Leveraging the "Multi" in Secure Multi-Party Computation

Jaideep Vaidya
Department of Computer Sciences and CERIAS
Purdue University
250 N University St
West Lafayette, IN 47907-2066
jsvaidya@cs.purdue.edu

Chris Clifton
Department of Computer Sciences and CERIAS
Purdue University
250 N University St
West Lafayette, IN 47907-2066
clifton@cs.purdue.edu

ABSTRACT

Secure Multi-Party Computation enables parties with private data to collaboratively compute a global function of their private data, without revealing that data. The increase in sensitive data on networked computers, along with improved ability to integrate and utilize that data, make the time ripe for practical secure multi-party computation. This paper surveys approaches to secure multi-party computation, and gives a method whereby an efficient protocol for two parties using an untrusted third party can be used to construct an efficient peer-to-peer secure multi-party protocol.

Categories and Subject Descriptors

K.4.4 [**Computers and Society**]: Electronic Commerce—*Security*

General Terms

Security

Keywords

Privacy, Secure Multi-party Computation, Secure Distributed Computation

1. INTRODUCTION

In the new era of enhanced privacy and security consciousness, secure distributed computing is gaining increasing attention. A plethora of situations exist where multiple parties have local data, and would like to share this data to obtain globally useful results. This desire often conflicts with privacy concerns; sharing and integrating this data may well violate privacy constraints. Theoretical results [10] show that it is possible to securely compute almost any function without revealing anything other than the output. However, performing this computation in a *practical* manner is quite another issue. The general method is inefficient for complex operations over large data sets.

This has motivated research on practical efficient protocols for secure computation in many domains. The emphasis has been on computation efficiency, communication efficiency, or both. Most work has been on two-party protocols that are both provably secure and efficient. With three or more (increasing number of) parties, the challenges (and solutions) become more difficult.

However, there is hope. We present a method where, by making certain assumptions about what can and cannot be revealed, we leverage the very *multiplicity* of parties in multi-party computation. Proposing a concrete solution for all problems is an extremely difficult task. Rather than doing so, we suggest a methodology that allows generation of efficient multiparty solutions provided

- a secure two-party solution to the problem exists, and
- a minor, and quantifiable, set of extra information is allowed to be revealed.

We begin with a short review of secure multi-party computation and discussion of related work. In Section 3, we give an informal definition of the privacy constraints enforced by secure multi-party computation, and describe various approaches to secure multi-party computation. Our method to generate multi-party protocols is presented in Section 4, followed by some demonstrative examples in Section 5. We conclude with a short discussion of ideas for future work.

2. RELATED WORK

There has been work in cooperative computation between entities that mutually distrust one another. This computation may be of any sort: scientific, data processing or even secret sharing. Secure two party computation was first investigated by Yao [22] and was later generalized to multi-party computation. The seminal paper by Goldreich et al. [10] proves the existence of a secure solution for *any* functionality. The approach used is as follows: the function f to be computed is first represented as a combinatorial circuit. The parties then run a short protocol for every gate in the circuit. Every participant gets (randomly chosen) shares of the values of the input and output for each gate; the exclusive or of the shares is the actual value. One share alone carries no information about the input or function value, as which party gets which share is determined randomly. At the end, the parties exchange their shares, enabling each to compute the final result. This protocol has been proven

WPES'03, October 30, 2003, Washington, DC, USA.

to give the desired result without disclosing anything other than the result. This approach, though appealing in its simplicity and generality, implies that the size of the protocol depends on the size of the circuit, which depends on the size of the input. Other general techniques have been proposed [4, 16]. However, the general methods are typically rather inefficient for large inputs, especially with a large number of parties.

The cost of circuit evaluation for large inputs has resulted in a slew of algorithms for more efficiently computing specific functionalities. Many protocols have been developed to solve specific problems, published both in the security literature as well as in the literature of application areas. Some examples include secure summation[19], scalar product[2, 12, 20], some scientific computation[6], secure set union[14], secure set intersection cardinality[21], private permutation[7], and computing entropy[9].

Lindell and Pinkas [15] examined the problem of computing the entropy for horizontally partitioned data using a clever application of the Taylor series expansion. Agrawal et al. also independently suggested techniques for computing the intersection, intersection size, join and join size for large databases [1].

