Secure Multi-Party Computation with Security Modules

Zinaida Benenson * Felix C. Gärtner Dogan Kesdogan

RWTH Aachen University, Germany

Abstract: We consider the problem of secure multi-party computation (SMC) in a new model where individual processes contain a tamper-proof security module. Security modules can be trusted by other processes and can establish secure channels between each other. However, their availability is restricted by their host, i.e., a corrupted party can stop the computation of its own security module as well as drop any message sent by or to its security module. In this model we show that SMC is solvable if and only if a majority of processes is correct. We prove this by relating SMC to the problem of Uniform Interactive Consistency among security modules (a variant of the Byzantine Generals Problem from the area of fault-tolerance). The obtained solutions to SMC for the first time allow to compute any function securely with a complexity which is polynomial only in the number of processes (i.e., the complexity does not depend on the function which is computed). We conclude that adding secure hardware does not improve the resilience of SMC but can effectively improve the efficiency.

1 Introduction

1.1 Motivation

The problem of *secure multi-party computation* (SMC) is one of the most fundamental problems in security. The setting is as follows: a set of n parties jointly wants to compute the result of an n-ary function F. Every party provides its own (private) input to this function but the inputs should remain secret to the other parties, except for what can be derived from the result of F. The problem is easy to solve if you assume the existence of a trusted third party (TTP) which collects the inputs, computes F and distributes the result to everyone. However, the problem is very challenging if you assume that there is no TTP available and parties can misbehave arbitrarily, i.e., they can send wrong messages or fail to send messages at all. Still, the protocol must correctly and securely compute F "as if" a TTP were available.

The problem was initially proposed by Yao in 1982 [Yao82]. In the first solution in 1987, Goldreich, Micali and Wigderson [GMW87] showed that in a synchronous system with

*Zinaida Benenson was supported by Deutsche Forschungsgemeinschaft as part of the Graduiertenkolleg "Software for mobile communication systems" at RWTH Aachen University.

cryptography a majority of honest processes can simulate a centralized trusted third party. Goldwasser [Gol97] and Goldreich [Gol02] provide comprehensive overview articles on this topic.

The obvious drawback of the general solutions is efficiency: All proposed protocols for SMC suffer from high communication complexities. This follows from their approach which involves extensive use of secret sharing and agreement protocols. To give an example, the most efficient solution [HMP00] can compute any function F defined over a finite field, but it requires communicating $O(m \cdot n^2)$ field elements (m is the number of multiplication gates in F) and at least $O(n^2)$ rounds of communication (in fact, the round complexity also depends on F). In this paper we revisit the problem of SMC in a stronger albeit still practical model and achieve a significant reduction in the communication complexity.

The context of this paper remains a model with no centralized TTP, but the task of jointly simulating a TTP is alleviated by assuming that parties have access to a local *security module*. Recently, manufacturers have begun to equip hardware with such modules: these include for instance smart cards or special microprocessors. These are assumed to be *tamper proof* and run a certified piece of software. Examples include the "Embedded Security Subsystem" within the recent IBM Thinkpad or the IBM 4758 secure co-processor board [DLP+01]. A large body of computer and device manufacturers has founded the Trusted Computing Group (TCG) [Tru03] to promote this idea. Security modules contain cryptographic keys so that they can set up secure channels with each other. However, they are dependant on their hosts to be able to communicate with each other.

1.2 Contributions

We formalize the system model of untrusted hosts and security modules and investigate the resilience and efficiency of SMC in that model. In this model the security modules and their communication network form a subnetwork with a more benign fault assumption, namely that of *general omission* [PT86]. In the general omission failure model processes may simply stop executing steps or fail to send or receive messages sent to them.

