Applications of Secure Multiparty Computation: Robotics as a Case Study

Thesis submitted in partial fulfillment
of the requirements for the degree of

MASTER of SCIENCE by RESEARCH
in
COMPUTER SCIENCE

by

SARAT CHANDRA ADDEPALLI
200605021
`sarat_a@research.iiit.ac.in`

CENTRE for SECURITY, THEORY and ALGORITHMS RESEARCH
International Institute of Information Technology
Hyderabad - 500 032, INDIA
DECEMBER 2009

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Applications of Secure Multiparty Computation: Robotics as a Case Study" by Sarat Chandra Addepalli, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Advisor: Dr. K. Srinathan

To My Parents and Brother

Acknowledgments

Acknowledgements goes here ...

Abstract

Abstract goes here ...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Introduction goes here...

1.1 First Section

Text of section 1 goes here...

This is to insert a table

This is to insert a figure

1.2 Second Section

Text of section 2 goes here...

Few suggestions

Method	Frobnability
Theirs	Frumpy
Yours	Frobbly
Ours	Makes one’s heart Frob

Table 1.1: Results. Ours is better.

1.2.1 Mathematics

Please number all of your sections and displayed equations. It is important for readers to be able to refer to any particular equation. Just because you didn't refer to it in the text doesn't mean some future reader might not need to refer to it. It is cumbersome to have to use circumlocutions like "the equation second from the top of page 3 column 1". (Note that the ruler will not be present in the final copy, so is not an alternative to equation numbers). All authors will benefit from reading Mermin's description of how to write mathematics (see math.pdf).

1.2.2 Footnotes

Please use footnotes[1] sparingly. Indeed, try to avoid footnotes altogether and include necessary peripheral observations in the text (within parentheses, if you prefer, as in this sentence). If you wish to use a footnote, place it at the bottom of the column on the page on which it is referenced. Use Times 8-point type, single-spaced.

1.2.3 References

List and number all bibliographical references in 9-point Times, single-spaced, at the end of your paper. When referenced in the text, enclose the citation number in square brackets, for example [2]. Where appropriate, include the name(s) of editors of referenced books.

1.2.4 Illustrations, graphs, and photographs

All graphics should be centered. Please ensure that any point you wish to make is resolvable in a printed copy of the paper. Resize fonts in figures to match the font in the body text, and choose line widths which render effectively in print. Many readers (and reviewers), even of an electronic copy, will choose to print your paper in order to read it. You cannot insist that they do otherwise, and therefore must not assume that they can zoom in to see tiny details on a graphic.

Referring to [1], we state that so and so.

1.2.5 Color

Color is valuable, and will be visible to readers of the electronic copy. However ensure that, when printed on a monochrome printer, no important information is lost by the conversion to grayscale.

For more suggestions to improve your document, see preparationGuide.pdf

[1]This is what a footnote looks like. It often distracts the reader from the main flow of the argument.

Chapter 2

Secure Multiparty Computation and its Primitives

2.1 SMPC Primitives

2.1.1 Oblivious Transfer

Oblivious transfer is a type of protocol in which a sender sends a potential subset of messages to the receiver but is oblivious as to whether which ones (if any) were received.

Michael Rabin [5] introduced the first kind of oblivious transfer protocol, in which the sender sends a message with probability $\frac{1}{2}$, but is oblivious whether the receiver received it or not. A more useful form of this protocol called the *1-2 Oblivious Transfer* was developed by Shimon Even, Oded Goldreich and Abraham Lempel [3]. This protocol addresses the following problem: the sender has two messages m_0 and m_1, and the receiver wants one of the messages m_b, but the sender needs to remain oblivious about b, and the reciever needs to be oblivious about the value of $m_{\bar{b}}$.

In the algorithm 1, we show one possible implementation of *1-2 Oblivious Transfer* using the RSA cryptosystem [6], but in general, the oblivious transfer protocol can be implemented using any "trap-door" functions, such as one-way-functions and public key cryptosystems, or even Shamir's secret sharing [8].

2.1.2 Shamir's Secret Sharing

This SMPC primitive addresses the following problem: suppose a group of treasure hunters would like to lock a safe in such a way that it can't be opened unless there are atleast five (say) of them present at any given time. How many locks and keys would be required for this?