Naor et al. [17] define a reduction from a multi-party scenario into a two-party solution. Kantarcioglu and Vaidya [13] also define a similar architecture that enables privacy preserving data mining for multiple parties. Both of these assume that all the parties will be able to agree on two specific parties to trust, and also elevates the security risk of losing all the information to the "cracking risk" of just two parties. The reduction in this paper requires that any two parties need to agree on at least one other party to trust. This is a feasible alternative to the need for an external trusted third party, which is often difficult to achieve.

Du and Atallah [8] is an excellent survey of secure multiparty computation problems and their applications while [5] suggests a practical approach to applying secure multiparty computation by accepting a compromise on security. We further extend this idea. Under relatively practical assumptions about privacy constraints, we demonstrate how to exploit the very complexity of a problem (specifically, the multiplicity of parties) to create an efficient solution.

3. MODELS FOR SECURE MULTI-PARTY COMPUTATION

We now give an overview of varieties of secure multiparty protocols. We will illustrate with a simple and well-studied example, the scalar dot product problem. While this is a two-party problem, the ideas readily extend to more parties. The problem is formally defined as follows:

Assume party A has the vector $\vec{X} = (x_1, \ldots, x_n)$ and party B has the vector $\vec{Y} = (y_1, \ldots, y_n)$. The parties wish to compute the scalar product (or dot product) $\vec{X} \cdot \vec{Y} = \sum_{i=1}^{n} x_i * y_i$, without revealing any information about $\vec{X}$ or $\vec{Y}$ that is not inherently revealed by the result $\vec{X} \cdot \vec{Y}$.

3.1 Secure protocols with a trusted third party

The simplest approach is based on the existence of a trusted third party. A trusted party is one to whom private values can be revealed without violating privacy constraints. If we have a single party that is trusted by *all* of the participants, an extremely simple protocol is available. Simply have each party send its input to the trusted third party, and the trusted party computes and shares the result. The same protocol can be utilized independent of the number of local parties.

In reality, many protocols work this way. A common example is payment escrow. The buyer sends payment to the escrow agent, who validates that payment is received and tells the seller to proceed with the transaction. Only when the goods are delivered does the escrow agent release payment. In more complex transactions (e.g., Real Estate), there may be additional parties, such as mortgage companies, and the computation required of the escrow agent may be more complex.

This is the "gold standard" for secure multiparty computation. With our running example, we assume a trusted third party T. Both A and B send their vectors to T which performs the dot product and returns the result. Clearly neither A nor B learns anything from the protocol except the result, nor (assuming secure communication) does any other party except T. This is also generally an efficient protocol.

The biggest disadvantage to this model is the loss of privacy – there must be a party trusted with all of the data. Legal issues, trade secrets, and privacy concerns combine to make existence of such a party doubtful, especially with many parties. The goal of secure multiparty computation is to approximate such a model in the absence of a trusted party.

3.2 Completely secure multi-party protocols

The opposite extreme is a computation carried out only by the parties who possess the data and desire the results. Thus, a secure two-party protocol is completely carried out by two parties, say, Alice and Bob, without involving any other parties in the computation. Each gets its own output, with nobody else getting anything. Typically, without simplifying assumptions, these protocols are quite inefficient.

There are several scalar product protocols in the literature for efficiently and securely computing scalar product using only two parties[2, 12, 20].

The protocol in [20] is completely algebraic: one party adds randoms to its values and sends it to the other party, which does some computation and sends data computed from its own values back. By clever choice of the original randoms, and proper values sent back, the first party is able to compute the scalar product. The total cost is about 1.5 times the input size. This protocol is not completely private, as the information seen in the protocol along with externally learning half of the private values reveals the other half. The other protocols use more expensive cryptographic approaches to achieve privacy approximating the untrusted third party model, assuming parties do not "cheat" in the execution of the protocol.

The Secure Multi-Party Computation literature has shown that virtually any functionality can be privately computed in this model[10, 16]. The general approaches do not scale well to complex problems over large data sets, leading to many specialized protocols for specific functionalities.

We now look at two alternative models, enabling practical and efficient solutions while maintaining an equivalent degree of privacy.

