

Fostering the Uptake of Secure Multiparty Computation in E-Commerce

Octavian Catrina
International University in Germany
Bruchsal, Germany
octavian.catrina(at)i-u.de

Florian Kerschbaum
SAP Research
Karlsruhe, Germany
florian.kerschbaum(at)sap.com

Abstract

Secure Multiparty Computation (SMC) protocols enable a group of mutually distrustful parties to perform a joint computation with private inputs. Novel e-commerce applications have emerged that could benefit from strong privacy protection, e.g., benchmarking, auctions, and collaborative supply chain management and planning.

However, the uptake of SMC in these applications is still rare. We argue that this is due to poor performance, functionality, and scalability, as well as architectures that do not meet the needs of e-commerce applications. This paper explores SMC approaches and research directions, aiming at providing better support for e-commerce applications.

1 Introduction

Secure Multiparty Computation (SMC) protocols enable a group of parties to perform a joint computation with private inputs. As a basic example, consider a group of n parties, P_1, P_2, \dots, P_n , who want to compute a function $f(x_1, x_2, \dots, x_n) = (y_1, y_2, \dots, y_n)$. Each party P_i submits the private input x_i and wants to obtain the output y_i .

The goal is to ensure that every party P_i learns its specified output y_i (correctness) and nothing else is revealed by carrying out the computation (privacy). Preserving privacy means that the information a party can obtain by participating in the joint computation is limited to its output and what can be inferred from its own input and output.

SMC has many potential applications, e.g., secure auctions [23], privacy-preserving forecasting and benchmarking [4], secure supply chain management [3], e-voting [11], privacy-preserving data mining [19].

Research on SMC started in the 1980s and has gained substantial momentum during the last decade. However, development of practical SMC protocols is still a challenge.

Research has mainly focused on finding cryptographic methods for building SMC protocols, and formally proving the protocols' properties in a suitable security model. A

lot of effort was dedicated to protocols that withstand very powerful adversaries (active and adaptive) and remain secure in any execution environment (UC security framework) [7, 9, 10, 14]. However, for many potential SMC applications, these theoretical protocols are not practical, due to poor performance, functionality, and scalability.

We explore in this paper a broader range of approaches, trying to identify methods for obtaining practical SMC protocols. These protocols should be suitable for implementation using common software and hardware technologies, and should meet the functional and performance requirements of a wider range of applications.

Purely cryptographic solutions have not met these objectives. Therefore, we take into account solutions that combine cryptographic and non-cryptographic methods, and architectures where service providers help the parties to achieve the computation.

2 E-Commerce Applications

Many e-commerce applications benefit from privacy. As examples, we describe several applications that are fairly thoroughly developed and have been widely adopted in non-privacy-preserving form: benchmarking, auctions, and supply chain management. The goal is to identify common properties of these applications and motivate possible improvements for SMC protocols.

Privacy-Preserving Benchmarking. Benchmarking is a comparison of one's own performance measurements to the statistics of the competition. A benchmarking e-commerce service offers to customers access to these statistics.

Privacy is important for benchmarking because certain performance measurements may be sensitive and should not be revealed to competitors or the general public (e.g., financial data), and because poor performance may be embarrassing and diminish the brand value. Privacy-preserving benchmarking protects the confidentiality of key performance indicators (KPI), such that they remain private to

their respective originators. This can be realized by SMC protocols that compute the statistics.

Privacy-Preserving Auctions. E-commerce auctions are now a multi-billion business.

For certain types of auctions privacy facilitates finding the optimal price. This is particularly the case for many public tenders. An auction is being conducted by an auctioneer who places the offer on a public website and receives the bids. SMC protocols can compute the winner of an auction and hide the bids of the other bidders.

Privacy-Preserving Supply Chain Management. It is well-known that collaborative supply chain planning, also called supply chain master planning (SCMP), can reduce costs compared to local decision making.

SCMP is an optimization problem of the production of and the delivery between multiple companies. It requires as input the costs for individual steps (e.g., production costs).

