Secure Computation

Joe Kilian

ABOUT THE AUTHOR: Joe Kilian is a professor in the department of Computer Science at Rutgers University. His research is in cryptology (theory and applied), algorithms and complexity theory.

Who are they trying to hire?

Suppose your university department is about to make a job offer, and a friend tells you that their department is also about to make an offer. Are you entering into a bidding war, or are the two departments interested in different people?

You open your mouth to ask, "Are you making an offer to X?" then shut it, imagining an answer of, "Actually, we were interested in someone else, but now that you've told me that X is available..."

Your friend is similarly closed-mouthed. You are at an impasse. You both want to find out if you are interested in the same person, but you don't want to reveal anything beyond this one bit of information. What can you do?

This problem, and many others like it, makes simultaneous demands on the privacy and usability of sensitive data that go beyond the capabilities of conventional cryptography. Encrypting sensitive data is analogous to placing jewels in a safe. While in the safe, the jewels are protected from theft, but you can't wear them to the ball. Conventionally encrypted data may be safe from prying eyes, but until it is decrypted it can't be used, even by legitimate parties for agreed upon purposes.

Secure computation

In the 1980s, Andrew Yao introduced the concept of secure computation. Imagine two people, Alice and Bob. Alice has private information x, and Bob has private information y. They wish to determine $f(x, y)$ for some function f, without revealing anything more about x and y than is implied by knowing $f(x, y)$. In our job offer example above, $f(x, y)$ is just the function that is 1 when $x = y$ and 0 otherwise. Yao created a generic solution to this problem that would work not just for this particular f, but for any efficiently computable f. The amount of work done by each party grows proportionally to the number of gates in a circuit computing f – essentially the number of single-bit operations needed to compute f.

Shortly thereafter, Goldreich, Micali and Wigderson generalized Yao's approach to multiple parties. In its most general form, we have n participants; Player i has private input x_i. At the end of the secure computation, player i learns the value of $f_i(x_1, \ldots, x_n)$, for some function f_i agreed upon ahead of time, and nothing else. Note that in general the secure computation may have different functions f_i for different players; each player may learn something different about the combined data. However, we will not use this generality in our later discussions.

Voting has been cast as a secure computation in two different ways. Suppose we are having a yes/no vote, as with a referendum. We can identify 1 with "yes" and 0 with "no". By computing the sum of these inputs, one can figure out how many "yes" votes were cast. For a full-blown election, with write-in votes, each vote may be some general string of characters (with some reasonable bound on the length). For this case, a natural (probabilistic) function to compute is one whose outputs is merely its inputs in random order. This approach is analogous to having each voter put their votes, written on identical slips of paper, into a box that is shaken well before opening.

The solutions of Yao, Goldreich-Micali-Wigderson, and hundreds of subsequent researchers, are beyond the scope of this article. To give some of the flavor of the techniques used, we give in an appendix some toy examples of protocols for testing equality and referendum voting.

Even stating precisely what secure computation should obtain is much harder than it looks, and continues to be the subject of active investigation ([1, 13, 16, 2, 17]). Not only must we consider what each participant learns when everyone follows the protocol, we must consider what a coalition of participants might learn if they pool all of their information together. Worse, we must consider what these colluding participants might learn, or how they might influence the outcome of the protocol, if they willfully violate the rules.

In lieu of a full definition for secure computation, we give the basic motivation used in most of the approaches. Imagine that the players all knew someone of impeccable character and discretion. Each player could then give this trusted center their private value (its input). The trusted center would then compute for each player the function of combined inputs that they were supposed to learn. Its job complete, the trusted center forgets everything that is has learned. This is the "ideal" scenario.

The ideal scenario is not immune from abuse. A colluding group of players can possibly gain some extra information or unduly influence the outcome of the secure computation by coordinating their inputs and pooling all of the information they obtain. But this is relatively innocuous abuse, if it even is abuse, and more to the point is essentially impossible to prevent in most scenarios, though some work has considered how to diminish even this modest level of collusion.

From a thousand feet up, the goal of the cryptographic protocol for secure computation is to ensure that even if some "not too big" group of players collude, they cannot learn more or influence the outcome more than they could in the ideal scenario. As to what constitutes "not too big," this varies from paper to paper, and depends on among other things the methods used, the precise goals, and the underlying communication network. A common requirement is that the colluders comprise less than half the total number of players: It would be terrible if a collusion of 10% of the voters could determine the outcome of the election. However, more general possibilities have been considered than just looking at the size of the possible sets of colluders (c.f. [11, 6]).

In a nutshell, there is Diogenes's approach – find an honest man and trust him to combine the secret data,[1] and there is the cryptographer's approach – develop a clever protocol to be run by untrusted entities. The goal of the cryptographer is to simulate Diogenes's trusted man even if he doesn't exist.

[1]We leave as an exercise for the reader how to double the efficiency of Diogenes's approach.

