CYBERNETICA
Institute of Information Security

Improved protocols for the SHAREMIND virtual machine

Dan Bogdanov, Margus Niitsoo, Tomas Toft, Jan Willemson

T-4-10 / 2010

Dan Bogdanov[1,2], Margus Niitsoo[1,2], Tomas Toft[3], Jan Willemson[1,4].
[1] Cybernetica, Institute of Information Security
[2] University of Tartu, Institute of Computer Science
[3] University of Aarhus
[4] Software Technology and Applications Competence Center

The research reported here was supported by:

1. Estonian Science foundation, grant(s) No. 8058,
2. the target funded theme SF0012708s06 “Theoretical and Practical Security of Heterogenous Information Systems”,
3. the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, and the Software Technology and Applications Competence Centre, STACC,

Cybernetica research reports are available online at
`http://research.cyber.ee/`

Mailing address:
AS Cybernetica
Akadeemia tee 21
12618 Tallinn
Estonia

Improved protocols for the SHAREMIND virtual machine

Dan Bogdanov, Margus Niitsoo, Tomas Toft, Jan Willemson

1 Introduction

The aim of secure multi-party computation (MPC) is to enable a number of networked players to carry out distributed computing tasks on their private information while being evesdropped on by a subset of malicious, colluding players.

The SHAREMIND (introduced in [1] and the extended version [2]) framework aims for a practical implementation of multi-party computation. It is based on share computing on additive shares from the ring $\mathbb{Z}_{2^{32}}$ and provides security in the honest-but-curious model. The framework has secure MPC protocols for performing private addition, multiplication, comparison and individual bit extraction operations on 32-bit integers or integer vectors. The protocols are designed to be universally composable, which allows running them sequentially and in parallel without compromising security.

Currently, the system works with 3 parties or miners. In this work, we propose new protocols for multiplication, conversion from bit shares to integer shares, bit extraction and equality testing that all make essential use of the 3-party assumption to provide substantial speed increases over the previously used protocols without compromising on the security assumptions. To simplify the security analysis of these protocols, we also introduce a new analysis framework that simplifies the security proofs to a great degree.

2 Proving Security of Sharemind Protocols

We will prove that all the proposed protocols are secure in the passive (honest-but-curious) model against one attacker. For that, we will use the framework established in [1], proving in fact, that all our protocols are perfectly simulatable.

Definition 1. *We say that a share computing protocol is* perfectly simulatable *if there exists an efficient universal non-rewinding simulator $\mathcal{S}$ that can simulate all protocol messages to any real world adversary $\mathcal{A}$ so that for all input shares the output distributions of $\mathcal{A}$ and $\mathcal{S}(\mathcal{A})$ coincide.*

Perfect simulatability is not enough when we need to compose the protocols. Namely, output shares of a perfectly simulatable protocol may depend on input shares. As a result, published shares may reveal more information about inputs than necessary. Therefore, we must often re-share the output shares at the end of each protocol. As proven in Lemma 2 of [1], this guarantees universal composability. However, in the protocols presented in the current paper, the final resharing is usually dropped.

In order to prove perfect simulatability, we consider the incoming views of all the computing parties and prove that they are independent of the input shares of the other

parties, hence proving existence of the simulator. We will use the sequences-of-games formalism in our proofs. Denote the distribution of the incoming view G as $[\![G]\!]$ and let the original incoming view of the party $\mathcal{P}$ be G_0. Then we are interested in finding a sequence $G_0, G_1, \ldots, G_n$ such that

$$[\![G_0]\!] = [\![G_1]\!] = \ldots = [\![G_n]\!]$$

and that the view G_n does not contain any references to input shares of the other parties.

The main tool that allows us to construct such sequences is the following lemma.

Lemma 1. *Let the incoming view G contain incoming messages $a_1 \pm r, a_2, \ldots, a_k$ where a_1, r are elements of finite additive group $\mathcal{A}$ and where r is a uniformly random element of $\mathcal{A}$, independent from all a_i. Then*

$$[\![G]\!] = [\![G[a_1 \pm r/r]]\!] \; .$$

Proof. If $r \in \mathcal{A}$ is uniformly distributed and independent from all a_i then so is $r \pm a_1$ since $f_r(x) := r \pm x$ is a bijective mapping for $\mathcal{A}$. □

In the following security proofs this lemma will be used for various groups, including $\mathbb{Z}_2$, $\mathbb{Z}_{2^n}$ and $(\mathbb{Z}_2)^n$. In the current implementation of SHAREMIND, $n = 32$ and thus $\log_2 n = 5$.

