How to Split a Shared Secret into Shared Bits in Constant-Round

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

June 23, 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 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.

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.

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 $\mathcal{O}(1)$.

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 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 $a_1, \ldots, a_\ell$ 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.

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

[*]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.

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 $b = \prod_{i=1}^{\ell} a_i$. 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 computing $\overset{?}{<}$ (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 $[a_0]_p, \ldots, [a_{\ell-1}]_p$ 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] $O(2^d m)$, 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 $\overset{?}{<}: (\mathbb{F}_p)^\ell \times (\mathbb{F}_p)^\ell \to \mathbb{F}_p, ((a_0, \ldots, a_{\ell-1}), (b_0, \ldots, b_{\ell-1})) \mapsto \sum_{i=0}^{\ell-1} a_i 2^i \overset{?}{<} \sum_{i=0}^{\ell-1} b_i 2^i$ 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 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 round, $[a_0]_p, \ldots, [a_{\ell-1}]_p$ from $[a]_p$ 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 $[b]_p$ 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

[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.

multiplication implies that the adversary structure must be $Q2$ which in the standard threshold case means we need honest majority.

This result 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 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 $2^{-\kappa}$ — 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 secure protocol*, except that with probability $2^{-\kappa}$ 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-round multiparty computation with honest majority, see for instance [BB89, FKN94, CD98, IK00, 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 [ACS02] Algesheimer, Camenisch and Shoup also present a protocol for securely computing $([a_0]_p, \ldots, [a_{\ell-1}]_p) = DB([a]_p)$. It however only works when a is guaranteed to be noticeably smaller than p. Furthermore, it is only passive secure and is not constant round.

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 more along the lines of [ACS02]. The protocol from [Kil05] is however only passive secure and appears to be considerably less efficiently than the protocol we present here.[3]

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 then list some known results and simple observations. In Section 5 we give a protocol for comparing two bitwise-shared numbers and in Section 5 we give the protocol for adding two bitwise-shared numbers.

[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 perfectly secure protocol. It, however, only runs in expected constant round, as the protocol is run c times with probability $(2^{-\kappa})^{c-1}$.

[3]After personal communication Eike Kiltz and the authors of the present paper all acknowledge that our different solutions are concurrent and independent.

2 Preliminaries

In this section with introduce some notation.

We assume that n parties are connected by perfectly secure channels in a synchronous network.

We use $\mathbb{F}$ to denote a finite field, and we let $f = \lceil \log(|\mathbb{F}|) \rceil$, where we let $\log = \log_2$ for the rest of the paper. 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 and unconditionally secure. We will measure round complexity in the number of rounds of invocations of the multiplications and we will measure communication complexity by the number of invocations of the multiplication protocol.

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 Γ of subset $C \subset [n]$ which the adversary might corrupt; It is $Q2$ if it holds for all $C \in \Gamma$ that $[n] \setminus C \notin \Gamma$.

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 $p \in [2^{\ell-1}, 2^\ell]$. We look at the bit-decomposition function $BD : \mathbb{F}_p \to (\mathbb{F}_p)^\ell, a \mapsto (a_0, \ldots, a_{\ell-1})$ 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 $a \in \{0, 1, \ldots, p-1\}$.

Below we show how to securely generate a random solved instance

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

where b is a uniformly random $b \in \mathbb{F}_p$ and $(b_0, \ldots, b_{\ell-1}) = BD(b)$. This can be done using 21 rounds and 96ℓ invocations of the multiplication protocol.

Below we use $[x]_\mathrm{B} = [x_0]_p, \ldots, [x_{\ell-1}]_p$ 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.

In Section 6 it is shown how to add bitwise sharings unconditionally secure in constant round. When $x, y \in \{0, \ldots, 2^\ell - 1\}$ the complexity 37 rounds and $55\ell\log(\ell)$ invocations. In Section 5 it is shown how to compare bitwise sharings unconditionally secure in constant round. The complexity is 19 rounds and 22ℓ invocations.