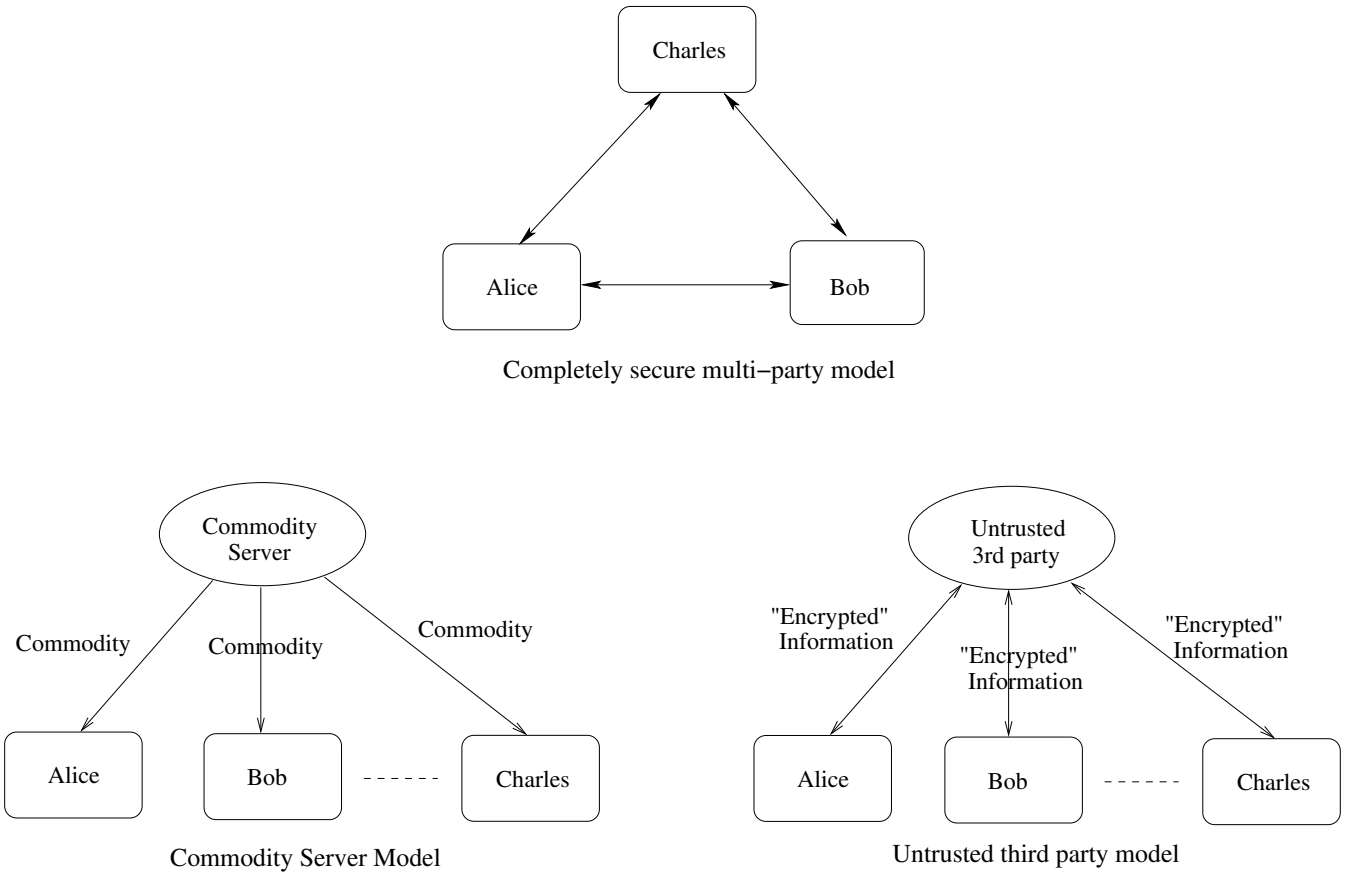

Figure 1: Different computation models

3.3 Two party protocols with an untrusted third party

The existence of an *un*trusted third party enables efficient protocols without revealing private information. The idea of an untrusted third party is that it is willing to perform some computation for the parties in the protocol (perhaps for a fee). It is not trusted with the data or the results. The trust placed in this party is that it:

1. not *collude* with any of the participating parties to violate information privacy, and
2. correctly executes the protocol.

Correct execution of the protocol is only required to guarantee correct results; even a dishonest third party is unable to learn private information in the absence of collusion.

Typically the third party is given some information in encrypted form. By encryption, we simply mean that the third party cannot make any sense of the data given to it without the assistance of the local parties involved in the protocol. The third party performs a computation on the encrypted data, possibly exchanging information with the other parties in the process. Finally, the encrypted computation result is revealed to the local parties, who are able to decrypt the result.