We derive a novel solution to SMC in a modular way: We show that SMC is solvable if and only if the problem of *Uniform Interactive Consistency* (UIC) is solvable in the network of security modules. UIC is closely related to the problem of *Interactive Consistency* [PSL80], a classic fault-tolerance problem. From this "equivalence" we are able to derive a basic impossibility result for SMC in the new model: We then show that UIC requires a majority of correct processes and from this can conclude that SMC is impossible in the presence of a dishonest majority. This shows that, rather surprisingly, adding security modules cannot improve the resilience of SMC. However, we prove that adding security modules can considerably improve the efficiency of SMC protocols. We give a novel solution to SMC which uses security modules and requires only $O(n)$ rounds of communication and $O(n^3)$ messages. This is the first solution for which the message and round complexity do not depend on the function which is computed.

Roadmap We first present the model in Section 2. In Section 3 we define the problem of Secure Multi-Party Computation followed by the definition of Uniform Interactive Consistency in Section 4. We discuss security issues which arise if security modules should help to solve security problems in Section 5. Section 6 presents the main equivalence result and Section 7 concludes this paper with a discussion of efficiency issues and ideas for future work.

2 Model

2.1 Processes and channels

The system consists of a set of processes interconnected by a synchronous communication network with secure reliable bidirectional channels. A secure reliable channel guarantees authenticity, integrity and confidentiality of all sent messages, and that no messages are lost or duplicated. Two processes connected by a channel are said to be adjacent.

In a synchronous network communication proceeds in rounds. In each round, a party first receives inputs from the user and all messages sent to it in the previous round (if any), processes them and may finally send some messages to other parties or give outputs to the user.

2.2 Untrusted hosts and security modules

The set of processes is divided into two disjoint classes: *untrusted hosts* (or simply *hosts*) and *security modules*. We assume that there exists a fully connected communication topology between the hosts, i.e., any two hosts are adjacent. We denote by n the number of hosts in the system. Furthermore, we assume that every host process H_A is adjacent to exactly one security module process M_A (i.e., there is a bijective mapping between security modules and hosts). In this case we say that H_A is *associated with* M_A (i.e., M_A is H_A's associated security module). We call the part of the system consisting only of security modules and the communication links between them the *trusted system* (see Fig. 1).

We call the part of the system consisting only of hosts and the communication links between them the *untrusted system*. The notion of association can be extended to systems, meaning that for a given untrusted system, the *associated trusted system* is the system consisting of all security modules associated to any host in that untrusted system.

In some definitions we use the term "process" to refer to both a security module and a host. We do this deliberately to make the definitions applicable both in the trusted and the untrusted system.

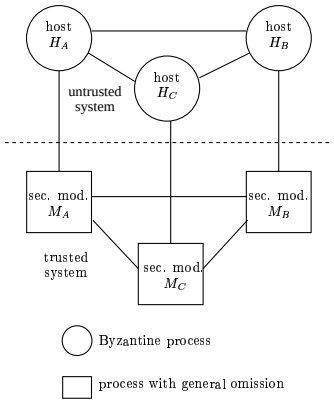

Figure 1: Hosts and security modules.

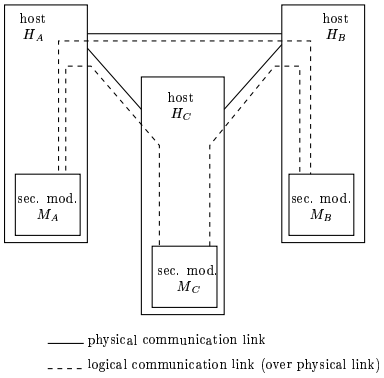

Figure 2: Internet hosts with tamper-proof hardware corresponding to Fig. 1.

2.3 Relation to Systems with Trusted Hardware.

The model sketched above can be related to the setup in practice as follows: untrusted hosts model Internet hosts and their users, whereas security modules abstract tamper proof components of user systems (like smart cards, see Fig. 2). Intuitively, security modules can be trusted by other security modules or hosts, and hosts cannot be trusted by anybody. Hosts may be malicious, i.e., they may actively try to fool a protocol by not sending any message, sending wrong messages, or even sending the right messages at the wrong time.

