Round-efficient OT and Oblivious Shuffle Protocols for Secure Multi-party Computation

Bingsheng Zhang[1,2]

Joint work with: Sven Laur[2] Jan Willemson[1,3]

[1]Cybernetica AS, Estonia
[2]University of Tartu, Estonia
[3]STACC, Estonia

Joint Estonian-Latvian Theory Days at Rakari
30.09.2010—03.10.2010

Outline

- Secure Multi-party Computation and Sharemind
- Existing Sharemind Elementary Protocols
- High degree Conjunction and Disjunction
- Oblivious Transfer Protocol
- Oblivious Shuffle Protocol

Multi-party Computation

Secure Multi-party Computation and Sharemind

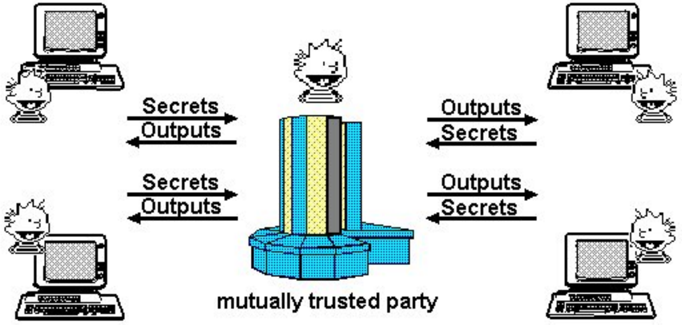


Figure: Ideal world

Secure Multi-party Computation and Sharemind

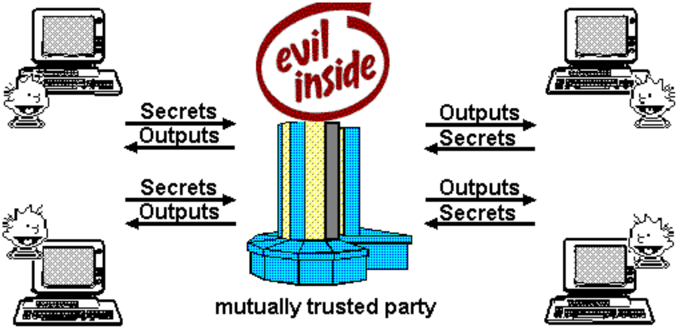


Figure: Ideal world

Secure Multi-party Computation and Sharemind

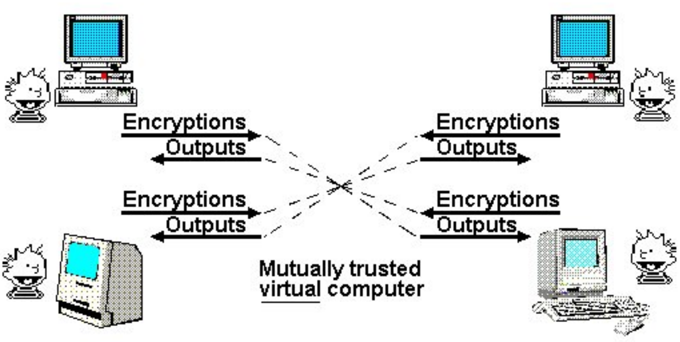


Figure: Real world

Secure Multi-party Computation and Sharemind

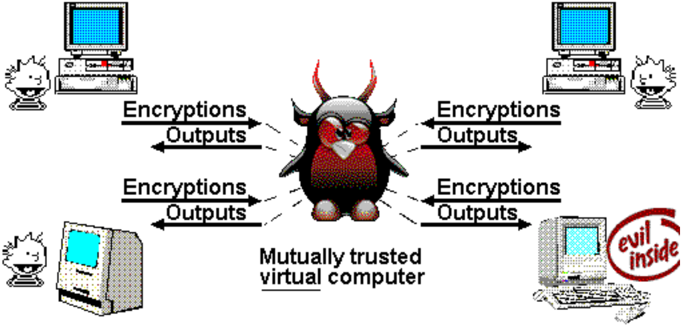


Figure: Real world

Secure Multi-party Computation and Sharemind

Linear Secret Sharing

A secret sharing scheme is specified by a randomized function Share, which takes in a secret value $x \in \mathbb{Z}_N$ and splits it into m pieces.

