A Management Framework for Secure Multiparty Computation in Dynamic Environments

Marcel von Maltitz, Stefan Smarzly, Holger Kinkelin and Georg Carle
Technische Universität München, Department of Informatics
Chair for Network Architectures and Services
85748 Garching b. München, Germany
{lastname}@net.in.tum.de

***Abstract*—Secure multiparty computation (SMC) is a promising technology for privacy-preserving collaborative computation. In the last years several feasibility studies have shown its practical applicability in different fields. However, it is recognized that administration, and management overhead of SMC solutions are still a problem. A vital next step is the incorporation of SMC in the emerging fields of the Internet of Things and (smart) dynamic environments. In these settings, the properties of these contexts make utilization of SMC even more challenging since some vital premises for its application regarding environmental stability and preliminary configuration are not initially fulfilled.**

We bridge this gap by providing FlexSMC, a management and orchestration framework for SMC which supports the discovery of nodes, supports a trust establishment between them and realizes robustness of SMC session by handling nodes failures and communication interruptions. The practical evaluation of FlexSMC shows that it enables the application of SMC in dynamic environments with reasonable performance penalties and computation durations allowing soft real-time and interactive use cases.

***Index Terms*—Cryptography, Secure Multiparty Computation, Privacy, Internet of Things, Orchestration**

I. Introduction

The Internet of Things (IoT) and Smart Buildings are emerging paradigms which aim for joining the physical and the digital world. An essential step towards this is to deploy a multitude of sensors in physical environments which then allow gaining insights into the environment's current state [1]. This data can be used to provide a fine-grained understanding of real-world processes, inform users, and (automatically) influence and manage the physical environment.

State-of-the-art cloud-based and local but centralized data stores, however, bear similar data protection problems: They constitute a single point of attack for all users' data. Likewise, conformance with data protection regulations can be a complex and challenging task. They might even completely prohibit valuable data usage, creating a trade-off between security and utility of collectable data.

SMC is a technical solution to these conflicts. Its application is possible due to the following observation: Often, collected data is only needed in post-processed form. Data points of different times or different locations are combined and numerical values are transformed into categorical (or boolean) decisions. While the raw input data might have been privacy-critical, often, the final output is not. The technology has already proved applicable [2]–[7] (cf. Section VII). However, publications stress that more focus should be laid on the practical problems of applying SMC: Administrative real-world challenges should gain more attention and more challenging settings like cloud-environments should be considered. Bogdanov et al. [2] "consider it an important challenge to reduce the administrative attention required for managing" a computing node to make "the technology easier to deploy in practice". Furthermore, Burkhart et al. [3] recognize that robustness of computations in the presence of host failures is a vital challenge which should be considered in future. We address exactly these problems in our work.

We capture this demand by focusing on smart environments as use case. They are understood to be dynamic settings where a completely dependable infrastructure cannot be assumed. Similarly, host and networking failures have to be taken into consideration when designing software for them. Nevertheless, the need for privacy-preserving computation is strongly given, which is preferably performed repeatedly and automatically without notable overhead of manual management. We use smart office buildings as running example: Information about building usage is distributedly collected in every office space. It is used to provide insights for building managers as well as the inhabitants (e.g. via public displays). Furthermore the same data can be forwarded to controllers and actuators influencing the state of the building (e.g. HVAC). However, privacy-preserving processing is needed as the collected data especially provides data about the employees working in the sensed area. It can contain presence information about individual employees and give insights about their working and moving behavior. Tracking and profiling becomes possible. [8]

In the following, we explicitly address the problems which emerge when trying to apply SMC in these contexts. Our contribution is the wrapper framework *FlexSMC* for SMC implementations which enables managing of participating nodes and the orchestration of collaborative computation. It facilitates setup including node discovery and cryptographic bootstrapping. During computation it performs monitoring and alive checks of

This work has been supported by the German Federal Ministry of Education and Research, project DecADe, grant 16KIS0538 and the German-French Academy for the Industry of the Future.

the peers and provides mechanisms for session recovery in case of failures. As a result, FlexSMC enhances SMC to be used as a dependable and self-managing service in dynamic and unstable environments. The concept of providing a seemingly centralized service providing access to distributed data is caught in the notion of *Virtual Centrality* (cf. Section III).