In [7], Shamir proposes a way of sharing a secret among n players, such that any k or more players can reconstruct the secret, but no set of $k-1$ or less players can do so. This is called a (k, n) *secret sharing scheme*, and is achieved by using $k-1$ degree polynomials as described follows:

The player who wishes to share a secret first chooses a $k-1$ degree secret random polynomial (by choosing the $k-1$ coefficients r_1 to r_k), say $f(x)$, and sets the constant term to the value of the secret. He then calculates the value of the "share" to be sent to each player i, as $f(i)$. With this, it is ensured

```
Require: A has two messages, $m_0, m_1$, and wants to send exactly one of them to B, but does not want to know which B receives.
  A generates a RSA key pair, comprising the modulus $N$, the public exponent $e$ and the private exponent $d$
  A also generates two random values, $x_0, x_1$ and sends them to B along with the public modulus and exponent.
  B picks $b$ to be either 0 or 1, and selects either the first or second $x_b$.
  B generates a random value $k$ and blinds $x_b$ by computing $v = (x_b + k^e) \mod N$, which he sends to A.
  A doesn't know which of $x_0$ and $x_1$ B chose, so she attempts to unblind with both of her random messages and comes up with two possible values for $k$: $k_0 = (v - x_0)^d \mod N$ and $k_1 = (v - x_1)^d \mod N$. One of these will be equal to $k$ since it will correctly decrypt, while the other will produce another random value that does not reveal any information about $k$.
  A blinds the two secret messages with each of the possible keys, $m'_0 = m_0 + k_0$ and $m'_1 = m_1 + k_1$, and sends them both to B.
  B knows which of the two messages can be unblinded with $k$, so he is able to compute exactly one of the messages $m_b = m'_b - k$
Ensure: each player $i$ has a share $v_i$ of the secret $v$
```

Algorithm 1: On 1-2 oblivious transfer

```
Require: A player has a secret value $v$ which he has to share
  select a random number $r$
  $f(x) = v + r_1 x + r_2 x^2 + \ldots + r_{k-1} x^{k-1}$
  for all players $i$ do
    send the value $v_i = f(i) = v + r_1 i + r_2 i^2 + \ldots + r_{k-1} i^{k-1}$ to player $i$
  end for
Ensure: each player $i$ has a share $v_i$ of the secret $v$
```

Algorithm 2: On sharing a secret

that each player has a "share" of the secret, which he may reconstruct if and only if atleast $k-1$ other players are willing to do so.

```
**Require:** Each player has a share $v_i$ of the secret. $\mathcal{P}$ is the set of players who wish to reconstruct the secret.
  **for** all players $i$ **do**
    all the players send their shares to players in $\mathcal{P}$
  **end for**
  **for** all players in $\mathcal{P}$ **do**
    construct a polynomial $f(x) = a + bx$ which passes through all the points $(i, v_i)$
    return the secret $v = a$
  **end for**
**Ensure:** each player knows the secret $v$
```

Algorithm 3: On reconstructing a secret

Notice, that a $k-1$ degree polynomial's equation can be reconstructed with the knowledge of any k points on the curve (as in the case of any k players colluding), but any set of $k-1$ or less points will yield no information about the equation of the curve (which means that any set of $k-1$ players or less will not be able to reconstruct the secret!), and thus the objective is achieved.

2.1.2.1 Secret Addition

Armed with the subroutines for secret sharing and reconstruction, we can do additional things with it. Notice that the "share"s are but numerical values, infact plot points on a polynomial curve. As with any polynomial, we can add, multiply, and do other arithmetic operations, but what does adding polynomials mean, in this sense?

What we mean by doing arithmetic operations on the polynomial is the following: Supposing two secrets a and b were shared by two different players according to their choice of random private polynomials $f_a(x)$ and $f_b(x)$. Let the shares of the i th player be a_i and b_i respectively, of the two secrets. Now if the each of the individual players were to add their own shares, this would result in each of them creating c_i, say, where $c_i = a_i + b_i$. But these individual shares would be plot points on the polynomial curve $f_c(x) = f_a(x) + f_b(x)$, which means that if the reconstruction subroutine were to be run on these new shares, it would result in the reconstruction of the value $a + b$! And similar to the case of sharing a secret, even when the shares are added, the privacy of the secret still remains intact, that is, all the players (except those two who shared a and b) still have no information about the value of the individual secrets.

