Survey on application models and usage scenarios for various SMC protocols

Dan Bogdanov[2] and Riivo Talviste[1,2⋆]

[1] University of Tartu, Institute of Computer Science
[2] Cybernetica AS, Institute of Information Security

1 Introduction

European Union funded research project Usable and Efficient Secure Multiparty Computation (UaESMC)[3] deliverable D1.1 [11] lists various application scenarios where secret sharing based secure multiparty computation (SMC) can be used. Inspired by their results, this work tries to add one more level of abstraction and compare different secure evaluation techniques from the point of applicability in different scenarios. We will consider the most suitable and most motivated case for each SMC technique and argue if and how other SMC techniques could be used to solve them.

2 SMC protocols

In the following multiparty protocols, each party has one or more of three roles:

1. Input party ($\mathcal{IP}$) - the initial owner of the secret data who inputs it into the system;
2. Computation party ($\mathcal{CP}$) - the party who participates in the cryptographic protocol evaluating a function on secret data;
3. Result party ($\mathcal{RP}$) - the one who is interested in the result of the computation.

2.1 Yao's construction

Assume that we have two parties $\mathcal{P}_1$ and $\mathcal{P}_2$ who want to compute an arbitrary functionality $f(x_1, x_2) = (y_1, y_2)$ so that $\mathcal{P}_i$ holds the input x_i and at the end, receives the output y_i $(i \in \{1, 2\})$. We assume that the functionality f is a boolean circuit and inputs x_1 and x_2 are bitstrings. In Yao's construction [17], $\mathcal{P}_1$ encrypts ("garbles") the circuit to preserve the privacy of the input values. For each wire in the circuit, two random values are chosen, one representing 0

⋆ This research was supported by European Social Fund Doctoral Studies and Internationalisation Programme DoRa and by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS.

[3] Usable and Efficient Secure Multiparty Computation – `http://www.usable-security.eu`.

and another representing 1. For a gate g with inputs $b_1 \in \{0,1\}$ and $b_2 \in \{0,1\}$, the random values of input wires of g corresponding to each pair of the values of b_1 and b_2 are used as keys to encrypt the value corresponding to $g(b_1, b_2)$ of the output wire of g. To hide the operation that the gate performs, each gate's computation table is also randomly permuted, so that it maps random inputs to the corresponding random output. $\mathcal{P}_1$ constructs the garbled circuit and sends it to $\mathcal{P}_2$ along with its encrypted inputs. While evaluating the garbled circuit, $\mathcal{P}_2$ uses 1-out-of-2 Oblivious Transfer (OT) for each of its input bits to get the corresponding encrypted input value from $\mathcal{P}_1$. If needed, $\mathcal{P}_2$ then forwards the circuit outcome back to $\mathcal{P}_1$. A more in-depth description of this method along with the security proof is given in [9].

2.2 Fully homomorphic encryption

Homomorphism is a property of some public key cryptosystems that allows one to modify encryptions in a meaningful way. For example, cryptosystems like Paillier [10], lifted ElGamal [6] and Damgård-Jurik [5] are additively homomorphic, meaning that one can add together encrypted values or multiply them with public constants without knowing the corresponding private key. However, for multiplication of encrypted values, help from the party holding the private key is required.

The first provably secure fully homomorphic encryption (FHE) scheme, i.e. a scheme that is both additively and multiplicatively homomorphic, was introduced by Gentry [7] in 2009. Gentry proposed a somewhat homomorphic scheme that was only able to evaluate low degree polynomials on encrypted data because of the cumulation of noise that would eventually render the encryption indecipherable. He then showed how to make this scheme *bootstrappable* by homomorphically recrypting the ciphertext yielding in a fresh noise free ciphertext. Unfortunately, this operation is costly and thus difficult to use in practical applications. Since then, this scheme has been built upon and refined to be more practical [15,14,8].

2.3 Secret sharing based SMC

Secret sharing based secure multiparty computation or simply secure multiparty computation (SMC) uses n computation parties that must be independent and not collude. Each input party *secret shares* his input x_i into n shares $[x_i]_1, \ldots, [x_i]_n$ and distributes them among the $\mathcal{CP}$-s so that $\mathcal{CP}_j$ gets share $[x_i]_j$. Depending on the secret sharing scheme used, all (for additive and bitwise sharing) or at least some *threshold* of shares (e.g. for Shamir's secret sharing scheme [13]) are required to reconstruct the original value.