For our scalar product protocol, consider the following: A and B together decide on a (possibly random) permutation π of n elements that is known only to them. A and B also generate a common random vector $\vec{R}$ of n elements. Now, A generates the vector $\vec{X}' = \pi(\vec{X} + \vec{R})$, while B generates $\vec{Y}' = \pi(\vec{Y} + \vec{R})$. A and B send $\vec{X}'$ and $\vec{Y}'$ to the third untrusted party U.

U computes the scalar product $S = \vec{X}' \cdot \vec{Y}'$. Identically permuting both vectors has no effect on the scalar product, since it effectively is only changing the order of additions in the sum. Thus, $S = \vec{X} \cdot \vec{Y} + \vec{X} \cdot \vec{R} + \vec{Y} \cdot \vec{R} + \vec{R} \cdot \vec{R}$. A and B also send $\vec{X} \cdot \vec{R}$ and $\vec{Y} \cdot \vec{R}$ to U. U subtracts these from S to get $S' = \vec{X} \cdot \vec{Y} + \vec{R} \cdot \vec{R}$. U sends S' back to A and B, who subtract $\vec{R} \cdot \vec{R}$ to get the final result. Since neither $\vec{R}$ nor the permutation π is known to U, it learns nothing of $\vec{X}$, $\vec{Y}$, or the result. This protocol is also very efficient in communication complexity, since the permutation π and the random vector R can be generated with a single random seed - the communication cost is $O(n)$, where n is the size of the vectors.

3.4 Two party protocols with commodity server

Beaver [3] introduced an alternative third party model – the commodity server model. In this model, a commodity server generates data that is used by the two parties, enabling them to "synchronize" their protocols. The trust requirements are the same as for an untrusted third party; the key practical difference is that the third party generally has a lower computation cost as a commodity server rather than as an untrusted third party.

A commodity-server based protocol is given in [9]. The

idea is that the commodity server generates two random vectors, $\vec{R_a}$ and $\vec{R_b}$, and two random shares r_a, r_b, of the scalar product of $\vec{R_a} \cdot \vec{R_b}$ (i.e., $r_a + r_b = R_a \cdot R_b$). The pair $(\vec{R_a}, r_a)$ is sent to A, while $(\vec{R_b}, r_b)$ is sent to B. A and B carry on a computation involving these pairs and their own input to find the scalar product $\vec{X} \cdot \vec{Y}$. The complete protocol is given in [9].

Commodity server protocols are in general less efficient than the class of protocols of subsection 3.3. However, a convincing proof that the third party does not learn anything is trivial, as it never receives any information.

3.5 Summary

Multi-party protocols without using any other parties are often complicated and inefficient. While a trusted third party allows us to easily solve problems, the assumption of such a completely reliable party is not always practical. Protocols using a commodity server or an untrusted third party allow elegant solutions that meet privacy constraints and achieve acceptable performance. The basic assumption of third party protocols is non-collusion: the third party will not work with one of the participants to violate the privacy of another participant. The different computation models are summarized in Figure 1.

4. A GENERAL, EFFICIENT, AND SECURE MULTI-PARTY PROTOCOL

One problem with untrusted third party and commodity server protocols is the requirement that an external party be willing to perform computation. While this may be an interesting business model, in this section we present an alternative. For associative functions, if we slightly relax the security guarantees, we can generate an efficient multi-party protocol for any function for which we have an efficient two-party protocol in the untrusted third party model. The basic idea is to utilize participants as untrusted third parties. By choosing what participants can see in their role as an untrusted third party, we can ensure they learn no more than an external party would.

4.1 Assumptions

The general protocol trades off security for efficiency. However, a small compromise in security results in a large amelioration of efficiency. Also, since this is a general technique rather than a particular solution, it requires actual two-party protocols to be used as plug-ins. These issues are now discussed in detail.

4.1.1 Function Restrictions

The general method presented here is not applicable to all problems. In fact, it is only applicable to a class of functions F_g (representing the class of functions solvable by our general method), the properties of which are defined below.

A k-input function $y = f(x_1, \ldots, x_k)$ belongs to the class of functions F_g, if and only if $y = f(x_1, \ldots, x_k) = x_1 \otimes x_2 \otimes \cdots \otimes x_k$, and further, that $\otimes$ is associative.

Examples of such functions include set union / intersection, as well as most additive/multiplicative functionalities.

