# How to Split a Shared Number into Bits in Constant Round and Unconditionally Secure

Ivan Damgård  Matthias Fitzi[*]  Jesper Buus Nielsen[†]
Tomas Toft[‡]

University of Aarhus
Department of Computer Science
IT-parken, Aabogade 34
DK-8200 Aarhus N, Denmark
{ivan|fitzi|buus|tomas}@daimi.au.dk

May 13, 2005

**Abstract**

We show that if a set of players hold shares of a value $a \in Z_p$ for some prime $p$ (where the set of shares is written $[a]_p$), it is possible to compute, in constant round and with unconditionally security, sharings of the bits of $a$, i.e. compute sharings $[a_0]_p, \ldots, [a_l]_p$ such that $l = \lceil \log_2(p) \rceil$, $a_0, \ldots, a_l \in \{0, 1\}$ and $a = \sum_{i=0}^{l-1} a_i 2^i$.

This result immediately implies solutions to other long-standing open problems, such as constant-round and unconditionally secure protocols for comparing shared numbers and deciding whether a shared number is zero.

If one can set $p$ optimally, the protocol uses $134 \cdot l^{2/15}\kappa + 54 \cdot l^{4/3}$ invocations of the multiplication protocol for the underlying secret sharing scheme, where $\kappa$ is a security parameter. In the worst case, where $p$ can be arbitrary, the number of invocations of the multiplication protocol is $35 \cdot l^{2/15}\kappa^2 + 31 \cdot l^{4/3}\kappa$.

Our protocol is secure against active adversaries and work for any linear secret sharing scheme with a multiplication protocol.

## 1 Introduction

Assume that $n$ parties have shared values $a_1, \ldots, a_l$ from some field $\mathbb{F}$ using some linear secret sharing scheme, such as Shamir's. Let $f : \mathbb{F}^l \to \mathbb{F}^m$. By 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 $(b_1, \ldots, b_m) = f(a_1, \ldots, a_l)$, while leaking no information on the values $a_1, \ldots, a_l$ or $b_1, \ldots, b_m$. The question *which functions can be computed with unconditional security on sharings, using a constant round protocol* is a long standing open problem.

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

[†]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.

However, a number of functions are known to have unconditionally secure, constant round 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).

The only non-trivial part needed in these solutions is unbounded fan-in multiplication $[\prod_{i=1}^{l} a_i] = \prod_{i=1}^{l} [a_i]$. If all $a_i$ are guaranteed to be non-zero this can be done in constant round using the techniques by Bar-Ilan and Beaver [BB89], which can also handle the case of general $a_i$ when the size of $\mathbb{F}$ is polynomial. When $\mathbb{F}$ is large and the $a_i$ can be arbitrary a technique by Cramer and Damgård is needed [CD98].

However, a number of functions do not have small constant-depth arithmetic solutions. Consider e.g. the function $\overset{?}{<}: \mathbb{F}_p \times \mathbb{F}_p \to \mathbb{F}_p$, where $(a \overset{?}{<} b) \in \{0, 1\}$ and $(a \overset{?}{<} b) = 1$ 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 this problem (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 round, by evaluating a constant-depth arithmetic circuit with $O(2^d g)$ gates.

If in particular the input $a$ is delivered as bitwise sharings $[a_0]_p, \ldots, [a_{l-1}]_p$ and $b = f(a)$ can be computed using a BBC with depth $d$ and $g$ gates, then sharings of the bits of $b = f(a)$ can be computed with complexity[1] $O(2^d m)$, unconditionally secure in constant round. 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 $\overset{?}{<}: (\mathbb{F}_p)^l \times (\mathbb{F}_p)^l \to \mathbb{F}_p, ((a_0, \ldots, a_{l-1}), (b_0, \ldots, b_{l-1})) \mapsto \sum_{i=0}^{l-1} a_i 2^i \overset{?}{<} \sum_{i=0}^{l-1} b_i 2^i$ unconditionally securely in constant round.

So, different representations of the inputs allow different classes of functions to be computed unconditionally securely in constant round — at least with our 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 [CD98] for linear algebra in constant round could not handle handle fields with large characteristic without assuming that the input was shared bit-wise 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.

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 $[b]_p$ the parties can compute a sharing $[a + b \bmod p]_p$ without interaction) and 2) there exists a 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 protocol will be actively secure too. The assumption on multiplication implies that the adversary structure must be $Q2$ which in the standard threshold case means we need honest majority.

Changing representations from bits to modulo $p$ is trivial. Assume that $0 \le a < p$ and that we are given sharings $[a_0]_p, \ldots, [a_{l-1}]_p$ of the bits of $a$. By computing $[a]_p = \sum_{i=0}^{l-1} 2^i [a_i]_p$ we

[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.

get a sharing of $a$ in constant round. Computing the other direction has however been a long standing open problem. We will show how to compute $[a_0]_p, \ldots, [a_{l-1}]_p$ from $[a]_p$ unconditionally secure in constant round. For $n$ parties and security parameter $\kappa$ and $p < 2^\kappa$, the complexity is bounded by $35 \cdot l^{2/15}\kappa^2 + 31 \cdot l^{4/3}\kappa$ invocations of the multiplication protocol, and can be as small as $134 \cdot l^{2/15}\kappa + 54 \cdot l^{4/3}$ if $p$ is close to a power of two, e.g. a Mersenne prime $2^k - 1$.

These results immediately imply that we can also in constant round compute a single shared bit containing the result of a comparison between two shared numbers, or containing the result of asking whether a shared number is zero. This last function was exactly what was missing in [CD98] in order to handle large characteristic fields.