Security modules are supposed to be cheap devices without their own source of power. They rely on power supply from their hosts. In principle, a host may cut off the power supply to its security module whenever he chooses, thereby preventing the security module from continuing to execute steps. Instead of doing this, a host may inhibit some or even *all* communication between its associated security module and the outside world.

2.4 Trust and adversary model

The setting described above is formalized using distinct failure models for different parts of the system. We assume that nodes in the untrusted system can act arbitrarily, i.e., they follow the Byzantine failure model [LSP82]. In particular, the incorrect processes can act together according to some sophisticated strategy, and they can pool all information they possess about the protocol execution. We assume however that hosts are computationally bounded, i.e., brute force attacks on secure channels are not possible.

For the trusted system we assume the failure model of *general omission* [PT86], i.e., processes can crash or fail by not sending messages or not receiving messages.

A process is *faulty* if it does not correctly follow the prescribed protocol. In particular, a security module is faulty if it crashes or commits send or receive omissions. Otherwise a

process is said to be *correct*. In a system with n processes, we use t to denote a bound on the number of hosts which are allowed to be faulty. In general, t should be at least 1 and at most n. Sometimes we restrict our attention to the case where $t < n/2$, i.e., where a majority of hosts is guaranteed not to misbehave. We call this the *correct majority assumption*.

3 Secure Multi-Party Computation

In *secure multi-party computation* (SMC), a set of processes $p_1, \ldots, p_n$, each starting with an input value x_i, wants to compute the result of a deterministic function F, i.e., $r = F(x_1, \ldots, x_n)$. Result r should be computed reliably and securely, i.e., as if they were using a trusted third party (TTP). This means that the individual inputs remain secret to other processes (apart from what is given away by r) and that malicious processes can neither prevent the computation from taking place nor influence r in favorable ways.

We assume that F is a well-known deterministic function with input domain X and output domain Y upon which all processes have agreed upon beforehand and that all correct processes jointly begin the protocol. We say that r is an *F-result* if r was computed using F. Since faulty processes cannot be forced to submit their input value, F may be computed using a special value $\perp \notin X$ instead of the input value of a faulty process.

Instead of defining SMC using a TTP [Gol02, Mau03], we now define SMC using a set of abstract properties.

Definition 1 (secure multi-party computation) *A protocol solves* secure multi-party computation (SMC) *if it satisfies the following properties:*

- *(SMC-Validity) If a process receives an F-result, then F was computed with at least the inputs of all correct processes.*
- *(SMC-Agreement) If some process p_i receives F-result r_i and some process p_j receives F-result r_j then $r_i = r_j$.*
- *(SMC-Termination) Every correct process eventually receives an F-result.*
- *(SMC-Privacy) Faulty processes learn nothing about the input values of correct processes (apart from what is given away by the result r and the input values of all faulty processes).*

Definition 1 defines SMC in terms of safety, liveness, and information-flow (security) properties, three distinct classes of system properties in the context of security [McL96]. From a security protocols perspective, the above definition can be considered slightly stronger than the usual (cryptographic) definitions of SMC since it demands that SMC-properties hold without any restriction. In the literature it is often stated that the probability of a violation of SMC-properties can be made arbitrarily small.

The properties of SMC are best understood by comparing them to a solution based on a TTP. There, the TTP waits for the inputs of all n processes and computes the value of F on all those inputs which it received. Since all correct processes send their input value to the TTP, F is computed on at least those values, which motivates SMC-Validity. After computing F, the TTP sends the result back to all processes. Hence, all correct processes eventually receive that result (SMC-Termination). Additionally, if a process receives a result from the TTP, then it will be the same result which any other process (whether correct or faulty) will receive. This motivates SMC-Agreement. SMC-Privacy is motivated by the fact that the TTP does all the processing and channels to the TTP are confidential: no information about other processes' input values leaks from this idealized entity, apart of what the result of F gives away when it is finally received by the processes.