We structure our work as follows: In Section II we provide background about SMC and outline the challenges for SMC in dynamic contexts. Section III discusses how these can be conceptually approached. In Section IV we give an overview over the architecture which is then described in detail in V. Section VI presents our evaluation showing the results of our performance measurements. Section VIII concludes the paper.

II. Secure Multiparty Computation

Secure multiparty computation (SMC) is a method which allows a group of parties $P_1 \dots P_n$ to collaboratively compute a common function f. Hereby, each party P_i can contribute an input x_i to the computation; no other party learns this value but all learn the result $f(x_i, \dots, x_n)$.

Yao [9] [10] was the first to question how two parties P_i and P_j can privately compare their values x_i and x_j so that both only learn which one is bigger without sharing the actual values. Later, this question was generalized to arbitrary computations. This resulted in the research field of SMC. Since then, research focused especially on feasibility of SMC in a variety of contexts, increasing its performance and improving its security in stronger adversary models.

SMC is executed as protocols between the participants. These protocols are typically organized in synchronized, sequential rounds. A round consists of a computation step, where every node performs the same calculations predefined by the protocol and often a communication step where some data is exchanged to proceed with the next round. This communication has to happen between every pair of participants.

Challenges in Dynamic Environments

The Internet of Things and smart environments bear characteristics which are initially incompatible with SMC. We consider solving these conflicts as the fundamental requirement in order to transform SMC into a valuable service in smart environments. The main conflicts are:

Before executing an SMC protocol, the nodes need to know each other (R.1). They must have an established trust relationship in order to create secure authenticated point-to-point channels (R.2). Compatibility for computation (R.3) between nodes must be ensured: They must be capable of the same SMC protocols for being syntactically able to interact and provide the same type of data for producing semantically sensible results. Coordination of the SMC protocol has to be performed from outside (R.4), at least the initiation of the computation has to be triggered at each node nearly synchronously. SMC is not necessarily robust against failing nodes during computation and recovery of sessions is not trivial. The set of participants cannot dynamically change (R.5) in a standard SMC computation. Consequently, the environment

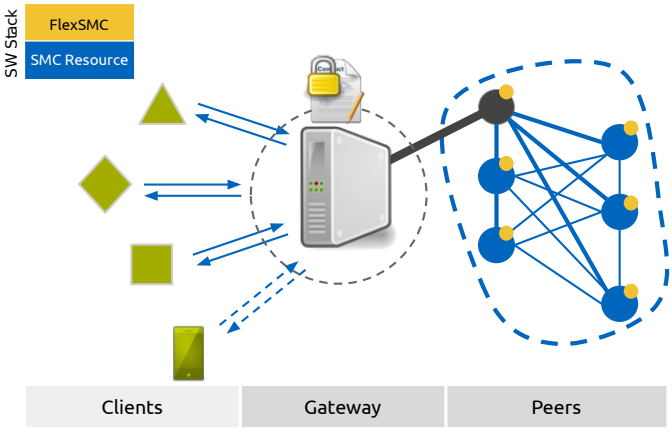

Figure 1: Overall architecture

where SMC is executed must be robust itself (R.6). Lastly, SMC technically uses peer-to-peer protocols, they are initially incompatible with a client-server architecture (R.7). I.e. clients cannot trivially request data from such a network. Furthermore, at the end of an SMC computation the result is available for each participant and no one outside this group. If other entities need access to the result it must be forwarded (R.8).

Smart environments on the other side are dynamic, i.e. nodes are not initially known to each other, no trust relationship between them exists initially, and nodes can unexpectedly join and leave the network. Every node might have different sensors attached, computation compatibility is not trivially given. Lastly, in an environment where other services shall depend on the results, a single point of contact is desirable.

III. Approach: Virtual Centrality

When formally proving security and correctness of SMC protocols, a *simulator-based* proof approach [11] is used. Roughly spoken, the proof shows that an adversary cannot differentiate whether it interacts with an SMC protocol or an actual Trusted Third Party (TTP).

We use this model as inspiration: TTPs bear characteristics which are desirable to address the management challenges mentioned in Section II. Hence we try to incorporate these properties while using SMC as core. Given by the scenario, there are distributed nodes acting as data sources. In the following, we denote them *peers*. Similarly, data consumers exist, they are *clients* of available services, like public displays or HVAC controller as exemplified before. For our solution, we now define a special further node – the gateway (GW) – which acts as a virtual server, shadowing the fact that data is actually stored in a distributed manner and requests are answered by collaborative on-the-fly computation by the data sources. The GW performs centralized coordination and orchestration of SMC computations but is not trusted with respect to the data. We call this concept *Virtual Centrality*.

