Multi-Party Computation with Conversion of Secret Sharing

Josef Pieprzyk

joint work with
Hossein Ghodosi and Ron Steinfeld

MACQUARIE UNIVERSITY
SYDNEY ~ AUSTRALIA

NTU, Singapore, September 2011

Road Map

- Introduction
- Background
- Our Contribution
- Building Blocks
    - Additive secret sharing
    - Multiplicative secret sharing
- Computations using Hybrid Secret Sharing
- Conversion of Multiplicative Shares into Additive Shares
- MPC Protocols with Hybrid Secret Sharing
- Conclusions

What is multi-party computation (MPC) protocol?

Assume that

- there is a collection of participants

$$\{P_1, P_2, \ldots, P_n\} \text{ and a function } Y = F(x_1, \ldots, x_n)$$

- each participant

$$P_i \text{ holds a private input } x_i \text{ for } i = 1, \ldots, n$$

- a MPC protocol allows participants to evaluate the function F in such a way that at the end of the protocol
  - all participants learn Y and
  - their inputs remain private

Introduction – Ideal Process

Assume that there is a trusted party (TP). Then we can run the following protocol:

- Participants submit their inputs to TP
- TP evaluates the function
- TP distributes the result to all participants

Problem:
What happens if the participants cannot agree on a TP?

Two possible frameworks:

- **computationally secure**
  breaking the security of the protocol implies that the adversary is able to solve a problem (in polynomial time) that is believed to be intractable
- **unconditionally secure**
  the adversary cannot break the system by any method better than by guessing private inputs

Two generic types of adversary

- **passive** – also called "honest but curious". The corrupted participants follow the protocol but they try to learn private information
- **active** – corrupted participants behave arbitrarily or maliciously

Background

Early developments

- **Yao**, 1982 – the concept of secure MPC
- **Goldreich, Micali and Wigderson**, 1987 – solution with computational security
- **Ben-Or, Goldwasser, and Wigderson** and independently **Chaum, Crepeau, and Damgård**, 1988 – solutions with unconditional security

Background – the BGW/CCD Solution

Assume that

$$Y = F(x_1, \ldots, x_n) = \sum x_1^{\alpha_1} \cdots x_n^{\alpha_n}$$

can be represented by a polynomial (sum of products) over $GP(p)$. The participants

- collectively evaluate products
- collectively evaluate the sums and finding shares of Y

Note 1

At the initial stage, each participant P_i distributes their shares x_i using Shamir secret sharing with the polynomial

$$f_i(x) = x_i + a_1 x + \ldots, + a_t x^t$$

Note 2

Computation of products is highly interactive – the multiplication of two polynomials of degree t gives a polynomial of degree $2t$. Reduction of the degree requires $n \geq 2t + 1$

Note 3

Computation of sums is easy.

Background – Unconditionally Secure MPC – Standard Model

1. In the presence of a passive adversary, no set of size

$$t < n/2$$

   of participants learns any additional information, other than what they could derive from their private inputs and the output of the protocol.

2. In the presence of an active adversary, no set of size

$$t < n/3$$

   of participants can learn any additional information or disrupt the protocol.

Background - Trusted Setup Assumptions

We study **the trusted model** when the participants may interact with trusted party BEFORE receiving their private inputs.

Previous results:

- **Killian**, 1988 – protocols that are secure against dishonest majority
- **Beaver**, 1995 – unconditionally secure OT protocols in trusted setup model

Background – MPC Efficiency

Two measures:

- **round complexity** – maximum number of rounds in the protocol
- **communication complexity** – maximum number of bits exchanged during a run of the protocol

MPC protocols in trusted setup model with Beaver's pre-computation construction are based on OT and have the following communication complexity

- $O(m \cdot n^2)$ field elements over $GF(2)$ or
- $O((\log p + k) \cdot m \cdot n^2)$ over $GF(p)$

where m is the number of multiplication gates.