$$x \equiv x_1 + x_2 + \cdots + x_m \mod N \ .$$

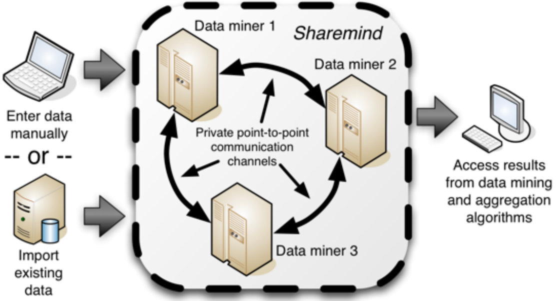
The vector of shares is commonly denoted by $[\![x]\!]$. In our case, $[\![x]\!] = (x_1, x_2, \ldots, x_m)$. N normally is p^e, where p is prime and $e \geq 1$. Otherwise, the Chinese Remainder Theorem allows us to split into a collection independent secret sharing schemes.

Secure Multi-party Computation and Sharemind

Sharemind

http://research.cyber.ee/sharemind/

Secure Multi-party Computation and Sharemind



- 3 parties that tolerant at most 1 passive corrupted party.
- Additive share in $\mathbb{Z}_{2^{32}}$.
- Currently used for privacy preserving datamining (PPDM).

Existing Sharemind Elementary Protocols

Existing Sharemind Elementary Protocols

Operation	Notation	Round count
Addition	$[\![x]\!] + [\![y]\!]$	$\tau_{\mathrm{ad}} = 0$
Multiplication	$[\![x]\!] \cdot [\![y]\!]$	$\tau_{\mathrm{mul}} = 1$
Smaller than	$[\![x]\!] \leq [\![y]\!]$	$\tau_{\mathrm{st}} = \mathrm{O}(\log \ell)$
Strictly less	$[\![x]\!] < [\![y]\!]$	$\tau_{\mathrm{sl}} = \mathrm{O}(\log \ell)$
Equality test	$[\![x]\!] \stackrel{?}{=} [\![y]\!]$	$\tau_{\mathrm{eq}} = \mathrm{O}(\log \ell)$
Bit-decomposition	$\mathrm{Decom}([\![x]\!])$	$\tau_{\mathrm{bd}} = \mathrm{O}(\log \ell)$

Table: Round complexity of common share-computing operations

High degree Conjunction

High degree Conjunction and Disjunction

Server's input: $[\![X_1]\!], \cdots, [\![X_k]\!]$ ($X_i \in \{0, 1\}$)

Server's output: $[\![Y]\!] = [\![X_1 \wedge \cdots \wedge X_k)]\!]$

1. All miners $\mathcal{M}_{p\in\{0,1,2\}}$ compute $[\![S]\!] = \sum_{i=1}^{k} [\![X_i]\!]$.
2. All miners $\mathcal{M}_{p\in\{0,1,2\}}$ call equality check protocol to compute $[\![S]\!] \stackrel{?}{=} k$.

Since addition can be done locally, the round complexity is $\tau_{\mathrm{eq}} = \mathrm{O}(\log \ell)$, where $\ell \geq \lceil \log k \rceil$. (We can do better, as k is public. Discussed in later slides.)

High degree Disjunction

High degree Conjunction and Disjunction

Server's input: $[\![X_1]\!], \cdots, [\![X_k]\!]$ ($X_i \in \{0, 1\}$)

Server's output: $[\![Y]\!] = [\![X_1 \vee \cdots \vee X_k)]\!]$

1. All miners $\mathcal{M}_{p \in \{0,1,2\}}$ compute $[\![S]\!] = \sum_{i=1}^{k} [\![X_i]\!]$.
2. All miners $\mathcal{M}_{p \in \{0,1,2\}}$ call equality check protocol to compute $[\![S]\!] \stackrel{?}{=} 0$.

Since addition can be done locally, the round complexity is $\tau_{\mathrm{eq}} = \mathrm{O}(\log \ell)$, where $\ell \geq \lceil \log k \rceil$. (We can do better, as k is public. Discussed in later slides.)

Oblivious Transfer

Oblivious Transfer Protocol

$(1, n)$ OT

Server's input: Shared database $[\![X_1]\!], \cdots, [\![X_n]\!]$

Server's output: $\perp$

Client's input: Shared index $[\![i]\!]$

Client's output: x_i

Oblivious Transfer Protocol

Query phase:

A client submits shares of i to the miner nodes.