4.1.2 Two party solution with protocol P

The primary assumption is that an efficient secure protocol P exists to solve the problem for two parties. The protocol may make use of a single untrusted third party or a single commodity server, i.e., protocol P is a protocol of the class of protocols described in either Section 3.3 or Section 3.4. This protocol P will be used as a plug-in to the general protocol.

It is allowable for P to provide the results to the third party rather than the original parties. Such asymmetric protocols allows use where $\otimes$ is invertible, i.e., given $a = x \otimes b$, we could determine x. In these cases, a symmetric protocol that gives the results to the parties with the data would automatically violate privacy. Examples of protocols where this is an issue are given in Section 5.

4.1.3 Adaptability of protocol P

Rather than an actual protocol, we give a methodology for constructing a protocol and proving it secure. This requires the integration of protocol P into the overall plan. Integration necessitates minor modifications to the protocol. The protocol P should be reasonably malleable to incorporate such modifications. This is not really an assumption, but rather one of the engineering requirements to get the whole thing to work.

4.1.4 Collusion

The core security assumption is identical to that needed in the untrusted third party / commodity server model. The chosen third party should not collude with either party against the other. Thus, every pair of two parties must be able to find at least one other party that it trusts not to collude with the other party in the pair. While pathologically dishonest and untrusting parties may find this difficult, in many practical situations it is feasible. For example, in business relationships this level of trust is common (e.g., non-disclosure agreements), but legal or contractual requirements may impose stronger constraints than can be satisfied by such agreements. This provides us with an alternative to finding a non-colluding external third party that is willing to participate in the protocol, which is often difficult to achieve.

4.2 The general framework

Given k parties, the goal is to compute a function $y \in F_g$, where $y = f(x_1, \ldots, x_k)$, where $x_1, \ldots, x_k$ are the local inputs of the k parties. Note, by assumption 4.1.1, since $y \in F_g$, $y = x_1 \otimes x_2 \otimes \cdots \otimes x_k$. By assumption 4.1.2, we have a protocol P to securely compute the two-input function $\otimes$.

The key idea is to create two partitions P_0 and P_1. Split the k parties equally into the two partitions. We can now use the parties in partition P_i as untrusted third parties to evaluate partition P_{1-i}. To visualize this, construct a binary tree on the partition P_i with the leaves being the parties in P_i (Figure 2). There can be at most $|P_i|-1$ interior nodes in the binary tree. Due to the (almost) equi-partitioning, the following invariant always holds: $|P_{1-i}| \geq |P_i| - 1$, for both values of i. Thus, there are sufficient parties in the other partition to act as interior nodes. The role of the parties in partition P_{1-i} is to act as the commodity server or untrusted third party for the parties in partition P_i.

In the first round, the $k/4$ of the parties from the other partition act as third parties for the $k/2$ parties in the first partition. For the remaining $\log k/2 - 1$ rounds the other $k/4$ parties of the 2nd partition act as third parties upwards along the tree. Each third party receives some form of the