Revelation of such data significantly impacts a company's negotiation position. Therefore, if privacy is not ensured, companies are reluctant to engage in collaborative supply chain management. The supply chain can be organized by a powerful company or a logistics provider. This party can run a privacy-preserving SCMP service.

Summary. All these applications benefit from privacy and can use SMC protocols to achieve it. Furthermore, each application is characterized by a central service provider that offers a privacy-preserving service to its customers.

This service provider plays a special role in the SMC protocol. It usually operates at a higher level of trust, although not as a fully trusted third party. It is therefore natural to investigate how this special server introduced by e-commerce applications affect the SMC protocols.

3 SMC Models and Methods

3.1 Trusted Party vs. Distributed SMC

The purpose of SMC is to enable a group of mutually distrustful parties to jointly perform a computation. Each party is supposed to supply an input and obtain an output. However, they cannot disclose their individual inputs. Moreover, they have reasons to doubt that everybody will honestly cooperate to achieve the computation.

Trusted party. Suppose that the parties can find an external entity whom they can trust to protect the privacy of their inputs and deliver correct outputs. For example, the trusted entity could be a specialized service provider, properly motivated and having suitable technology.

If a trusted party is available, the computation can be done using a very simple and efficient protocol: the trusted party privately collects the secret inputs, performs the computation, and (privately) delivers the output to each party. Then, it deletes the secret data used in the computation. If followed strictly, the protocol obviously fulfills the privacy and correctness requirements.

The caveat of this simple solution is that in real-life a provider offering sufficient guarantees might be too expensive or simply not available. We have to look for solutions that do not require placing so much trust in an external party.

Cryptographic protocols instead of trusted party. A radically different approach is to enable the parties to perform the computation themselves, without the help of a trusted entity. Instead, they rely on cryptographic protocols.

Most of the research on SMC has explored this path. However, the results after two decades are mainly theoretical. Most of the SMC cryptographic protocols proposed so far have never been used in practice.

Several approaches have recently received more attention, and have actually been implemented (mainly for experimental purposes): SMC using linear secret-sharing schemes [9, 12], homomorphic public-key encryption [10, 11], and Yao's garbled circuit [20, 22].

Trusted party model as reference solution. From the security viewpoint, the trusted party model helps to formulate a rigorous specification of the task to be achieved: a simple, ideal protocol that meets the security requirements in an obvious way, assuming an ideal, incorruptible trusted party (an ideal functionality). We can then define a strong security notion, requiring that real protocols must be equivalent to the ideal protocol [7].

On the other hand, it is also an optimal solution from the point of view of implementation and performance: it enables the simplest implementation and provides the best performance we can hope to achieve.

Then, the question is: How can we build SMC protocols that do not rely on an *idealized* trusted party, but preserve some of the attractive features of the model?

3.2 Adversarial Behavior

In real-life multiparty computations, a party consists of a human and a computer process. We must take into account both components. The human chooses the program being run and its inputs, and controls to some extent its execution. A human has partial knowledge about the behavior of the program he is running. Both of them can perform malicious actions, in cooperation or independently. An external adversary can also interfere with a multiparty computation, by

controlling network communications and by running malicious agents on the parties' computers.

Honest parties, malicious parties. The traditional security models for cryptographic protocols divide the participants into semi-honest parties ("honest but curious") and malicious parties. Different kinds of adversarial behavior are usually modeled as a single adversary, who controls the communications and corrupts a subset of parties.

Semi-honest parties follow the protocol but try to learn additional information from data collected while running the protocol (passive adversary model). Malicious parties can deviate arbitrarily from the protocol in their attempts to break it (active adversary model). In both cases, subsets of corrupted parties can collude, pulling together their individual resources and coordinating their actions, to achieve unauthorized objectives.

The theory of secure multiparty computation suggests the design of a protocol in two stages [8, 14]. We start with a simpler protocol that meets the security requirements if all the parties are semi-honest. Then, we enhance this protocol to make it secure even when some parties are malicious. Essentially, this is done by adding cryptographic mechanisms that enforce semi-honest behavior.