The information-flow property SMC-Privacy covers *all* possible (direct and indirect) sources of non-authorized information-flow about input values from one process to another. Direct information flow exists if a process receives the secret input of another process directly in a message. Indirect information flow occurs if some process can, for example, by observing another process' external behavior deduce the first bit of that process' input value.

4 Uniform Interactive Consistency

The problem of *Interactive Consistency* (IC) was introduced by Pease, Shostak and Lamport in 1980 [PSL80]. It is one of the classical problems of reliable distributed computing since solutions to this problem can be used to implement almost any type of fault-tolerant service [Sch90]. Every process starts with an initial value v_i. To solve the problem, the set of processes needs to agree on a vector D of values, one per process (Agreement property). Once vector D is output by process p, we say that p *decides* D. The i-th component of this vector should be v_i if p_i does not fail, and can be $\perp$ otherwise. IC is equivalent to the (also classic) *Byzantine Generals Problem* [LSP82].

The original version of IC was defined for the Byzantine failure model, i.e., where faulty processes can act arbitrarily. In such systems the Agreement property must be restricted to the set of correct processes. Definition 2 considers the version of IC in the context of the general omission failure model which allows to achieve a stronger agreement property called *Uniform Agreement*. Uniform Agreement demands that *all* processes should decide the same (if they decide) — it does not matter whether they are correct or faulty.

Definition 2 (uniform interactive consistency) *A protocol solves* uniform interactive consistency (UIC) *if it satisfies the following properties:*

- *(UIC-Termination) Every correct process eventually decides.*
- *(UIC-Validity) The decided vector D is such that $D[i] \in \{v_i, \perp\}$, and is v_i if p_i is not faulty.*
- *(UIC-Uniform Agreement) No two different vectors are decided.*

Parvédy and Raynal [PR03] studied the problem of UIC in the context of general omission failures. They give an algorithm that solves UIC in such systems provided a majority of processes is correct. Since their system model is the same as the one used for trusted systems in this paper, we conclude:

Theorem 1 ([PR03]) *If $t < n/2$ then UIC is solvable in the trusted system.*

Parvédy and Raynal also show that Uniform Consensus (UC), a problem closely related to UIC, can be solved in the omission failure model only if $t < n/2$. We conclude:

Theorem 2 *UIC is solvable in the trusted system only if $t < n/2$.*

Due to space limit, we omit the proof and refer the reader to [BGK04].

5 Maintaining Secrecy in Trusted Systems

The problem of UIC, which was introduced in the previous section, will be used as a building block in our solution to SMC. The idea is that a protocol for SMC will delegate certain security-critical actions to the trusted system in which the UIC protocol runs. One could think that the security features of the tamper proof devices together with the confi dentiality of secure channels between security modules offer suffi cient protection from eavesdropping on the data exchanged within the trusted system. Since critical data items (like the hosts' inputs to SMC) are never transmitted in cleartext, how can a host learn anything about these items? In this section we argue that we have to carefully analyze the security properties of the protocols which run within the trusted system. We show that, in general, there can be non-negligible information flow out of the trusted system and what can be done to prevent it.

5.1 Examples of Unauthorized Information Flow

Assume that a host H_A hands its input value v_A of SMC to its associated security module M_A and that M_A sends v_A over a secure channel to the security module M_B of host H_B. Since all communication travels through H_B (see Fig. 2), H_B can derive some information about v_A even if all information is encrypted. For example, if no special care is taken, H_B could deduce the size (number of bits) of v_A by observing the size of the ciphertext of v_A. This may be helpful to exclude certain choices of v_A and narrow down the possibilities in order to make a brute-force attack feasible.