The main functionality of the GW can be divided into two parts (cf. Figure 1): Firstly, it provides a unified API for data consumers, hiding SMC from them. Clients can

request preliminary metadata which nodes and which data are available. Based on this information, clients can issue computation requests, i.e. requests of data which is then newly computed by a corresponding SMC session between the peers. Clients finally receive their answers via the API. Secondly, it must coordinate the computations performed by the data sources. It acts as a discoverable management node which initially collects metadata about the capabilities of the connecting peers and establishes control connections to them. Upon a computation request, it initiates and orchestrates a corresponding computation, finally receiving the result itself but nothing else. Due to the mutual trust relationship among all nodes, and discovery and bootstrapping mechanism, any node is potentially eligible for the gateway's leader role. In use cases where a preliminary infrastructure can be assumed (e.g. smart offices or homes), the gateway can be assigned rather statically to a dedicated node. In highly volatile environments like ad-hoc-networks, the gateway role can be chosen completely dynamically at runtime.

IV. Overview

In the following section we present the overall architecture of FlexSMC, our approach to practically realize Virtual Centrality and address the challenges discussed in Section II.

A. Entities

We elaborate further on the roles of entities implied by our use case and added by our solution:

1) Peers: Peers are sensor platforms which gather raw information. They can differ regarding their capabilities, i.e. the attached sensors. This and other information are available at the peers as metadata. The peers carry out the SMC sessions.

2) Gateway: The *gateway* is a central yet untrusted *peer*-like node. It functions as described in Section III. It presents itself as a monolithic service for all clients and coordinates and orchestrates the SMC sessions for all participating peers.

3) Clients: Clients are data consumers which request and finally receive the collected data to perform actions upon them or make them visible to users.

B. Functionality

We implement a GW which provides functionality to cover all problems stated before in Section II. It contains an orchestration module which solves R.1, R.2, and R.3 during a setup phase for every joining peer. During runtime it initiates computations requested from outside R.4. The monitoring module addresses R.5 and R.6. The GW will also participate in the computations and also obtain the final result. I.e, after having received a request and translated it into an SMC session (R.7), it is also able to response with the result at the end (R.8).

FlexSMC acts as a wrapper for existing SMC frameworks. I.e., we do not duplicate SMC functionality but provide an environment which mitigates the conflicts between dynamic contexts and the premises of SMC application. FlexSMC then connects via a local socket to an SMC instance and controls it via an adapter (cf. Figure 4). This decouples FlexSMC from specific SMC implementations. This is also beneficial with respect to improvements on SMC protocols. As long as no fundamental changes in the structure of sessions are made, FlexSMC remains compatible.

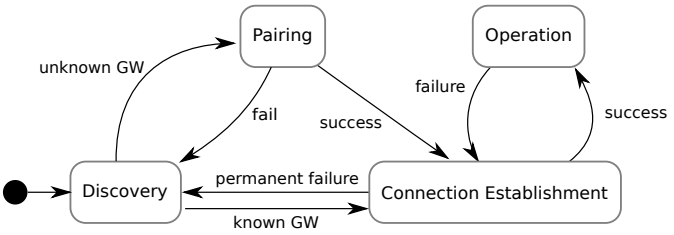

Figure 2: State machine of peers

V. Architecture

In this section, we explain the aforementioned modules of FlexSMC in detail.

A. Orchestration

When new nodes join they need to find an appropriate GW instance. This GW then has to gather information about the new node, and prepare it for later computation sessions. The peer-side process is depicted in Figure 2.

For *discovery* of GWs we employ mDNS [12][13]. The GW acts as a service which announces itself over the network. Since multiple GWs can be present which may have different locations, purpose, etc., GWs also send this metadata during this announcement. This facilitates selection of the right GW for new peers. Selection can happen manually or automatically on the peer-side. When a peer has selected a well-suited GW, it establishes a connection to the GW and starts the pairing.