The bit decomposition of $[a]_p$ proceeds as follows. First the parties generate a random solved instance $[b]_p$ and $[b]_\mathrm{B}$. Then 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 $(c_0, \ldots, c_{\ell-1}) = BD(c)$. Then using Section 6 the parties compute a bitwise sharing

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

of $d = b + c \in \{0, \ldots, 2^{\ell+1} - 1\}$. Clearly $d = a + qp$, where $q \in \{0, 1\}$. Using Section 5 the parties compute a sharing

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

Then the parties compute a bitwise sharing $[g]_{\mathrm{B}} = [g_0]_p, \ldots, [g_{\ell-1}]_p$ of $(2^\ell - qp) \bmod 2^\ell$ as follows. Let $(f_0, \ldots, f_{\ell-1}) = BD(2^\ell - p)$ and for $i = 0, \ldots, \ell - 1$, compute

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

The parties now have the followings bitwise sharings

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

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

Using again Section 6 they compute

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

It is easy to see that $h = a + q2^\ell$. So, by dropping the sharing $[h_\ell]_p$ of the most significant bit the parties obtain a bitwise sharing $[a]_{\mathrm{B}}$, as desired.

As for the complexity we generated one solved instance, had two applications of Section 6 and one application of Section 5. This yields a total complexity of 114 rounds and $110\ell \log(\ell) + 118\ell$ invocations. Assuming that $\log(\ell) \geq 4$, meaning that we compute on at least 16-bit numbers, the number of invocations can be bounded by $140\ell \log(\ell)$.

3.1 Generating Random Solved Instances

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

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

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

First the parties generate

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

i.e. sharings of ℓ uniformly random bits. We show in Section 4 how to do this with 2 rounds and 2ℓ invocations. Then using Section 5 the parties compute and reveal

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

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

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

If $\sum_{i=0}^{\ell-1} 2^i b_i \geq p$, then the protocol aborts. This clearly yields a uniformly random $b \in \mathbb{F}_p$ when the protocol does not abort.

Overall, this costs 21 rounds and 24ℓ invocations.

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 $2^{-\kappa}$. Though the Mersenne primes soon become sparse, 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. Other primes close to the 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 $b \in_R \mathbb{Z}_{2^\ell}$ 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 that $n/4$ of them satisfy $b < p$ is upper bounded by $2^{-\kappa}$. This means that the amortised complexity for generating one solved instance goes up to 21 rounds and 96ℓ invocations.

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 $[c_0 + \sum_{i=1}^{\ell} c_i a_i]$ from $[a_1], \ldots, [a_l]$ by the parties doing local computations. We write this computation as $c_0 + \sum_{i=1}^{\ell} c_i [a_i]$.

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 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 Γ 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 $\mathbb{F} = \mathbb{F}_p$ 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 bitstring representation of the elements. Define a map $\sqrt{\ } : Q(\mathbb{F}) \to \mathbb{F}$ by $\sqrt{a} \in SQRT(a)$ and $\sqrt{a} \geq -\sqrt{a}$. 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, and 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 $d \in \{0, 1\}$. 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 $|\mathbb{F}|^{-1}$. In the following estimates we assume that $|\mathbb{F}|^{-1} \leq 2^{-\kappa}$.

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 is 2 rounds and 2 invocations.

If it is not the case that $|\mathbb{F}|^{-1} \leq 2^{-\kappa}$, then, using that at least $|\mathbb{F}|^{-1} \leq 1/2$, it follows from a Chernoff bound that by generating $n = 12\kappa$ candidates, at least $n/4$ of them will be successful, except with probability $2^{-\kappa}$. This means that we can always get the same amortised complexity, except for a factor 4.

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 $[a]_\mathbb{F} = [b]_\mathbb{F}$. Notice that at the same price we can compute $[a^{-1}]_\mathbb{F}$ as $d^{-1}[c]_\mathbb{F}$.

If $d \notin \mathbb{F}^*$, then the algorithm aborts. This happens with probability less than $2/|\mathbb{F}|$. Again we assume that this is less than $2^{-\kappa}$, but if it is not we can solve it as for random bits and suffer only a factor 4 in the number of invocations. The complexity is 2 rounds and 3 invocations.

