Secure Two-Party Context Free Language Recognition

Anshuman Singh, Siddharth Barman, and K.K. Shukla

Dept. of Computer Sc. and Engg., Institute of Technology, Banaras Hindu University, Varanasi, India - 221005
anshum4n@yahoo.com, siddharth.barman@cse05.itbhu.org, shukla@ieee.org

Abstract. The growth of the internet provides opportunities for cooperative computation, it also requires development of protocols that can accomplish this task among mutually untrusting parties. The aim is to develop methods which ensure both the correct evaluation of the function and privacy of individual inputs. Multiparty Computation protocols help to achieve the aim without using a trusted third party.

In this paper we consider the problem of context-free language recognition in a two-party setting. Alice has the description of a context-free language L while Bob has a secret string whose membership in L is to be checked. Neither Alice nor Bob is ready to disclose his/her input to the other. Here we propose a protocol which accomplishes secure two party context-free language recognition. The novelty of this paper lies in the use of formal languages based approach for multiparty computations.

1 Introduction

The development of computer networks and consequently the Internet has opened the wide area of distributed computation. Internet allows computers from far off places to interact and opens possibilities that were unknown before. A scenario is conceivable where some parties want to compute a function over data which is distributed among the parties, but none of the parties want to disclose their private data. A naive solution would be to send the data to a trusted party who performs the computation and returns the results to respective parties. However a trusted agency may not be available or affordable. In such a case we can use cryptographic techniques of Secure Multiparty Computation.

Secure multi-party computation (MPC) was introduced by Yao in [1]. It deals with the problem of securely computing an arbitrary function f over the private inputs of n players. Here security means guaranteeing the correctness of the output as well as the privacy of the player's inputs, even when some players cheat. Assuming we have inputs $x_1, x_2, \ldots x_n$ where player i knows x_i, we want to compute $f(x_1, x_2, \ldots x_n) = (y_1, y_2, \ldots y_n)$ such that player i is guaranteed to learn y_i, but can get no more information. A number of cooperative computation problems have been shown to be plausible, in a secure way, using multi party computational techniques [2, 5]. Generic solutions have been developed [3, 8] and

G. Chakraborty (Ed.): ICDCIT 2005, LNCS 3816, pp. 117–124, 2005.

much effort has been directed towards reducing the communication complexity [7] and round complexity [6] of these solutions.

In this paper we are concerned with Context Free Language Recognition (CFLR) in a two-party setting. In a traditional setting the problem of CFLR is to determine the membership of a string **w** in a context free language **L**. In the two party setting the problem remains the same but the inputs are distributed among two different parties, Alice and Bob. Now Alice has a private context free grammar **G** while Bob has a private string **w**. Bob wants to check the membership of **w** in **L(G)**, the language generated by **G**. Also neither of the two is willing to disclose his/her private input to the other.

CFLR is an interesting problem and has various applications. For instance, the problem of checking the syntactic correctness of a program can be posed as CFLR. Also pattern matching and recognition queries can be posed as CFLR. Numerous other decision problems can be solved using CFLR. The basic methodology is to describe some class of objects using a context free grammar **G**. When a new object is encountered, we determine it's membership in **L(G)**. This tells us whether the new object belongs to the same class or not. A solution for the two-party version of CFLR can be used to solve the above problems in a secure two-party manner.

It is the first time that a formal language based approach has been used for solving multiparty computation problems. The completeness of multiparty protocols has been shown in [2]. The solution given in [2] can be used to solve the membership question for recursive languages, i.e. the languages recognized by a turing machine. Since context free languages are a proper subset of recursive languages, CFLR can also be solved using such a method. However, the protocol that we present here generates a more efficient solution of membership question for context free languages than promised by the generic approach. This is because our solution utilizes the specific properties of context free grammars, used in description of a context free language.

2 Preliminaries

In this section we give the notation and definitions of the terms used in this paper. Most of the content of this section from basic formal language theory. We have endeavored to use standard notations throughout. A superscript on a vector, such as S^m, denotes the m^{th} bit of it. Also $|V|$ represents the size of set V.