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 $\kappa$, 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 $2^{-\kappa}$ — i.e. with probability $1 - 2^{-\kappa}$ 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 private protocol*, which however with probability $2^{-\kappa}$ terminates with some abort symbol $\perp$.[2]

## 1.1 Related Work

There has been a considerable amount of previous work on unconditionally secure constant-round multiparty computation with honest majority, see for instance [BB89], [FKN94], [CD98], [IK00] and [IK02]. As mentioned, this work has shown that some functions can indeed be computed in constant round with unconditional security, but this has been limited to restricted classes of functions, such as $NC1$ or non-deterministic log-space.

In concurrent independent work, Eike Kiltz sets out to solve essentially the same set of problems we look at here [Kil05], using a quite different technique. We have learned, however, from personal communication with Kiltz that in the current version, a central part of the protocol does not work. He expects to be able to solve the problem in a forthcoming version.

## 1.2 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 then list some known results and simple observations. In Section 5 we give a protocol for the so-called sub-sequence-product problem. In Section 6 we use Section 5 to give a protocol for all so-called preprocessed carry functions, and in Section 7 we then give protocols for adding bitwise shared numbers and comparing bitwise shared number within the promised complexities.

[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 probability $2^{-\kappa}$. 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 probability $2^{-\kappa}$. Finally, to get a perfect protocol rerun the protocol when it aborts. This gives a completely perfect protocol. It however only runs in expected constant round, as the protocol is run $c$ times with probability $(2^{-\kappa})^{c-1}$.

## 2 Preliminaries

In this section with introduce some notation.

We assume that $n$ parties are connected by perfectly secure channels in some network. We assume that the network is synchronous.

We use $\mathbb{F}$ to denote a finite field, and we let $f = \lceil \log_2(|\mathbb{F}|) \rceil$. By $[a]_\mathbb{F}$ we denote a secret sharing of $a \in \mathbb{F}$ over $\mathbb{F}$.

We assume that the secret sharing scheme allows to compute a sharing $[a + b]_\mathbb{F}$ from $[a]_\mathbb{F}$ and $[b]_\mathbb{F}$ without interaction, and that it allows to compute $[ab]_\mathbb{F}$ from $a \in \mathbb{F}$ and $[b]_\mathbb{F}$ without interaction. We also assume that the secret sharing scheme allows to compute a sharing $[ab]_\mathbb{F}$ from $[a]_\mathbb{F}$ and $[b]_\mathbb{F}$ in constant round.

If our protocols should 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 $\Gamma$ of subset $C \subset [n]$ which the adversary might corrupt; It is $Q2$ if it holds for all $C \in \Gamma$ that $[n] \setminus C \not\in \Gamma$.

We assume that all parties have access to uniformly random coins $c \in_R \mathbb{Z}_n$ for any integer $n$. This is not to disable perfect security because of the simple fact that no finite computation can sample a uniformly random $c \in \mathbb{Z}_n$ given only uniformly random coins $c' \in \mathbb{Z}_2$, unless $n$ is a power of 2. We will e.g. need to sample uniformly random numbers modulo a large prime $n$.

## 3 Bit-Decomposition

Let $p$ be a prime $p \in [2^{l-1}, 2^l]$. We look at the bit-decomposition function $f : \mathbb{F}_p \to (\mathbb{F}_p)^l, a \mapsto (a_0, \ldots, a_{l-1})$ given by $a_0, \ldots, a_{l-1} \in \{0, 1\} \subseteq \mathbb{F}_p$ and $a = \sum_{i=0}^{l-1} a_i 2^i$ when $a \in \mathbb{F}_p$ is considered a residue $a \in \{0, 1, \ldots, p - 1\}$.

Assume first that the parties are able to securely generate a random solved instance

$$[b]_p, [b_0]_p, \ldots, [b_{l-1}]_p \ ,$$

where $b$ is a uniformly random $b \in \mathbb{F}_p$, $b_0, \ldots, b_{l-1} \in \{0, 1\}$ and $b = \sum_{i=0}^{l-1} b_i 2^i$.

Below we use $[x]_\mathrm{B} = [x_0]_p, \ldots, [x_{l-1}]_p$ to denote a bitwise sharing of an integer $x$, and we use $[z]_\mathrm{B} = [x]_\mathrm{B} + [y]_\mathrm{B}$ to denote computing a bitwise sharing of $x + y$ from the bitwise sharings, $[x]_\mathrm{B}$ and $[y]_\mathrm{B}$, of integers $x$ and $y$. Finally we use $[x \overset{?}{<} y]_p$ to denote computing a sharing of the bit $(x \overset{?}{<} y) \in \{0, 1\}$, where $(x \overset{?}{<} y) = 1$ iff $x < y$, starting from the bitwise sharings, $[x]_\mathrm{B}$ and $[y]_\mathrm{B}$, of integers $x$ and $y$. It is shown in Section 7 how to add and compare bitwise sharings unconditionally secure in constant round within the complexity $33l^{17/15}\kappa/f + 18l^{4/3}$ when $x, y \in \{0, \ldots, 2^l - 1\}$ and $\kappa$ is the security parameter, $f = \log_2(|\mathbb{F}_p|)$ and the complexity is measured in the number of invocations of the multiplication protocol.

Given a random solved instance $[b]_p$ and $[b]_\mathrm{B}$, the protocol proceeds as follows. First the parties compute

$$[a - b]_p = [a]_p - [b]_p$$

and reveal

$$c = a - b \bmod p \ .$$

This leaks no information as $b$ is uniformly random.