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

First use the above method to generate $[b_0 \in_R \mathbb{F}^*]_\mathbb{F}, [b_1 \in_R \mathbb{F}^*]_\mathbb{F}, \ldots, [b_\ell \in_R \mathbb{F}^*]_\mathbb{F}$, along with $[b_0^{-1}]_\mathbb{F}, [b_1^{-1}]_\mathbb{F}, \ldots, [b_\ell^{-1}]_\mathbb{F}$, using 2 rounds and $3(\ell + 1)$ invocations. For simplicity we will use the estimate 3ℓ invocations.

Then for $i = 1, \ldots, \ell$ compute and reveal $[c_i]_\mathbb{F} = [b_{i-1}]_\mathbb{F}[a_i]_\mathbb{F}[b_i^{-1}]_\mathbb{F}$, using 2 rounds and 2ℓ invocations.

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 $[a_{i_0,i_1}]_\mathbb{F} = [b_{i_0-1}^{-1}]_\mathbb{F} d_{i_0,i_1}[b_{i_1}]_\mathbb{F}$, using 1 round and A invocations.

The overall complexity is 5 rounds and $5\ell + A$ invocations.

5 Bitwise Less-Than

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

5.1 Symmetric Functions

Assume that we have inputs $[a_1]_{\mathbb{F}}, \ldots, [a_\ell]_{\mathbb{F}}$ and want to compute a symmetric Boolean function on these.

A symmetric Boolean function can be written as $f(x_1, \ldots, x_\ell) = \phi(\sum_{i=1}^{\ell} x_i)$ for some function $\phi : \{0, 1, \ldots, \ell\} \to \{0, 1\}$. This allows a particularly efficient secure computation. First compute $[a]_{\mathbb{F}} = 1 + \sum_{i=1}^{\ell} [a_i]_{\mathbb{F}}$. Then $a \in \{1, \ldots, \ell+1\}$. So, we can compute $[a]_{\mathbb{F}}, [a^2]_{\mathbb{F}}, \ldots, [a^{\ell+1}]_{\mathbb{F}}$ using an unbounded fan-in multiplication. After that we can compute for free $[f(a)]_{\mathbb{F}}$ for any $f(X) = \sum_{i=0}^{\ell+1} \alpha_i X^i \in \mathbb{F}[X]$, as $[f(a)]_{\mathbb{F}} = \alpha_0 + \sum_{i=1}^{\ell+1} \alpha_i [a^i]_{\mathbb{F}}$. In particular we can do this for the polynomial $f(X)$ of degree $\ell + 1$ for which $f(i) = \phi(i - 1)$ for $i = 1, \ldots, \ell + 1$.

The complexity is 5 rounds and 6ℓ invocations.

5.2 Prefix-Or

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

We use the method by Chandra, Fortune and Lipton [CFL83a]. For notational convenience, assume that $l = \lambda^2$ for an integer λ. Index the bits a_i as $a_{i,j} = a_{\lambda(i-1)+j}$ for $i, j = 1, \ldots, \lambda$. For $i = 1, \ldots, \lambda$, compute $[x_i]_{\mathbb{F}} = \vee_{j=1}^{\lambda} [a_{i,j}]_{\mathbb{F}}$ using λ parallel applications of the previous section. Then for $i = 1, \ldots, \lambda$ compute $[y_i]_{\mathbb{F}} = \vee_{k=1}^{i} [x_k]_{\mathbb{F}}$ using another λ parallel applications of the previous section. Now $y_i = 1$ iff some block $a_{i',1}, \ldots, a_{i',\lambda}$ with $i' \leq i$ contains a $a_{i',j} = 1$, i.e. $a_{i',1}, \ldots, a_{i',\lambda}$ is not all-zero.

We had 2λ applications of the previous section, in two rounds and on problems of size λ. This gives a total complexity until now of 10 rounds and 12ℓ invocations.

