Yao's Garbled Circuits: Recent Directions and Implementations

Peter Snyder
University of Illinois at Chicago
Chicago, Illinois, USA
psndye2@uic.edu

ABSTRACT

Secure function evaluation, or how two parties can jointly compute a function without any other party learning about any other party's inputs, has been an active field in cryptography. In 1986 Andrew Yao presented a solution to the problem called *garbled circuits*, based on modeling the problem as a series of binary gates and encrypting the result tables. This approach was initial treated as theoretically interesting but too computationally expensive for practical use. However, in the decades since Yao's solution was initially published, much work has gone into both optimizing the protocol for practical use, and further securing the protocol to make it further secure.

This paper provides a thorough explanation of both Yao's original protocol and and its security characteristics. The paper then details additions to the protocol to make it both practical for computation and secure against untrusted parties. Implementations of Yao's protocol are also discussed, though the paper's emphasis is on the underlying enabling improvements to the protocol.

1. INTRODUCTION

Secure function evaluation (SFE) referrers to the problem of how can two parties collaborate to correctly compute the output of a function without any party needing to reveal their inputs to the function, either to each other or to a third party. A common example of this problem is the "millionaires problem", in which two millionaires wish to determine which of them has more money, without either party revealing how much money they have[9].

Many solutions have been developed for SFE. One category of solution is function specific, and depends on specific attributes of the function being executed to provide security[6]. These solutions, while interesting, are by definition of less general interest, since they apply to only a limited set of problems.

Another category of approach is more general, and seeks to provide a general solution for SFE by transforming arbitrary functions into secure functions. Approaches in this category include homomorphic encryption systems[3] which allow for arbitrary execution on encrypted data. Yao's *garbled circuits* protocol fits in this second category.

Yao's *garbled circuits protocol* (GCP) transforms any function into a function that can be evaluated securely by modeling the function as a boolean circuit, and then encrypting the inputs and outputs of each gate so that the party executing the function cannot discern any information about the inputs or intermediate values of the function. The protocol is secure as long as both parties follow the protocol. A full description of the protocol and the related security definitions are provided later in this paper.

1.1 History of Protocol

Interestingly, Yao never published his GCP. Several of his publications discuss approaches to the SFE problem generally, specifically papers from 1982[9] and 1986[10]. These papers are much broader in scope and are much more abstract than providing a protocol that could be implemented. Yao first discussed the *garbled circuits* approach in a public talk on the latter paper, as a concrete example of how his broader strategies could be applied[1]. Only later and by other researchers would the protocol be documented formally[5], though still crediting Yao for the approach.

Yao having developed this foundational protocol, but never having published it, presents authors with the tricky question of what to cite when crediting to the GCP approach. The common approach seems to be to cite Yao's two papers discussing his general approach the problem, even though those papers make no mention of garbled circuits or any similar concept.

1.2 Aims of the Paper

This paper aims to provide a full description of Yao's GCP and its security characteristics, namely what security the protocol does and does not provide. This paper also provides detailed explanations of related work done by other authors to improve the performance and security provided by the protocol.

This paper presumes no previous familiarity with Yao's protocol or cryptography in general in the explanation explanation of the protocol, beyond the general concepts of symmetric and asymmetric cryptography. Some background in cryptography is assumed in the sections on improvements and additions to the protocol. Formal proofs of the underlying concepts are not discussed and are left to their originating papers.

Some discussion is included of existing implementations of Yao's protocol. However, the focus here is on the promises, improvements and general techniques of the implementations, and not on implementation details like programming languages or hardware characteristics. Discussion of the implementations is mainly meant to inform how the protocol has developed and been improved, as opposed to a detailed comparison of how different implementations compare with each other.

1.3 Organization of the Paper

The remainder of the paper is structured as follows. Section 2 provides some security definitions used throughout the rest of the paper. Section 3 discusses oblivious transfer (OT), its role in the protocol, and a method for achieving OT in a manner that is compatible with the security guarantees of the standard version of Yao's protocol. Section 4 then provides a full explanation of the Yao's protocol and how to use *garbled circuits* to solve the *SFE* problem. Section 5 discusses some of the security and performance limitations of Yao's protocol, and sections 6 and 7, respectively, discuss subsequent developments to Yao's protocol to address these issues. Section 8 provides a brief overview of some implementations of the protocol, and section 9 concludes.

2. SECURITY DEFINITIONS

This section defines several security related terms that are used through out this paper. The terminology is not identical throughout the literature, but have mappings onto similar, equivalent terms.

2.1 Properties of SFE System

Attempting to abstractly but precisely defining the characteristics of a SFE protocol is difficult and can quickly devolve into a long enumeration of characteristics a SFE system should *not* do. Instead, Yao suggests[10] that a correct system should be compared to an ideal-oracle that fulfills three properties, and that a SFE system is correct if it performs identically to this imagined ideal-oracle.

This imagined ideal-oracle takes a function to execute (f), the first party (*P1*)'s input (i_{P1}) and the second party (*P2*)'s input (i_{P2}), executes the given function with the values provided, and then returns the function's output to both parties ($u \leftarrow f(i_{P1}, i_{P2})$).

2.1.1 Validity

A SFE system must perform indistinguishably from an ideal-oracle in being able to correctly calculate the given function. Note that this does not guarantee a correct result, since the function being computed in a secure manner could itself have a logic error in it, nor does it guarantee to produce any answer, if one of the parties submits an invalid input to the computation. This *validity* requirement merely requires that the function produce the same result as the insecure (or "pre-secured") version of the function being evaluated, given the same inputs.