Let $c_0, \ldots, c_{l-1} \in \{0, 1\}$ be given by $c = \sum_{i=0}^{l-1} c_i 2^i$. Then using Section 7 the parties compute a bitwise sharing

$$[d]_\mathrm{B} = [d_0]_p, \ldots, [d_l]_p \ ,$$

of $d = b + c \in \{0, \ldots, 2^{l+1} - 1\}$. Clearly $d = a + qp$, where $q \in \{0, 1\}$. Using Section 7 the parties compute a sharing

$$[q]_p = [d \overset{?}{<} p]_p \ .$$

Then the parties compute a bitwise sharing $[g]_{\mathrm{B}}$ of $(-qp) \bmod 2^l$ as follows. Define $f_0, \ldots, f_{l-1} \in \{0, 1\}$ by $2^l - p = \sum_{i=0}^{l} f_i 2^i$. Then for $i = 0, \ldots, l$, compute

$$[g_i]_p = f_i [q]_p \ .$$

The parties now have the followings bitwise sharings

$$[d]_{\mathrm{B}} = [a + qp]_{\mathrm{B}}$$

$$[g]_{\mathrm{B}} = [(2^l - qp) \bmod 2^l]_{\mathrm{B}} \ .$$

Using again Section 7 they compute

$$[a']_{\mathrm{B}} = [d]_{\mathrm{B}} + [g]_{\mathrm{B}} \ .$$

As $q \in \{0, 1\}$, we have that $a' = a + qp + (2^l - qp \bmod 2^l) = a + q2^l$. So, if we only compute the $l$ least significant bits of the sum (just ignore the sharing of the most significant bit), then the term $q2^l$ will disappear and we will have that $a' = a$, as desired.

The complexity for one addition or one comparison is upper bounded by $33 l^{17/15} \kappa / f + 18 l^{4/3}$. This gives a total complexity of $99 l^{17/15} \kappa / f + 54 l^{4/3}$. Since we here have that $l = f$, we can write the complexity as

$$99 f^{2/15} \kappa + 54 f^{4/3} \ .$$

## 3.1 Generating Random Solved Instances

This whole thing hinged on our ability to generate a random solved instance

$$[b \in_R \mathbb{F}_p]_p, [b]_{\mathrm{B}} = [b_0]_p, \ldots, [b_{l-1}]_p \ ,$$

where $b_0, \ldots, b_{l-1} \in \{0, 1\}$ and $b = \sum_{i=0}^{l-1} b_i 2^i$. This is done as follows.

First the parties generate

$$[b_0 \in_R \{0, 1\}]_p, \ldots, [b_{l-1} \in_R \{0, 1\}]_p \ ,$$

i.e. sharings of $l$ uniformly random bits. We show in Section 4 how to do this within the complexity $l(2\kappa/f) = 2\kappa$. Then using Section 7 the parties compute and reveal

$$[\sum_{i=0}^{l-1} 2^i b_i \overset{?}{<} p]_p \ .$$

If $\sum_{i=0}^{l-1} 2^i b_i < p$, then they compute

$$[b]_p = \sum_{i=0}^{l-1} 2^i [b_i]_p \ .$$

The complexity of this is $33 f^{2/15} \kappa + 18 f^{4/3}$. If $\sum_{i=0}^{l-1} 2^i b_i \geq p$, then the protocol aborts.

This clearly yields a uniformly random $b \in \mathbb{F}_p$ when the protocol does not abort.

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 $p = 2^l - 1$ for some $l > \kappa$. In that case the probability that $b \geq p$ is less than $2^{-\kappa}$. Though the distance between the Mersenne primes soon becomes large, this would work for small values of $l$. At the time of writing $p = 2^{24036583} - 1$ is the largest $p$ for which we know this works.

In the worst-case, where we have no control over $p$, our only guarantee is that $p \in [2^{l-1}, 2^l]$ for some $l$. In that case the probability that $b \leq p$ when $b \in_R \mathbb{Z}_{2^l}$ can be as large as $1/2$. This is dealt with by generating $\kappa$ candidates and adopting one of the successful ones.

The total complexity for one attempt can be bounded by $33f^{2/15}\kappa + 18f^{4/3} + 2\kappa < 35f^{2/15}\kappa + 18f^{4/3}$. The reduction from a given instance to a random solved instance had complexity $99f^{2/15}\kappa + 54f^{4/3}$. This means that if $p$ and $f$ have been set such that the abort-probability is $2^{-\kappa}$, then the total complexity of a bit decomposition can be bounded by

$$134f^{2/15}\kappa + 54f^{4/3} \ .$$

If $\kappa$ attempts are needed, then the complexity is bounded by $35f^{2/15}\kappa^2 + (99f^{2/15} + 18f^{4/3})\kappa +$ $+54f^{4/3}$. For $f, \kappa \geq 10$, this is bounded by

$$35f^{2/15}\kappa^2 + 31f^{4/3}\kappa \ .$$

## 4 Some Simple Observations

In this section we list some known techniques and simple observations.

**Linear Functions.** We assumed that it is possible to compute additions without any communication. This means that given $c_0, c_1, \ldots, c_l \in \mathbb{F}$ it is possible to compute $[c_0 + \sum_{i=1}^{l} c_i a_i]$ from $[a_1], \ldots, [a_l]$ by the parties doing local computations. We write this computation as $c_0 + \sum_{i=1}^{l} c_i [a_i]$.

**Multiplication.** We assume that it is possible to compute multiplications perfectly secure in constant round. We write $[c] = [ab] = [a][b]$. We will measure complexity by invocations of the multiplication protocol.

**Random Elements** The parties can share a uniformly random, unknown field element. We write $[a \in_R \mathbb{F}]_\mathbb{F}$. This is done by letting each party $P_i$ deal a sharing $[a_i \in_R \mathbb{F}]_\mathbb{F}$, and letting $[a]_\mathbb{F} = \sum_{i=1}^{n} [a_i]_\mathbb{F}$. The communication complexity of this is given by $n$ dealings, which we assume is upper bounded by the communication 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 $H = [n] \setminus C$ for some $C \in \Gamma$. Since $\Gamma$ is Q2 it follows that $H \notin \Gamma$. So, at least one honest party will contribute to the sum, which is sufficient to argue privacy.