Let $[f_1]_{\mathbb{F}} = [x_1]_{\mathbb{F}}$ and for $i = 2, \ldots, \lambda$, let $[f_i]_{\mathbb{F}} = [y_i]_{\mathbb{F}} - [y_{i-1}]_{\mathbb{F}}$. Now $f_i = 1$ iff $a_{i,1}, \ldots, a_{i,\lambda}$ is the first block containing a $a_{i,j} = 1$. Let i_1 be such that $f_{i_1} = 1$. We can compute $[a_{i_1,1}]_{\mathbb{F}}, \ldots, [a_{i_1,\lambda}]_{\mathbb{F}}$ by computing $[a_{i_1,j}]_{\mathbb{F}} = \sum_{i=1}^{\lambda} [f_i]_{\mathbb{F}} [a_{i,j}]_{\mathbb{F}}$. This can be done using ℓ multiplications in parallel.

We can then compute $[b_{i_1,1}]_{\mathbb{F}}, \ldots, [b_{i_1,\lambda}]_{\mathbb{F}}$, where $b_{i_1,j} = \vee_{k=1}^{j} a_{i_1,k}$, using λ parallel applications of the previous section on problems of size λ.

Now let $[s_i]_{\mathbb{F}} = [y_i]_{\mathbb{F}} - [f_i]_{\mathbb{F}}$. Then $s_i = 1$ iff $i > i_1$. It follows that the results can be computed as $[a_{i,j}]_{\mathbb{F}} = [f_i]_{\mathbb{F}} [a_{i_1,j}]_{\mathbb{F}} + [s_i]_{\mathbb{F}}$. This can be done using ℓ multiplications in parallel.

The total complexity is 17 rounds and 20ℓ invocations.

5.3 Bitwise Less-Than

Assume then that sharings $[a_0]_{\mathbb{F}}, \ldots, [a_{\ell-1}]_{\mathbb{F}}$ and $[b_0]_{\mathbb{F}}, \ldots, [b_{\ell-1}]_{\mathbb{F}}$ are given with $a_0, \ldots, a_{\ell-1}$, $b_0, \ldots, b_{\ell-1} \in \{0, 1\}$. Let $a = \sum_{i=0}^{\ell-1} a_i 2^i$ and $b = \sum_{i=0}^{\ell-1} b_i 2^i$ and let $d = a \overset{?}{<} b$, where $d \in \{0, 1\}$ and $d = 1$ iff $a < b$. We want to compute $[d]_{\mathbb{F}}$.

First for $i = 0, \ldots, \ell - 1$ compute $c_i = a_i \oplus b_i = (a_i - b_i)^2$ in 1 round using ℓ multiplications.

Let $c_{-1} = 1$ and let i_0 denote the largest $-1 \leq i \leq \ell$ for which $c_i = 1$. Using the previous section, it is straight-forward to compute $[e_0]_{\mathbb{F}}, \ldots, [e_{\ell-1}]_{\mathbb{F}}$ where $e_i = 1$ if $i = i_0$ and $e_i = 0$ otherwise. This costs 17 rounds and 20ℓ invocations.

We can then compute $[d]_{\mathbb{F}} = \sum_{i=0}^{\ell-1} [e_i]_{\mathbb{F}} [b_i]_{\mathbb{F}}$ in 1 round using ℓ invocations. Assume namely that $i_0 = -1$. Then $a = b$ and $\sum_{i=0}^{\ell-1} e_i b_i = 0$, as desired. If $i_0 \geq 0$, then i_0 is the index of the most significant bit in which a and b differs, so the sought result is seen to be $d = b_{i_0} = \sum_{i=0}^{\ell-1} e_i b_i$.

All in all we used 19 rounds and 22ℓ invocations.

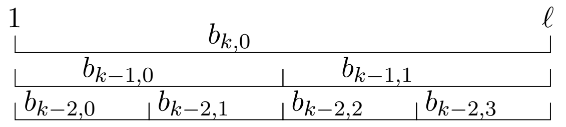
6 Bitwise Sum

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

6.1 Generic Prefix Computations

Assume that we have inputs $[a_1]_{\mathrm{B}}, \ldots, [a_\ell]_{\mathrm{B}}$, where $a_i \in \{0,1\}^n$. I.e. $[a_i]_{\mathrm{B}} = [a_{i,1}]_{\mathbb{F}}, \ldots, [a_{i,n}]_{\mathbb{F}}$ consists of n sharings of bits. Assume furthermore that an associative binary operator $\circ : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is given and that we want to compute $[b_1]_{\mathrm{B}}, \ldots, [b_\ell]_{\mathrm{B}}$, where $b_i = \circ_{j=1}^{i} a_j$. Assume that it is possible to securely compute $b_\ell = \circ_{j=1}^{\ell} a_j$ with complexity R rounds and $C(\ell)$ invocations. Assume for notational convenience that $\ell = 2^k$ for some k.