2.1.2 Privacy

A SFE system must also perform indistinguishably from an ideal-oracle in preventing preventing *P2* from learning about i_{P1} provided *P1* follows the protocol. The same must also hold for preventing *P1* from learning about i_{P2}.

Note that that this definition of *privacy* does not guarantee that *P1* is not able to learn *P2*'s input by examining the function's result (if the function being executed allows for such reverse engineering). If, for example, the function being evaluated securely is multiplication, the fact that *P1* can learn i_{P2} through u/i_{P1} does not violate this *privacy* property; *P1* could learn i_{P2} in this scenario given an ideal-oracle as well. This does not mean that SFE cannot be used to protect the *privacy* of each parties inputs, only that some functions (such as integer multiplication) do not make sense for two-party SFE.

2.1.3 Fairness

Finally, a SFE system must perform indistinguishably from an ideal-oracle in preventing one party from learning the output of f while learning it themselves. In order words, *P1* should not be able to learn the output of f while denying it to *P2*, and vise-versa.

2.2 Adversary Models

In addition to defining the properties a SFE should have, its necessary to define under which conditions those properties must hold. While the relevant literature contains many different terms for an adversary's willingness to deviate from the protocol, and many gradations between 100% honest and 100% malicious, this paper generalizes the types of attackers into two categories, at the extremes of the attacker spectrum.

A SFE protocol is said to be secure against under a given adversary model if the given SFE protocol can provide the three above mentioned security properties against any party following the assumptions of the adversary model.

2.2.1 Semi-Honest

A *semi-honest* adversary is assumed to follow all required steps in a protocol, but will also look for all advantageous information leaked from the execution of the protocol, such as intermediate values, control flow decisions, or values derivable from the same[4]. Additionally, *semi-honest* adversaries are assumed to be selfish, in that they will take any steps that will benefit themselves if the benefit is greater than the harm, within the constraints imposed by the protocol.

2.2.2 Malicious

A *malicious* adversary is assumed to arbitrarily deviate from the protocol at any point, and in whenever way it might benefit them[4]. This includes proving deceptive or incorrect values, aborting a protocol at anytime, or otherwise taking any steps that could reach a desirable outcome. This is the most difficult type of adversary to secure against; a system that is secure against *malicious* adversaries is also therefor secure against *semi-honest* adversaries.

3. OBLIVIOUS TRANSFER

OT refers to methods for two parties to exchange 1 of several values, with the sending party blinded to what value was selected, and the receiving party blinding to all other possible values that could have, but were not selected.

While OT and SFE are approaches to distinct (though related) problems, understanding the Yao's GCP and the security properties requires some understanding of OT and how it is used in the protocol. Its a cryptographic primitive that serves as a building block that the security of Yao's GCP relies on.

This section provides a brief overview of the OT problem, a simple protocol that provides a solution to the OT problem against *semi-honest* adversaries. The role of OT in Yao's protocol is discussed in section 4.

3.1 Problem Definition

A general form of OT is *1-out-of-N oblivious transfer*, a two party protocol where *P1*, the sending party, has a collection of values. *P2* is able to select one of the values from this set to receive, but is not able to learn any of the other values.

More formally, a *1-out-of-N oblivious transfer* protocol takes as inputs a set of values N from *P1*, and i form *P2*,

where $0 \leq i < |N|$. The protocol then outputs nothing to *P1*, and N_i to *P2* in a manner that prevents *P2* from learning another other values in N.

A special case of the above is the *1-out-of-2 oblivious transfer* problem, where N is fixed at 2. Here *P1* has just two values, and *P2* is accordingly limited to $i \in \{0, 1\}$. All versions of Yao's GCP discussed in this paper rely on *1-out-of-2 oblivious transfer* protocols.

3.2 Example 1-out-of-2 Protocol

The problem of *1-out-of-2 OT* was first addressed by Rabin[8] in 1981 using an online approach with multiple rounds of message passing, but was later adapted into an offline approaches using an techniques similar to the Diffie-Hellman key exchange protocol[2].

The following protocol[7] is a very simple *1-out-of-2 OT* protocol that is secure against *semi-honest*. It is included here to help in the next section's explanation of how the full GCP works, and to provide a easy-to-understand example of OT to build from later.

Semi-Honest 1-out-of-2 Oblivious Transfer

1. *P1* has a set of two strings, $S = \{s_0, s_1\}$.
2. *P2* selects $i \in \{0, 1\}$ corresponding to whether she wishes to learn the first or second value in S.
3. *P2* generates two pairs of public / private keys, $(k_0^{pub}, k_0^{pri}), (k_1^{pub}, k_1^{pri})$.
4. *P2* then destroys k_{i-1}^{pri}, preventing her from recovering any information encrypted under k_{i-1}^{pub}.
5. *P1* generates $c_0 = E_{k_0^{pub}}(s_0)$ and $c_1 = E_{k_1^{pub}}(s_1)$ and sends c_0 and c_1 to *P2*.
6. *P2* computes $s_i = D_{k_i^{pri}}(c_i)$.

Note that that as long as all parties follow the protocol, the protocol is secure by the *semi-honest* definition. *P2* is able to recover the desired string from S but is not able to recover the other value from S. Similarly, *P1* does not know which value from S *P2* learned.

References

[1] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796. ACM, 2012.

[2] W. Diffie and M. E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[3] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[4] O. Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 1998.

[5] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[6] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, volume 201, 2011.

[7] Y. Lindell. Secure two-party computation in practice. Lecture given at Technion-Israel Institute of Technology TCE Summer School 2013, `https://www.youtube.com/watch?v=YvDmGiNzV5E`, 2013.

[8] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[9] A. C.-C. Yao. Protocols for secure computations. In *FOCS*, volume 82, pages 160–164, 1982.

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