3 Improved Protocols

3.1 Multiplication

Instead of using the Du-Atallah multiplication protocol proposed in [1], we propose a new protocol described in Algorithm 1.

Theorem 1. *Algorithm 1 is correct and secure against a passive attacker.*

Proof. For correctness we note that

$$\begin{aligned} w = w_1 + w_2 + w_3 &= u'_1v'_1 + u'_1v'_3 + u'_3v'_1 + u'_2v'_2 + u'_2v'_1 + u'_1v'_2 + u'_3v'_3 + \\ &\quad + u'_3v'_2 + u'_2v'_3 = \\ &= (u'_1 + u'_2 + u'_3)(v'_1 + v'_2 + v'_3) = \\ &= (u_1 + r_{12} - r_{31} + u_2 + r_{23} - r_{12} + u_3 + r_{31} - r_{23}) \times \\ &\quad (v_1 + s_{12} - s_{31} + v_2 + s_{23} - s_{12} + v_3 + s_{31} - s_{23}) = \\ &= (u_1 + u_2 + u_3)(v_1 + v_2 + v_3) = uv \; . \end{aligned}$$

To prove security, we note that Algorithm 1 is symmetric for all the parties. Thus, it will be enough to consider just the incoming view of $\mathcal{P}_1$. Using Lemma 1 two times, we get

$$\left[\!\!\left[\begin{matrix} r_{31} \\ s_{31} \\ u_3 + r_{31} - r_{23} \\ v_3 + s_{31} - s_{23} \end{matrix}\right]\!\!\right] = \left[\!\!\left[\begin{matrix} r_{31} \\ s_{31} \\ r_{23} \\ v_3 + s_{31} - s_{23} \end{matrix}\right]\!\!\right] = \left[\!\!\left[\begin{matrix} r_{31} \\ s_{31} \\ r_{23} \\ s_{23} \end{matrix}\right]\!\!\right] \; .$$

Since the last view does not depend on private inputs other than u_1 and v_1, we conclude that the protocol can be perfectly simulated. □

Algorithm 1: Protocol for multiplying two shared values

Data: Shared values $[\![u]\!]$ and $[\![v]\!]$.
Result: Shared value $[\![w]\!]$ such that $w = uv$.
`Round 1`
$\mathcal{P}_1$ generates $r_{12}, s_{12} \leftarrow \mathbb{Z}_{2^n}$.
$\mathcal{P}_2$ generates $r_{23}, s_{23} \leftarrow \mathbb{Z}_{2^n}$.
$\mathcal{P}_3$ generates $r_{31}, s_{31} \leftarrow \mathbb{Z}_{2^n}$.
All values $*_{ij}$ are sent from $\mathcal{P}_i$ to $\mathcal{P}_j$.
`Round 2`
$\mathcal{P}_1$ computes $u'_1 \leftarrow u_1 + r_{12} - r_{31}$, $v'_1 \leftarrow v_1 + s_{12} - s_{31}$.
$\mathcal{P}_2$ computes $u'_2 \leftarrow u_2 + r_{23} - r_{12}$, $v'_2 \leftarrow v_2 + s_{23} - s_{12}$.
$\mathcal{P}_3$ computes $u'_3 \leftarrow u_3 + r_{31} - r_{23}$, $v'_3 \leftarrow v_3 + s_{31} - s_{23}$.
$\mathcal{P}_1$ sends u'_1 and v'_1 to $\mathcal{P}_2$.
$\mathcal{P}_2$ sends u'_2 and v'_2 to $\mathcal{P}_3$.
$\mathcal{P}_3$ sends u'_3 and v'_3 to $\mathcal{P}_1$.
`Round 3`
$\mathcal{P}_1$ computes $w_1 \leftarrow u'_1v'_1 + u'_1v'_3 + u'_3v'_1$.
$\mathcal{P}_2$ computes $w_2 \leftarrow u'_2v'_2 + u'_2v'_1 + u'_1v'_2$.
$\mathcal{P}_3$ computes $w_3 \leftarrow u'_3v'_3 + u'_3v'_2 + u'_2v'_3$.