Secure computation in real life

But can we do it? For the first decade or so since the invention of secure computation, the answer was a resounding, "Yes!...but not really." For a theorist, a solution that works in time proportional to the number of bit operations is absolutely wonderful. However, in practice we have been spoiled by modern computers that can perform a trillion gate operations in the blink of an eye. Even seemingly modest functions take many, many bit operations to compute. For each bit operation, the generic protocol of Yao required the work equivalent of computing several encryptions. Performing the work of a few encryptions per bit operation is an extremely daunting slowdown.

But times have changed. First, computers have become faster while the bit complexity of interesting functions have remained unchanged. We are catching up. For example, truly private sealed-bid auctions require relatively modest computations. Systems have now been developed for specifying and implementing generic secure computation in practice (e.g., [12]).

For a number of useful secure computation problems, special-purpose protocols are vastly more efficient than the generic constructions. In particular, the techniques for electronic voting protocols have been continuously refined for nearly a quarter of a century since Chaum's seminal work on mix-nets [3]) Modern protocols would allow modern PCs to act as vote counters for large scale elections (c.f., [4, 8, 14, 15]).

Whither voting?

It would be misleading to suggest that we can take the results for electronic voting out of the box. The real world has far more complicated constraints in terms of what we want to get out of our voting systems and what assumptions we are willing to make about the reliability of systems. The major systems issues aside, basic issues of protocol design remain, such as how best to allow a voter to verify that their vote has been counted while preventing the selling of votes. The ideal electronic voting scheme is still a matter for the future and will be shaped as much by social concerns as by technical considerations.

Yet it appears that we are stepping into the age of electronic voting, ready or not. It must be pointed out, vociferously, that the electronic voting machine you are likely to encounter employs essentially none of the techniques for secure electronic voting that have been developed over the last 25 years. A detailed discussion of current systems is well beyond the scope of this article. (An appraisal of one commonly used system was given in [10], with additional material given in [18].) But from the theoretician's thousand-foot high perch, some comments may be made.

From the theoretician's viewpoint, these machines are essentially trusted entities. Trusted entities are comparatively easy to model and work with. However, it is an axiom of cryptography (and one that surely predates even this ancient art) that the greater the trust, the greater the opportunity for betrayal. If at the end of the day, the trusted entity says that no one has voted, or that more people voted than could ever have shown up, or that it has forgotten the vote count, there is no guarantee of any recourse – no groundwork has been laid for a "plan B." This is not an observation about a specific product or technology – it applies almost equally well to old-fashioned mechanical voting machines as it does to modern touch-screens.

The lesson of the last 25 years is that we can do better. For example, electronic voting machines can as a backup send out a data stream that would allow for much better auditing, recounting and failure recovery than is possible using previous technologies, while protecting the privacy of individual voters.

It would be a truly exceptional circumstance if the best technology for a problem did not incorporate the last few decades of research on that problem. However exceptional, even magical the study of secure computation can be, we believe it fits the norm if only in this one respect.

References

[1] D. Beaver, Foundations of secure interactive computing, In Advances in Cryptology – Proceedings of Crypto '91,377–391, Springer Verlag LNCS, vol. 576, 2001.

[2] R. Canetti, Universally Composable Security: A New Paradigm for Cryptographic Protocols, IEEE Symposium on Foundations of Computer Science, 136–145, 2001.

[3] D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms", Communications of the ACM 24, 1981 .

[4] R. Cramer, R. Gennaro and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Advances in Cryptology – Proceedings of EUROCRYPT'97, 103–118, Springer Verlag LNCS 1233, 1997.

[5] U. Feige, J. Kilian, and M. Naor, A Minimal Model for Secure Computation, Proceedings of the 26th Symposium on the Theory of Computing (STOC), 554-563, ACM Press, 1994.

[6] M. Fitzi, M. Hirt and U. Maurer, General Adversaries in Unconditional Multi-Party Computation Advances in Cryptology - ASIACRYPT '99, 232–246, Springer-Verlag LNCS 1716, 1999.

[7] R. Fagin, M. Naor and P. Winkler, Comparing Information Without Leaking It, Communications of the ACM, vol 39, May 1996, pp. 77-85.

[8] J. Furukawa and K. Sako, An Efficient Scheme for Proving a Shuffle In Advances in Cryptology – Proceedings of Crypto 2001, 368 – 387, Springer LNCS 2139, 2001.

[9] O. Goldreich, S. Micali and A. Wigderson, How to play ANY mental game, Proceedings of the nineteenth annual ACM conference on Theory of computing, 218-229, ACM Press, 1987.

[10] T. Kohno, A. Stubblefield, A. D. Rubin, D. S. Wallach, Analysis of an Electronic Voting System, IEEE Symposium on Security and Privacy, Oakland, CA, May, 2004.

[11] M. Hirt and U. Maurer, Complete characterization of adversaries tolerable in secure multi-party computation. In Proceedings, 16th ACM Symposium on Principles of Distributed Computing (PODC), 25–34, 1997.

[12] D. Malkhi, N. Nisan, B. Pinkas and Y. Sella, Fairplay — a secure two-party computation system, Proceedings, Usenix Security Symposium 2004.