Our Contribution

MPC protocols in unconditionally secure setting to evaluate

$$F(x_1, \ldots, x_n) = F_L(x_1, \ldots, x_n) + F_{C_1}(x_1, \ldots, x_n) + \ldots + F_{C_\ell}(x_1, \ldots, x_n)$$

where

- $F_L(x_1, \ldots, x_n)$ denotes the linear component
- $F_{C_i}(x_1, \ldots, x_n)$ denotes monomials, where $i = 1, \ldots, \ell$

Our Contribution – Details

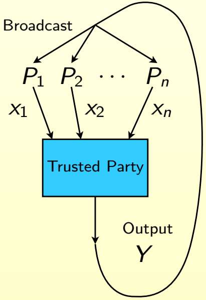
- The linear component, and every monomial (regardless of its depth), can be computed with no interaction using **hybrid secret sharing**
- The value of function is computed by converting

multiplicative secret sharing into **an additive secret sharing**

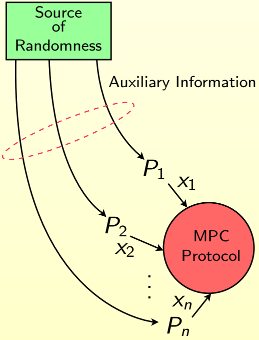
  with a help of **auxiliary information** distributed to the participants in a trusted setup phase
- Our MPC protocol allows the adversary to corrupt up to $n-1$ participants and does not use OT
- The communication complexity of our protocol is $O(\ell \cdot n^2)$ field elements

Our Contribution – Comparison of Models

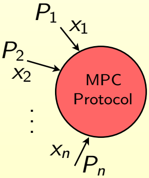
Ideal Process ($t < n$)



Hybrid Model ($t < n$)



Distributed Model ($t < n/2$)



Building Blocks – Assumptions

- We have complete synchronous network with private channels available between every pairs of n collaborating participants.
- The adversary is passive with unlimited computing capabilities.

Definition

A MPC protocol is t-private if after completion of the protocol

no subset of t participants

learns any information (about uncorrupted participant private inputs) more than what they could derive from their private inputs and the output of the protocol.

Building Blocks – Hybrid Secret Sharing

Definition

Let $\mathcal{K}$ be the domain of possible secrets, and let $\mathcal{S}$ be the domain of possible shares. A hybrid (t, n)-threshold scheme determines two sets of functions

$$F_A : \mathcal{S}^t \to \mathcal{K} \quad \text{and} \quad G_A : \mathcal{S}^t \to \mathcal{K}$$

defined for every $A \subseteq \{1, \ldots, n\}$ with $|A| = t$, such that for any given set of t shareholders each function defines the value of the secret, i.e.,

$$K = F_A(s_{i_1}, \ldots, s_{i_t}) = G_A(s'_{i_1}, \ldots, s'_{i_t})$$

We refer to such secret sharing scheme as a (F, G)-hybrid (t, n)-threshold scheme.

We use the following instantiations

- F is the modular addition, and
- G is the modular multiplication over $GF(p)$

Building Blocks – Additive (n, n) Secret Sharing

Share Distribution

The dealer chooses $n-1$ shares $s_1, \ldots, s_{n-1}$ at random from all possible values in $GF(p)$, and computes

$$K = s_n + \Sigma_{i=1}^{n-1} s_i \pmod{p}$$

The dealer sends (privately) share s_i to participant P_i $(i = 1, \ldots, n)$.

Secret Reconstruction

All participants pool their shares and reconstruct the secret

$$K = \Sigma_{i=1}^{n} s_i \pmod{p}$$

Share Distribution

The dealer chooses $n-1$ independent and uniformly random shares $s_1, \ldots, s_{n-1}$ from $GF(p)^*$, and computes

$$s_n = K \times (\Pi_{i=1}^{n-1} s_i)^{-1} \pmod p$$

For $i = 1, \ldots, n$, the dealer privately sends share s_i to participant P_i.