Formal Language Basics. A grammar $G = (V, T, S, P)$ is said to be context-free if all its productions are of the form $A \rightarrow x$ where $A \in V$ and $x \in (V \cup T)^*$. Here V is the set of variables, T is the set of terminals, S is the starting symbol and P are the production/rewrite rules. A language L is said to be context-free if and only if there exists a context-free grammar G, such that $L = L(G)$. Here $L(G)$ denotes the set of strings that can be produced by the grammar. This can also be written as "$L = \{w \in T^* | S \Rightarrow^* w\}$?" where the symbol '$\Rightarrow^*$' stands for 'derives'.

Given a context grammar G and a string w the problem of CFLR is to determine the answer to the following question "Does $w \in L(G)$?". We now give a definition for the two-party version of CFLR.

Secure Two-Party Context Free Language Recognition Protocol. Alice and Bob, determine whether Bob's secret string w is present in Alice's secret Context Free Language $L(G)$. At the end of the protocol the following properties must hold.

- Bob knows whether $\mathbf{w} \in L(G)$
- Alice gains no information about **w**
- Bob gains only as much information about **L(G)** as can be determined from the output, i.e. whether **w** is accepted by **L(G)** or not.

Solving Context Free Language Recognition: The CYK Membership Algorithm. There are many existing algorithms for solving CFLR. One of the standard methods is the CYK membership algorithm [11]. It's time complexity is cubic in the size of the input. There exist some efficient (linear time) membership algorithms that can solve some restricted versions of CFLR. We selected CYK for it's generality. The CYK algorithm requires the context free grammar to be converted to chomsky normal form. A context free grammar $G = (V, T, S, P)$ is in chomsky normal form if all it's productions are of the form $A \rightarrow BC$ or $A \rightarrow a$ where $A, B, C \in V$ and $a \in T$. Any context-free grammar can be written in chomsky normal form following a straightforward set of rules [11]. The CYK algorithm first converts a given grammar in CNF and then utilizes it to determine membership. We now describe the CYK algorithm.

Assume that we have a grammar $G = (V, T, S, P)$ in Chomsky Normal Form and a string $w = w_1 w_2 ... w_n$. We define sub-string $w_{ij} = w_i ... w_j$ and subsets of V, $S_{ij} = \{A \in V : A \Rightarrow^* w_{ij}\}$. Clearly $w \in L$ if and only if $S \in S_{1n}$. To compute S_{ii}, observe that $A \in S_{ii}$ if and only if G contains a production $A \rightarrow w_i$. Therefore S_{ii} can be computed for all $1 \leq i \leq n$ by inspection of w and the productions of the grammar. To continue notice that for $j > i$, A derives w_{ij} if and only if there is a production $A \rightarrow BC$, with $B \Rightarrow^* w_{ik}$ and $C \Rightarrow^* w_{k+1j}$ for some k with $i \leq k < j$. In other words

$$S_{ij} = \bigcup_{k \in \{i, i+1, \ldots, j-1\}} \{A : A \rightarrow BC, \text{ with } B \in S_{ik}, C \in S_{k+1j}\} \tag{1}$$

An inspection of indices show that the above equation can be used to compute all the S_{ij}s if we proceed in the sequence

1. Compute $S_{11}, S_{22}, ..., S_{nn}$
2. Compute $S_{12}, S_{23}, ..., S_{n-1n}$
3. Compute $S_{13}, S_{24}, ..., S_{n-2n}$

and so on.

Cryptographic Assumptions and Oblivious Transfers. The security of our protocol is based on oblivious transfers. Oblivious Transfers are a basic cryptographic primitive that has proved necessary for many of the protocols [10]. It allows multiple parties to get individual secrets from a single seller. There are different definitions of oblivious transfers [4, 12]. In it's most primitive form, the sender has an input $(b_1, b_2, ..., b_k)$ and the receiver has an input $i \in \{1, 2, ..., k\}$. The goal is to transfer the i^{th} bit to the receiver without letting the receiver obtain knowledge of any other bit and without letting the sender obtain the knowledge of the identity of the bit required by the receiver. Assuming the existence of trapdoor permutation, a protocol for the above functionality can be constructed as given in [12]. The above version of Oblivious transfer functionality is a main ingredient of our construction. The existence of trapdoor permutation [12] is the only assumption we make for security of our protocol.