The computation parties then execute an interactive protocol to evaluate a function on the input values. As a result, each $\mathcal{CP}$ learns a single share of the output that can be used in later computations or sent to the result party who can reconstruct the function output value.

3 Application models

In this paper we will consider three common scenarios where SMC is considered suitable. We will argue how each of the different SMC protocols can be used in each scenario and what are their advantages and disadvantages in each case.

The three scenarios are

1. a case with many input parties and separate result parties that receive aggregated results from the SMC instance. For example, auctions and many other scenarios where aggregated results are computed from sensitive data;
2. a case where a single input party is outsourcing extensive computations on encrypted data, for example, to a cloud;
3. a case with two input/result parties who want to compute a joint function on their sensitive data, e.g. Yao's millionaire problem [16].

3.1 Case 1

The first scenario involves several input parties and one or more result parties that are interested in some function results computed from inputs. The application model is shown in Figure 1a. The number of computation parties depends on the SMC technique that is used. In case of secret sharing based SMC and multiparty Yao, there are three of more $\mathcal{CP}$-s, while with using FHE one $\mathcal{CP}$ might be enough, depending on the amount of homomorphic computations needed.

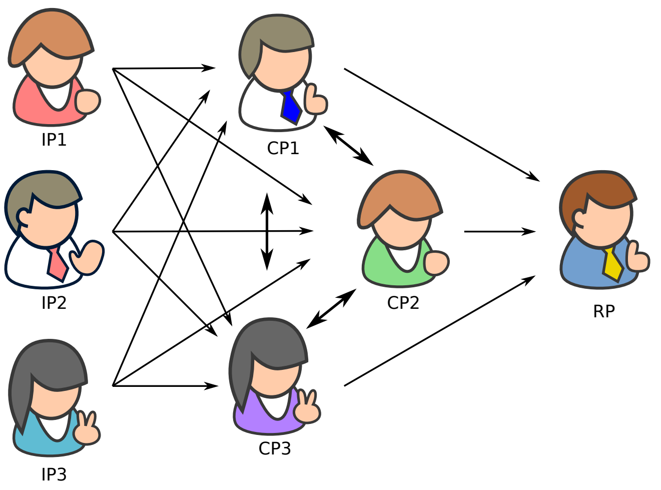
The intuition that this case is most suitable for secret sharing based SMC is illustrated by the fact that it has been successfully used to solve real life problems. For the first time, it was used in 2008 in Denmark to conduct a secure double auction [4] and the second time was in 2011 in Estonia where the Association of ICT companies used it to see how their sector is doing financially [3]. In both these cases there were several parties interested in the computation result, but from the application model's view in the second case there was a single result party that ordered the computation and received the result. It was then responsible for distributing it to the others.

With fully homomorphic encryption and a single result party (see Figure 1b) this application scenario is natural — all input parties encrypt their values with the public key of the result party, the computing party perform homomorphic operations on those and the result party is able to decrypt the outcome with its private key. In that sense, the result party "owns" the data as it is the only one who can read it. However, with several result parties with different private keys, it comes increasingly difficult to realize it. Different input parties may encrypt their data under different public keys and the computation party can use the bootstrapping function to recrypt them under another public key, but in the end the result must still be encrypted under a single public key so whoever has the corresponding private key, is the data owner. As a possible solution, one may use a hybrid setup where result parties secret share the "master" private key so they must cooperate to decrypt the final result [7].

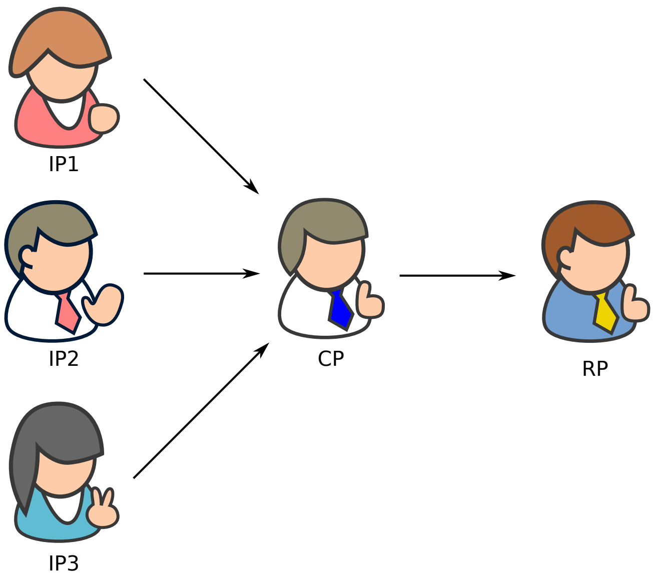
Similarly, with several input and/or result parties, this application scenario is not realizable with classical two-party Yao's construction. However, there exist multiparty extensions for Yao's construction [1]. For example, FairplayMP [2] allows several input and result parties that use secret sharing and computation parties that use secure multiparty computation to produce a separate garbled circuit for each result party that can then evaluate it. In this case the communication model is similar to using secret sharing based SMC.