Secret Reconstruction

All participants pool their shares and reconstruct the secret

$$K = \Pi_{i=1}^{n} s_i \pmod p$$

Computations for Linear Functions

Given two secrets x_i and x_k shared using (n, n)-threshold SS

$$P_j \xleftarrow{s_{i,j}} x_i \text{ and } P_j \xleftarrow{s_{k,j}} x_k$$

- In order to compute shares of $x_i + x_k$, each participant P_j computes

  $$P_j \xleftarrow{s_j^{i+k} = s_{i,j} + s_{k,j}} (x_i + x_k),$$

  where s_j^{i+k} is the share of P_j associated with the secret $x_i + x_k$. This is because the additive (n, n)-threshold scheme is $(+, +)$-homomorphic.
- For every known scalar $c \in GF(p)$ and each secret input x_i, then

  $$P_j \xleftarrow{c \cdot s_{i,j}} c \cdot x_i$$

Computations for Linear Functions

- Given a secret x_i and a scalar $c \in GF(p)$, how to computer shares of
$$c + x_i$$
This can be done at least in two ways:
  (i) Share the value c amongst all participants, using the additive (n, n)-threshold scheme, i.e.
  $$P_j \xleftarrow{c_j} c$$
  Then each participant
  $$P_j \quad \xleftarrow{c_j + s_{i,j}} \quad (c + x_i),$$
  where $j = 1, \ldots, n$.
  (ii) A more efficient way is that only a designated participant, P_ℓ, $\ell \in \{1, \ldots, n\}$ (who is chosen by all participants) adds c to his share from x_i, i.e., computes $c + s_{i,\ell}$.
- Computation of an additive inverse – each participant P_j computes the additive inverse of his share.

Thus, every linear function with n inputs can be computed with no interaction.

Computation of Monomials

Given n secret inputs $x_1, \ldots, x_n$, of participants $P_1, \ldots, P_n$. They are shared using the multiplicative (n, n)-threshold scheme.
Assume that

$$P_j \xleftarrow{m_{i,j}} x_i \text{ and } P_j \xleftarrow{m_{k,j}} x_k$$

Then

$$P_j \xleftarrow{m_j^{i+k} = m_{i,j} \cdot m_{k,j}} x_i \cdot x_k,$$

This can be done as the multiplicative (n, n)-threshold scheme is $(\times, \times)$-homomorphic.

Computations of Monomials

- Given a secret x_i and a scalar $c \in GF(p)$, how to computer shares of
$$c \cdot x_i$$
This can be done at least in two ways:
  (i) Share the value c amongst all participants, using the multiplicative (n, n)-threshold scheme, i.e.
  $$P_j \xleftarrow{c_j} c$$
  Then each participant
  $$P_j \quad \xleftarrow{c_j \cdot s_{i,j}} \quad (c \cdot x_i),$$
  where $j = 1, \ldots, n$.
  (ii) A more efficient way is that only a designated participant, P_ℓ, $\ell \in \{1, \ldots, n\}$ (who is chosen by all participants) multiplies c by his share of x_i, i.e., computes $c \cdot s_{i,\ell}$.
- Computation of an multiplicative inverse – each participant P_j computes the multiplicative inverse of his share.

Thus, every multiplication gate, regardless of its depth, can be computed with no interaction.

Inputs:

- **Shares –** Each participant P_j ($j = 1, \ldots, n$) owns a share m_j associated to a multiplicative (n, n)-threshold scheme over $GF(p)$, such that

$$m_1 \times \ldots \times m_n = K \pmod{p}$$

  where $K \in GF(p)^*$ is the secret.