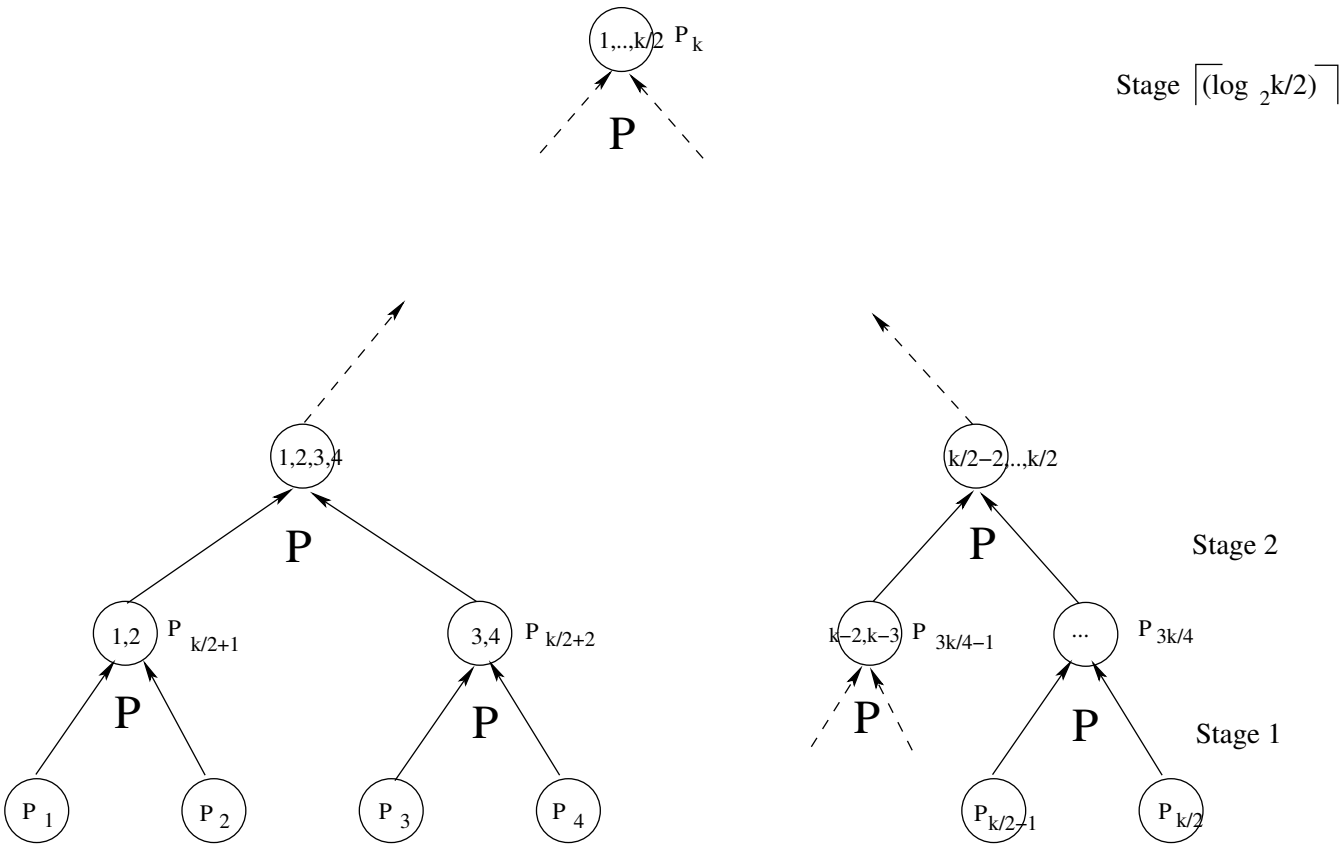

Figure 2: The general protocol process applied on partition P_0

intermediate result, and utilizes it in the next round. It is important to analyze the amount of data revealed to the third party at this point and modify the protocol if necessary to limit the information disclosure. The entire process is illustrated in Figure 2, where we show the process for partition P_0 consisting of the first $k/2$ parties. Thus, all of the parties $P_{k/2+1}, \ldots P_k$ act as third parties / commodity servers in a single call to the protocol P when applied to the parties at the leaf nodes. There are a total of $\log k/2$ rounds in which several calls to the protocol P are made in parallel.

Once a similar process is done for the other partition P_1, the two topmost representatives of the two parties use a secure two party protocol P' to compute the final result. Every party possibly acquires some information about a few of the other parties, which goes against the precept of secure multi-party computation. But as long as the information revealed is held within strict (and provable) bounds, it is often worthwhile to trade this limited information disclosure for efficiency and practicality. As we shall show in Section 5, the intermediate information disclosed can generally be presented in a form that does not reveal private information.

4.2.1 Security

Given a secure multi-party computation (SMC) style proof of security for protocol P, it is straightforward to provide a proof of security for the entire protocol. The proof depends upon applying the composition theorem[11], considering P to be the subroutine called. The views of interior nodes will generally be straightforward to simulate, given that the results of the protocol P using that node as the untrusted party are viewed as part of the result seen by that party. This allows a simple simulation proof showing no other data is revealed. An example of such a proof is given in [21].

5. EXAMPLES

In this section, we give some examples of applying the general methodology to practical problems. Section 5.1 provides a solution to the rather trivial problem of computing the XOR. Section 5.2 provides a much more concrete demonstration of the methodology to compute the set intersection.

5.1 A trivial application - XOR

Computing the exclusive or of a distributed set of bits provides a simple demonstration of this methodology. The problem is as follows: Given k parties $P_1 \ldots P_k$, each having its own private bit b_i, securely compute $s = b_1 \oplus b_2 \oplus \cdots \oplus b_k$, where $\oplus$ signifies the XOR operation. Securely computing XOR makes no sense for only two parties, as the result reveals the other party's input, however it is a viable problem for $n > 2$ parties, or for two parties if some third party (and only the third party) is allowed to view the result.