We use the method by Chandra, Fortune and Lipton [CFL83b]. For $i = 1, \ldots, k$ and $j = 0, \ldots, \ell/2^i - 1$, let $b_{i,j} = \circ_{m=j\cdot 2^i+1}^{j\cdot 2^i+2^i} a_m$.



There are $\ell - 1$ of the sums $b_{i,j}$, 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 is thus R rounds and $\sum_{i=1}^{k} 2^i C(\ell \cdot 2^{-i})$ invocations.

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 sums $b_{i,j}$. This costs another R rounds and at most $\ell C(k)$ invocations. Therefore the total complexity is upper bounded by $2R$ rounds and $\sum_{i=1}^{\log(\ell)} 2^i C(\ell \cdot 2^{-i}) + \ell C(\log(\ell)) \leq \log(\ell) C(\ell) + \ell C(\log(\ell))$ invocations.

6.2 Bitwise Sum

Assume we are given sharings $[a_0]_{\mathbb{F}}, \ldots, [a_{\ell-1}]_{\mathbb{F}}$ and $[b_0]_{\mathbb{F}}, \ldots, [b_{\ell-1}]_{\mathbb{F}}$ with $a_0, \ldots, a_{\ell-1}, b_0, \ldots, b_{\ell-1} \in \{0,1\}$. Let $a = \sum_{i=0}^{\ell-1} a_i 2^i$ and $b = \sum_{i=0}^{\ell-1} b_i 2^i$ and let $d = a + b$. Define $d_0, \ldots, d_\ell \in \{0,1\}$ by $d = \sum_{i=0}^{\ell} d_i 2^i$. We want to compute $[d_0]_{\mathbb{F}}, \ldots, [d_\ell]_{\mathbb{F}}$.

For $i = 0, \ldots, \ell$, define the carry $c_i \in \{0,1\}$ by $c_i = 1$ iff $\sum_{j=1}^{i-1} 2^j (a_j + b_j) > 2^i$. It is straight-forward to verify that given a bitwise sharing of the carries we can compute a bitwise sharing of the sum by computing $[d_\ell]_{\mathbb{F}} = [c_\ell]_{\mathbb{F}}$ and computing $[d_i]_{\mathbb{F}} = [a_i]_{\mathbb{F}} + [b_i]_{\mathbb{F}} + [c_i]_{\mathbb{F}} - 2[c_{i+1}]_{\mathbb{F}}$ for $i = 0, \ldots, \ell - 1$. This costs no interaction, so we now focus on computing the carries.

We use the well-known *carry set/propagate/kill* algorithm. For each $i = 0, \ldots, \ell - 1$, define bits s_i, p_i and k_i, where $s_i = 1$ iff a carry is set at position i (i.e. $a_i + b_i = 2$), and $p_i = 1$ iff a carry is propagated at position i (i.e. $a_i + b_i = 1$), and $k_i = 1$ iff a carry is killed at position i, (i.e. $a_i + b_i = 0$). We can compute these bits in 1 round using ℓ invocations, as $[s_i]_{\mathbb{F}} = [a_i]_{\mathbb{F}} [b_i]_{\mathbb{F}}$, $[p_i]_{\mathbb{F}} = [a_i]_{\mathbb{F}} + [b_i]_{\mathbb{F}} - 2[s_i]_{\mathbb{F}}$ and $[k_i]_{\mathbb{F}} = 1 - [s_i]_{\mathbb{F}} - [p_i]_{\mathbb{F}}$.