- **Auxiliary information –** Each participant P_j ($j = 1, \ldots, n$) is given a set of n elements $\alpha_{1,j}, \ldots, \alpha_{n,j}$, such that

$$\Sigma_{i=1}^{n} u_i \equiv 1 \pmod{p}, \text{ where } u_i \equiv \Pi_{j=1}^{n} \alpha_{i,j} \pmod{p}.$$

Conversion of Multiplicative Shares to Additive Shares

The $\alpha_{i,j}$'s are generated as follows:

- Pick $u_1, \ldots, u_n$ in $GF(p)$ as shares for an additve (n, n)-threshold sharing of 1, i.e. pick $u_1, \ldots, u_{n-1}$ independently and uniformly at random from $GF(p)$ and compute

$$u_n \equiv 1 - \sum_{i=1}^{n-1} u_i \pmod{p} \in GF(p)$$

- For $i = 1, \ldots, n$, pick $n - 1$ independent and uniformly random elements $\{\alpha_{i,j}\}_{j \neq i}$ from $GF(p)^*$ and compute

$$\alpha_{i,i} \equiv u_i \cdot \Big(\prod_{j \neq i} \alpha_{i,j}\Big)^{-1} \pmod{p} \in GF(p)$$

(note that $\alpha_{i,i} = 0$ if and only if $u_i = 0$).

Conversion of Multiplicative Shares to Additive Shares

Conversion:

- Each participant P_j $(j = 1, \dots, n)$ sends $v_{i,j} = \alpha_{i,j} m_j \pmod{p}$ (for $i = 1, \dots, n$) to participant P_i.

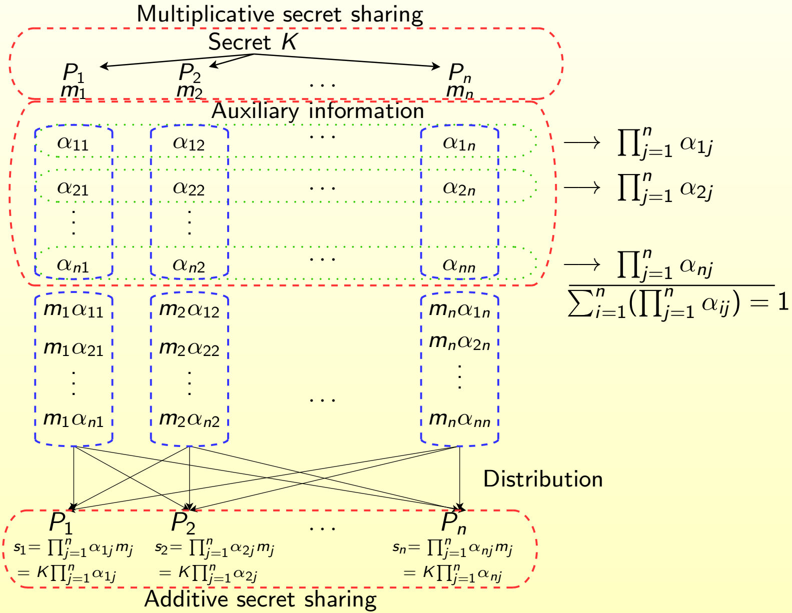
Outputs:

- Participant P_i $(i = 1, \dots, n)$ computes

$$s_i = \prod_{j=1}^{n} v_{i,j} = \Pi_{j=1}^{n} \alpha_{i,j} m_j = u_i K \pmod{p}$$

as his share of K, associated to an additive (n, n)-threshold scheme.

Conversion of Multiplicative Shares to Additive Shares



Correctness – Each participant P_i $(i = 1, \ldots, n)$ receives $n - 1$ values $\alpha_{i,j} m_j$ from participants P_j $(j = 1, \ldots, n,\ j \neq i)$. Knowing $\alpha_{i,i}$, m_i, and the received information, P_i computes

$$s_i = \Pi_{j=1}^{n} \alpha_{i,j} m_j = \Pi_{j=1}^{n} \alpha_{i,j} K,$$

as his share corresponding to an additive (n, n)-threshold scheme. The conversion protocol is correct, because at the end of the protocol, the sum of the computed shares of all participants is:

$$\Sigma_{i=1}^{n} s_i = \Sigma_{i=1}^{n} \left(\Pi_{j=1}^{n} \alpha_{i,j} K \right) = \left(\Sigma_{i=1}^{n} \left(\Pi_{j=1}^{n} \alpha_{i,j} \right) \right) K = K \pmod{p}.$$

Security – Let $P_1, \ldots, P_{n-1}$ be the set of $n-1$ participants who collude in order to breach the privacy of the proposed conversion protocol via learning some information about the secret, K.

