Easily programmable secure multi-party computation on integers, strings and floating point numbers

Dan Bogdanov
Sharemind project lead
dan@cyber.ee

https://sharemind.cyper.ee/
privacy preservation in statistics and data mining
Providing Security-as-a-Service

The **sharemind** secure multi-party database

Data owners

People with queries

Data

Query

Results
A typical problem statement

How do we jointly analyze data without showing it to others?
A typical (but insecure) solution
A typical (but insecure) solution

This requires that the party hosting the data warehouse is trusted by everyone.
Our specific goal

- Secure data aggregation
  - Analyze data collected from several sources
  - Build services that package this technology.

- Simple statistics and complex algorithms
  - Compute sums and averages, use filtering.
  - Perform complex analyses like market basket analysis, clustering, regression and so on.
Security measures and guarantees

- The data entry application protects input data.
  - Only the data owner sees the input values.

- The database of each server leaks no data.
  - Defense against insiders (e.g. system administrators).
  - Some degree of protection against malicious hacking.

- The servers run only agreed-to computations.
  - Protection against malicious queries.
overview of sharemind 2
Secure computation à la sharemind

- We use additive secret sharing on 32-bit unsigned integers [BLW08].

- Both public and private values are from $\mathbb{Z}_{2^{32}}$.

- Three miner servers store the data and perform secure multi-party computation.

- Any number of controller applications provide data and request computations.

- Ideally, we can show information-theoretic security.

Getting data into **sharemind**
Features of controller applications

- Controller applications are built using the *controller library*.

- Different controller libraries exist for desktop and web applications [TB09].

- Mobile versions of the controller library are planned.

- The controller application automatically handles secret sharing when data is entered and when results are received.

---

The **sharemind** built-in database

**Miner 1 database**
- **Person**
  - gender
  - age
  - education
  - incomeRange
- **ShoppingBasket**
  - date
  - items

Contains one share of each secret

**Miner 2 database**
- **Person**
  - gender
  - age
  - education
  - incomeRange
- **ShoppingBasket**
  - date
  - items

Contains one share of each secret

**Miner 3 database**
- **Person**
  - gender
  - age
  - education
  - incomeRange
- **ShoppingBasket**
  - date
  - items

Contains one share of each secret
Processing data on **sharemind**
The data analyst receives shares of the final result and nothing else, and the result is reconstructed.
Secure operations on sharemind

- Additive secret sharing is additively homomorphic so we get addition and multiplication by constant for free.

- We use custom protocols for all other operations.

- We have security and correctness proofs for these protocols together with universal composability proofs that allow them to be used in a programmable system.

- The current protocol suite is not yet published [BNTW].

## Performance in lab conditions (LAN)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Rounds</th>
<th>SISD</th>
<th>SIMD</th>
<th>SIMD Hz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>local operation</td>
<td>-</td>
<td>0,015 μs</td>
<td>66 MHz</td>
</tr>
<tr>
<td>Multiplication w public</td>
<td>local operation</td>
<td>-</td>
<td>0,006 μs</td>
<td>166 MHz</td>
</tr>
<tr>
<td>Cast bool to int</td>
<td>1</td>
<td>15,3 ms</td>
<td>0,8 μs</td>
<td>1,25 MHz</td>
</tr>
<tr>
<td>Multiplication w private</td>
<td>2</td>
<td>25,9 ms</td>
<td>1,8 μs</td>
<td>555 KHz</td>
</tr>
<tr>
<td>Equality</td>
<td>l + 2</td>
<td>101 ms</td>
<td>5,0 μs</td>
<td>200 KHz</td>
</tr>
<tr>
<td>Greater-than</td>
<td>l + 3</td>
<td>113 ms</td>
<td>51 μs</td>
<td>20 KHz</td>
</tr>
<tr>
<td>Bit decomposition</td>
<td>l + 3</td>
<td>122 ms</td>
<td>15,7 μs</td>
<td>64 KHz</td>
</tr>
<tr>
<td>Division w public</td>
<td>l + 4</td>
<td>124 ms</td>
<td>44 μs</td>
<td>23 KHz</td>
</tr>
<tr>
<td>Division w private</td>
<td>4l + 9</td>
<td>390 ms</td>
<td>534 μs</td>
<td>1,9 KHz</td>
</tr>
</tbody>
</table>

Note: All operations are on 32-bit unsigned integers.