For non-trivial tasks, the protocols for semi-honest parties are often very complex. Extensions needed to handle malicious parties can substantially increase the complexity, making a protocol impractical. This suggests we should look for other means to control the parties' behavior.

Classical feasibility theorems show that any functionality can be securely computed in the presence of an adaptive, active adversary corrupting less than $n/2$ parties in the cryptographic model, and less than $n/3$ parties in the information-theoretic model.

The practical value of such guarantees is difficult to assess. For example, if all the parties run the protocol on the same type of platform, they are all exposed to new attacks that exploit vulnerabilities of their platform. Massive Internet-based attacks carried out during the last decade can affect the majority of the parties, rather than a minority.

Rational parties. Rational secure computation proposes a different model, using concepts borrowed from game theory [1, 16]. A secure computation is modeled as a game with rational players who want to maximize their payoffs. The game must be designed such that its equilibrium corresponds to the specified outcome of the computation (players cannot improve their payoff by choosing other strategies). Dishonest behaviors (input substitution, aborting the game, denying the outputs to some players) should be dominated strategies, that rational players would not choose.

Rational security is an important theoretical development, with interesting applications (e.g., mechanism design

[16]). However, practical feasibility and efficiency have not been properly addressed so far. Initial research focused on new techniques, able to handle rational parties instead of semi-honest parties. Proposed protocols are more complex, even for simple utility functions (e.g., selfish parties [1]).

A more general model includes both rational parties, expected to follow the protocol given suitable incentives, as well as malicious parties, with unknown utility functions and hence arbitrary behavior [21]. Protocols designed in this security model are inherently more complex than those designed in the classical model.

Rational secure computation focuses on the rational behavior of humans and neglects the computer process. We can try to control the behavior of a computer process, by combining cryptography and a physically protected execution environment. This can also substantially restrict the human's options for malicious actions.

4 Towards Practical Protocols

In order to reduce the gap between theoretical and practical SMC protocols we have to consider the following issues.

Functionality and performance. We need practical protocols, able to meet the functional and performance requirements of the applications. Many of the protocols proposed in the literature remain, essentially, mathematical solutions to SMC problems: valuable theoretical contributions, but insufficient for practical applications.

For example, there are various generic solutions based on secure evaluation of boolean circuits, in different cryptographic frameworks. A circuit is evaluated gate by gate by composition of sub-protocols for individual gates (e.g., XOR and AND). These protocols are very useful for studying fundamental theoretical aspects of secure computation [8, 14]. However, they are much less relevant from the practical viewpoint: they can only handle (with satisfactory performance) very simple computations and few parties.

We need sub-protocols able to directly compute more complex functions, with minimum interaction, taking advantage of local processing resources (e.g., perform arithmetic operations without having to evaluate gate by gate boolean combinatorial circuits).

Secure protocol composition. A critical challenge is to design sub-protocols that can be securely composed, and used as building blocks for more complex protocols [7].

Composition is straightforward if the sub-protocols use the same cryptographic framework, e.g., secret-sharing or homomorphic encryption. However, it seems difficult to obtain a full range of sub-protocols for mathematical computations using the same framework.

On the other hand, if we try to use different frameworks for different operations, composition can become hard or impossible, without revealing intermediate results (we need to convert the outputs of a protocol in one framework to inputs of a protocol in another framework).

Adversarial model. Cryptographic SMC protocols designed to withstand worst case adversarial capabilities (e.g., adaptive active adversary) are too complex for many applications. Performance is modest even for the passive adversary model, simple tasks, and few parties [12, 22].

The adversarial model should be more realistic and pragmatic. Also, we can consider combining cryptography with other methods. For example, running a protocol task in a tamper-resistant security module can help to ensure honest behavior (instead of using zero-knowledge proofs), or to avoid inefficient cryptographic protocols (e.g., for generating threshold decryption keys).

Communications and setup. Theoretical specifications of SMC protocols assume communication models and setup that are not generally available in real-life networks.