As another example, suppose M_A only sends v_A to M_B if v_A (interpreted as a binary number) is even. Since we must assume that H_B knows the protocol which is executed on M_A and M_B, observing (or not observing) a message on the channel at the right time is enough for M_B to deduce the lowerest order bit of v_A. In this example, the control flow of the algorithm unintentionally leaks information about secrets.

In the Privacy property of SMC we are required to prevent any type of unauthorized information flow about the input values of processes.

5.2 Security Properties of Protocols in the Trusted System

A protocol running in the trusted system needs to satisfy two properties to be of use as a building block in SMC:

- (Content Secrecy) Hosts cannot learn any useful information about other hosts' inputs from observing the *messages* in transit.
- (Control Flow Secrecy) Hosts cannot learn any useful information about other host's inputs from observing the *message pattern*.

Both types of secrecy can be implemented in two ways. Either we give an algorithm that transforms any algorithm in the trusted system into an algorithm that satisfies the two secrecy properties, or we can adapt a known algorithm to satisfy these properties.

We now show that the UIC protocol of Parvédy and Raynal [PR03] can be modified to fulfill the two above secrecy properties. At the end of this section we will briefly sketch how to transform any UIC algorithm to one that satisfies the two secrecy properties.

5.3 Making Parvédy and Raynal's UIC Protocol Secure

We first analyze the UIC algorithm given by Parvédy and Raynal [PR03] and show that this algorithm already satisfies the required properties, if following changes are made to it:

- All sent messages must be end-to-end encrypted such that the consecutive encryption of the same input (i.e., plain text) results in different outputs (cipher text).
- All sent messages must have the same length.

The first change is easy fulfilled by our formal model (see Section 2), since we consider only end-to-end encrypted messages between the processes and any standard probabilistic encryption scheme that prevents replay attacks satisfies this property [MOV97]. The second change can be simply achieved by giving a constant maximum length of messages, so that all messages have to be padded (before encryption) to the given maximum length.

We now briefly explain the protocol by Parvédy and Raynal: It goes through $t + 1$ synchronous rounds:

- In the first round every process p_i sends his initial value v_i to all other processes and initializes the n-ary vector D_i to hold the result of the protocol, i.e. $D_i[i] := v_i$, $D_i[j] = \perp$ for all $i \neq j$. Upon receiving values from other processes, p_i includes their values into D_i.

- In all subsequent rounds, if p_i receives some value v_j for the first time, it includes this value into $D_i[]$ and relays his knowledge to all processes which are not *suspected*. In case p_i does not receive any new values in some round, it just sends an "empty" message meaning that he has not failed yet.

In each round each process keeps a record of those processes which he *suspects* to have failed. These are all processes from which p_i did not receive any messages in some round. If p_i suspects more than t processes, it realizes that he himself must be faulty and quits the protocol without deciding on any vector. Otherwise, p_i outputs D_i at the end of round $t + 1$.

In [BGK04] we show that the augmented protocol of Parvédy and Raynal satisfies Content Secrecy and Control Flow Secrecy.

5.4 Making an Arbitrary UIC Protocol Secure

To provide the two secrecy properties in general, it is sufficient to use a communication protocol between the processes that ensures *unobservability*. Unobservability refers to the situation when an adversary cannot distinguish meaningful protocol actions from "random noise" [PK01]. In particular, unobservability assumes that an adversary knows the protocol which is running in the underlying network. It demands that despite this knowledge and despite observing the messages and the message pattern on the network it is impossible for the adversary to figure out in what state the protocol is. The term "state" refers to all protocol variables including the program counter, e.g., the mere fact whether the protocol has started or has terminated must remain secret.

Obviously, if unobservability is fulfilled during the application of UIC then Content Secrecy and Control Flow Secrecy are fulfilled. There are known techniques in the area of unobservable communication that guarantee perfect unobservability [PK01]. It is out of scope of this paper to present these techniques or to give a new technique providing unobservability.

6 Solving SMC with Security Modules

In this section we show the main result of this paper.