3 The Protocol

Our protocol is a secure two-party version of the CYK Algorithm. Observe that if we can securely compute S_{ii} for $1 \leq i \leq n$ and provide a secure protocol to compute S_{ij} given S_{ik} and S_{k+1j} where $i \leq k < j$ we can use them to carry out CYK in a two party setting.

Let the context free grammar with Alice be $G = \{V, T, S, P\}$ where $V = \{V_1, V_2, \ldots V_{|V|}\}$, $T = \{T_1, T_2, \ldots T_{|T|}\}$, S is the starting symbol and P is the set of rewrite rules in chomsky normal form. The sets S_{ij}, as defined in (1), are maintained as a $|V|$ length 0/1 vector Γ_{ij} where $\Gamma_{ij}^m = 1$ if and only if $V_m \in S_{ij}$. Note that here (and in the remainder of the paper) a superscript m denotes the m^{th} bit of the corresponding vector. These vectors are shared by a simple xor scheme such that if A_{ij} is Alice's share and B_{ij} is Bob's share, then $A_{ij} \oplus B_{ij} = \Gamma_{ij}$. The Γ_{ii}s in the first step are constructed using a 1-out-of-n Oblivious Transfer protocol as shown next.

Alice builds up a vector S_t for each of the terminal t. S_t^m is 1 if and only if $V_m \rightarrow t$. She also chooses a random 0/1 vector A_{ii} for each $1 \leq i \leq n$ which forms her share of Γ_{ii}. A_{ii} when xored with the S_t for each $t \in T$, yield a set of vectors $B_{ii} = \{B_{ii}^{t_1}, B_{ii}^{t_2}, \ldots B_{ii}^{t_{|T|}}\}$. Bob is allowed to select $B_{ii}^{w_i}$ from this set (depending upon his character w_i) using 1-out-of-$|T|$ oblivious transfer protocol. This forms his share of Γ_{ii}. Thus $S_{11}, S_{22}, ..., S_{nn}$ are shared between Alice and Bob.

The protocol now proceeds in phases and after each phase the new Γ_{ij}, as given in the sequence for CYK, are computed. Shares of Γ_{ij} can be computed provided Γ_{ik} and Γ_{k+1j} for all $k \in i, i+1, ..j-1$ have already been shared. For each production of the form $V_X \rightarrow V_Y V_Z$ Alice and Bob co-operatively update the x^{th} bit of shares A_{ij} and B_{ij}. Let the new shares be called $A_{ij}^{x\ new}$ and $B_{ij}^{x\ new}$. Then the x^{th} bit of the updated share can be written as

$$B_{ij}^{x\ new} = \left([(A_{ik}^y \oplus B_{ik}^y) \wedge (A_{k+1j}^z \oplus B_{k+1j}^z)] \vee [A_{ij}^x \oplus B_{ij}^x]\right) \oplus r^A \tag{2}$$

$$A_{ij}^{x\ new} = r^A, \text{ where } r^A \text{ is chosen randomly by Alice} \tag{3}$$

Equations (2) and (3) are the same as (1), except for the notations. $(A^y_{ik} \oplus B^y_{ik})$ gives whether $V_y \in S_{ik}$ while $(A^z_{k+1j} \oplus B^z_{k+1j})$ gives whether $V_z \in S_{k+1j}$. If both the above expressions evaluate to true Γ^x_{ij} must be one and otherwise it should remain as it is, as is emphasized by oring the above expression with $[A^x_{ij} \oplus B^x_{ij}]$. Finally, we xor it with a random term r^A chosen by Alice, which forms her private share(A^x_{ij}) for Γ^x_{ij}.

Equations (2) and (3) can be calculated using the general circuit evaluation protocol [3]. Finally, Alice and Bob check out whether Γ_{1n} contains the starting symbol S or not.