Consider two parties A and B with the source data x and y, and a third party U that is allowed to know the result $x \oplus y$ but is not trusted with either x or y. A suitable protocol P follows: A and B jointly agree on two random bits r_1 and r_2. Then, A sends $M_a = x \oplus r_1 \oplus r_2$ to U, while B sends

$M_b = y \oplus r_1 \oplus \overline{r_2}$ to U. U can now easily find the result:

$$\begin{aligned} M_a \oplus M_b &= (x \oplus r_1 \oplus r_2) \oplus (y \oplus r_1 \oplus \overline{r_2}) \\ &= (x \oplus y) \oplus (r_1 \oplus r_1) \oplus (r_2 \oplus \overline{r_2}) \\ &= (x \oplus y) \oplus 0 \oplus 1 \\ &= \overline{x \oplus y} \end{aligned}$$

U can take the inverse of the XOR of the two messages it receives to get the desired result $x \oplus y$.

This protocol can easily be extended to more than two parties by simple application of the general methodology. The complete set of parties is separated into two partitions. The interior nodes for the computation tree of one partition will be the parties from the other partition. Repeated application of the protocol P at all levels gives the desired result. Each party does learn some additional information (the XOR of the bits of the subtrees below it), but since none of these includes its own bits, it gains no information about the individual data values at the leaves of the tree. The exception is the root, as its bit is included in the result it learns, however since all it learns is the desired final result what it learns is inescapable.

A slight variation would allow a completely secure protocol. Instead of just choosing r_1 and r_2, parties would choose additional randoms in collaboration with the other leaf nodes such that the random values would only cancel out at the root.

5.2 Secure Set Intersection

A more useful example is securely obtaining the size of the intersection of several sets. The problem is defined as follows. There are k parties, $P_1, \ldots, P_k$, each with a local set S_k drawn from a common (global) universe U. They wish to compute $|\cap_{j=1}^{k} S_j|$, i.e., the cardinality of the common intersection set. This is useful for several applications for example data mining association rules (see [21] for details.)

Since set intersection is associative, we can use the protocol of Section 4 provided we have a secure two-party protocol using an untrusted third party. We now outline a two party protocol P using a third untrusted party to compute $|S_a \cap S_b|$ for two parties A and B. The key idea behind protocol P is to use a commutative encryption scheme (e.g., RSA[18] with both keys kept private). Commutativity ensures that $E_{k1}(E_{k2}(x)) = E_{k2}(E_{k1}(x))$. Parties A and B generate encryption keys E_a and E_b respectively. A encrypts the items in its set S_a with E_a and sends them to B. Similarly, B encrypts the items in S_b with E_b and sends them to A. Each site now encrypts the received items with its own key, and sends the doubly-encrypted sets S'_a and S'_b to U. U now finds the intersection of these two sets. Because of commutativity of the encryption, an item $x \in S_a \cap S_b$ will correspond to an item $E_a(E_b(x)) = E_b(E_a(x))$ that appears in both S'_a and S'_b. Therefore, the size of the intersection $|S'_a \cap S'_b| = |S_a \cap S_b|$. Thus U learns the size of the intersection, but learns nothing about the items *in* the intersection.

Extending this to more than two parties is simple. The lowest layer proceeds as above. At the higher layers, the parties encrypt with the keys of their sibling's children. Since a party never sees any of the values from the sibling's children (even after encryption), knowing the keys gives no information. More details are given in [21].

A similar protocol can be constructed to determine the size of set union.

6. CONCLUSION

Protecting privacy while doing meaningful computation over distributed data is a tough task. However, there have been great strides in research on secure distributed computation. Theoretically, both two-party and multi-party protocols are possible for *almost any* computation. Practical solutions have been found for several types of two-party computations. However, secure multi-party computations are not as well explored.

We have presented a method for converting two-party protocols that use an untrusted third party into secure multi-party protocols that do not require a third party. Through use of the composition theorem of [11] and the provably secure protocol of Section 4, it is straightforward to generate and prove secure a multi-party protocol given a secure two-party protocol with an untrusted third party. The method requires only the assumption of non-collusion, and a willingness to reveal a small and provably limited amount of additional information. This method works for any associative function. Developing a general method which goes beyond associative functions is an interesting future direction for research.