3.2 Multiplication 2

Algorithm 1 can also be transformed to have one less round, see Algorithm 2

Correctness and security of Algorithm 2 are proved exactly the same way as for Algorithm 1.

3.3 Share Conversion

An improved version of share conversion routine is presented as Algorithm 3

Theorem 2. *Algorithm 3 is correct and secure against one passive attacker.*

Proof. For correctness, we first note that

$$u = u_1 \oplus u_2 \oplus u_3 = b \oplus m \oplus b_{12} \oplus s_{23} \oplus b_{13} \oplus s_{32} = m \oplus s\,.$$

Hence, if $s = 1$, we have

$$v = v_1 + v_2 + v_3 = 1 - m_{12} - m_{13} = 1 - m\,,$$

which is equal to $m \oplus 1$ when embedded to $\mathbb{Z}_{2^n}$. If $s = 0$ we have

$$v = v_1 + v_2 + v_3 = m_{12} + m_{13} = m\,,$$

which is equal to $m \oplus 0$ when embedded to $\mathbb{Z}_{2^n}$.

To prove security, we will consider all the three computing parties and prove that their incoming views can be perfectly simulated.

Algorithm 2: Protocol for multiplying two shared values

Data: Shared values $[\![u]\!]$ and $[\![v]\!]$.
Result: Shared value $[\![w]\!]$ such that $w = uv$.
`Round 1`
$\mathcal{P}_1$ generates $u_{13}, v_{13} \leftarrow \mathbb{Z}_{2^n}$.
$\mathcal{P}_2$ generates $u_{21}, v_{21} \leftarrow \mathbb{Z}_{2^n}$.
$\mathcal{P}_3$ generates $u_{32}, v_{32} \leftarrow \mathbb{Z}_{2^n}$.
$\mathcal{P}_1 \rightarrow \mathcal{P}_2 : u_1 - u_{13}, v_1 - v_{13}$; $\mathcal{P}_2 \rightarrow \mathcal{P}_1 : u_{21}, v_{21}$.
$\mathcal{P}_2 \rightarrow \mathcal{P}_3 : u_2 - u_{21}, v_2 - v_{21}$; $\mathcal{P}_3 \rightarrow \mathcal{P}_2 : u_{32}, v_{32}$.
$\mathcal{P}_3 \rightarrow \mathcal{P}_1 : u_3 - u_{32}, v_3 - v_{32}$; $\mathcal{P}_1 \rightarrow \mathcal{P}_3 : u_{13}, v_{13}$.
`Round 2 (post-processing)`
$\mathcal{P}_1$ and $\mathcal{P}_2$ compute $u'_1 \leftarrow u_1 - u_{13} + u_{21}$ and $v'_1 \leftarrow v_1 - v_{13} + v_{21}$.
$\mathcal{P}_2$ and $\mathcal{P}_3$ compute $u'_2 \leftarrow u_2 - u_{21} + u_{32}$ and $v'_2 \leftarrow v_2 - v_{21} + v_{32}$.
$\mathcal{P}_3$ and $\mathcal{P}_1$ compute $u'_3 \leftarrow u_3 - u_{32} + u_{13}$ and $v'_3 \leftarrow v_3 - v_{32} + v_{13}$.
$\mathcal{P}_1$ computes $w_1 \leftarrow u'_1 v'_1 + u'_1 v'_3 + u'_3 v'_1$.
$\mathcal{P}_2$ computes $w_2 \leftarrow u'_2 v'_2 + u'_2 v'_1 + u'_1 v'_2$.
$\mathcal{P}_3$ computes $w_3 \leftarrow u'_3 v'_3 + u'_3 v'_2 + u'_2 v'_3$.