3.1 Initialization Step

1. Alice prepares vectors S_t for each $t \in T$ such that $S^k_t = 1$ if and only if $V_k \rightarrow t$.
2. Alice prepares random $|V|$ length vectors A_{ii} for each $1 \leq i \leq n$. These form her share of Γ_{ii}.
3. Alice constructs a set of vectors $B^j = \{B^{t_1}_{jj}, B^{t_2}_{jj}, \ldots B^{t_{|T|}}_{jj}\}$ for all $j \in \{1, 2, ..., n\}$ where $B^{t_k}_{jj} = S_{t_k} \oplus A_{jj}$.
4. For each $j \in \{1, n\}$, Bob selects a vector $B^{w_j}_{jj}$ from B^j using oblivious transfer protocol. This forms his share B_{jj}.
5. Thus Alice and Bob share the initial Γ_{jj} as A_{jj} and B_{jj} for $j \in \{1, n\}$. This completes the initialization step.

3.2 Computing Γ_{1n}

We now describe the crux of the protocol, the computation of Γ_{1n}. We give the description in pseudocode as it is easier to understand and more expressive this way.

1. for d= 1 to n-1 do
2. for i = 1 to n-d do
3. j=i+d
4. for k = i to j-1 do
5. for each Production $x \rightarrow yz$ with Alice do
6. Alice chooses a random bit r^A
7. Alice and Bob use secure circuit evaluation protocol to compute

$$B^{x\,new}_{ij} = \left([(A^y_{ik} \oplus B^y_{ik}) \wedge (A^z_{k+1j} \oplus B^z_{k+1j})] \vee [A^x_{ij} \oplus B^x_{ij}]\right) \oplus r^A \quad (4)$$

$$A^{x\,new}_{ij} = r^A \quad (5)$$

8. endfor
9. endfor
10. endfor
11. endfor

3.3 Final Step

Using the initialization step and then computing each of Γ_{ij} as above, Alice and Bob obtain the shares for Γ_{1n}. At this point Alice sends her share of the bit corresponding to the starting symbol S from A_{1n} to Bob. Bob xors it with the corresponding bit in his share. The result tells Bob whether $S \in S_{1n}$ and hence whether w is generated by Alice's grammar or not.

4 Security

We prove the security of our protocol in the semihonest model with passive adversary. Such a protocol can be compiled into a protocol secure against a dishonest party and in presence of malicious adversary [9] using verifiable secret sharing and zero-knowledge proofs [2]. It is a standard procedure to construct a secure protocol in semihonest model and then convert it to a secure protocol in malicious model. However such a conversion increases the communication and computation cost of the protocol. Below we give an informal proof of security for the proposed protocol.

The protocol consists of three distinct phases the initialization step, the updation step and the final step. Without loss of generality we can consider the case where the language $L(G)$ consists of only two alphabets 0 and 1. In the initialization step, Alice prepares B^0_{jj} and B^1_{jj} for $j \in \{1, 2, ..., n\}$. One of this is selected by Bob based on his input w_j using 1-out-of-2 oblivious transfer protocol. If oblivious transfers were carried out correctly, there is no information gain for Alice as she doesn't know whether Bob has chosen B^0_{jj} or B^1_{jj}. Also Bob remains ignorant of the variables in S_{jj} as B_{jj} has been xored with a random vector A_{jj} which forms Alice's share. Hence there is no gain of information for either parties in the initialization step.

The updation step is based on the secure circuit evaluation protocol. During the computation of the circuit no information is revealed to either party. Finally the result of the evaluation $B^{x\ new}_{ij}$ is revealed only to Bob. But this is xored with a random bit r^A, known only to Alice. Hence the information content in $B^{x\ new}_{ij}$ is nil for Bob.

In the final step Alice sends the bit corresponding to S (starting symbol) in her share A_{1n} to Bob. This transfer doesn't increase her information in any way. Bob then xors this bit with the corresponding bit in his share to obtain one bit of information namely whether $w \in L(G)$. Hence during whole of the protocol the information gained by Bob is one bit.