Practical use of secure multi-party computation will require the ability to easily generate *new* computations corresponding to the requirements of a particular set of participants. The task of generating such protocols and proving them secure must be made easier. One way to do this is by providing a toolkit of secure protocols, and methods to combine them. We have presented once such combination method. Further work along these lines will enable secure multi-party computation to become a significant enabler for electronic collaboration that preserves privacy.

7. REFERENCES

[1] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, San Diego, California, June 9-12 2003.

[2] M. J. Atallah and W. Du. Secure multi-party computational geometry. In *Seventh International Workshop on Algorithms and Data Structures (WADS 2001)*, Providence, Rhode Island, USA, Aug. 8-10 2001. [Online]. Available: http://www.cerias.purdue.edu/homes/duw/research/paper/wads2001.ps

[3] D. Beaver. Commodity-based cryptography (extended abstract). In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing.* ACM Press, 1997, pages 446–455.

[4] D. Chaum, C. Crepeau, and I. Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the 20th ACM Symposium on the Theory of Computing.* ACM Press, 1988, pages 11–19.

[5] W. Du and Z. Zhan. A practical approach to solve secure multi-party computation problems. In *Proceedings of New Security Paradigms Workshop*, Virginia Beach, virginia, USA, September 23-26 2002.

[6] W. Du and M. J. Atallah. Privacy-preserving cooperative scientific computations. In *14th IEEE Computer Security Foundations Workshop*, Nova Scotia, Canada, June 11-13 2001, pages 273–282. [Online]. Available: http://portal.acm.org

[7] W. Du and M. J. Atallah. Privacy-preserving statistical analysis. In *Proceeding of the 17th Annual Computer Security Applications Conference*, New Orleans, Louisiana, USA, December 10-14 2001. [Online]. Available: http://www.cerias.purdue.edu/homes/duw/research/paper/acsac2001.ps

[8] W. Du and M. J. Atallah. Secure multi-party computation problems and their applications: A review and open problems. In *New Security Paradigms Workshop*, Cloudcroft, New Mexico, USA, September 11-13 2001, pages 11–20. [Online]. Available: http://www.cerias.purdue.edu/homes/duw/research/paper/nspw2001.ps

[9] W. Du and Z. Zhan. Building decision tree classifier on private data. In *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, C. Clifton and V. Estivill-Castro, Eds., vol. 14. Maebashi City, Japan: Australian Computer Society, Dec. 9 2002, pages 1–8. [Online]. Available: http://www.jrpit.flinders.edu.au/CRPITVolume14.html

[10] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game - a completeness theorem for protocols with honest majority. In *19th ACM Symposium on the Theory of Computing*, 1987, pages 218–229. [Online]. Available: http://doi.acm.org/10.1145/28395.28420

[11] O. Goldreich. Secure multi-party computation. Sept. 1998, (working draft). [Online]. Available: http://www.wisdom.weizmann.ac.il/~oded/pp.html

[12] I. Ioannidis, A. Grama, and M. Atallah. A secure protocol for computing dot-products in clustered and distributed environments. In *The 2002 International Conference on Parallel Processing*, Vancouver, British Columbia, Aug. 18-21 2002.

[13] M. Kantarcioglu and J. Vaidya. An architecture for privacy-preserving mining of client information. In *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, C. Clifton and V. Estivill-Castro, Eds., vol. 14. Maebashi City, Japan: Australian Computer Society, Dec. 9 2002, pages 37–42. [Online]. Available: http://crpit.com/Vol14.html

[14] M. Kantarcıoğlu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Trans. Knowledge Data Eng.*, to appear.

[15] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO 2000*. Springer-Verlag, Aug. 20-24 2000, pages 36–54. [Online]. Available: http://link.springer.de/link/service/series/0558/bibs/1880/18800036.htm

[16] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, Heraklion, Crete, Greece, July 6-8 2001. [Online]. Available: http://doi.acm.org/10.1145/380752.380855

[17] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*. ACM Press, 1999.

[18] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. [Online]. Available: http://doi.acm.org/10.1145/359340.359342

[19] B. Schneier, *Applied Cryptography*, 2nd ed. John Wiley & Sons, 1995.

[20] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, July 23-26 2002, pages 639–644. [Online]. Available: http://doi.acm.org/10.1145/775047.775142

[21] J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining. *ACM Transactions on Information Systems Security*, submitted.

[22] A. C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*. IEEE, 1986, pages 162–167.