Note: $l = \log_2(\text{numberOfBitsInDataType})$
Saturation points in performance

Mult
- Old protocol
- New protocol

Performance on an international cloud

- We deployed Sharemind internationally, with miners in:
  - United States (West coast)
  - United Kingdom (London)
  - Japan (Tokyo)

<table>
<thead>
<tr>
<th>Protocol</th>
<th>SIMD (100 000 parallel ops)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cast bool to int</td>
<td>18 μs per operation</td>
</tr>
<tr>
<td>Multiplication w private</td>
<td>36 μs per operation</td>
</tr>
<tr>
<td>Equality</td>
<td>78 μs per operation</td>
</tr>
<tr>
<td>Greater-than</td>
<td>380 μs per operation</td>
</tr>
<tr>
<td>Bit decomposition</td>
<td>1,58 ms per operation</td>
</tr>
</tbody>
</table>
tools for creating secure applications
Deployment of a Sharemind system

Data model

- Enter data manually
- Import existing data

Business logic

Private point-to-point communication channels

Data miner 1

Data miner 2

Data miner 3

Access results from data mining and aggregation algorithms
Programming secure computations

- The secure functionality is programmable in an assembly language that is interpreted by Sharemind.

- Internally, Sharemind has a private stack and public and private registers to support the implementation of algorithms.

- All registers store vectors to better support SIMD operations.

- The design is described in [BL10].

The SecreC language

```cpp
public int count (private int[[1]] data, public int value)
{
    public int length = size (data);
    private int matchcounter = 0;
    public int i = 0;
    for (i = 0; i < length; i++) {
        private bool match = (data[i] == needle);
        matchcounter += match;
    }
    return declassify (matchcounter);
}
```

The SecreCIDE developer tool

```java
// The function requires a one-dimensional array and a value
private int count (private int[] data, public int value) {
    public int length = size(data); // number of private values
    public int[] values(length) = value; // expand the input to a vector
    private bool[] matches = (data == values); // parallel comparisons
    private int matchcounter = vecSum (matches); // add up ones and zeroes
    return matchcounter; // return the number of matches
}

// Counts the number of times a value occurs in a database column
public void main (public string db, public string table,
                 public string column, public int value) {
    dbLoad (db); // load the database by name
    private int[] data = dbGetColumn(column, table); // get the data
    private int result = count (data, value); // call our counting function
    public int publicresult = declassify (result); // make the result public
    publish ("countresult", publicresult); // send it to the client
}
```

The **sharemind** SDK is freely available

- Sharemind SDK version 2012.04 is the latest version.

- It contains:
  - a developer version of the Sharemind 2.1 machine,
  - a compiler for the SecreC programming language,
  - a controller library for C++ applications,
  - example SecreC code and applications

- See [https://sharemind.cyber.ee/](https://sharemind.cyber.ee/) for downloads.
applications
A Sharemind installation deployed by three independent members of the ITL consortium performs secure computations.
Secure computation algorithms used

- The analyses were implemented in SecreC.

<table>
<thead>
<tr>
<th>Analysis operation</th>
<th>Applied secure computation primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oblivious filtering to process only values that were entered by the user.</td>
<td>Boolean values, integer values, casting booleans to integers, multiplication.</td>
</tr>
<tr>
<td>Sorting individual data vectors</td>
<td>Oblivous array sorting using a sorting network. Requires addition, multiplication, and comparison.</td>
</tr>
<tr>
<td>Calculating a composite indicator added value per employee</td>
<td>Division of secret values</td>
</tr>
<tr>
<td>Time series for financial indicators</td>
<td>Oblivous matrix sorting by a key column</td>
</tr>
</tbody>
</table>


Frequent itemset mining

• FIM and association mining are used in problems like collaborative filtering and shopping basket analysis.

• We implemented four frequent itemset mining algorithms on Sharemind (Apriori, Eclat and hybrids).