**Random Bits.** It is possible to efficiently generate a sharing $[a]_\mathbb{F}$ of a uniformly random $a \in \{0, 1\} \subseteq \mathbb{F}$ unconditionally secure in constant round. Here we treat the case where $\mathbb{F}$ does not have characteristic 2. Since we will later restrict our study to $\mathbb{F} = \mathbb{Z}_p$ for an odd prime $p$, this is sufficient. If $\mathbb{F}$ has characteristic 2, a slightly different technique is needed.

First some notation. Let $\mathbb{F}^*$ be the set of non-zero elements of $\mathbb{F}$ and let $Q(\mathbb{F}) \subset \mathbb{F}^*$ be the subset of squares. For $a \in Q(\mathbb{F})$, let $SQRT(a) = \{b \in \mathbb{F}^* | b^2 = a\}$. For each $a \in Q(\mathbb{F})$ we have that $|SQRT(a)| = 2$. Impose an arbitrary ordering $<$ of the elements in $\mathbb{F}$, e.g. the lexicographical ordering on the bit-string representation of the elements. Define a map $\sqrt{\ }$ : $Q(\mathbb{F}) \to \mathbb{F}$ by letting $\sqrt{a} = b$ where $b$ is the smaller element of $SQRT(a)$. Notice that given any element $b \in SQRT(a)$ we can compute $\sqrt{a}$ as the smaller element of $b$ and $-b$.

Extend the map by $\sqrt{0} = 0$ and let $S : \mathbb{F} \to \mathbb{F}$ be given by $S(0) = 0$, $S(x) = 1$ if $x \in \mathbb{F}^*$ and $x = \sqrt{x^2}$, and $S(x) = -1$ if $x \in \mathbb{F}^*$ and $x \neq \sqrt{x^2}$. Notice that it holds for all $x \in \mathbb{F}$ that $x = S(x)\sqrt{x^2}$.

It is straight-forward to verify that if $a \in_R \mathbb{F}^*$ is a uniformly random non-zero element, then $S(a)$ is uniformly random in $\{1, -1\}$. Furthermore, $S(a) = a(\sqrt{a^2})^{-1}$.

This suggests the following protocol. First the parties compute $[a \in_R \mathbb{F}]_\mathbb{F}$. Then the parties compute $[a^2]_\mathbb{F} = [a]_\mathbb{F}[a]_\mathbb{F}$ and reveal $a^2$. Then the parties compute $b = \sqrt{a^2}$. If $b = 0$, then abort, otherwise compute $[c]_\mathbb{F} = b^{-1}[a]_\mathbb{F} = [S(a)]_\mathbb{F}$ and then compute $[d]_\mathbb{F} = 2^{-1}([c]_\mathbb{F} + 1)$.

When the protocol does not abort, this clearly yields a uniformly random $d$. Furthermore, no information is leaked on $S(a)$, so no information is leaked on $d$.

If $b = 0$, then the protocol aborts. This happens with probability $2^{-f}$, where $f = \log_2(|\mathbb{F}|)$. To make the abort probability $2^{-\kappa}$ one generates $\kappa/f$ candidates and adopts the first one for which $b \neq 0$. If no such candidate arises the protocol aborts.

The complexity of generating $[a \in_R \mathbb{F}]_\mathbb{F}$ is bounded by the complexity of one multiplication. Then one multiplication is needed to compute $[a^2]_\mathbb{F}$. The rest is for free. The total complexity with abort probability $2^{-\kappa}$ is thus $2\kappa/f$.

**Random invertible elements.** The parties can share a uniformly random, unknown, invertible field element using [BB89]. We write $[a \in_R \mathbb{F}^*]_\mathbb{F}$.

This is done by first generating two elements $[b \in_R \mathbb{F}]_\mathbb{F}$ and $[c \in_R \mathbb{F}]_\mathbb{F}$. Then the parties compute and reveal $[d]_\mathbb{F} = [b]_\mathbb{F}[c]_\mathbb{F}$. If $d \in \mathbb{F}^*$, then $(b, c)$ is a uniformly random element from $\mathbb{F}^* \times \mathbb{F}^*$ for which $bc = d$, and thus $b$ is a uniformly random element in $\mathbb{F}^*$ independent of $d$. Therefore we can set $[a]_\mathbb{F} = [b]_\mathbb{F}$.

If $d \notin \mathbb{F}^*$, then the algorithm aborts. This happens with probability less than $2/|\mathbb{F}| = 2^{-f+1}$. The abort-probability can be forced down to $2^{-\kappa}$ by running $(\kappa + 1)/f$ attempts in parallel and using the first $[b_i]_\mathbb{F}$ for which $d_i \in \mathbb{F}^*$. The complexity in multiplications is $3(\kappa + 1)/f$. For simplicity we will use the estimate $3\kappa/f$ in the following.

**Inverting.** It is possible to invert an invertible element using [BB89]. Assume that we have input $[a \in \mathbb{F}^*]_\mathbb{F}$. First generate $[b \in_R \mathbb{F}^*]_\mathbb{F}$ and compute and reveal $[c]_\mathbb{F} = [a]_\mathbb{F}[b]_\mathbb{F}$. Then compute $[a^{-1}]_\mathbb{F} = c^{-1}[b]_\mathbb{F}$. The complexity is $3\kappa/f + 1$.

**Unbounded Fan-In Multiplication.** Using the technique from [BB89] it is possible to do unbounded fan-in multiplication in constant round.

