{i=0}^{\ell-1} (a_i x^{2^i} + 1 - a_i) \in \mathbb{F}_p \tag{1}$$

where the shares $[x^{(2^i)}]_p$ can be computed (in parallel for $1 \le i \le l-1$) with the exponentiation protocol above. If x is non-zero the protocol MULT* can be used for the unbounded fan-in multiplication, and the "exception trick" can be used use to deal with the case were x can be zero.

7.3 Modulo reduction

Let $m \in [2, p-1]$ be a public integer. In this subsection we look at the "modulo m" function, $\mathrm{mod}_m : \mathbb{F}_p \to \mathbb{F}_p, x \mapsto x \bmod m \in \{0, \dots, m-1\}$, where $x \in \mathbb{F}_p$ is considered a residue $x \in \{0, 1, \dots, p-1\}$. We show how to privately compute the modulo m function in constant rounds, i.e., the players are given $[x]_p$ and want to compute $[\mathrm{mod}_m(x)]_p$ for a public integer m.

The players first compute shares $[x]_B = [x_0]_p, \dots, [x_{\ell-1}]_p$ of the bits of x, i.e. $x = \sum_{i=0}^{\ell-1} x_i 2^i$. Note that if m is a power of 2, i.e., $m = 2^a$, then $[x \bmod m]_p$ can be computed using the equation $x \bmod 2^a = \sum_{i=0}^{a-1} 2^i x_i$. Otherwise define $y = \sum_{i=0}^{\ell-1} x_i (2^i \bmod m) \in \mathbb{Z}$. Then clearly $x \bmod m = (\sum_{i=0}^{\ell-1} x_i (2^i \bmod m)) \bmod m = y - tm$ for some integer t in the range $t \in [0, \ell-1]$. Define $y^{(i)} = y - im \in \mathbb{Z}$. The shares $[y^{(i)}]_B$ can be computed in parallel for all $i \in [0, \ell-1]$ using the bitwise sum protocol from Section 6. The value $x \bmod m$ is now the unique $y^{(t)}$ such that $0 \le y^{(t)} < m$. Shares of such an $[x \bmod m]_B = [y^{(t)}]_B$ can be found using (ℓ parallel applications of) the comparison function and one conversion to shares of $[x \bmod m]_p$.

7.4 Private modulo reduction

The players are given shares $[x]_p$ and shares $[m]_p$ of an integer m of known bit-size $\ell_0 \ll \ell$. The problem is to compute shares $[x \bmod m]_p$. There already exists an efficient protocol to approximate $[x \bmod m]_p$ due to [ACS02] but it does not run in a constant number of rounds. In this section we note that combining the techniques of this paper with the results from [ACS02] and [KLM05] (the latter one approximates the fractional part of $1/m$ by a Taylor polynomial), we get an efficient constant-rounds protocol to compute an approximation of $[x \bmod m]_p = [x - \lfloor \frac{x}{m} \rfloor \cdot m]_p$. Shares of the exact value of $x \bmod m$ may then be obtained by running an appropriate number of comparison protocols to make sure that result lies in the interval $[0, m-1]$. With the results from Section 7.2 this enables us to build a constant-rounds protocol that privately computes shares $[x^a \bmod m]_p$, where all three inputs, x, a, and m are given as shares (together with the bit-size ℓ_0 of m). Here we only consider the case of prime m. First compute shares $[x \bmod m]_p$ and $[a \bmod m-1]_p$. The prime p has to be large enough (of bit-size $\ell > \ell_0^2$) such that in Eqn. (1) no wrap-around modulo p appears: after computing $[x^a]_p$, the modulo reduction protocol is used again to compute shares $[x^a \bmod m]_p$.

7.5 Unrestricted conversion to additive shares over the integers

Informally, additive shares over the integers are $(n-1)$-out-of-n shares where each party P_j holds a random share $x_j \in [-2^\rho A, 2^\rho A]$ (where ρ is some security parameter). The secret x is then defined as $x = \sum_{j=1}^{n} x_j \in [-A, A]$ over the integers. We use $[x]_\mathbb{Z}$ to denote additive shares over the integers. See [ACS02] for a formal definition of additive shares and for applications.

Let p be a prime. We want to note that we now can give a constant-rounds protocol that converts shares $[x]_p$ to shares $[x]_\mathbb{Z}$. Prior to our work, by a result from [ACS02], this could only be done in constant rounds when x is guaranteed to be considerably smaller than the modulus p. As the protocol in [ACS02] our protocol is only passively secure.

First compute shares $[x]_B = [x_1]_p, \ldots, [x_{\ell-1}]_p$ of the bits of x. Then (in parallel) convert the shares $[x_j]_p$ to shares $[x_j]_\mathbb{Z}$ over the integers using the technique from [ACS02] (note that this can be carried out since the shares of the bits are now "small enough" compared to the modulus p). Finally, the integer shares of x can be computed without interaction via $[x]_\mathbb{Z} = \sum_{i=0}^{\ell-1} 2^i [x_i]_\mathbb{Z}$.

Acknowledgments

The authors would like to thank the anonymous referees for many useful suggestions, which helped improve the presentation considerably.

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, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432, Santa Barbara, CA, USA, August 18–22, 2002. Springer-Verlag, Berlin, Germany.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proc. ACM PODC'89*, pages 201–209, 1989.

[Bea00] Donald Beaver. Minimal latency secure function evaluation. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 335–350, Bruges, Belgium, May 14–18, 2000. Springer-Verlag, Berlin, Germany.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, 14–17 October 2001. IEEE.

[CD01] Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In J. Kilian, editor, *Advances in Cryptology - Crypto 2001*, pages 119–136, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 2139.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multiparty computation from any linear secret-sharing scheme. In Bart Preneel, editor, *Advances in Cryptology - EuroCrypt 2000*, pages 316–334, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1807.

[CFL83a] Ashok K. Chandra, Steven Fortune, and Richard J. Lipton. Lower bounds for constant depth circuits for prefix problems. In *Proceedings of ICALP 1983*, pages 109–117. Springer-Verlag, 1983. Lecture Notes in Computer Science Volume 154.

[CFL83b] Ashok K. Chandra, Steven Fortune, and Richard J. Lipton. Unbounded fan-in circuits and associative functions. In *15th Annual ACM Symposium on Theory of Computing*, pages 52–60, Boston, Massachusetts, USA, April 25–27, 1983. ACM Press.

[DN03] Ivan Damgård and Jesper B. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In D. Boneh, editor, *Advances in Cryptology - Crypto 2003*, Berlin, 2003. Springer-Verlag. Lecture Notes in Computer Science.

[FKN94] Uri Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation. In *Proc. ACM STOC*, pages 554–563, 1994.

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *Proc. 5th Israel Symposium on Theoretical Comp. Sc. ISTCS*, pages 174–183, 1997.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science*, pages 294–304, Las Vegas, Nevada, USA, November 12–14, 2000. IEEE Computer Society Press.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *Proceedings of ICALP 2002*, pages 244–256, Berlin, 2002. Springer-Verlag. Lecture Notes in Computer Science Volume 2380.

[KLM05] Eike Kiltz, Gregor Leander, and John Malone-Lee. Secure computation of the mean and related statistics. In *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 283–302, Cambridge, MA, USA, February 10–12, 2005. Springer-Verlag, Berlin, Germany.

[NWKo] Martin Nowak, Georg Woltman, Scott Kurowski, and others. Mersenne.org project discovers new largest known prime number $2^{25,964,951} - 1$. Press release.