Trust and resources. With secret sharing based SMC there is a need to choose and set up computation parties so that they will not collude with each other. Depending on the given scenario, collaborating parties may take the role of $\mathcal{CP}$-s, eliminating the need to search for trusted external parties. For example, in the double auction held in Denmark in 2008 the buyer took the role of one $\mathcal{CP}$, the representative of the sellers took the role of the second $\mathcal{CP}$ and an independent third party responsible for the technical details of the auction played the role of third $\mathcal{CP}$. The aforementioned project in Estonia in 2011 also had three computation parties. However, since the collaborating parties were all ICT companies capable of hosting a computation service, all the the $\mathcal{CP}$-s were elected among themselves. As implementing this case with Yao's garbled circuits requires a hybrid model with many computation parties and secret sharing, the same trust and resource handling principles apply in this case.

With a single computation and a single result party implementing this case with fully homomorphic encryption is easiest as only one $\mathcal{CP}$ instance has to be set up. Also, there are no trust issues as everything that the $\mathcal{CP}$ sees is encrypted under $\mathcal{RP}$'s public key. Unfortunately, FHE implies huge computational cost. With multiple result parties and a hybrid model with secret shared $\mathcal{RP}$'s private key, the same principles apply as for secret sharing based SMC and multiparty Yao model.



(a) Secret sharing based SMC and multiparty Yao

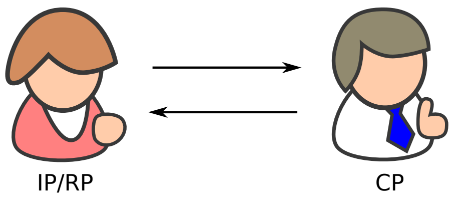


(b) FHE

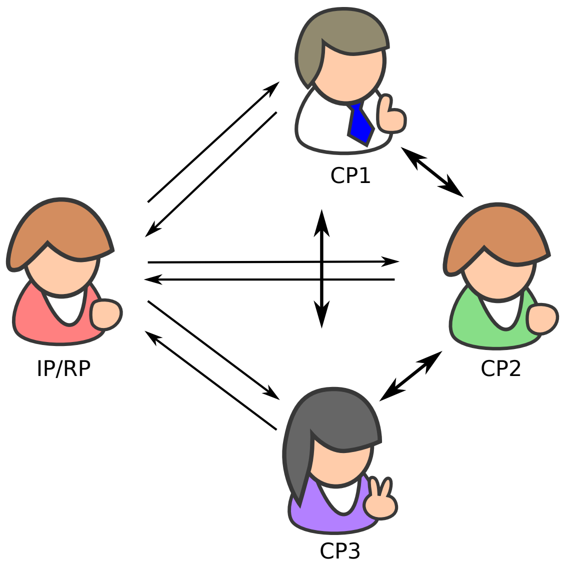
Fig. 1: Interactions between parties in Case 1, involving one or more computation parties.

3.2 Case 2

In the second case there are only two parties — a single input and result party as one entity and a single computing party. This case illustrated on Figure 2a is a typical "killer application" for the fully homomorphic encryption setting, where a party outsources operations on sensitive data to a public cloud by first encrypting the data.



(a) FHE



(b) Secret sharing based SMC

Fig. 2: Interactions between parties in Case 2.

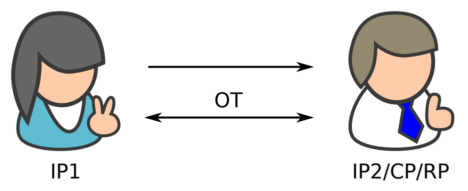
With Yao's construction, this scenario does not make much sense although there are essentially two parties. Since only one party has input, generating a garbled circuit and interactively evaluating it does not suit well with the goal of this case.

It is trivial to solve this case also with secret sharing based SMC by splitting the role of $\mathcal{CP}$ between three or more parties (Figure 2b).