They usually require synchronous group communications with complex interaction patterns. In a protocol round, each party sends messages to all the others, addressed 1-to-1 or 1-to- n (for $n + 1$ parties). A round may consist of $n + 1$ interactions 1-to- n (each party to all) or n -to-1 (all parties to each one), or combinations of other patterns. Security requirements include guaranteed, timely, authentic, and confidential delivery. There are specific requirements for particular interaction types, e.g., simultaneous broadcast.

These abstractions are necessary in formal specifications and proofs, but the practical implications must be correctly assessed. Generally available network services only offer 1-to-1 reliable channels, optionally authenticated and confidential. All the other features must be provided by sub-protocols included in the SMC implementation.

5 SMC Using Service Providers

5.1 Server-based vs. Fully Distributed SMC

There are strong reasons to consider solutions where the parties are helped by service providers, rather than doing everything themselves. This is a natural setting for e-commerce applications. The providers can support SMC with communications and cryptographic services, or can perform secure computations on behalf of the parties.

Protocol overhead and resources. Fully distributed SMC solutions do not scale up for a large number of parties and non-trivial tasks. They require too much interaction and

processing for each party. Moreover, in SMC applications the parties interact over the Internet. This implies longer delays, lower bandwidth, and basic communications services. Server-based solutions can simplify the protocols, and improve their performance and scalability.

SMC needs substantial cryptographic processing, which increases with the number of parties. Some SMC solutions use expensive public-key cryptography (e.g., homomorphic encryption, oblivious transfer).

Providers can set up servers with much more processing and communications resources than those available to parties (including powerful cryptographic co-processors). For some applications the parties might need to use mobile devices with limited resources.

Network access. Direct (peer-to-peer) interactions between parties are hindered by NAT gateways and firewalls. They require special arrangements to enable incoming connections, like running the SMC protocols on a bastion host in the DMZ or using VPN technologies. A server-based solution avoids these hurdles. In this case, a party only needs to establish an outgoing connection to the server.

Common technologies. A client-server solution is easier to implement using common technologies, and deploy. Since modern business applications are often implemented as Web Services [13] in a Service-Oriented Architecture [15], this point is very important. A party runs the client side of the protocol, which can be considerably simplified, such that single threaded solutions are usually feasible.

5.2 Split Responsibilities

One way to address these issues is to split the responsibilities of the ideal trusted party between a service provider and a protocol run by the parties. We can distinguish the following main tasks: coordination and communications, correctness of the computation, and protection of data privacy.

Coordination and communications. An obvious partition of the tasks assigns the communications services to the provider and leaves correctness and privacy to a cryptographic protocol run by the parties. This partition follows the usual approach to SMC design, which starts by assuming a certain communication model. In the UC framework [7], it corresponds to designing a hybrid protocol, with an ideal functionality that provides synchronous group communications to an SMC protocol.

These communications services should be generic. The server does not participate in computing the output, just facilitates the interactions between parties. The parties are responsible for authenticating and encrypting their messages.

Correctness. We can consider extending the provider’s responsibilities regarding the correctness of the computation, and leave only privacy to the cryptographic protocol run by the parties. For example, we could use Yao’s garbled circuit technique as follows: the parties submit to the server their encrypted inputs and the garbled circuit, the server evaluates the circuit and delivers to the parties the encrypted outputs, then the parties decrypt the outputs.

Distributed trust. The parties might not trust a single provider to suitably protect the privacy of their data. However, they can distribute the trust among several providers. For example, when using SMC based on secret-sharing, the parties can delegate the responsibility for the computation to a group of servers. Each server receives input shares from every party, the servers jointly perform the computation and deliver output shares to the parties, then each party constructs its output from the received shares.

5.3 Ideal Architecture

We introduce in the following an ideal architecture for e-commerce SMC applications. We focus on the typical scenario where a service provider wants to offer privacy-preserving services to customers.

A server-based solution offers clear benefits, and is an obvious choice for our architecture. Ideally, only a single server would be involved in a computation.

Given the inputs, the server would compute the output without further interaction with the clients, and would carry most of the computation effort. Privacy of the inputs delivered to the server is ensured by the SMC protocol. The server acts as another party without input to the protocol, not as a trusted party responsible for preserving privacy.