Assume first that we have inputs $[a_1]_\mathbb{F}, \ldots, [a_l]_\mathbb{F}$, where $a_i \in \mathbb{F}^*$. For $1 \leq i_0 \leq i_1 \leq l$, let $a_{i_0,i_1} = \prod_{i=i_0}^{i_1} a_i$. We are interested in computing $a_{1,l}$, and the method allows to compute any other $a_{i_0,i_1}$ at the cost of two extra multiplications (we use $A$ to denote the number of $a_{i_0,i_1}$ which we want to compute).

First generate $[b_0 \in_R \mathbb{F}^*]_\mathbb{F}, [b_1 \in_R \mathbb{F}^*]_\mathbb{F}, \ldots, [b_l \in_R \mathbb{F}^*]_\mathbb{F}$, and for $i = 1, \ldots, l$ compute $[b_i^{-1}]_\mathbb{F}$ and then compute and reveal $[c_i]_\mathbb{F} = [b_{i-1}]_\mathbb{F}[a_i]_\mathbb{F}[b_i^{-1}]_\mathbb{F}$. Now we have that $d_{i_0,i_1} = \prod_{i=i_0}^{i_1} d_i = b_{i_0-1}(\prod_{i=i_0}^{i_1} a_i)b_{i_1}^{-1} = b_{i_0-1}a_{i_0,i_1}b_{i_1}^{-1}$, so we can compute $[a_{i_0,i_1}]_\mathbb{F} = [b_{i_0-1}^{-1}]_\mathbb{F} d_{i_0,i_1} [b_{i_1}]_\mathbb{F}$.

The complexity without the multiplication to compute $[a_{i_0,i_1}]_{\mathbb{F}} = [b_{i_0-1}^{-1}]_{\mathbb{F}} d_{i_0,i_1} [b_{i_1}]_{\mathbb{F}}$ is $(l+1)((3\kappa/f) + (3\kappa/f+1)) + 2l = l(6\kappa/f+3) + 6\kappa/f + 1$. The total complexity for computing $A$ of the values $[a_{i_0,i_1}]_{\mathbb{F}}$ is thus $l(6\kappa/f+3)+6\kappa/f+A+1$, which we can bound by $7l\kappa/f+3l+A$, except for very small parameter values.

Assume now that we have inputs $[a_1]_{\mathbb{F}}, \ldots, [a_l]_{\mathbb{F}}$, where $a_i \in \mathbb{F}$ (i.e. the elements are no longer guarantee to be invertible). We use a technique from [CD98]. Assume that we know distinct elements $b_0, \ldots, b_l$ such that $\{b_0, \ldots, b_l\} \cap \{a_1, \ldots, a_l\} = \emptyset$.[3]

For $i = 1, \ldots, l$ define the polynomial $f_i(X) = a_i - X$. Define the polynomials $f_{i_0,i_1}(X) = \prod_{i=i_0}^{i_1} f_i(X)$. Notice that $f_{i_0,i_1}(0) = \prod_{i=i_0}^{i_1} a_i = a_{i_0,i_1}$.

We proceed to compute the $A$ desired $[f_{i_0,i_1}(0)] = [a_{i_0,i_1}]$. For $i = 1, \ldots, l$ and $j = 0, \ldots, l$, compute $[f_i(b_j)] = [a_i] - b_j$. Notice that $f_i(b_j) \neq 0$. So, use an unbounded fan-in multiplication of invertible elements to compute $[f_{i_0,i_1}(b_j)] = \prod_{i=i_0}^{i_1} [f_i(b_j)]$.

Now for each $(i_0, i_1)$ we know $(b_0, [f_{i_0,i_1}(b_0)]), \ldots, (b_l, [f_{i_0,i_1}(b_l)])$. Since $f$ has degree $l$ we can therefore use Lagrange interpolation to compute $[f_{i_0,i_1}(0)]$ without further interaction.

This costs $l+1$ invocations of the unbounded fan-in protocol for invertible elements. To keep our expressions reasonable to look at we will use the bound which would have arisen from $l$ invocations of the unbounded fan-in protocol for invertible elements, which is $7l^2\kappa/f + 3l^2 + lA$.

## 5 Sub-Sequence Product

Assume that we have inputs $[a_1]_{\mathbb{F}}, \ldots, [a_l]_{\mathbb{F}}$ and want to compute sharings $[a_{i_0,i_1}]_{\mathbb{F}}$ of a number $A$ of sub-sequence products $a_{i_0,i_1} = \prod_{i=i_0}^{i_1} a_i$.

If $A = l$ this can be done with complexity $7l^2\kappa/f + 4l^2$ using an unbounded fan-in multiplication. We will however later need to solve this problem for $A = l^2$, where the complexity using unbounded fan-in multiplication would be $7l^2\kappa/f + 3l^2 + l^3$. We describe an algorithm which does considerably better.

### 5.1 Prefix Product

We first demonstrate our technique for prefix product. Assume that we have inputs $[a_1]_{\mathbb{F}}, \ldots, [a_l]_{\mathbb{F}}$, where $a_i \in \mathbb{F}$, and that we want to compute $[b_1]_{\mathbb{F}}, \ldots, [b_l]_{\mathbb{F}}$, where $b_i = \prod_{j=1}^{i} a_j$. Assume for notational convenience that $l = \lambda^3$ for an integer $\lambda$.

We will first work on $\lambda^2$ blocks of size $\lambda$, then a problem size of $\lambda^2$, and then do some post-processing.

For $i = 1, \ldots, \lambda^2$, let $\lambda_i = (i-1)\lambda$ and for $1 \le j \le \lambda$, compute the products

$$[b_{i,j}]_{\mathbb{F}} = \prod_{k=\lambda_i+1}^{\lambda_i+j} [a_i]_{\mathbb{F}} \ .$$

Using the unbounded fan-in algorithm these $A = \lambda - 1$ products for each block of size $\lambda$ can be computed with complexity $7\lambda^2\kappa/f + 3\lambda^2 + \lambda(\lambda-1) = 7\lambda^2\kappa/f + 4\lambda^2 - \lambda$, giving a total complexity of $7\lambda^4\kappa/f + 4\lambda^4 - \lambda^3$ for the $\lambda^2$ blocks.