Other Security Issues. One can say that after the protocol, Alice knows the length of Bob's string while Bob knows the exact number of variables, productions in Alice's automaton. This gives them some idea of the complexity of the other's input. However such information can easily be hidden. Alice can add some dubious variables and productions in G that do not affect the language L(G) generated by G. Bob can also add some random symbols after/before his actual input string. In such an instant Alice allows Bob to choose one of the bits

corresponding to S in A_{ij} for all $1 \leq i, j \leq n$ using 1-out-of-N oblivious transfer protocol. Bob will choose the bit in A_{ij}, w_{ij} being his actual string.

5 Analysis

The number of communication rounds required is $O(|w|^3)$ for each of S_{ij}. Also for the calculation of each S_{ij} the communication required is $O(|P|)$ where $|P|$ gives the number of productions in the grammar. Each round requires $O(|V|)$ communication for carrying out 1-out-of-$|V|$ oblivious transfers. Hence the total communication complexity of the protocol is $O(|w|^3|P||V|)$. Thus the multiparty version of CYK is slower by a factor of $O(|V|)$.

In an implementation over a data network, instead of running the protocol in a step by step manner, we can run steps 4 to 9 at once. Hence all the updates are made to the vectors concurrently. We can parallelize these steps because they are independent from each other and can be carried in any order we please. Taking network latency into account, this gives performance benefits over a network as sending chunks of data is more efficient than sending it bit by bit. The round complexity of the protocol is reduced to $O(|w|)$ without affecting the communication complexity. Thus the actual running time of the protocol is reduced.

6 Applications

A two-party CFLR, as discussed in this paper, can be used for providing web services over internet. It can also help in protecting intellectual property for both the parties. Consider a case where Alice has discovered the context free grammar that can accurately describe a disease. Using the protocol she can keep the discovery to herself while making it available for use through a web service. The interesting part of such a service would be that the patient can be diagnosed without revealing his syndromes. Also the result of the diagnosis would be known only to him. Such a protocol can be useful in a social scenario.

Another use for the protocol can be for providing a compilation service over the network where a user can submit his program to get it syntax checked. Our protocol is stricter than required for this case. In such a case the CFG is public and it is only the input that needs to be hidden.

References

1. A. Yao. Protocols for secure computations: In Proceedings of the twenty-third annual IEEE Symposium on Foundations of Computer Science, pages 160-164. IEEE Computer Society, 1982.
2. M. Ben-Or, S. Goldwasser, and A. Wigderson: Completeness theorems for non-cryptographic fault- tolerant distributed computation. In Proc. of 20th STOC, pp. 1-10, 1988.

3. A. Yao: How to generate and exchange secrets. In Proceedings of the twenty-seventh annual IEEE Symposium on Foundations of Computer Science, pages 162-167. IEEE Computer Society, 1986
4. Bruce Schneier: Applied Cryptography, 2^{nd} Ed., John Wiley & Sons Pte Ltd, pp. 96, pp. 543
5. David Chaum, Claude Cropeau, and Ivan Damgard: Multiparty unconditionally secure protocols. In Proceedings of the twentieth annual ACM symposium on Theory of computing, pages 11-19. ACM Press, 1988.
6. D. Beaver, S. Micali, and P. Rogaway: The round complexity of secure protocols. In Proc. of 22nd STOC, pp. 503-513, 1990.
7. M. Franklin and M. Yung. Communication complexity of secure computation. In Proc. of 24th STOC, pp. 699-710, 1992.
8. O. Goldreich, S. Micali, and A. Wigderson: How to play any mental game (extended abstract). In Proc. of 19th STOC, pp. 218-229, 1987.
9. R.Canetti, U.Fiege, O.Goldreich and M.Naor: Adaptively Secure Computation, Proceedings of STOC 1996.
10. J.Kilian: Founding Cryptography on Oblivious Transfer, Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pages 2031, Chicago, Illinois, 24 May 1988.
11. Peter Linz: An Introduction to Formal Languages and Automata, Narosa Publications.
12. O. Goldreich: Secure Multiparty Computation, Version 1.4, available at http://www.wisdom.weizmann.ac.il/~oded/PS/prot.ps