Is Multiparty Computation Any Good In Practice?*

(Draft version of a paper appeared at ICASSP 2011.)

Claudio Orlandi

Abstract

The aim of this paper is to present some of the recent progress in *efficient* secure multiparty computation (MPC). In MPC we have a set of parties owning a set of private inputs. The parties want to compute a function of their inputs, but they do not trust each other, therefore they need a cryptographic protocol to perform the computation in a way that 1) the output is correct and 2) cheating parties will not be able to learn any information about the honest parties inputs.

Even though this problem has been formulated and essentially solved almost 30 years ago, practical solutions that can be relevant for real-world applications have been discovered only in the last few years. We will present some of these advances, trying to explain to a non-specialized audience the significance of the several existing security notions.

1 Introduction

Secure Multi Party Computation (MPC)[1] is arguably the most general and important problem in cryptography: We have a set of parties $P_1, \ldots, P_n$ each with his own private input $x_1, \ldots, x_n$ that want to compute some function of their inputs $y = f(x_1, \ldots, x_n)$. However the parties do not trust each other, and they do not want to reveal their inputs to the other parties. A possible solution is for the parties to agree on a commonly trusted entity T: then T could gather the inputs, compute the function on behalf of the parties, and announce the result. The feasibility results from the '80s tell us that the parties can replace the trusted entity T with a cryptographic protocol while still preserving the same level of security that the trusted entity intuitively provides. These early protocols are however very far from being practical.

Electronic elections, electronic auctions, secure benchmarking and secure signal processing are only few of the many applications that can be seen as instances of the MPC problem – finding practically efficient protocols is therefore of prime importance.

For many years general purpose MPC protocols (i.e. protocols that allow to compute *any* function) have been considered only of theoretical interest, and some researchers focused on cryptographic protocols for specific tasks, such as voting, data-mining, set operations and so on.

In the last few years this belief has been attacked by drastic improvements and implementations of MPC protocols that showed how, even if still computationally costly, MPC is of practical interest: In January 2008, for the first time MPC has been used to move real money in the in the context of a nation-wide auction for sugar-beets contracts among farmers in Denmark [BCD+09]: some 1200 farmers employed an MPC protocol to determine the market price of sugarbeets contracts without having to reveal their (sensitive) selling and buying prices and without resorting to an external trusted party. The whole computation lasted around 30 minutes, more than a reasonable time for a once-a-year kind of task.

[1] Due to space constraint, references will be provided only for results that are not covered by standard cryptographic textbooks, such as [Gol04, KL08].

Our thesis is that generic MPC protocol are getting ready to be deployed in the real world, and they will only become faster in the next years, allowing to solve more and more real privacy sensitive scenarios: researchers and engineers in all field of computations should therefore be aware of what MPC is and what MPC can do for them.

2 Models of Computation

The world of generic MPC protocols is split into protocols that allow to evaluate Boolean circuits and protocols that evaluates arithmetic circuits (with $+, \times$) gates. Both model can express the same kind of functions, up to a polynomial blow-up (arithmetic operations can be performed by a Boolean circuit and vice versa). However, the number of cryptographic operations (that dominates the complexity of MPC protocols) are usually proportional to the number of gates in the circuit, and therefore one can gain in efficiency by choosing the right model of computation: for instance securely computing a multiplication between ℓ-bit numbers with a Boolean circuit might be up to ℓ^2 more expensive than with an arithmetic one. On the other hand there are plenty highly non-linear operations, like comparisons, that can be computed more efficiently using the Boolean representation. It is also possible to use a mixed approach, where different parts of the function are computed using different representations as shown by [KSS10].

3 Security Models

What is a *secure* cryptographic protocol? Intuitively, we would like that the protocol cannot be successfully attacked by any adversary. Unfortunately, such a statement cannot be checked empirically in the way we empirically check if the protocol produce the right output. How can we be sure that we tried every possible attack?

Sometimes there is no way around it, and security is essentially conjectured: as an example, we believe that the RSA cryptosystem is secure essentially because no one found a way to break it – we don't even know if breaking RSA is as hard as factoring! On the other hand, *some* assumption has to be trusted if one wants to use cryptography at all, so it might as well be the RSA assumption – especially given that it has been studied extensively.

On the other hand, it is a really bad idea to use an MPC protocol that is only conjectured to be secure: an MPC protocol is a complex object, where things can go wrong in so many different ways. Therefore, MPC protocols come with a proof of security, or *reduction*, to a simpler, well studied assumption. A reduction is basically a program that uses an adversary that breaks the MPC protocol as a subroutine to break the underlying assumption. It means that if the adversary is able to break our protocol, then the adversary is essentially able to break the cryptographic assumption (e.g. RSA). As we believe that there is no efficient algorithm to break RSA, it means that there are no ways of attacking our protocol. In other words, we *need* users to believe in our conjecture about the hard problem: however, one should not ask the users for *any more trust than what is strictly needed*.