During *pairing*, self-signed certificates of the new peer and the GW are exchanged which provide cryptographically secure identities for the peers (R.2).[1] Later, when invoking a computation, all necessary certificates are distributed among the participating peers in order to enable computations performed over secure channels. Furthermore, the peer provides metadata which enables creation of semantically sensible SMC groups of peers. Examples for this data are the peer's location and its capabilities. Each group gets a label which enables clients to address this very group by its name. It is the task of the GW to resolve the name again when a computation is requested. Lastly, when *connecting* a permanent control channel is established to enable the GW to later provide instructions to this peer.

When *operation* has been started, peers are ready to execute computations. Incoming requests from clients are preprocessed by the GW, which hides SMC for the clients completely (R.7). The group label is resolved and the input data type and the choice of protocol are extracted. Afterwards, the GW informs the corresponding set of peers about the upcoming computation including its metadata and triggers execution (R.4). This includes the identities and connection endpoints of all other participants (R.1). Peers can then prefetch the input

[1]This can be substituted by a Public Key Infrastructure if available.

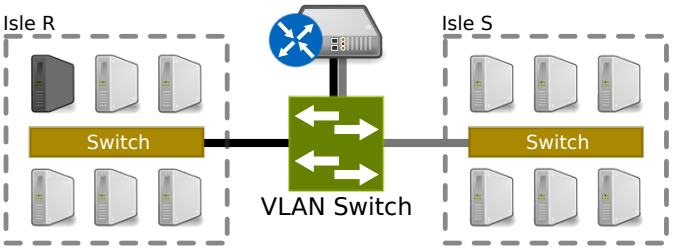

Figure 3: Topology of the test setup

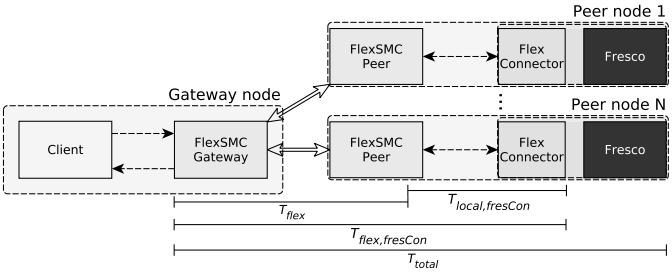

Figure 4: Test metric for the complete communication path

data and perform local setup of the SMC implementation. On each peer, the computation is delegated to a dedicated SMC implementation (R.3). It uses the metadata to connect to the participants and collaboratively execute the selected protocol. When the computation has finished, all participants, including the GW, have obtained the result. The GW forwards the result to the initially requesting client (R.8).

B. Monitoring

Monitoring mainly observes the state of and the connection to already joined nodes. It addresses the challenges of [2], [3] mentioned in Section I.

1) State Monitoring: The availability of a joined node is monitored in a two way fashion: Regularly, peers send heart beat messages to the GW. When the connection drops, peers will take notice of it, clean up the connection and transition back into discovery mode (cf. Figure 2) trying to reestablish a connection the same or – in case of a permanent error – to another GW. On the side of the GW, missing heart beat messages or connection loss in the control channel indicate errors which also cause the cleanup of the corresponding channel. The peer is then unlisted as active and available.

2) Computation Monitoring: We delegate the computation to a SMC framework which we assume to be available on each peer. As a consequence the actual computation is carried out via other channels than the channels FlexSMC controls. Therefore, SMC communication cannot be monitored directly.

However, FlexSMC is connected to the SMC framework so that exceptions can be retrieved from the computation session.

Assuming that SMC sessions cannot recover themselves, the GW then instructs to cleanup the failed session and to begin recovery (R.6). Using the information from state monitoring (Section V-B1) the GW optionally removes vanished or failed peers (R.5) and restarts the computation a predefined, finite time.

VI. Performance Evaluation

In order to assess feasibility of our solution we performed measurements focusing on absolute durations for the processing of requests and the overhead induced by FlexSMC.

1) Setup: We used 12 physical hosts in a topology as depicted in Figure 3. There is a switch per subnet; both are in turn connected by a central switch. Traffic between subnets needs to pass the router. The left upper node of Subnet R acts as a dedicated GW. Each host possesses 16 GB main memory and an Intel Xeon CPU with 8 cores having 2.50 GHz and 8192 KB cache size. The operating system is a headless Debian 8.5 Jessie using a Kernel of version 3.16.0-4. They are connected via a 1 Gbit ethernet network.