Algorithm 3: Protocol for converting a share from $\mathbb{Z}_2$to $\mathbb{Z}_{2^n}$

Data: Shared value $[\![u]\!]$ in bit shares.
Result: Shared value $[\![v]\!]$ such that $[\![u]\!] = [\![v]\!]$ and $[\![v]\!]$ is shared in full shares.
`Round 1`
$\mathcal{P}_1$ generates random $b \leftarrow \mathbb{Z}_2$ and sets $m \leftarrow b \oplus u_1$.
$\mathcal{P}_1$ locally converts m to $\mathbb{Z}_{2^n}$, generates random $m_{12} \leftarrow \mathbb{Z}_{2^n}$ and computes $m_{13} = m - m_{12}$.
$\mathcal{P}_1$ generates random $b_{12} \leftarrow \mathbb{Z}_2$ and computes $b_{13} = b - b_{12} = b \oplus b_{12}$.
All values $*_{ij}$ are sent from $\mathcal{P}_i$ to $\mathcal{P}_j$.
`Round 2`
$\mathcal{P}_2$ sets $s_{23} \leftarrow b_{12} \oplus u_2$.
$\mathcal{P}_3$ sets $s_{32} \leftarrow b_{13} \oplus u_3$.
All values $*_{ij}$ are sent from $\mathcal{P}_i$ to $\mathcal{P}_j$.
`Round 3 (Post-processing)`
$\mathcal{P}_1$ sets $v_1 \leftarrow 0$
$\mathcal{P}_2$ and $\mathcal{P}_3$ set $s \leftarrow s_{23} \oplus s_{32}$
if $s = 1$ **then**
  $\mathcal{P}_2$ sets $v_2 \leftarrow (1 - m_{12})$.
  $\mathcal{P}_3$ sets $v_3 \leftarrow (-m_{13})$.
else
  $\mathcal{P}_2$ sets $v_2 \leftarrow m_{12}$.
  $\mathcal{P}_3$ sets $v_3 \leftarrow m_{13}$.
end

– The view of party $\mathcal{P}_1$ contains no incoming messages, so the corresponding simulator is trivial.
– The incoming view of $\mathcal{P}_2$ can be perfectly simulated, since using Lemma 1 we see that its distribution

$$\left[\!\!\left[\begin{matrix} m_{12} \\ b_{12} \\ b \oplus b_{12} \oplus u_3 \end{matrix} \right]\!\!\right] = \left[\!\!\left[\begin{matrix} m_{12} \\ b_{12} \\ b \end{matrix} \right]\!\!\right]$$

is independent of private inputs other than u_2.
– Similarly, the incoming view of $\mathcal{P}_3$ can be perfectly simulated, since using Lemma 1 we see that its distribution

$$\left[\!\!\left[\begin{matrix} b \oplus u_1 - m_{12} \\ b \oplus b_{12} \\ b_{12} \oplus u_2 \end{matrix} \right]\!\!\right] = \left[\!\!\left[\begin{matrix} m_{12} \\ b \oplus b_{12} \\ b_{12} \oplus u_2 \end{matrix} \right]\!\!\right] = \left[\!\!\left[\begin{matrix} m_{12} \\ b \\ b_{12} \oplus u_2 \end{matrix} \right]\!\!\right] = \left[\!\!\left[\begin{matrix} m_{12} \\ b \\ b_{12} \end{matrix} \right]\!\!\right]$$

is independent of private inputs other than u_3.

□

3.4 Bit extraction

An improved version of bit extraction routine is presented in Algorithm 4. In the algorithm, lower indices are used to refer to the computing parties and they are handled modulo 3 (i.e., where necessary we identify $\mathcal{P}_4 = \mathcal{P}_1$ and $\mathcal{P}_0 = \mathcal{P}_3$). All the binary operations ($\oplus$ and $\wedge$) on 32-bit values are performed bitwise i.e. as operations in $(\mathbb{Z}_2)^n$ for integer values in $\mathbb{Z}_{2^n}$.