It is now clear that if $I \bmod \lambda = 0$, then

$$b_I = \prod_{i=1}^{I/\lambda} b_{i,\lambda} \ ,$$

[3]If one does not know such elements they can be generated by moving the computation to an extension field of $\mathbb{F}$, but we will not need this, as we will later have that $a_i \in \{0,1\}$ and that $\mathbb{F}$ is large enough to pick the $b_j \in \mathbb{F} \setminus \{0,1\}$.

and that otherwise

$$b_I = (\prod_{i=1}^{\lfloor I/\lambda \rfloor} b_{i,\lambda}) b_{\lfloor I/\lambda \rfloor+1, I \bmod \lambda} \ .$$

The $A = (\lambda^2 - 1)$ prefix products $\prod_{i=1}^{\lfloor I/\lambda \rfloor} b_{i,\lambda}$ can be computed with complexity $7(\lambda^2)^2\kappa/f + 3(\lambda^2)^2 + \lambda^2(\lambda^2 - 1) = 7\lambda^4\kappa/f + 4\lambda^4 - \lambda^2$. Then at most one additional multiplication is needed to compute each of the $\lambda^3$ results $[b_I]_{\mathbb{F}}$. Therefore the total complexity is within $\big(7\lambda^4\kappa/f + 4\lambda^4 - \lambda^3\big) + \big(7\lambda^4\kappa/f + 4\lambda^4 - \lambda^2\big) + \lambda^3 < 14\lambda^4\kappa/f + 8\lambda^4 = 14l^{4/3}\kappa/f + 8l^{4/3}$.

## 5.2 Sub-Sequence Product

We then look at computing all the sub-sequence products. Assume that we have inputs $[a_1]_{\mathbb{F}}, \ldots, [a_l]_{\mathbb{F}}$, where $a_i \in \mathbb{F}$. For notational convenience we set $\lambda = l^{2/5}$ and ignore rounding issues. We will first work on $l/\lambda$ blocks of size $\lambda$, then a problem size of $l/\lambda$, and then do some post-processing.

For $i = 1, \ldots, l/\lambda$, let $\lambda_i = (i-1)\lambda$ and for $1 \le i_0, i_1 \le \lambda$, compute the products

$$[b_{i,i_0,i_1}]_{\mathbb{F}} = \prod_{k=\lambda_i+i_0}^{\lambda_i+i_1} [a_i]_{\mathbb{F}} \ .$$

Using the unbounded fan-in algorithm these $A < \lambda(\lambda-1)$ products for each block of size $\lambda$ can be computed with complexity $7\lambda^2\kappa/f + 3\lambda^2 + \lambda\lambda(\lambda-1) = 7\lambda^2\kappa/f + 2\lambda^2 + \lambda^3$. This is done for $l/\lambda$ blocks, giving a complexity of $7l\lambda\kappa/f + 2l\lambda + l\lambda^2 = 7l^{7/5}\kappa/f + 2l^{7/5} + l^{9/5}$.

Then for $1 \le i_0, i_1 \le l/\lambda$ compute the products

$$[b_{i_0,i_1}]_{\mathbb{F}} = \prod_{i=i_0}^{i_1} [b_{i,1,\lambda}]_{\mathbb{F}} \ .$$

Having $l/\lambda$ elements and $A \le l/\lambda(l/\lambda - 1)$, this can be done with complexity $7(l/\lambda)^2\kappa/f + 3(l/\lambda)^2 + (l/\lambda)l/\lambda(l/\lambda - 1) = 7l^{6/5}\kappa/f + 2l^{6/5} + l^{9/5}$, allowing us to bound the complexity so far by $14l^{7/5}\kappa/f + 4l^{7/5} + 2l^{9/5}$.

It is now clear that each $a_{i_0,i_1}$ can be expressed as $a_{i_0,i_1} = b_{i,I_0,I_1}$ for some $i, I_0, I_1$ or can be expressed as $b_{I_0,j_0,\lambda} b_{I_0+1,I_1-1} b_{I_1,1,j_1}$ for some $I_0, j_0, I_1, j_1$. I.e. each of the $< l^2$ values $a_{i_0,i_1}$ can be computed using at most 2 multiplications. This gives an overall complexity of $14l^{7/5}\kappa/f + 4l^{7/5} + 2l^{9/5} + 2l^2 < 14l^{7/5}\kappa/f + 5l^2$, except for very small values of $l$.

# 6 Carry Functions

Assume that we are given inputs $[a_1 \in \{0,1\}]_{\mathbb{F}}, \ldots, [a_l \in \{0,1\}]_{\mathbb{F}}, [b_1 \in \{0,1\}]_{\mathbb{F}}, \ldots, [b_l \in \{0,1\}]_{\mathbb{F}}$ and $[f_0 \in \{0,1\}]_{\mathbb{F}}$ and want to compute $[f_1 \in \{0,1\}]_{\mathbb{F}}, \ldots, [f_l \in \{0,1\}]_{\mathbb{F}}$, where for some known function $f$ we can express $f_i$ as $f_i = f(a_i, b_i, f_{i-1})$ for $i = 1, \ldots, l$. We call this a carry function.