Hard Problems. It seems unlikely that we will ever be able to construct MPC protocols using symmetric cryptographic primitives (e.g. hash functions, symmetric ciphers) only. We need to rely on the intractability of some factoring, discrete logarithms or lattice related problem. Without entering into the details, discrete logarithm over an elliptic curve seem to be an harder problem than factoring or discrete logarithm in other finite fields, allowing one to obtain the same security level using much shorter public keys: a 200 bit long elliptic curve public key offers roughly the same security as a 2048 bit long RSA public key [II10]. The recent introduction of lattice based cryptography [MR09] is interesting especially because, while an adversary using a quantum computer could efficiently factor and solve discrete logarithms, there is no known quantum attack to solve lattice problems.

Defining Security. For an MPC protocol to be secure some properties must be satisfied: input privacy, output correctness, etc. How do we know when the list is complete? It turns out that it is better to define the security of an MPC protocol using the *simulation* paradigm: we define an ideal world where a trusted entity runs the computation for the parties. As we assume the third party to be honest, this world is secure by definition. Then we have the real world, where the parties run the cryptographic protocol and an adversary can try to attack it. We say that the protocol is as secure as the ideal world if, for every real adversary that attacks the protocol there is an ideal adversary that can *simulate* the same attack in the ideal world. The existence of such a simulator guarantees that the protocol is secure, as the ideal world is secure by default.

How many adversaries? We usually consider a *monolithic* adversary i.e., if two or more parties are corrupted, we assume that they all cooperate with each other, and we would like to guarantee security even when there is only one honest party left (this is clearly the meaningful security goal in the crucial two-party case). Lower bounds show that protocols that are secure when only one party is honest are inherently expensive, as they require a considerable amount of expensive cryptographic operations.

If one assumes that the adversary is restricted in corrupting a fraction of the parties, the lower bound can be beaten. In particular, if more than $1/2$ of the parties are honest, then it is possible to use MPC protocol that do not require *any cryptographic assumption*: this is interesting for two reasons: 1) this kind of protocols are secure no matter what the computing power of the adversary is and 2) these protocols only use simple arithmetic operations and they run at least thousands of times faster than protocols that makes extensive use of cryptographic primitives. In a way, these protocols can be thought as the *one-time-pad* of MPC: perfect security is possible only under really strong assumptions.

Can I trust you tomorrow? Another distinction is made between *static* or *adaptive adversaries*: the former kind chooses which parties to corrupt *before* the protocol starts, while the latter can choose to corrupt a new party at any time, eventually based on the information gathered so far. Adaptive security is extremely expensive to achieve (even for simple tasks as encryption) and in practice static security is accepted as the proper goal to shoot for.

How dishonest? We can classify adversaries according to their strategies in at least four different ways: the most general adversary is an *active* one: an active adversary can deviate from the protocol specification as he wants, and will try to collect as much information as he can about the honest parties inputs. A weaker adversary, a *passive* one, will instead follow the protocol specification and try to extract more information from his view in the protocol. Considering passive adversaries makes sense as it is known (from a theoretical point of view) how to compile a protocol secure against a passive adversary into one secure against an active adversary. Also, there might be scenarios where passive security is enough. In between a passive and an active adversary one can imagine a *covert* adversary, that does not necessarily follow the protocol specification, but does not want to be caught in doing so: would a company risk to loose its credibility to learn some secret if they will be caught with probability 99.9%? Depending on the actual value of the secret inputs, one can choose whether covert security is enough for the specific application. Protocols for covert security are usually more efficient than protocol for active security. Another kind of adversary is the *rational* one, that has some well defined objective (specified by an utility function) and will cheat in the protocol only in order to maximize this utility function. This is a new area of research that brings together cryptography and game theory, and can produce interesting results for specific applications such as electronic auctions [MNT09].

Person-In-The-Middle. Even if only parties $P_1, \ldots, P_n$ are supposed to take part in the protocol, we cannot exclude that other parties are connected to the same network (as it is the case over the Internet) that can run their own attacks to the protocol. Also, 2 copies of the same protocol could be running – potentially with different sets of parties – and an adversary could use the information that he obtains in one run to attack the other run of the protocol. This is an extremely sensitive issue, and there are numerous

examples of protocols that were proved secure in a world with only two parties that were later broken by person-in-the-middle kind of attack.