We employed FRESCO [14] (version 0.2) as SMC framework. The basic computation operations are provided by the BGW protocol [15]. Being guided by the smart office example, we perform a simple summation of all participant's input values as it is typically necessary for generating statistics from collected information, e.g. calculating the average. Each measurement has been repeated 1000 times.

2) Method: We dissect the steps from request to computation corresponding to Figure 4, gaining different test cases. Here, T always denotes the time for a bi-directional communication flow. Dotted edges visualize local communication on a host while solid edges between FlexSMC instances depict any routed and switched network. T_{flex} denotes the time which is needed to inform peers about an upcoming computation and to invoke the computation on the peers. Adding the time spent while communicating via a local socket with the SMC implementation yields $T_{flex,fresCon}$. The full time taken, also including the actual execution of the SMC protocol, is T_{total}. In this work, we will provide measurements for T_{flex} and T_{total}.

Communication is carried out simultaneously with every peer. However, during the SMC computation each round has to be synchronized among all peers. Practically, this means that the slowest peer determines the overall performance. Due to this reason all our results always depict the maximum duration, i.e. the time the slowest peer took per measurement.

3) Results: We provide three insights: a) The overall time a protocol execution costs. b) The fraction FlexSMC adds to this amount. c) The scaling behavior with respect to the number of peers, an important parameter in our setting.

Figure 5 shows that a sequence of 10 consecutive "echo" requests sent to the FlexSMC peers via individual control channels involving all FlexSMC framework components in an end-to-end view but without connection to the local adapter and actual computation costs around 4 ms. This value increases very slightly when more peers are added. Actual forwarding of these messages via the local socket adds further 3 ms to the overall duration. Regarding scaling behavior, the offset is correspondingly shifted and the slope is increased.

The next measurements included the actual SMC computation. Figure 6 shows that the corresponding durations have an essentially higher offset. The setting with 3 peers starts with

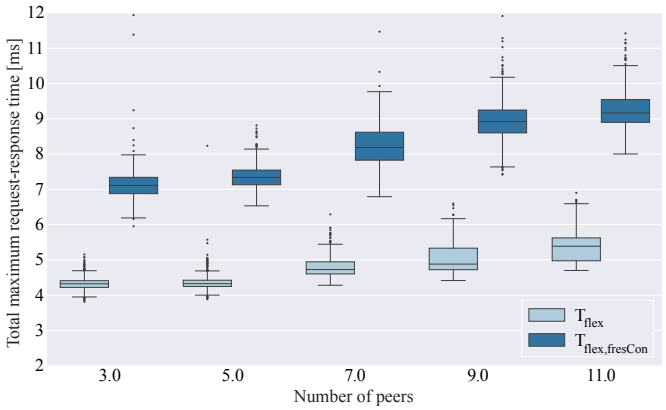

Figure 5: Total time for 10 consecutive echo requests

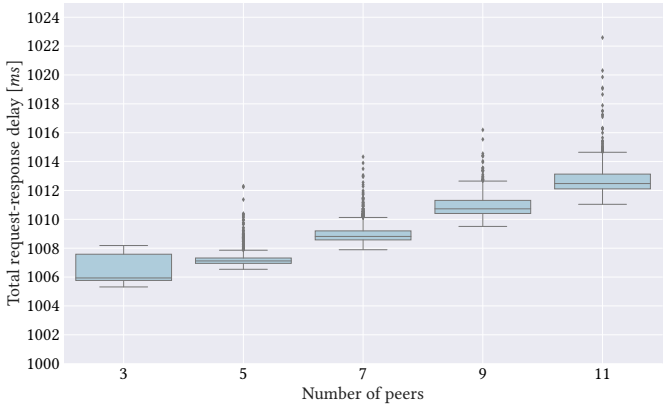

Figure 6: Computation of a single secure sum per round with different number of participants

1006 ms and increases to 1012 ms for 11 peers. Investigating this issue, we found out that this is neither SMC-specific nor directly caused by the performed computation. Instead, the networking layer of FRESCO always waits a full second before checking whether all necessary channels between each pair of peers has been established. This is also the reason why this is only an offset which does not influence the slope.