This process is illustrated in the algorithm 4, where each of the individual players share their private secrets, and add the individual shares. This results in the addition of each of the secrets, so that at the end of this routine, the group of individual players as a whole are each left with a share of the total sum of the private inputs of all of the players.

```
Require: $n$ is the number of players, each with private input $s_i$.
  {Phase 1: Sharing}
  for each player $i$ do
    Share secret $s_i$
    for each player $j$ do
      send $j$'s share $s_{ij}$ to player $j$
    end for
  end for
  for each player $j$ do
    $sum_j = \sum_{i=1}^{n} s_{ij}$
  end for
Ensure: each player is left with $sum_j$, the share of the sum of the private inputs $s_i$.
```

Algorithm 4: Computing Secure Addition

2.1.2.2 Secret Multiplication

For multiplication, we have a protocol given by [4], which takes multiple rounds of communication, and is a little complex. Here, we present a simple and elegant protocol, with an initial preprocessing stage. The preprocessing uses the protocol of [4], but only once. All subsequent multiplications can be executed, using the protocol 5. The advantage with this protocol is that there is lesser communication complexity, and the computation can be done on the local shares.

We look at the simpler case of multiplying two secrets, α and β, which are assumed to be already shared among the n players. In the initial preprocessing stage, we share two random numbers x and y, and their product xy, while keeping the values themselves secret. The protocol given in [4] is used to calculate xy.

The numbers α, β, x, y, xy are now shared among the players. Now each player i computes the values $\lambda_i = \alpha_i - x_i$ and $\lambda'_i = \beta_i - y_i$, where η_i stands for the player i's share of the value of η. Now the values of λ and λ' are reconstructed, that is, made public among the players. This affords no information gain about the values of α and β, as x and y are still private.

Due to the linearity of the secret sharing protocol, we know that $\lambda = \alpha - x$ and $\lambda' = \beta - y$. Also,

$$\begin{aligned}
\alpha\beta &= (\alpha - x + x)(\beta - y + y) \\
&= (\alpha - x)(\beta - y) + y(\alpha - x) + x(\beta - y) + xy \\
&= \lambda\lambda' + y\lambda + x\lambda' + xy
\end{aligned}$$

Since the respective shares of x, y and xy are already with the players, the computation of $\alpha\beta$ (rather the computation of the *shares* of $\alpha\beta$) can be done locally. Since all the players have the shares of the product, the required product can be reconstructed easily. This minimizes the cost of repeated multiplication, which is very communication expensive. Also, the shares of x and y can be reused for all further multiplications, without any loss of information.

```
Require: All the players have shares of random numbers $x$ and $y$ and their product $xy$. They also have
  shares of $\alpha$ and $\beta$, of which they need to evaluate the product.
  {Phase 1: SHARING}
  for player $i$ do
    $\lambda_i = \alpha_i - x_i$
    $\lambda'_i = \beta_i - y_i$
    reconstruct the values $\lambda$ and $\lambda'$
    $\alpha\beta_i = \lambda\lambda' + y_i\lambda + x_i\lambda' + xy_i$
  end for
```

Algorithm 5: Computing Secure Multiplication

2.1.3 Privacy Preserving Union

Owing to the protocols given above for secure addition and multiplication, we can rest assured that any algebraic function can be evaluated securely, since every function is but a series of repeated parametric additions and multiplications, and their inverses (subtractions and divisions). Also, assuming that every player can locally generate a random number, we can also safely say that every probabilistic polynomial time algorithm can be executed securely by the players. Formal proofs of these claims exist in literature, but they are out of the scope of this paper. However, in addition to these protocols, we also present some specific purpose protocols such as this one for Secure Union.