We let each triple (s_i, p_i, k_i) represent an element $e_i \in \Sigma = \{S, P, K\}$ where $e_i = S$ iff $s_i = 1$, $e_i = P$ iff $p_i = 1$ and $e_i = K$ iff $k_i = 1$. We define an operator $\circ : \Sigma \times \Sigma \to \Sigma$ by $S \circ x = S$, $K \circ x = K$ and $P \circ x = x$. This is the carry-propagation operator and it is clearly associative.

First we compute $f_0, \ldots, f_{\ell-1}$, where $f_i = \circ_{j=0}^{i} e_j$, where again f_i is represented by a triple of bits (S_i, P_i, K_i). We show below how to compute $f = \circ_{i=1}^{\ell} e_i$ with complexity 18 rounds and 27ℓ invocations. Using Section 6.1 this allows us to compute all $f_0, \ldots, f_{\ell-1}$ with complexity 36 rounds and $54\ell \log(\ell)$ invocations. It is straight-forward to verify that then we can then compute the carries without interaction by letting $c_0 = 0$ and letting $c_i = S_{i-1}$ for $i = 1, \ldots, \ell$.

All in all we used 37 rounds and $55\ell \log(\ell)$ invocations.

6.3 Unbounded Fan-In Carry Propagation

We show how to securely compute $f = \circ_{i=1}^{\ell} e_i$ with complexity 18 rounds and 27ℓ invocations.

Again, let e_i be represented by (s_i, p_i, k_i) and let f be represented by (S, P, K). We clearly have that $[P]_{\mathbb{F}} = \wedge_{i=1}^{\ell}[p_i]_{\mathbb{F}}$, which we can compute in 5 rounds and 6ℓ invocations using Section 5.2.

We have that $K = 1$ iff there exists some i such that $k_i = 1$ and $p_\ell = 1, \ldots, p_{i+1} = 1$. I.e. $K = \vee_{i=1}^{\ell}(k_i \wedge (\wedge_{j=i+1}^{\ell} p_j))$. Since k_i and p_i are never 1 simultaneously it can be seen that at most one of the expressions $k_i \wedge (\wedge_{j=i+1}^{\ell} p_j)$ equals one. This implies that $K = \sum_{i=1}^{\ell} k_i \cdot \wedge_{j=i+1}^{\ell} p_j$. From $[p_1]_{\mathbb{F}}, \ldots, [p_\ell]_{\mathbb{F}}$ we can compute $[q_1]_{\mathbb{F}}, \ldots, [q_\ell]_{\mathbb{F}}$ such that $q_i = \wedge_{j=i+1}^{\ell} p_j$ using Section 5.2. This costs 17 rounds and 20ℓ invocations. We can then compute $[K]_{\mathbb{F}} = \sum_{i=1}^{\ell}[k_i]_{\mathbb{F}}[q_i]_{\mathbb{F}}$ using 1 round and ℓ invocations. The overall complexity for computing $[K]_{\mathbb{F}}$ is thus 18 rounds and 21ℓ invocations.

Having computed $[P]_{\mathbb{F}}$ and $[K]_{\mathbb{F}}$ we can compute $[S]_{\mathbb{F}} = 1 - [P]_{\mathbb{F}} - [K]_{\mathbb{F}}$ without further interaction. Since we can compute P and K in parallel, the overall complexity for an unbounded fan-in carry propagation is 18 rounds and 27ℓ invocations.

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 M. Yung, editor, *Advances in Cryptology - Crypto 2002*, pages 417–432, Berlin, 2002. Springer-Verlag. Lecture Notes in Computer Science Volume 2442.

[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.

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

[CFL83b] Ashok K. Chandra, Steven Fortune, and Richard J. Lipton. Unbounded fan-in circuits and associative functions. In *Proceedings of the Fifthteenth Annual ACM Symposium on Theory of Computing*, pages 52–60, New York, NY, 1983.

[FKN94] Uri Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 554–563, Montréal, Québec, Canada, 23–25 May 1994.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. pages 294–304, 2000.

[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.

[Kil05] Eike Kiltz. Unconditionally secure constant round multi-party computation for equality, comparison, bits and exponentiation. Cryptology ePrint Archive 2005/066, February 2005.