A notion of security that accounts for this is the universally composable (UC) security framework [Can01]. Unfortunately, is is impossible to construct MPC protocols with UC security, unless some *trusted setup* is available, like the presence of a common random string (CRS) or some form of trusted public key infrastructure (PKI): when the protocol is deployed, it is crucial that the way the trusted setup is implemented is as close as possible to the idealized version – for instance, measuring some physical phenomenon to extract a random string. Even if the PKI model seem to be a strong assumption, one has to remember that most protocols for the CRS model assume that parties are connected via secure and authenticated channels. In order to implement these channels, some kind of public key infrastructure is needed already, and therefore one could in fact claim that the PKI model is more realistic than the CRS model.

Should we even compute this? A final, important remark is that the way we define security for MPC ensures only that running the protocol is as good as letting a trusted party run the computation. It does not say whether *the computation itself* is going to leak information about the secret inputs. For instance, even if one runs an MPC protocol to compute the sum of two inputs, the adversary will always be able to learn the other input only by looking at the input/output of the computation. A different line of research, called *differential privacy* [Dwo08] focus instead on how much information is leaked by specific kind of computation, e.g. with respect to querying a database containing privacy-sensitive data.

4 MPC Protocols

Boolean MPC. Virtually every MPC protocol that allows to evaluate a Boolean circuit is based on Yao's garbled circuits: we have two parties, a circuit constructor A, and a circuit evaluator B. A encrypts, or garbles, a Boolean circuit and send it to B. Then B receives from A a key X_a associated to A's input a. B needs to get the key Y_b, but A should not learn B input bit b: thus B retrieves his keys using an *oblivious transfer* i.e. a protocol where A inputs Y_0, Y_1 and B inputs b and learns Y_b, while A learns nothing. The gates are garbled as follow: if A wants to garble an AND gate A computes four ciphertexts $C_{00} = E_{X_0,Y_0}(Z_0), C_{01} = E_{X_0,Y_1}(Z_0), C_{10} = E_{X_1,Y_0}(Z_0), C_{11} = E_{X_1,Y_1}(Z_1)$, permutes them at random and sends them to B, that can decrypt only the ciphertext encrypted using X_a, Y_b and therefore learns $Z_{a \wedge b}$. Here we assume that B can distinguish a correct decryption from a bad decryption: this can be easily obtained by appending some 0's after the plaintext. Now a ciphertext decrypted using the wrong keys will produce a random output and will not be of the right form, while only the ciphertext decrypted with the right key will contain the 0's. The new key can be used as input to a new layers of encrypted gates, or can be "decrypted" by simply having A announce to B the pairs $(0, Z_0), (1, Z_1)$.

Yao's protocol is only secure against passive adversaries. The most promising attempt of making Yao's protocol secure against an active adversary in an efficient way can be found in [LP11]. Without entering in the details, a problem with Yao's protocol is that A can send to B a circuit that compute a function different from the one they agreed upon. To deal with this, a *cut-and-choose* technique can be employed, where A sends multiple copies of the garbled circuit to B. Then B selects a random subset of the garbled circuits, and A "opens" them to show that they are actually computing the desired function. If B detects any incongruence, the protocol aborts; otherwise, with very high probability a majority of the unopened circuits are also computing the right function, and B proceeds in evaluating them.

Arithmetic MPC. Another common approach to build MPC protocol is to start from some additively homomorphic encryption: this is and encryption scheme where if $C_1 = E(x)$ and $C_2 = E(y)$ then $C_1 \otimes C_2 = E(x+y \mod p)$, where $\otimes$ is some well defined operation between ciphertexts and p is some integer modulus. Additively homomorphic encryption allows for some kind of secure computation: A can send $C_1 = E(a)$ to B, that using the additively homomorphic property can compute $C_2 = E(ab - r)$. Now A can decrypt and obtain $s = ab - r \mod p$, and therefore A and B now have a *additive* secret sharing r, s of the product

between their inputs a, b s.t. $r + s = ab \mod p$. Note that we want the output to be secret shared among the parties, so that it can be used again to compute new products and additions in an iterative fashion.