Theorem 3. *Algorithm 4 is correct and secure against one passive attacker.*

Proof. The basic working principle of Algorithm 4 is the same of the bitwise addition protocol explained in [2]. During the first two rounds, u is represented as

$$u = u_1 + u_2 + u_3 = (r + r_2 + r_3) + (v_2 - r_2) + (v_3 - r_3) = v_2 + v_3 + r = v + r\,,$$

where r has a known shared bit decomposition $r = \overline{r}_2 \oplus \overline{r}_3$. Thus, in order to find the bits of u, we can use bitwise addition to compute the bits of $v + r$. To do that, one needs to compute the carry bits, and this is done using the carry-lookahead algorithm exactly as in [2].

First, we set $s = v \wedge r$ and $p = v \oplus r$ (see lines number 9 to 11). Then in the main cycle (lines 13 to 21), the values p and s first get reshared (lines 14 to 16). Indeed, note that

$$p_1 \oplus q_1^k \oplus q_3^k \oplus p_2 \oplus q_2^k \oplus q_1^k \oplus p_3 \oplus q_3^k \oplus q_2^k = p_1 \oplus p_2 \oplus p_3 = p\,,$$

and similarly for s.

Lines 18 to 22 implement the carry-lookahead algorithm. It works by iteratively merging the adjacent blocks of bits of p and s; see [2] for the full description of the algorithm.[1] The major technical detail is working bitwise over the field $\mathbb{Z}_2$, so that

[1] Do we need to explain the algorithm here in detail?

Algorithm 4: Protocol for bit extraction.

Data: Additively shared value $[\![u]\!]$.
Result: Bitwise shares $[\![w^{(j)}]\!]$ of u.

1 `Round 1`
2 $\mathcal{P}_1$ generates random $r, r_2, \overline{r}_2 \leftarrow \mathbb{Z}_{2^n}$ and computes $r_3 \leftarrow u_1 - r - r_2$, $\overline{r}_3 \leftarrow r \oplus \overline{r}_2$.
3 $\mathcal{P}_1$ sends $r_i, \overline{r}_i$ to $\mathcal{P}_i$ $(i = 2, 3)$.
4 Each $\mathcal{P}_i$ chooses $q_i^0, t_i^0 \leftarrow \mathbb{Z}_{2^n}$ and sends them to $\mathcal{P}_{i+1}$.
5 `Round 2`
6 $\mathcal{P}_i$ $(i = 2, 3)$ computes the share $v_i \leftarrow u_i + r_i$ and sends it to $\mathcal{P}_{6/i}$.
7 `Rounds 3 to` $\log_2 n + 2$
8 $\mathcal{P}_2, \mathcal{P}_3$ compute $v = v_2 + v_3$.
9 $\mathcal{P}_1$ sets $p_1 \leftarrow 0$, $s_1 \leftarrow 0$.
10 $\mathcal{P}_2$ sets $p_2 \leftarrow v \oplus \overline{r}_2$, $s_2 \leftarrow v \wedge \overline{r}_2$.
11 $\mathcal{P}_3$ sets $p_3 \leftarrow \overline{r}_3$, $s_3 \leftarrow v \wedge \overline{r}_3$.
12 All parties $\mathcal{P}_i$ perform the following computations:
13 **for** $k \leftarrow 0$ **to** $\log_2 n - 1$ **do**
14 $p_i \leftarrow p_i \oplus q_i^k \oplus q_{i-1}^k$.
15 $s_i \leftarrow s_i \oplus t_i^k \oplus t_{i-1}^k$.
16 Send p_i, s_i to $\mathcal{P}_{i+1}$.
17 Generate new random items $q_i^{k+1}, t_i^{k+1} \leftarrow \mathbb{Z}_{2^n}$ and send them to $\mathcal{P}_{i+1}$.
18 **for** $\ell \leftarrow 0$ **to** $2^k - 1$ **do**
19 **for** $m \leftarrow 0$ **to** $\frac{n}{2^{k+1}} - 1$ **do**
20 $[\![s^{(2^k+\ell+2^{k+1}m)}]\!] \leftarrow [\![s^{(2^k+\ell+2^{k+1}m)}]\!] \oplus [\![p^{(2^k+\ell+2^{k+1}m)}]\!] \wedge [\![s^{(2^k+2^{k+1}m-1)}]\!]$
21 $[\![p^{(2^k+\ell+2^{k+1}m)}]\!] \leftarrow [\![p^{(2^k+\ell+2^{k+1}m)}]\!] \wedge [\![p^{(2^k+2^{k+1}m-1)}]\!]$
22 //`Note that` $([\![a]\!] \wedge [\![b]\!])_i$ `can be computed locally as` $(a_i \wedge b_i) \oplus (a_i \wedge b_{i-1}) \oplus (a_{i-1} \wedge b_i)$)
23 Define ${s_i}^{(-1)} = 0$
24 **for** $j \leftarrow 0$ **to** $n - 1$ **do**
25 In $\mathcal{P}_1$: ${w_1}^{(j)} \leftarrow {s_1}^{(j-1)}$
26 In $\mathcal{P}_2$: ${w_2}^{(j)} \leftarrow {v}^{(j)} \oplus {\overline{r}_2}^{(j)} \oplus {s_2}^{(j-1)}$
27 In $\mathcal{P}_3$: ${w_3}^{(j)} \leftarrow {\overline{r}_3}^{(j)} \oplus {s_3}^{(j-1)}$
28 Run share conversion protocol (Algorithm 3) on all $[\![w^{(j)}]\!]$.