They collectively know

- $n-1$ shares $m_1, \ldots, m_{n-1}$ associated with a multiplicative (n,n)-threshold scheme
- auxiliary information $\alpha_{i,j}$ $(i = 1, \ldots, n$ and $j = 1, \ldots, n-1)$ and
- $n-1$ values $v_{i,n} \equiv m_n \alpha_{i,n} \pmod{p}$ $(i = 1, \ldots, n-1)$ received from the honest participant P_n.

To demonstrate the security, we need to show that all these known values can be perfectly simulated by the collusion $P_1, \ldots, P_{n-1}$ by itself, independently of the secret K.

MPC Protocols with Hybrid Secret Sharing

- Initialization – Each participant P_i $(i = 1, \ldots, n)$ distributes his private input $x_i \in GF(p)^*$ amongst all participants, using the additive and multiplicative (n, n)-threshold schemes
- Computation – In order to compute the function $F(x_1, \ldots, x_n) = F_L(.) + F_{C_1}(.) + \ldots, F_{C_\ell}(.)$, each participant P_i $(i = 1, \ldots, n)$ computes $F_L(.)$ and all monomials $F_{C_j}(.)$ $(j = 1, \ldots, \ell)$.
- Reconstruction – Let $A_{i,j}$ be the share of participant P_i associated with monomial $F_{C_j}(.)$, in an additive (n, n)-threshold format. Now, P_i computes $Y_i = A_{i,0} + A_{i,1} + \ldots, A_{i,\ell}$, where $A_{i,0}$ is the share of P_i associated with the linear component $F_L(.)$ (if it exists). They can pool their shares and compute the function value, using

$$Y = \Sigma_{i=1}^{n} Y_i \pmod{p}.$$

Corollary

- *Let $F : (GF(p)^*)^n \to GF(p)$ denote a n-variate polynomial over $GF(p)$ (with inputs restricted to $GF(p)^*$) having ℓ non-linear monomials.*
- *Assume a setup phase in which an auxiliary information (which is independent of the function inputs and consists of $O(\ell \cdot n^2)$ elements of $GF(p)$) is privately distributed among the n participants.*

Then the function F can be computed by the n participants such that

- *no subset of $n-1$ participants can learn any additional information, other than what they can learn from their inputs and the protocol's output.*
- *the protocol has a total communication complexity of $O(\ell \cdot n^2)$ elements of $GF(p)$.*

Remark 1 – The protocol can still be used for $GF(2)$. To see that this is possible, it suffices to show that a two-input NAND gate can be encoded into a polynomial over non-zero inputs over the larger field.
Consider

$$h(x_1, x_2) = 2x_1^2x_2^2 + 3x_1x_2 + 2$$

over the field $GF(5)$. It is easy to verify that

$$h(2,2) = h(1,2) = h(2,1) = 1 \text{ and } h(1,1) = 2$$

so h computes an encoding of the $GF(2)$ NAND function over $GF(5)$, where we encode the $GF(2)$ values 0 (respectively 1) as the $GF(5)$ non-zero values 2 (respectively 1).

Remark 2 – For security reasons, any set of auxiliary information should be used only once. That is, for computing a function containing ℓ monomials, ℓ sets of auxiliary information should be provided to the participants.

Conclusions

- How to extend our results for active adversary?
- Hybrid secret sharing is an interesting tool, and its properties need more investigation. How the conversion depends on the access structure and required homomorphic properties.
- If we would like to efficiently extend our approach to arithmetic circuits of an arbitrary depth, then we need a conversion of additive secret sharing into its multiplicative version. So far, we do not know how to do this.