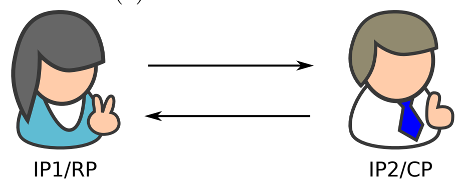
Trust and resources. With FHE only one $\mathcal{CP}$ (cloud provider) is required and there are no trust issues as the $\mathcal{CP}$ only sees encrypted data. Using FHE introduces great computational overhead, whereas using SMC introduces communication overhead between the parties sharing the role of $\mathcal{CP}$. This must be taken into account as both resources (CPU time and network bandwidth) cost on a cloud. Moreover, in the case on SMC, the input/result party has to be sure that the computation parties do not collude which in this case means finding three or more competing cloud providers.

3.3 Case 3

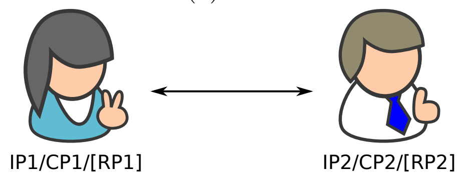
The third case (Figure 3) is motivated by Yao as a millionaires' problem [16], where two parties both have secret input (their wealth) and they want to compute a function (greater than comparison) on their inputs. As such, it is most suitable for Yao's construction (Figure 3a) where one party constructs the garbled circuit and sends it to the other one who then evaluates it with the help of oblivious transfers (OT) with the first party.



(a) Yao's construction



(b) FHE



(c) Secret sharing based SMC

Fig. 3: Interactions between parties in Case 3.

A similar construction also works with FHE (Figure 3b), where one party encrypts his input with his public key and sends it to the other party who then evaluates the necessary function on it using homomorphic operations and his own input. In case of both Yao's construction and FHE only one of the two parties gets the result. In former the one evaluating the garbled circuit gets the result. However, in case of FHE the $\mathcal{CP}$ only gets the encrypted output that it has to send back to $\mathcal{RP}$ who has the corresponding private key. In both cases it is up to $\mathcal{RP}$ if it forwards the result also to the other party.

As there exists multiparty construction that uses Yao's circuits [2], there is also a secret sharing based SMC construction that use only two parties [12] (Figure 3c). In this scenario both parties get only a share of the final result so they have to decide how to publish those, i.e. whether only one or both of the parties get to know the result.

Trust and resources. The millionaires' problem has two parties who are interested in the outcome of the computation. As such, there is no need to include any external parties when using Yao's construction, FHE or 2-party secret sharing scheme. This also eliminates the threat of colluding parties when using secret sharing based SMC.

Of course, there is a possibility to also add one or more external non-colluding parties and use the more common secret sharing scheme with three or more parties.

4 Conclusion

This work compares three secure multiparty computation techniques — Yao's construction, fully homomorphic encryption and secret sharing based SMC — by taking a signature application model for each one and try to solve the problem with the other SMC techniques. As expected, each SMC technique is most suitable for its own case but with a few exceptions, the other techniques are also applicable.

References

1. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In Harriet Ortiz, editor, *STOC*, pages 503–513. ACM, 1990.
2. Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 257–266, New York, NY, USA, 2008. ACM.
3. Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis (short paper). In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security. FC'12*, pages 57–64, 2012.
4. Peter Bogetoft, Dan Lund Christensen, Ivan Damgrd, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068, 2008. `http://eprint.iacr.org/`.

5. Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2001.
6. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.
7. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.
8. Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. Cryptology ePrint Archive, Report 2011/680, 2011. `http://eprint.iacr.org/`.
9. Yehuda Lindell and Benny Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *J. Cryptology*, 22(2):161–188, 2009.
10. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.
11. Pille Pruulmann-Vengerfeldt, Liina Kamm, Riivo Talviste, Peeter Laud, and Dan Bogdanov. Capability Model, March 2012. UaESMC Deliverable 1.1.
12. Pille Pullonen, Dan Bogdanov, and Thomas Schneider. The design and implementation of a two-party protocol suite for Sharemind 3. Technical Report T-4-17, Cybernetica, `http://research.cyber.ee/`., 2012.
13. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
14. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.
15. Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2009/616, 2009. `http://eprint.iacr.org/`.
16. Andrew C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.
17. Andrew C. Yao. How to generate and exchange secrets (extended abstract). pages 162–167, 1986.