addition is represented by $\oplus$ and multiplication by $\wedge$. In order to compute the multiplication over $\mathbb{Z}_2$, we will essentially use the idea behind Algorithm 1 by computing $[\![a]\!] \wedge [\![b]\!]$ locally as $(a_i \wedge b_i) \oplus (a_i \wedge b_{i-1}) \oplus (a_{i-1} \wedge b_i)$. Indeed, adding all the shares we get

$$\begin{aligned} &(a_1 \wedge b_1) \oplus (a_1 \wedge b_3) \oplus (a_3 \wedge b_1) \oplus (a_2 \wedge b_2) \oplus (a_2 \wedge b_1) \oplus \\ &\oplus (a_1 \wedge b_2) \oplus (a_3 \wedge b_3) \oplus (a_3 \wedge b_2) \oplus (a_2 \wedge b_3) = \\ &= (a_1 \oplus a_2 \oplus a_3) \wedge (b_1 \oplus b_2 \oplus b_3) = a \wedge b\,. \end{aligned}$$

Note also that all the shares required for this computation on lines 20 and 21 are local and that all these computations can be run in parallel.

As a result of the main cycle, the bit vector s will contain exactly the carry bits from the corresponding positions when computing $v + r$, hence it remains to add these bits

to the shared bitwise $\oplus$ of v and r, which is done on lines 23 to 27. Indeed, we see that

$$w^{(j)} = {s_1}^{(j-1)} \oplus v^{(j)} \oplus \overline{r}_2{}^{(j)} \oplus {s_2}^{(j-1)} \oplus \overline{r}_3{}^{(j)} \oplus {s_3}^{(j-1)} = v^{(j)} \oplus \overline{r}^{(j)} \oplus s^{(j-1)} .$$

As the last step of the Algorithm, share conversion is run to represent the bit shares as elements of the ring $\mathbb{Z}_{2^n}$.

To prove security, we will consider all the three computing parties and prove that their incoming views can be perfectly simulated.