Theorem 3 *SMC is solvable for any deterministic F in the untrusted system if and only if UIC is solvable in the associated trusted system.*

The above theorem shows that SMC and UIC are "equivalent" in their respective worlds. Due to space limit, we only give intuition for the proof and refer the reader to [BGK04].

Fig. 3 shows the protocol which is executed within the security module and solves SMC using a secure solution to UIC. The hosts first give their inputs for SMC to their security

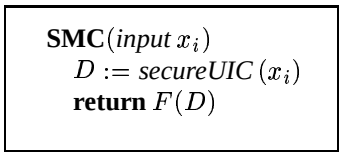

Figure 3: Implementing SMC using UIC on security modules. Code for the security module of host H_i. The term "secure UIC" refers to a UIC protocol that satisfies Content Secrecy and Control Flow Secrecy.

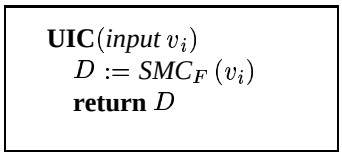

Figure 4: Implementing UIC on security modules using SMC for the function $F(v_1, \ldots, v_n) = (v_1, \ldots, v_n)$ in the untrusted system. Code for the security module of host H_i.

modules. Security modules run "secure UIC" (i.e., UIC which satisfies Message Secrecy and Control Flow Secrecy, see Section 5) on these inputs, compute F on the decided vector and give the result to their hosts.

We now show that if SMC is solvable in the untrusted system, then UIC is solvable in the trusted system. First note that if SMC is solvable in the untrusted system, then SMC is trivially also solvable in the trusted system. This is because the assumptions available to the protocol are much stronger (general omission failures instead of Byzantine).

To solve UIC, we let the processes compute the function $F(v_1, \ldots, v_n) = (v_1, \ldots, v_n)$ (see Fig. 4). We now show that the properties of UIC follow from the properties of SMC.

Theorem 3 allows us to derive a lower bound on the resilience of SMC in the given system model.

Corollary 1 *There is no solution to SMC in the untrusted system if* $t \geq n/2$.

7 Analysis and Conclusions

Theorem 3 and Corollary 1 show that adding security modules cannot improve the resilience of SMC compared to the standard model without trusted hardware [GMW87]. For the model of perfect security (i.e., systems without cryptography) our secure hardware has a potential to improve the resilience from a two-thirds majority [BOGW88, CCD88] to a simple majority. However, this would rely on the assumption that security modules can withstand *any* side-channel attack, an assumption which can hardly be made in practice.

Since F is computed locally, the main efficiency metric for our solution is message and round complexity of the underlying UIC protocol. We estimate the worst case message complexity of the protocol of Parvédy and Raynal [PR03] (see Section 5). Each process forwards its input value and input values of other processes at most once which means that at most n^2 messages containing input values are sent during the protocol. Additionally, each process sends "I'm still correct"-messages to all unsuspected processes in each of $t + 1$ rounds if it has not received any new input values in the previous round. Thus $O(n)$ processes send $O(n)$ "I'm still correct"-messages during the $O(n)$ rounds. Therefore, the worst case message complexity of the protocol is $O(n^3)$. This worst case complexity also

holds for the private and fair modification introduced in Section 5.

Thus, our solution requires $O(n)$ rounds and $O(n^3)$ messages, whereas the most efficient solution to SMC without secure hardware requires at least $O(n^2)$ rounds and $O(mn^3)$ messages where m is the number of multiplications in F [HMP00].

This analysis shows the main benefit of using secure hardware: In contrast to previous solutions to SMC the message and round complexity does not depend on F anymore.

Recently, Avoine *et al.* [AGGV04] studied the problem of Fair Exchange in the same system model as used in this paper. Similar to the technique in this paper, they showed that Fair Exchange is equivalent to a problem called *Biased Consensus*. Using this equivalence they showed that Fair Exchange is possible if and only if $t < n/2$. In a sense, Theorem 3 is a generalization of their findings.