The most efficient solution we are aware of for this setting is the BeDOZa protocol [BDOZ11]. In this work the authors define a relaxation of the homomorphic property defined above, so that many cryptosystems (based on factoring, discrete logarithm, or lattice based assumptions) can be used to implement the protocol. Moreover, the protocol offers a preprocessing flavor: in a first phase, where neither the inputs nor the function to be computed have to be specified, some random multiplications are computed together with some control values. In a second phase, this precomputed material can be used to evaluate the actual circuit using only inexpensive computation. Very informally one can think that a multiplication gate costs 2 seconds of preprocessed computation and only 10ms during the online phase (additions are essentially for free). This offline/online approach is particularly useful in settings where parties know in advance that some computation has to be performed, and *low online latency* is desired. A classical example is the one of an airline company that wants to check the list of passengers on a flight against a database of blacklisted passengers (and nor the list of passenger nor the blacklist should be publicly disclosed). Here the final list of passengers might be ready only few minutes before take-off, while the flight has been scheduled way in advance.

Recently a fully-homomorphic encryption scheme, that is a cryptosystem that allows to compute any function inside the ciphertext (as opposed to addition only) has been proposed [Gen09]: this approach leads to huge improvements in communication complexity. However, with the current state of the art, the computational overhead is too big for practical applications (public keys are gigabytes of data).

5 Public Implementations

Here is a list of implementations of MPC protocols available online: The **Fairplay** project [BDNP08, MNPS04] (`http://www.cs.huji.ac.il/project/Fairplay/`), allows for computation of Boolean circuits. The system can be used for two or more parties, with passive security. In the two party case, it is secure also against covert adversary (where if the adversary cheats he will get caught with probability 90%). The **VIFF** Project [DGKN09] (`viff.dk`), allows instead to compute with arithmetic circuits: born as a solution for honest majority only, it contains now runtimes for dishonest majority and active adversaries. **Sharemind** [BLW08] (`http://research.cyber.ee/sharemind/`) is a framework for 3 parties, with at most 1 passive adversary. The **TASTY** [HKS+10] (`tastyproject.net`) project covers the 2-party case (passive security), but interestingly allows to compute using both the Boolean and the arithmetic representation.

6 Conclusions

Generic MPC is a fast moving field: In 2009 the first implementation of MPC for 2 parties with active security [PSSW09], was able to evaluate a circuit of $\sim 3 \cdot 10^4$ gates in $\sim 10^3$ seconds. Only two years after, the same circuit is evaluated in less than 5 seconds [NNOB11]. Looking at it from a different angle, we went from securely emulating a trusted third party running a 30Hz CPU, to emulating a trusted third party running a 6kHz CPU.

Even if the price for security is still high (when compared to the situation where a trusted third party run the computation for the parties), the actual speed of MPC protocols is still very promising, especially considered what could be done only a few years ago.

Thanks to the continuous improvement in protocol design, the careful implementations of the cryptographic primitives and the constant increase in computing power, we expect MPC protocols to run faster and faster in the future. It is not unreasonable to expect to see MPC protocols running at megahertz speed soon, thus allowing to use MPC in more and more scenarios where mutually distrusting parties desire to perform some cooperative computation over their inputs.

References

[BCD+09] P. Bogetoft, D. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. Nielsen, J. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In *Financial Cryptography*, 2009.

[BDNP08] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM CCS*, 2008.

[BDOZ11] R. Bendlin, I. Damgrd, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, 2011.

[BLW08] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, 2008.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[DGKN09] I. Damgård, M. Geisler, M. Krøigaard, and J. Nielsen. Asynchronous multiparty computation: Theory and implementation. In *PKC*, 2009.

[Dwo08] C. Dwork. Differential privacy: A survey of results. In *TAMC*, 2008.

[Gen09] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[Gol04] O. Goldreich. *Foundations of cryptography, Volume II, Basic Applications*. Cambridge University Press, 2004.

[HKS+10] W. Henecka, S. Kögl, A. Sadeghi, T. Schneider, and I. Wehrenberg. Tasty: Tool for automating secure two-party computations. In *ACM CCS*, 2010.

[II10] ECRYPT II. Yearly report on algorithms and keysizes, 2010.

[KL08] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2008.

[KSS10] V. Kolesnikov, A. Sadeghi, and T. Schneider. Modular design of efficient secure function evaluation protocols. Cryptology ePrint Archive, Report 2010/079, 2010.

[LP11] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *TCC*, 2011.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, 2004.

[MNT09] P. Miltersen, J. Nielsen, and N. Triandopoulos. Privacy-enhancing auctions using rational cryptography. In *CRYPTO*, 2009.

[MR09] D. Micciancio and O. Regev. Lattice-based cryptography. In *Post-Quantum Cryptography*. 2009.

[NNOB11] J. Nielsen, P. Nordholt, C. Orlandi, and S. Sheshank Burra. A new approach to practical active-secure two-party computation. Manuscript, 2011.

[PSSW09] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *ASIACRYPT*, 2009.