In a simple overview: (1) Clients drop their input at the server. (2) The server computes the result. (3) Clients retrieve the result. This matches most closely the ideal model.

Ideally, the structure of the protocol would not be fixed: a client could come any time and the three steps would be executed. In a more strict form, the three steps correspond to rounds in the protocol. Although farther from the ideal model, this seems still acceptable for many applications.

The SMC service might need to record the output of the computation at the server side for future use. In this case, the output would ideally be obtained at the server side (rather than at the client side).

In the distributed trust approach, the parties use a group of mutually distrustful providers, and the protocol ensures that security is preserved even if a subset of them are corrupted and collude. The underlying assumption is that corrupting several providers is harder than a single provider.

In practice, setting up a consortium of providers is more expensive and complicated. While the parties are better

assured by distributing trust among several providers, the providers might not be willing to accept joint responsibility. If a privacy violation occurs, how are the providers going to “distribute” the liabilities?

5.4 Anonymity

Anonymity can be defined as the absence of a static identifier that refers to a party. Anonymity of parties is a rarely considered question in SMC. It seems counter-intuitive at first glance, since most SMC protocols require secure and authenticated channels.

Mix networks and onion routing can anonymize the sender of a message in IP networks and e-mail, but they cannot anonymize the recipient. Therefore, in a full mesh where each party needs to send a message to each other party, anonymity is not possible.

The server model enables anonymity among clients, i.e., a client does not know the identity of the other clients (all know the identity of the server). This could be achieved in many protocols, but has not been thoroughly researched.

Applications and providers might need to hide the identities of the clients to each other, so anonymity among clients should be a feature of the ideal SMC implementation.

6 Communications Services

We consider in this section SMC protocols that use a service provider for coordination and communications services. In the UC framework, this corresponds to a hybrid protocol that uses an ideal functionality providing synchronous group communications.

The parties trust the provider for guaranteed and timely message delivery, and possibly for other synchronization requirements (e.g., simultaneous multicast). These services can substantially simplify an SMC protocol and facilitate its implementation, deployment, and operation.

On the other hand, the parties take care of the end-to-end cryptographic protection of their messages (authentication, confidentiality). The provider cannot modify the protocol messages without being detected, does not have access to private data, and is not aware of what is being computed.

Misbehavior of the provider can affect the fair termination of the computation and the correctness of the output. However, it can often be detected, and a dishonest provider risks losing its reputation, and hence its business.

Physical topology. Full mesh interconnection topology is replaced by star topology. Each party establishes a single connection to a server, regardless of group membership, and all messages are exchanged on this connection. The star topology simplifies session management and protocol implementation, and improves scalability and performance.

Logical topology. Since the computation is done by the parties, the logical topology is still a mesh of secure point-to-point channels and/or broadcast channels. Processing associated to secure channels has an important contribution to protocol overhead, and is not reduced by these services.

Reliable broadcast. Typical SMC protocols need reliable broadcast channels. Native multicast is not generally available in the Internet. In a mesh topology, parties have to implement secure broadcast channels as “multi-unicast” on point-to-point secure channels.

Dishonest parties might not send the same message to all the other parties. Enforcing reliable broadcast in a group with dishonest parties requires complex cryptographic protocols. A trusted server can offer more efficient solutions, e.g., in some SMC protocols a sender can submit a single copy of a message to the server, which replicates it to all the other parties. This implies reliable broadcast, because dishonest parties do not have the opportunity to cheat.

Synchronized communications. Many SMC protocols require synchronized unicast and/or multicast communications. In a protocol round r , each party receives messages from the other parties (sent at the end of round $r - 1$), does a local computation, and then sends messages to the other parties (to be processed in round $r + 1$).

A basic synchronization requirement is that all parties receive (in the beginning of) a round r all messages sent to them in the previous round. A communications hub can simplify this synchronization.