– The incoming view of party $\mathcal{P}_1$ is

$$\left[\!\!\left[\begin{array}{l} q_3^0 \\ t_3^0 \\ p_3 \oplus q_3^0 \oplus q_2^0 \\ s_3 \oplus t_3^0 \oplus t_2^0 \\ q_3^1 \\ t_3^1 \\ p_3 \oplus q_3^1 \oplus q_2^1 \\ s_3 \oplus t_3^1 \oplus t_2^1 \\ \dots \\ q_3^{\log_2 n-1} \\ t_3^{\log_2 n-1} \\ p_3 \oplus q_3^{\log_2 n-1} \oplus q_2^{\log_2 n-1} \\ s_3 \oplus t_3^{\log_2 n-1} \oplus t_2^{\log_2 n-1} \\ q_3^{\log_2 n} \\ t_3^{\log_2 n} \end{array}\right]\!\!\right] = \left[\!\!\left[\begin{array}{l} q_3^0 \\ t_3^0 \\ p_3 \oplus q_3^0 \oplus q_2^0 \\ s_3 \oplus t_3^0 \oplus t_2^0 \\ q_3^1 \\ t_3^1 \\ p_3 \oplus q_3^1 \oplus q_2^1 \\ s_3 \oplus t_3^1 \oplus t_2^1 \\ \dots \\ q_3^{\log_2 n-1} \\ t_3^{\log_2 n-1} \\ q_2^{\log_2 n-1} \\ t_2^{\log_2 n-1} \\ q_3^{\log_2 n} \\ t_3^{\log_2 n} \end{array}\right]\!\!\right] = \dots = \left[\!\!\left[\begin{array}{l} q_3^0 \\ t_3^0 \\ p_3 \oplus q_3^0 \oplus q_2^0 \\ s_3 \oplus t_3^0 \oplus t_2^0 \\ q_3^1 \\ t_3^1 \\ q_2^1 \\ t_2^1 \\ \dots \\ q_3^{\log_2 n-1} \\ t_3^{\log_2 n-1} \\ q_2^{\log_2 n-1} \\ t_2^{\log_2 n-1} \\ q_3^{\log_2 n} \\ t_3^{\log_2 n} \end{array}\right]\!\!\right] = \left[\!\!\left[\begin{array}{l} q_3^0 \\ t_3^0 \\ q_2^0 \\ t_2^0 \\ q_3^1 \\ t_3^1 \\ q_2^1 \\ t_2^1 \\ \dots \\ q_3^{\log_2 n-1} \\ t_3^{\log_2 n-1} \\ q_2^{\log_2 n-1} \\ t_2^{\log_2 n-1} \\ q_3^{\log_2 n} \\ t_3^{\log_2 n} \end{array}\right]\!\!\right]$$

  Since last view only contains elements generated randomly and independently during the protocol, it can be perfectly simulated.
– The incoming view of $\mathcal{P}_2$ looks mostly the same as that of $\mathcal{P}_1$, only the initial part differs. We use Lemma 1 again to see that

$$\left[\!\!\left[\begin{array}{l} r_2 \\ \overline{r}_2 \\ u_3 + r_3 \\ \dots \end{array}\right]\!\!\right] = \left[\!\!\left[\begin{array}{l} r_2 \\ \overline{r}_2 \\ r_3 \\ \dots \end{array}\right]\!\!\right]$$

  which does not depend on any input values and can hence be perfectly simulated.
– Similarly, for party $\mathcal{P}_3$ we have the initial part of the incoming view

$$\left[\!\!\left[\begin{array}{l} u_1 - r - r_2 \\ r \oplus \overline{r}_2 \\ u_2 + r_2 \\ \dots \end{array}\right]\!\!\right] = \left[\!\!\left[\begin{array}{l} u_1 - r - r_2 \\ \overline{r}_2 \\ u_2 + r_2 \\ \dots \end{array}\right]\!\!\right] = \left[\!\!\left[\begin{array}{l} r \\ \overline{r}_2 \\ u_2 + r_2 \\ \dots \end{array}\right]\!\!\right] = \left[\!\!\left[\begin{array}{l} r \\ \overline{r}_2 \\ r_2 \\ \dots \end{array}\right]\!\!\right]$$

  which does not depend on any input values once again.

□

We note that this protocol can also be used to improve the efficiency of comparison protocols which usually use bit extraction as a subroutine [2].

3.5 Equality testing

We note that equality testing can be accomplished fairly easily via bit extraction. However, since it is used fairly often in practical applications, it makes sense to provide a separate and more efficient protocol specifically designed for that task. The new equality testing protocol for two shared values is given in Algorithm 5.

Algorithm 5: Protocol for evaluating the equality predicate.

Data: Shared values $[\![u]\!]$ and $[\![v]\!]$.
Result: Shares $[\![w]\!]$ such that $w = \text{EQ}(u, v)$.