[13] S. Micali and P. Rogaway, Secure computation, In Advances in Cryptology – Proceedings of Crypto '91, 392–494, Springer Verlag LNCS 576, 1991.

[14] A. C. Neff, A verifiable secret shuffle and its application to e-voting. In CCS 01: Proceedings of the 8th ACM conference on Computer and Communications Security, 116–125, ACM Press, 2001.

[15] K. Peng, C. Boyd and E. Dawson, Simple and Efficient Shuffling with Provable Correctness and ZK Privacy, Proceedings, Advances in Cryptology – Crypto 2005, 188–204, Springer LNCS 3621, 2005..

[16] B. Pfitzmann and M. Waidner, Composition and integrity preservation of secure reactive systems, ACM Conference on Computer and Communications Security, 245–254, ACM Press, 2000.

[17] M. Prabhakaran and A. Sahai, New notions of security: Achieving universal composability without trusted setup. In STOC '04: Proceedings of the 36th annual ACM symposium on Theory of computing, 242–251, ACM Press, 2004.

[18] A. D. Rubin, http://avirubin.com/vote/.

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

Appendix: Some toy solutions

Testing Equality

The simplest way to test equality is to enlist the aid of a trusted third party, tell them the names, and have them tell you if they are equal. However, the third party then knows who you are talking about. Here is a way to make the third party learn only whether they are the same (what you need to know), but nothing else.

There are many approaches to this problem, see [7] for a large collection of methods. The one we present is a special case of a framework given in [5] (though this specific method predates that paper).

First, you must agree on some encoding system to convert names into numbers, and choose some upper bound on the largest number you might reasonably come up with (certainly, nearly all names can be encoded by a 1000 bits number). Choose a prime p bigger than this upper bound. For the rest of this discussion, we consider the essentially equivalent problem of determining whether $x \equiv y \bmod p$, without revealing x or y, where Player 1 knows x and Player 2 knows y.

First, there is a setup phase. Together the two players choose $a \bmod p$ such that $a \not\equiv 0 \bmod p$. They also choose $b \bmod p$ at random (here, b may equal $0 \bmod p$. They do *not* reveal a or b to the trusted third party.

Next, Player 1 sends $m_1 = ax + b \bmod p$ to the trusted third party; similarly, Player 2 sends $m_2 = ay + b \bmod p$. The trusted third party announces "Equal" if $m_1 = m_2$ and "Not Equal" if $m_1 \neq m_2$. It is not hard to show that if $x = y$ then $m_1 = m_2$ and is distributed randomly over numbers mod p. If $x \neq y$ then m_1 will be distributed randomly over numbers mod p, and m_2 will be distributed randomly over numbers mod p that are different from m_1. Thus, all information about x and y except for their equality (and a trivial upper bound on these numbers) is wiped out by the randomization process. So even if the third party isn't so trustworthy, it won't learn anything. Note, however, that if it can enlist the aid of one of the players, they can learn the other player's number.

Referendum Voting

Suppose that Player i has an input x_i, where $x_i = 1$ denotes "yes" and $x_i = 0$ denotes "no". Suppose that everyone is honest and is willing to abide by the rules. Then computing the number of "yes" votes amounts to computing the sum $x_1 + \cdots + x_n$ (n is the number of players). Here is a way n honest parties can do so without anyone learning who voted what; this protocol, or a version of it, has been attributed to Rabin.

First, they choose and $m > n$ arbitrarily (m need not be prime). Player i chooses n numbers

$$x_{i,1}, \ldots, x_{i,n},$$

mod m, at random subject to the constraint

$$x_i \equiv x_{i,1} + \cdots + x_{i,n} \bmod m.$$

One way to do this is to choose $x_{i,1}, \ldots, x_{i,n-1}$ at random mod m, and then choosing

$$x_{i,n} \equiv x_i - (x_{i,1} + \cdots + x_{i,n-1}) \bmod m.$$

Next, each Player i sends to each Player j the value of $x_{i,j}$. After this step, Player i knows $x_{j,i}$ for $1 \le j \le n$.

Finally, each Player computes and announces the partial sum,

$$\text{sum}_i = \sum_{i=1}^{n} x_{j,i} \bmod m,$$

and the total sum is given by

$$x_1 + \cdots + x_n = \sum_{j=1}^{n} \text{sum}_j \bmod m.$$

The above equality can be easily shown by

$$\begin{aligned} \sum_{i=1}^{n} x_i &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_{i,j} \\ &= \sum_{j=1}^{n}\sum_{i=1}^{n} x_{i,j} \\ &= \sum_{j=1}^{n} \text{sum}_i, \end{aligned}$$

where all arithmetic is done mod m. We leave the privacy of this protocol as an exercise for the reader.

It should be noted that if any party is dishonest, they can, for example, cast a double "yes" vote by choosing $x_{i,1}, \ldots, x_{i,n}$ that sums to 2 mod m.

Also, note that there is nothing special about x_i being 0 or 1. Any set of nonnegative integers (with some known upper bound) can be privately added by choosing m larger than the largest possible sum. As an application, a group of players can compute their average salary.