• We benchmarked the result on three datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Transactions</th>
<th>Items</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>mushroom</td>
<td>8124</td>
<td>119</td>
<td>19.3%</td>
</tr>
<tr>
<td>chess</td>
<td>3196</td>
<td>75</td>
<td>49.3%</td>
</tr>
<tr>
<td>retail</td>
<td>88163</td>
<td>16470</td>
<td>0.06%</td>
</tr>
</tbody>
</table>

To find frequent 11-item sets with support 2000 from the mushroom dataset Apriori needs to perform:

- 71 548 068 secure multiplications
- 8 926 secure greater-than comparisons

introducing
sharemind 3

This research was, in part, funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

Distribution Statement “A” (Approved for Public Release, Distribution Unlimited)
Adding more integer types

• The 32-bit integer type was hardcoded in Sharemind 2.

• However, our arithmetic protocols work in $\mathbb{Z}_{2^{2n}}$.

• Therefore, we can let the applications use booleans, 8-bit integers, 16-bit integers and 64-bit integers too.

• Smaller integers are more efficient since they use less communication and storage space.

• Currently, we have used unsigned integers, but we could adapt some of the protocols to signed integers.
Moving from integers to real numbers

- Given that we now have secret values with different bit depths, we can consider more complex data types.

- Our first target is to support floating point numbers.

- The standard IEEE 754 32-bit floating point number looks like this:

  1-bit sign  
  | 8-bit exp | significand (23 bits) |

- In Sharemind 3, we will use a modified version:

  1-bit sign  
  | exponent (16 bits) | significand (32 bits) |

Operations on floating point numbers

• We created data-independent circuits for adding and multiplying secret-shared floating point numbers.

• We do not process Not-a-Number cases. The protocols will not leak anything, but the result is undefined.

• After implementing addition and multiplication, we used Taylor series to compute basic functions such as sine, natural logarithm and square root.

Benchmarks on floating point ops

- An unoptimized addition takes 1.4 seconds.
- An unoptimized multiplication takes 0.4 seconds.
- Sine computation (5 elements) takes 8.8 seconds.
- Natural logarithm (6 elements) takes 13.5 seconds.
- Square root (5 elements) takes 10.1 seconds.
- All operations will be more efficient if parallelized.

Private string operations

• Given that we have different bit depth integers, we can use their arrays to implement ASCII and UTF strings.

• We can have two options for strings:
  
  • fixed length - potentially hides message length
  
  • variable length - string is as long as the array allocated for its storage

• Algorithms are slightly different for both cases.
Challenges for string algorithms

- Character manipulation, especially with UTF strings, requires bit-level operations to be efficient.

- However, if we use additive secret sharing, we need to use an expensive bit decomposition to manipulate bits.

- Therefore, for strings we will consider using XOR instead of addition in the secret sharing scheme.

- The same approach is useful also for other operations that need bit-level access to data - like secure AES.

Challenges for string algorithms

• Standard string algorithms can make decisions based on the data and jump ahead (e.g., in searching).

• To guarantee data independence, we often need to run naïve versions of algorithms and brute force searches.

• However, brute force can typically be heavily parallelized with SIMD operations.

• The alternative is to leak some (possibly aggregated) bits about the string’s contents, but such decisions should be driven by applications.
Adding support for new paradigms

- Sharemind 2 is limited to three parties and solutions based on secret sharing. The limits will be removed.

- However, there are interesting protocols and primitives out there that may work in the same application model:
  
  - Homomorphic encryption can help reduce the number of required servers and the communication.
  
  - Security in the consistent or malicious model will help protect against outages and hacking.

- We will be happy to implement interesting protocols.
Keeping it all easy for the developers

• We want all new data types to be available in the SecreC language for ease of use.

• We want the programming experience to be similar to existing techniques. Sharemind should be perceived similarly to a database and application server.

• Programs written in SecreC should be forward-compatible with new results in cryptography.

• The Sharemind machine is responsible for hiding the complexities of scheduling protocols.
Future work

• In the coming years we will be looking for novel practical protocol designs for inclusion in Sharemind.

• We want to extend the practical applicability of secure computation technology by building more prototypes.

• The technology can be used to solve real-life problems and we will jump to the opportunity of doing so.

• We will maintain a freely available toolkit for creating secure computation applications for academic use.

• We welcome all collaboration opportunities.
https://sharemind.cyber.ee/
sharemind@cyber.ee