Processing phase:

1. For $j \in \{1, \ldots, n\}$, miners evaluate in parallel: $[\![y_i]\!] \leftarrow [\![x_j]\!] \cdot (j \stackrel{?}{=} [\![i]\!])$.
2. Miners compute the shares of the reply: $[\![z]\!] \leftarrow [\![y_1]\!] + \cdots + [\![y_n]\!]$.

Reconstruction phase:

Miners send the shares of z to the client who reconstructs and outputs z.

Outline

Oblivious Transfer Protocol

$(1, n)$ OT

Whenever the database elements $x_i \in \{0,1\}^\ell$ and the index i can be embedded into the ring $\mathbb{Z}_N$, we can use high-degree conjunction to represent oblivious transfer as an arithmetic circuit

$$x_i = \sum_{j=1}^{n} (i \stackrel{?}{=} j) \cdot x_j = \sum_{j=1}^{n} \bigwedge_{k=1}^{\lceil \log(n+1) \rceil} (i_k \stackrel{?}{=} j_k) \cdot x_j \tag{1}$$

where i_k and j_k respectively denote the kth bit of i and j.

$$i_k \stackrel{?}{=} j_k \equiv \begin{cases} 1 - i_k, & \text{if } j_k = 0 \ , \\ i_k, & \text{if } j_k = 1 \ . \end{cases}$$

Oblivious Transfer Protocol

Theorem

The round complexity of the OT is $\tau_{\mathrm{eq}} + \tau_{\mathrm{mul}} + 1$ *where* τ_{mul} *and* τ_{eq} *are the round complexities of multiplication and equality test protocols. The protocol achieves security against malicious data donors and clients provided that the miner nodes follow the assumptions of share computing protocols.*

Next Problem

How to ensure the client's input are valid?

Oblivious Transfer Protocol

Lemma

$x_i \in \mathbb{Z}_{p^t}$ For uniformly chosen $r_i \in \mathbb{Z}_{p^t}$,
$\Pr[x_1 r_1 + \cdots + x_\ell r_\ell = 0] \leq \frac{1}{p}$ provided that some $x_k \neq 0$.

Public zero test batch

1. All miners $\mathcal{M}_{p\in\{0,1,2\}}$ compute $[\![S_t]\!] = [\![x_1]\!] r_{1,t} + \cdots + [\![x_\ell]\!] r_{\ell,t}$, for $t = \{1, 2, \cdots, \kappa\}$, where κ is security parameter. r_i is uniformly chosen from $\mathbb{Z}_{p^t}$.
2. All miners $\mathcal{M}_{p\in\{0,1,2\}}$ open $[\![S_t]\!]$. If all $S_t = 0$, then $x_1 = 0, \cdots, x_\ell = 0$.

Oblivious Transfer Protocol

Range Proof

There are 3 ways to ensure that the client's index bits $i_k \in \{0, 1\}$.

- Require the client send index bits that shared in $\mathbb{Z}_2$, then the server do share conversion to $\mathbb{Z}_{p^t}$.
- Require the client send the entire index $[\![i]\!]$, then the server calls Bit-decomposition protocol Decom($[\![i]\!]$) to i_k.
- Allow the client to send shared index bits in $\mathbb{Z}_{p^t}$, then the server ZK proves $i_k \in \{0, 1\}$.

Outline

Oblivious Transfer Protocol

Range Proof

- Miners can securely compute $\alpha = [\![i_1]\!](1 - [\![i_1]\!])r_1 + \cdots + [\![i_\ell]\!](1 - [\![i_\ell]\!])r_\ell$.
- Check whether $\alpha \stackrel{?}{=} 0$.
- Repeat κ times.

Note that is $b \in \{0, 1\}$ then $b * (1 - b) = 0$, and it reveals nothing about b. Therefore, we can even directly open $[\![b]\!] * (1 - [\![b]\!])$.

Equality test batch

To check $[\![x_1]\!] \stackrel{?}{=} [\![y_1]\!], \cdots, [\![x_\ell]\!] \stackrel{?}{=} [\![y_\ell]\!]$ can be done as

$$[\![S_t]\!] = ([\![x_1]\!] - [\![y_1]\!])r_{1,t} + \cdots + ([\![x_\ell]\!] - [\![y_\ell]\!])r_{\ell,t}$$

and check $[\![S_t]\!] \stackrel{?}{=} 0$, for $t = \{1, 2, \cdots, \kappa\}$.