Avoine *et al.* also present a solution to Fair Exchange for a dishonest majority of processes in which the probability of unfairness can be made arbitrarily small. It would be interesting to derive a similar result for SMC. Also interesting future work is to extend the work in this paper to asynchronous systems in which SMC has been already studied in the general model [BOCG93].

Acknowledgments We thank Neeraj Mittal for helpful comments regarding the security properties of SMC.

References

[AGGV04] G. Avoine, F. C. Gärtner, R. Guerraoui, and M. Vukolic. Gracefully Degrading Fair Exchange with Security Modules. Technical Report 200426, Swiss Federal Institute of Technology (EPFL), School of Computer and Communication Sciences, Lausanne, Switzerland, March 2004.

[BGK04] Z. Benenson, F. C. Gärtner, and D. Kesdogan. Secure Multi-Party Computation with Security Modules. Technical Report 2004-10, RWTH Aachen University of Technology, 2004.

[BOCG93] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computations. In *Proceedings of the 25th Annual Symposium on Theory of Computing (STOC)*, pages 52–61, San Diego, CA, USA, May 1993. ACM Press.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual Symposium on Theory of Computing (STOC)*, pages 1–10, Chicago, IL USA, May 1988. ACM Press.

[CCD88] D. Chaum, C. Crepeau, and I. Damgård. Multiparty unconditionally secure protocols. In Richard Cole, editor, *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 11–19, Chicago, IL, May 1988. ACM Press.

[DLP+01] J. G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, and S. Weingart. Building the IBM 4758 secure coprocessor. *IEEE Computer*, 34(10):57–66, October 2001.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proceedings of the 19th ACM Symposium on the Theory of Computing (STOC)*, pages 218–229, 1987.

[Gol97] S. Goldwasser. Multi party computations: Past and present. In *Proceedings of the sixteenth annual ACM Symposium on Principles of Distributed Computing*, pages 1–6. ACM Press, 1997.

[Gol02] O. Goldreich. Secure Multi-Party Computation. Internet: `http://www.wisdom.weizmann.ac.il/~oded/pp.html`, 2002.

[HMP00] M. Hirt, U. Maurer, and B. Przydatek. Efficient secure multi-party computation. In *Proceedings of Asiacrypt*, 2000.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[Mau03] U. Maurer. Secure Multi-Party Computation Made Simple. In G. Persiano, editor, *Third Conference on Security in Communication Networks (SCN'02)*, volume 2576 of *Lecture Notes in Computer Science*, pages 14–28. Springer-Verlag, 2003.

[McL96] J. McLean. A General Theory of Composition for a Class of "Possibilistic" Properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, January 1996. Special Section—Best Papers of the IEEE Symposium on Security and Privacy 1994.

[MOV97] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1997.

[PK01] A. Pfitzmann and M. Köhntopp. Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology. In H. Federrath, editor, *Anonymity 2000*, number 2009 in Lecture Notes in Computer Science, pages 1–9, 2001.

[PR03] P. R. Parv´edy and M. Raynal. Uniform Agreement Despite Process Omission Failures. In *17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*. IEEE Computer Society, April 2003. Appears also as IRISA Technical Report Number PI-1490, November 2002.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching Agreements in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[PT86] K. J. Perry and S. Toueg. Distributed Agreement in the Presence of Processor and Communication Faults. *IEEE Transactions on Software Engineering*, 12(3):477–482, March 1986.

[Sch90] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[Tru03] Trusted Computing Group. Trusted Computing Group Homepage. Internet: `https://www.trustedcomputinggroup.org/`, 2003.

[Yao82] A. C. Yao. Protocols for Secure Computations (Extended Abstract). In *23th Annual Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164, Los Alamitos, Ca., USA, November 1982. IEEE Computer Society Press.