1 `Round 1`
2 $\mathcal{P}_1$ generates random $r_2 \leftarrow \mathbb{Z}_{2^n}$ and computes $r_3 \leftarrow (u_1 - v_1) - r_2$.
3 $\mathcal{P}_1$ sends r_i to $\mathcal{P}_i$ $(i = 2, 3)$.
4 Each $\mathcal{P}_i$ chooses $q_i^0 \leftarrow \mathbb{Z}_{2^n}$ and sends them to $\mathcal{P}_{i+1}$.
5 `Rounds 2 to` $\log_2 n + 1$
6 $\mathcal{P}_2, \mathcal{P}_3$ compute $e_i = (u_i - v_i) + r_i$.
7 $\mathcal{P}_1$ sets $p_1 \leftarrow 2^n - 1$.
8 $\mathcal{P}_2$ sets $p_2 \leftarrow e_2$.
9 $\mathcal{P}_3$ sets $p_3 \leftarrow (0 - e_3)$.
10 All parties $\mathcal{P}_i$ perform the following computations:
11 **for** $k \leftarrow 0$ **to** $\log_2 n - 1$ **do**
12 $p_i \leftarrow p_i \oplus q_i^k \oplus q_{i-1}^k$.
13 Send p_i to $\mathcal{P}_{i+1}$.
14 Generate new random $q_i^{k+1} \leftarrow \mathbb{Z}_{2^n}$ and send it to $\mathcal{P}_{i+1}$.
15 **for** $m \leftarrow 0$ **to** $\frac{n}{2^{k+1}} - 1$ **do**
16 $[\![p^{(2^{k+1}m)}]\!] \leftarrow [\![p^{(2^{k+1}m)}]\!] \wedge [\![p^{(2^{k+1}m+2^k)}]\!]$
17 //`Note that` $[\![p]\!]$ `can be computed locally`
18 Run share conversion protocol (Algorithm 3) on $[\![p^{(0)}]\!]$ and return that as $[\![w]\!]$.

Theorem 4. *Algorithm 5 is correct and secure against one passive attacker.*

Proof. For correctness note first that

$$e_2 + e_3 = (u_2 - v_2) + r_2 + (u_3 - v_3) + r_3 = (u_2 - v_2) + (u_3 - v_3) + (u_1 - v_1) = u - v\,,$$

hence $u = v$ iff $e_2 = 0 - e_3$. Algorithm 5 compares u and v by comparing $p_2 = e_2$ and $p_3 = (0 - e_3)$ bitwise. For that, bitwise sum (xor) p of $p_1 = 2^n - 1 = 111\ldots1$, p_2 and p_3 is analyzed. We see that $u = v$ iff all the bits of p are 1, which is exactly the case when the product $w = \prod_{i=0}^{n-1} p^{(i)}$ of the bits of p is 1. This is exactly what the cycle in lines 11 to 16 does.

To prove security, we will consider all the three computing parties and prove that their incoming views can be perfectly simulated.

The incoming view of party $\mathcal{P}_1$ is

$$\left[\!\!\left[\begin{array}{l} q_3^0 \\ p_3 \oplus q_3^0 \oplus q_2^0 \\ q_3^1 \\ p_3 \oplus q_3^1 \oplus q_2^1 \\ \ldots \\ q_3^{\log_2 n-1} \\ p_3 \oplus q_3^{\log_2 n-1} \oplus q_2^{\log_2 n-1} \\ q_3^{\log_2 n} \end{array} \right]\!\!\right] = \left[\!\!\left[\begin{array}{l} q_3^0 \\ q_2^0 \\ q_3^1 \\ q_2^1 \\ \ldots \\ q_3^{\log_2 n-1} \\ q_2^{\log_2 n-1} \\ q_3^{\log_2 n} \end{array} \right]\!\!\right]$$

following the same argumentation as in the preceding proof.

The incoming view of $\mathcal{P}_2$ is again pretty much the same as that of $\mathcal{P}_1$ with the only exception of receiving one extra independently and uniformly chosen element r_2, which is trivial to simulate. The same goes for $\mathcal{P}_3$ who receives $r_3 = (u_1 - v_1) - r_2$ which can be replaced by r_2 by Lemma 1. □

References

1. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, volume 5283 of *LNCS*, pages 192–206. Springer, 2008.
2. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: a framework for fast privacy-preserving computations. Cryptology ePrint Archive, Report 2008/289, 2008. `http://eprint.iacr.org/`.