A stronger requirement is to ensure that a party does not receive any message in round r before sending its own messages for the previous round. For example, in SMC using secret-sharing, fair output reconstruction cannot be achieved without synchronized (simultaneous) broadcast. A related issue is to ensure input independence. The problem can be solved by a trusted server that collects messages from all the parties before delivering them. A mesh approach would require complex cryptographic protocols.

7 Privacy and Correctness Using Providers

We explore in the following server-based variants of the main SMC approaches: linear secret-sharing, homomorphic public-key encryption, Yao’s garbled circuit. We consider a group of $n > 2$ parties, P_1, \dots, P_n , who want to compute a function $f(x_1, \dots, x_n) = (y_1, \dots, y_n)$.

7.1 SMC Using Secret-Sharing

In SMC based on secret-sharing [9], each party shares its private inputs among all the parties that participate in

the computation. Then, the parties use SMC protocols to perform a joint computation that takes on shared inputs and produces shared outputs (based on the linearity of the secret-sharing scheme).

The parties create a distributed state of the computation, with the secret values shared among them (inputs and intermediate outputs), and operate on this state until they obtain shared final outputs. The secret-sharing scheme and the SMC protocols ensure that the secret values can be reconstructed only by authorized sets of parties.

We can easily obtain a server-based solution with several mutually distrustful providers (but not with a single server). Essentially, the parties outsource the computation to a group of servers, instead of running the protocol themselves.

Suppose that the parties run an SMC protocol using secret-sharing with threshold t , i.e., secret reconstruction requires the shares of $t + 1$ parties. The computation can be done using $m > 2t$ servers, as follows. Every party shares its input among the m servers, the servers run the protocol and obtain shares of the outputs, then every party receives its output shares and constructs the output. Privacy is preserved if no coalition of corrupted servers is larger than t .

7.2 SMC Using Homomorphic Encryption

In SMC based on homomorphic public-key encryption [10, 11], each party distributes encryptions of its private inputs to the other parties. The parties use SMC protocols that perform a joint computation taking on encrypted inputs and producing encrypted outputs (using the homomorphic properties of the encryption function).

The parties create a distributed state of the computation with the secret values encrypted (inputs and intermediate outputs), and operate on this state to obtain encryptions of the final outputs. For $n > 2$ parties, a threshold decryption scheme ensures that secret values can only be decrypted by authorized sets of parties.

We can obtain a server-based solution with multiple servers as in the case of SMC based on secret-sharing.

The servers run a key generation protocol to obtain a public encryption key and shares of the matching decryption key. The parties receive the public key, encrypt their inputs, and send them to the servers. The servers jointly perform the computation, operating on encrypted data. When the computation is finished, the servers deliver to each party shares of the decrypted output (for this purpose, each server uses its share of the decryption key). Finally, each party combines the received shares and obtains its output.

Efficient protocols using homomorphic encryption and a single server are an open problem. In this case, only the parties can decrypt, so they must generate the key. Few operations with encrypted data can be locally done by the server. The others need expensive interaction with the parties.

7.3 SMC Using Yao’s Protocol

Yao’s protocol [20, 22] allows a secure evaluation of any function represented as a boolean circuit. The core component is a technique that transforms a circuit C into a garbled circuit EC by “encrypting” the gates’ wires and truth tables. We consider variants for $n > 2$ parties. Given EC and its encrypted inputs $E_{i_1}(x_1), \dots, E_{i_n}(x_n)$, any party can compute the encrypted outputs $E_{o_1}(y_1), \dots, E_{o_n}(y_n)$, without knowing the secret keys used to obtain EC . During evaluation, a party does not learn anything about the values for which the function is evaluated.

This technique can be used in various ways in a server-based setting. We start with the most efficient. Each party locally computes the garbled circuit using a common secret seed. Then, it sends the circuit and its own encrypted inputs to a server (without the keys). After checking the consistency of the received data, the server evaluates the garbled circuit and obtains the encrypted outputs. Finally, each party receives its encrypted output and locally decrypts it.

The server does not know the circuit’s keys, so it cannot decrypt the value of any wire. Each party knows the keys, but does not learn the encrypted data of other parties. However, privacy is compromised if the server colludes with any party and discloses the encrypted data.