Assume that instead of $[a_1 \in \{0,1\}]_{\mathbb{F}}, \ldots, [a_l \in \{0,1\}]_{\mathbb{F}}, [b_1 \in \{0,1\}]_{\mathbb{F}}, \ldots, [b_l \in \{0,1\}]_{\mathbb{F}}$ we get inputs

$$[f_0]_{\mathbb{F}}, (([c_1 \in \{0,1\}]_{\mathbb{F}}, [d_1 \in \{0,1\}]_{\mathbb{F}}), \ldots, ([c_l \in \{0,1\}]_{\mathbb{F}}, [d_l \in \{0,1\}]_{\mathbb{F}})) \ ,$$

where $c_i = f(a_i, b_i, 0)$ and $d_i = f(a_i, b_i, 1)$. Below, we call this an instance of the preprocessed carry problem.

We show how to compute $[f_1 \in \{0,1\}]_{\mathbb{F}}, \ldots, [f_l \in \{0,1\}]_{\mathbb{F}}$ unconditionally secure in constant round given an instance of the preprocessed carry problem. The solution is independent of the function $f$.

First, express $f_i$ as

$$\begin{aligned} f_i &= f(a_i, b_i, f_{i-1}) \\ &= f_{i-1}(f(a_i, b_i, 1) - f(a_i, b_i, 0)) + f(a_i, b_i, 0) \\ &= f_{i-1}(d_i - c_i) + c_i \\ &= f_{i-1}e_i + c_i \ , \text{ where } e_i = d_i - c_i \ . \end{aligned}$$

Letting $c_0 = f_0$ and expanding this expression we get that

$$f_i = \sum_{j=0}^{i} \left( c_j \prod_{k=j+1}^{i} e_k \right) \ .$$

The sub-sequence products $[\prod_{k=j+1}^{i} e_k]_{\mathbb{F}}$, for $i = 1, \ldots, l$ and $j = 1, \ldots, i$, can be computed using a sub-sequence product with complexity $14l^{7/5}\kappa/f + 5l^2$. Then each of the $\leq l^2 - l$ sub-sequence products should be multiplied by some $[c_j]_{\mathbb{F}}$, costing another $l^2 - l$ multiplications. Summing up the $[f_i]_{\mathbb{F}}$ is then for free. All in all this results in a complexity bounded by $14l^{7/5}\kappa/f + 6l^2$.

We can optimize this considerably. Let $\lambda > 1$ be some integer and assume for notational convenience that $l/\lambda$ is also an integer. We change the way of indexing elements $x_1, \ldots, x_l$ by letting $x_{i,j} = x_{\lambda i + j}$. For each $i = 0, \ldots, l/\lambda$ we get a block of the problem given by $(([c_{i,1}]_{\mathbb{F}}, [d_{i,1}]_{\mathbb{F}}) \ldots, ([c_{i,\lambda}]_{\mathbb{F}}, [d_{i,\lambda}]_{\mathbb{F}}))$.

If we knew $[f_{i,0}]_{\mathbb{F}} = [f_{i-1,\lambda}]_{\mathbb{F}}$ we could solve this block by solving the following instance of the preprocessed carry function

$$[f_{i,0}]_{\mathbb{F}}, (([c_{i,1}]_{\mathbb{F}}, [d_{i,1}]_{\mathbb{F}}), \ldots, ([c_{i,\lambda}]_{\mathbb{F}}, [d_{i,\lambda}]_{\mathbb{F}})) \ .$$

If we denote the result by $[f_{i,\cdot}]_{\mathbb{F}} = ([f_{i,1}]_{\mathbb{F}}, \ldots, [f_{i,\lambda}]_{\mathbb{F}})$, then the solution to the overall problem is seen to be $([f_{0,\cdot}]_{\mathbb{F}}, \ldots, [f_{l/\lambda-1,\cdot}]_{\mathbb{F}})$.

We do however not know $[f_{i,0}]_{\mathbb{F}}$. Instead what we do is solve the following two instances of the preprocessed carry problem

$$[0]_{\mathbb{F}}, (([c_{i,1}]_{\mathbb{F}}, [d_{i,1}]_{\mathbb{F}}), \ldots, ([c_{i,\lambda}]_{\mathbb{F}}, [d_{i,\lambda}]_{\mathbb{F}})) \ ,$$

$$[1]_{\mathbb{F}}, (([c_{i,1}]_{\mathbb{F}}, [d_{i,1}]_{\mathbb{F}}), \ldots, ([c_{i,\lambda}]_{\mathbb{F}}, [d_{i,\lambda}]_{\mathbb{F}})) \ .$$

Let the results be

$$([f^0_{i,1}]_{\mathbb{F}}, \ldots, [f^0_{i,\lambda}]_{\mathbb{F}})$$

respectively

$$([f^1_{i,1}]_{\mathbb{F}}, \ldots, [f^1_{i,\lambda}]_{\mathbb{F}}) \ .$$

Let $T(m)$ denote the complexity of solving an instance of the preprocessed carry problem of length $m$. Then the overall complexity for doing the above for each of the $l/\lambda$ blocks is $2l/\lambda T(\lambda)$.

Assume now again that we knew $[f_{i,0}] = [f_{i-1,\lambda}]$. In that case we could compute $[f_{i,\cdot}] = ([f_{i,1}], \ldots, [f_{i,\lambda}])$ by letting

$$[f_{i,j}] = [f_{i,0}]([f^1_{i,j}] - [f^0_{i,j}]) - [f^0_{i,j}] \ .$$

Overall, this would require another $l$ multiplications, bring the complexity up to $2l/\lambda T(\lambda) + l$.

By the above observation we in particular have that

$$[f_{i,\lambda}] = [f_{i,0}]([f^1_{i,\lambda}] - [f^0_{i,\lambda}]) - [f^0_{i,\lambda}] \ .$$

Since $f_{i,\lambda} = f_{i+1,0}$ this means that

$$[f_{i+1,0}] = [f_{i,0}]([f^1_{i,\lambda}] - [f^0_{i,\lambda}]) - [f^0_{i,\lambda}] \ .$$

Since we already know $[f^1_{i,\lambda}]$ and $[f^0_{i,\lambda}]$ and know $[f_{0,0}] = [f_0]$ we can therefore compute all $[f_{i,0}]$ by solving an instance of the preprocessed carry problem of size $l/\lambda$. This brings the overall complexity up to $2l/\lambda T(\lambda) + l + T(l/\lambda)$.