Let every player i have a private set S_i, where $S_i \subset S$ is the universal set, of size k. Without loss of generality, we assume that the set S is ordered, and that $s_i | 1 \leq i \leq k$ are the elements. Each player creates a binary array, $A_i = [a_{ij}]$, where $a_{ij} = 1$ if $s_j \in S_i$ and 0 otherwise. Now it remains to logically OR the arrays A_i bitwise, and create the union set U. Note that, in binary operations, the AND gate corresponds to multiplication, and the XOR gate corresponds to addition. We have already described protocols for multiplication and addition. The only difference here will be that the coefficients, operands and operations are in binary logic. Also, the OR gate can be decomposed to the AND and XOR gate. And since we know the protocols for these gates, the OR gate can also be computed. The algorithm for the construction of secure union set is given below.

The protocol for secure intersection is similar to this, where instead of constructing the XOR of the bits, just an AND of the bits is enough. Which is to say that a modified multiplication protocol will be the same as a secure intersection protocol.

2.1.4 Secure Maximisation

We suppose that the n players each have a private input value, a probability, in the range $[0, 1]$, and that they wish to find out the maximum value amongst them, and the player who holds it. The protocol mimics the binary search over a given range. First, the interval is divided into two halves, $[0, 0.5]$ and $[0.5, 1]$, and the number of players in the greater interval are counted. Then, the interval with the largest

```
**Require:** player $i$ has the array $A_i = [a_{ij}]$ which represents the set $S_i$
  **for** bit $j$ from $j = 1$ to $k$ **do**
    **for** all players **do**
      share bit $a_{ij}$ in binary $\{$Use the protocol 4$\}$
      construct bit $XORa_j = XOR^n_{t=1} a_{ij}$ using protocol 4
      construct bit $ANDa_j = AND^n_{t=1} a_{ij}$ using protocol 5
      construct bit $ORa_j = XORa_j XORANDa_j$ using protocol 4
      element $u_j = ORa_j$
    **end for**
  **end for**
  return set $U = \{u_j | j = 1$ to $k\}$
```

Algorithm 6: Constructing Secure Union

value is subdivided into two halves, and the same process is repeated recursively, until only one player remains in the greater interval. This value is then shared.

All the above mentioned algorithms and protocols end with the actual answer being shared. For the answer to be revealed, one can always run the reconstruction algorithm (protocol 3).

```
Require: each player has a private value $p_i$ within the range $[0, 1]$
  $A = 0, B = 1$
  loop
    $C = (A + B)/2$
    for each player $i$ do
      $a_i = 0$; $b_i = 0$
      if value $p_i < C$ then
        $a_i = 1$
      else
        $b_i = 1$
      end if
      {Use the Secure Addition protocol 4}
      add all players' private $a_i$'s and $b_i$'s, call them $a$ and $b$
      reconstruct $a$ and $b$
      if $b \geq 1$ then
        $A = C$ and continue loop
      else if $b = 1$ then
        exit loop {Player with max value found}
      else if $a \geq 1$ then
        $B = C$ and continue loop
      else
        exit loop {Player with max value found}
      end if
    end for
  end loop
  {All players know for themselves whether they are or not the one with the max value}
  for all players $i$ do
    if $a_i = 1$ or $b_i = 1$ then
      share value $p_i$
    else
      share 0
    end if
  end for
```

Algorithm 7: Maximisation of probabilities

Chapter 3

Robotics and its Primitives

Chapter 4

A Framework for Secure Localization

Chapter 4 goes here ...

Chapter 5

Conclusions

Conclusion goes here ....

Related Publications

Bibliography

[1] A. Alpher. Frobnication. *Journal of Foo*, 12(1):234–778, 2002.

[2] Authors. The frobnicatable foo filter. 2006. ECCV06 submission ID 324. Supplied as additional material `eccv06.pdf`.

[3] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.

[4] R. Gennaro, M. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of 17th ACM Symposium on Principles of Distributed Computing (PODC)*, 1998.

[5] M. O. Rabin. How to exchange secrets with oblivious transfer. Technical report, 1981.

[6] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21:120–126, February 1978.

[7] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22:612–613, 1979.

[8] B. Shankar, K. Srinathan, and C. P. Rangan. Alternative protocols for generalized oblivious transfer. In *ICDCN* [8], pages 304–309.