We can avoid this vulnerability by enabling the parties to jointly generate a garbled circuit without learning the keys. A variant suitable for this purpose was proposed in [5]. The protocol allows each party to obtain a share of the garbled circuit and of its keys. Then, they exchange shares such that each party creates a local copy of the encrypted inputs and the garbled circuit (without the keys), and the means for decrypting its own output. Thus, each party can locally evaluate the circuit and obtain its output.

This is the most secure solution, but it is not efficient. The joint generation of the garbled circuit requires the exchange of large volumes of data. Performance is slightly improved by using a server to collect the shares from the parties, build the circuit, and evaluate it.

Finally, we can use a server G that generates the garbled circuit and obviously transfers to each party its encrypted inputs. Then, G sends the garbled circuit to the parties, they exchange the encrypted inputs, and each party locally evaluates the circuit to obtain its output. However, security is no better than in the first case, and the communication overhead is much higher due to the oblivious transfers.

A limited improvement can be obtained by adding a server S that gets the garbled circuit from G and the encrypted inputs from the parties, evaluates the circuit, and delivers to each party its output [23]. This variant distributes the trust among two servers. The parties do not know the keys of the garbled circuit. However, if G and S collude, they can learn the private data of all the parties.

7.4 SMC Using Tamper-Resistant Devices

The solutions discussed so far meet some of the objectives of the ideal architecture, but require a group of mutually distrustful providers running peer-to-peer SMC protocols. Setting up and operating such a consortium is rather difficult in practice. Also, the cryptographic methods used in these protocols offer limited, rather poor support for real-life applications, even for the passive adversary model.

We can extend the methods and tools used in SMC by including tamper-resistant cryptographic co-processors. These devices combine cryptographic methods and physical protection to provide integrity of executable code, data secrecy, and data integrity. Several solutions using tamper-resistant devices have recently been proposed [6, 18], but this line of research has not received sufficient attention.

In a server-based solution, we propose the use of such devices to realize ideal functionalities for secure computation tasks, as trusted implementations of the tasks, running in a closed, secure execution environment.

A powerful cryptographic co-processor (e.g., [2]) has sufficient resources to run internally a secure computation, for simple applications and a small number of parties. The device can interact with the parties using the ideal protocol in Section 3.1, extended to enable the parties to verify that the device is genuine and runs trusted code.

These devices are expensive and their resources are inherently limited. A more general solution uses hardware security modules only for a small set of tasks, that cannot be (efficiently) done by cryptographic protocols, such as generation and distribution of threshold decryption keys and a comprehensive set of arithmetic operations and mathematical functions needed in business applications.

Using this approach, straight-line programs for mathematical computations could be done quite easily in a single-server setting. Handling control flow and decryption of the results without compromising privacy seems more difficult. Threshold decryption could still help, and if the server may learn the result, direct interaction between parties is not necessary for decryption.

8 Conclusion

Many e-commerce applications could benefit from secure multiparty computation. Poor performance and functionality, and inadequate architectures prevent the uptake of SMC protocols. Overcoming these obstacles should benefit both consumers and producers and enable better privacy.

We reviewed current security models and methods used for SMC protocols, and analyzed them for practical applicability in e-commerce applications.

Practical aspects of distributed Internet applications have been neglected in the design of SMC protocols. We would

like to bring them back to focus, so that realistic and pragmatic solutions enable future uptake of SMC protocols.

We suggested possible ways to improve efficiency, including mixed solutions, where the parties are helped by service providers, and the protocols combine cryptography with other methods to meet the security requirements. We hope to stimulate future research in this direction.