Having $T(m) = 14m^{7/5}\kappa/f + 6m^2$ and letting $\lambda = l^{1/3}$, this gives an overall complexity of

$$\begin{aligned}
&2l^{2/3}(14(l^{1/3})^{7/5}\kappa/f + 6(l^{1/3})^2) + l + 14(l^{2/3})^{7/5}\kappa/f + 6(l^{2/3})^2 \\
&= 2l^{2/3}(14l^{7/15}\kappa/f + 6l^{2/3}) + l + 14l^{14/15}\kappa/f + 6l^{4/3} \\
&= 28l^{17/15}\kappa/f + 12l^{4/3} + l + 14l^{14/15}\kappa/f + 6l^{4/3} \\
&< 33l^{17/15}\kappa/f + 18l^{4/3} \ .
\end{aligned}$$

## 7 Addition, Bitwise

Assume that sharings $[a_0]_\mathbb{F}, \ldots, [a_{l-1}]_\mathbb{F}$ and $[b_0]_\mathbb{F}, \ldots, [b_{l-1}]_\mathbb{F}$ are given with $a_0, \ldots, a_{l-1}, b_0, \ldots, b_{l-1} \in \{0,1\}$. Let $a = \sum_{i=0}^{l-1} a_i 2^i$ and $b = \sum_{i=0}^{l-1} b_i 2^i$ and let $d = a + b$. Define $d_0, \ldots, d_l \in \{0,1\}$ by $d = \sum_{i=0}^{l} d_i 2^i$. We want to compute sharings $[d_0]_\mathbb{F}, \ldots, [d_l]_\mathbb{F}$. The addition can be done rather efficiently in $l$ rounds by using the addition with carry iteratively. We want a constant round protocol.

Assume first that we have already computed

$$[c]_\mathrm{B} = [c_0]_\mathbb{F}, \ldots, [c_l]_\mathbb{F} \ ,$$

where $c_i$ is the $i$'th carry, i.e. $c_i = 1$ iff $\sum_{j=0}^{i-1} a_j 2^j + \sum_{j=0}^{i-1} b_j 2^j \geq 2^i$. We can then compute the result as

$$[d]_\mathrm{B} = [a_0]_\mathbb{F} \oplus [b_0]_\mathbb{F} \oplus [c_0]_\mathbb{F}, \ldots, [a_{l-1}]_\mathbb{F} \oplus [b_{l-1}]_\mathbb{F} \oplus [c_{l-1}]_\mathbb{F}, [c_l]_\mathbb{F} \ ,$$

where $\oplus$ denotes addition modulo 2.

It is straight-forward to verify that $a_i \oplus b_i \oplus c_i = a_i + b_i + c_i - 2c_{i+1}$. So, computing the results from the carry-vector can be done without interaction.

We describe how to compute sharings of the bits $c_i$. Clearly $c_0 = 0$, so use a dummy sharing of 0 as $[c_0]_\mathbb{F}$. For the remaining values $c_i$ it can be verified that if $c_{i-1} = 0$, then $c_i = c_i^0 = a_i \wedge b_i = a_i b_i$ and if $c_{i-1} = 1$, then $c_i = c_i^1 = a_i \vee b_i = a_i + b_i - a_i b_i = a_i + b_i - c_i^0$. This allows to compute $[0]_\mathbb{F}, (([c_0^0]_\mathbb{F}, [c_0^1]_\mathbb{F}), \ldots, ([c_{l-1}^0]_\mathbb{F}, [c_{l-1}^0]_\mathbb{F}))$ using $l$ multiplications.

Then $([c_0]_\mathbb{F}, \ldots, [c_{l-1}]_\mathbb{F})$ can be computed by solving an instance of the preprocessed carry problem in complexity $33l^{17/15}\kappa/f + 18l^{4/3}$. The bound $33l^{17/15}\kappa/f + 18l^{4/3}$ was rather conservative and we can therefore neglect the initial $l$ multiplications and assume that the complexity of a bitwise addition is bounded by $33l^{17/15}\kappa/f + 18l^{4/3}$, except for very small values of $l$.

It is straight-forward to verify that by solving an instance of the preprocessed carry problem one can securely compute a number of other functions on $[a]_\mathrm{B} = [a_0]_\mathbb{F}, \ldots, [a_{l-1}]_\mathbb{F}$ and $[b]_\mathrm{B} = [b_0]_\mathbb{F}, \ldots, [b_{l-1}]_\mathbb{F}$ within this complexity. This includes e.g. the function $[a \overset{?}{<} b]_\mathbb{F}$, where $(a \overset{?}{<} b) \in \{0,1\}$ and $(a \overset{?}{<} b) = 1$ iff $a < b$.

## References

[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.

[CD98] Ronald Cramer and Ivan Damgaard. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free. In Hugo Krawczyk, editor, *Advances in Cryptology - Crypto '98*, pages 424–441, Berlin, 1998. Springer-Verlag. Lecture Notes in Computer Science Volume 1462.

[FKN94] Uri Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *Proc.* 26*th STOC*, pages 554–563. ACM, 1994.

[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.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proc. 41st FOCS*, pp. 294–304, 2000.

[Kil05] Eike Kiltz: Unconditionally Secure Constant Round Multi-Party Computation for Equality, Comparison, Bits and Exponentiation, Cryptology ePrint Archive: Report 2005/066, February 28, 2005.