Outline

Oblivious Transfer Protocol

Tweaked Version

As a first step towards lower communication complexity note that the right hand side of Eq. (1) can be viewed as multivariate polynomial with arguments $i_1, \ldots, i_\ell$:

$$f(i_1, \ldots, i_l) = \sum_{j=1}^{n} \prod_{k=1} (1 - i_k - j_k + 2 i_k j_k) \cdot x_j = \sum_{\mathcal{K} \subseteq \{1,\ldots,\ell\}} \alpha_{\mathcal{K}} \cdot \prod_{k \in \mathcal{K}} i_k$$

where the coefficients before monomials are linear combinations

$$\alpha_{\mathcal{K}}(x_1, \ldots, x_n) = \alpha_{\mathcal{K},1} x_1 + \cdots + \alpha_{\mathcal{K},n} x_n$$

with public constants $\alpha_{\mathcal{K},1}, \ldots, \alpha_{\mathcal{K},n} \in \{-1, 1\}$. For example, for the three element database

$$f(i_1, i_2) = x_1 i_1 + x_2 i_2 + (x_3 - x_2 - x_1) i_1 i_2 \quad .$$

Oblivious Shuffle Protocol

Oblivious Shuffle Protocol

Oblivious Shuffle Protocol

Server's input: Shared database $[\![x_1]\!], \cdots, [\![x_n]\!]$

Server's output: Shuffled database $[\![x_{\pi(1)}]\!], \cdots, [\![x_{\pi(n-1)}]\!]$

1. For $p \in \{0, 1, 2\}$, all miners do:
    1. $\mathcal{M}_p$ shares its shares additively to other parties.
    2. $\mathcal{M}_{p-1}, \mathcal{M}_{p+1}$ compute additives 2-out-of-2 shares of $x_1, \ldots, x_n$.
    3. $\mathcal{M}_{p-1}$ and $\mathcal{M}_{p+1}$ jointly pick a random permutation π_p. They permute the shared database locally and set $[\![x_i]\!] \leftarrow [\![x_{\pi_p(i)}]\!]$
    4. $\mathcal{M}_{p-1}$ and $\mathcal{M}_{p+1}$ share their shares additively for all parties $\mathcal{M}_{p\in\{0,1,2\}}$.
    5. All parties compute additive sharings of $x_{\pi(1)}, \ldots, x_{\pi(n)}$.

Outline

Oblivious Shuffle Protocol

Theorem

For any linear secret sharing scheme, there exists a oblivious shuffle protocol secure in the semihonest model such that round complexity and computational complexity is $\mathrm{O}(2^m n \log n)$ *where n is the database size and m is the number of miner nodes.*

Implementation for random permutation

We propose a solution using a block cipher, say, 128-bit AES. In our solution, the parties, Alice and Bob, pick the random keys k_a and k_b, respectively, and send them to each other. Then they compute $\sigma(i) \leftarrow AES_{k_a \oplus k_b}(i)$ $(i = 0, \ldots, n-1)$. Note that, in practice, since $n << 2^{128}$, $\sigma(i)$ is sparse. Hence, in order to get a real permutation, Alice and Bob can make an array of pairs $(i, \sigma(i))$ and sort the array according to $\sigma(i)$. The resulting permutation of the first elements of the pairs will be π.

Oblivious Shuffle Protocol

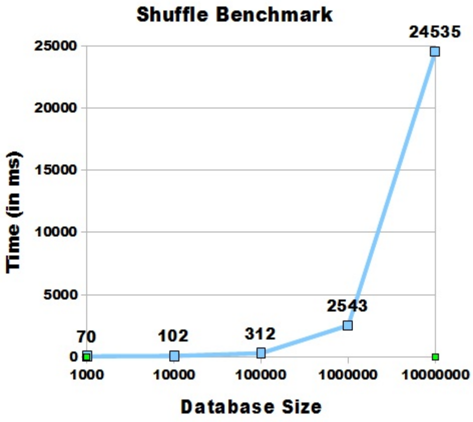


Figure: Benchmark for oblivious shuffle protocol

Oblivious Shuffle Protocol

Thank You!

Questions?