References

- [1] I. Abraham, D. Dolev, R. Gonen, J. Y. Halpern. Distributed Computing Meets Game Theory: Robust Mechanisms for Rational Secret Sharing and Multiparty Computation. In *Proc. PODC'06*, 2006.
- [2] T.W. Arnold, L.P. Van Doorn. The IBM PCIXCC: A new cryptographic coprocessor for the IBM eServer. *IBM Journal of Research and Development*, Vol. 48, No. 3/4, 2004.
- [3] M. Atallah, M. Blanton, V. Deshpande, K. Frikken, J. Li, L. Schwarz. Secure Collaborative Planning, Forecasting, and Replenishment (SCPFR). In *Proc. of Multi-Echelon/Public Applications of Supply Chain Management Conference*, 2006.
- [4] M. Atallah, M. Bykova, J. Li, K. Frikken, M. Topkara. Private Collaborative Forecasting and Benchmarking. In *Proc. of WPES'04*, 2004.
- [5] D. Beaver, S. Micali, P. Rogaway. The Round Complexity of Secure Protocols. In *Proc. of STOC*, 1990.
- [6] Z. Benenson, et. al. TrustedPals: Secure Multiparty Computation Implemented with Smart Cards. In *Proc. of the 11th European Symposium on Research in Computer Security (ESORICS 2006)*, Germany, 2006.
- [7] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols, December 2005 (revised). <http://eprint.iacr.org/2000/067>
- [8] R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. In *Proc. of STOC*, 2002.
- [9] R. Cramer, I. Damgård, U. Maurer. General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme. In *Advances in Cryptology - EUROCRYPT'00*, LNCS vol. 1807, pp. 316-334, Springer-Verlag, 2000.
- [10] R. Cramer, I. Damgård, J.B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology - EUROCRYPT'01*, LNCS vol. 2045, pp. 280-300, Springer-Verlag, 2001.
- [11] R. Cramer, R. Gennaro, B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology - EUROCRYPT '97*, LNCS vol. 1233, Springer-Verlag, 1997.
- [12] I. Damgård, et. al. A Practical Implementation of Secure Auctions Based on Multiparty Integer Computation. In *Proc. of Financial Cryptography 2006*, LNCS vol. 4107, pp. 142-147, Springer-Verlag, 2006.
- [13] C. Ferris, and J. Farrell. What are Web Services? *Communications of the ACM* 46(6), 2003.
- [14] O. Goldreich. Secure Multi-party Computation, 2002. <http://www.wisdom.weizmann.ac.il/oded/pp.html>
- [15] M. Huhns, and M. Singh. Service-Oriented Computing: Key Concepts and Principles. *IEEE Internet Computing* 9(1), 2005.
- [16] S. Izmalkov, S. Micali, M. Lepinski. Rational secure computation and ideal mechanism design. In *Proc. of the 46th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 585-595, 2005.
- [17] S. D. Gordon, J. Katz. Rational secret sharing revisited. In *Proc. of the 5th Conference on Security and Cryptography for Networks (SCN'06)*, LNCS vol. 4116, Springer-Verlag, pp. 229-241, 2006.
- [18] J. Katz. Universally Composable Multi-party Computation Using Tamper-Proof Hardware. *Advances in Cryptology - EUROCRYPT '07*, LNCS 4515, Springer-Verlag, pp. 115-128, 2007.
- [19] Y. Lindell, B. Pinkas. Privacy Preserving Data Mining. In *Advances in Cryptology - CRYPTO'00*, LNCS vol. 1880, pp. 20-24, Springer-Verlag, 2000.
- [20] Y. Lindell, B. Pinkas. A Proof of Yao's Protocol for Secure Two-Party Computation. *Cryptology ePrint Archive: Report 2004/175*.
- [21] A. Lysyanskaya, N. Triandopoulos. Rationality and adversarial behavior in multi-party computation. In *Advances in Cryptology - CRYPTO'06*, LNCS vol. 4117, Springer-Verlag, pp. 180-197, 2006.
- [22] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella. Fairplay - A Secure Two-Party Computation System. In *Proc. of the 13th USENIX Security Symposium*, 2004.
- [23] M. Naor, B. Pinkas, R. Sumner. Privacy preserving auctions and mechanism design. In *Proc. of 1st ACM Conference on Electronic Commerce*, 1999.
- [24] A. C. Yao. How to generate and exchange secrets. In *Proc. of the 27th IEEE Symposium on Foundations of Computer Science*, pp. 162-167, 1986.