Besides absolute durations comparison with a TTP solution as given in the state-of-the-art is of interest. We can approximate its cost by assuming that the GW would act as a TTP, performing the computations centrally. For this purpose, each peer would forward its data to the GW. Using this approximation and neglecting the central computational costs, utilization of a TTP would then be around T_{flex} since its corresponding commmunication is not SMC-specific. It is, however, vital to see that the costs would be nearly independent of the number of peers contribution to the computation. All peers would be able to send their values simultaneously. This is in stark contrast to the SMC solution: Carrying out the SMC protocol requires communication between all participating peers. This causes a linear increase of the duration when adding further peers.[2]

[2]The initially quadratic overhead is reduced to linear due to parallel communication.

Due to the waiting time in FRESCO, client requests would be answered in roughly more than a second. Here, FlexSMC imposes only minimal overhead, and with further versions of FRESCO, the overall time can even decrease to some milliseconds. These performance characteristics enable serveral use cases: Batch data processing and interactive applications become feasible. Similarly, continuous applications requiring soft real-time like public displays or actuators for HVAC control would be well realizable.

The obtained results give first insights in the performance behavior. Regarding transferability of the performance results to real world deployments we consider three aspects. The network properties like packet loss and transmission rate are quite similar to the anticipated intranet setup. The hosts in our setup possess more computing power than IoT devices. However, we expect this divergence to be acceptably small: Due to the high communication overhead of SMC, computation performance is typically bound by the network parameters. Lastly, real-world intranets can feature a more complex topology than our setup. This can influence network latency, which, in turn, might negatively influence computation speed. A similar influence is expected when scaling the number of nodes beyond our current test setup.

VII. RELATED WORK

Over the last decade there have been multiple feasibility studies demonstrating practical SMC application. However, it becomes apparent that most solutions presented in related work have been executed manually in highly controlled environments. Only few propose architectures for continuous real-world deployment and automated computations. Among them, some in turn do not consider realistic constraints of their environment.

In constrast, we deliberately chose dynamic environments and automated computation as our scenario in order to examine and address infrastructural challenges for SMC present in smart environments. We provide a solution which allows continuous and automatic application of robust SMC computations in dynamic contexts. We evaluated our prototype in a real network and proved its feasibility for practical purposes.

The first practical and large-scale application of SMC happened in 2008 [4]. Its purpose was to perform a multiple seller multiple bidder auction. Data from 1200 users was collected and split into shares locally in a Java applet. The computation has then been performed by three laptops in a local network. The whole computation, working with 1229 bids encompassing 9 million individuals numbers took 30 minutes in an 100Mbps intranet setting. They do not give insights how the results were distributed to the initial data input parties.

Martin Burkhart developed and applied SEPIA to event correlation for network data [3] in 2010. The overall goal is generation of network traffic statistics and anomaly detection. The collected data stems from 140 input parties while varying the computation parties between 3 and 9. Their evaluation was performed in a very controlled setting: Measurements have been performed in intranet and internet settings (using

PlanetLab), however without real interaction via SMC with the ISPs which provided the data.

Bogdanov et al. [2] set up SMC in 2012 for computing statistical financial indicators of companies. Sharemind [16] was used for realizing ranking operations. They provided a web-based solution which created the shares locally in the browser via JavaScript before submitting them to three computational nodes. These were located in three participating companies. They successfully deployed it as an application for continuous use, several computations have been carried out.

Djatmiko et al. [5] reused SEPIA in 2013 to perform collaborative outage detection. They extended an existing approach to work privately with multiple inputs. The core operation is multiset union operation which they realize with *counting bloom filters (CBFs)*. Their CBF works on an integer array which has to be generated from equally-shaped, local arrays. The necessary summations of the interger elements are performed via SMC. Real-world data from an ISP is used and evaluated in a controlled and manually setup cloud setting.

In 2016, Zanin et al. [6] applied SMC to transmit CO_2 emission allowances between airlines. They used SEPIA implementing the necessary comparison functions. Most notably, they have realized a real web service interacting with actual stakeholders. While the interaction is neither fully automated nor the SMC computations are performed automatically, they provide an infrastructure which addresses a concrete use case and allows real interactions. The same is reflected in their measurements, where also organizational overhead is assessed which includes peer discovery and authentication.

In 2017, Bonawitz et al. [7] from Google published a solution for SMC-based privacy-preserving training of neural networks (NN). A trained model of a NN can be represented as a (long) integer vector. Training such a model *privately* means that a global vector is build from a multitude of local input vectors without making these available to anyone. In their approach they address real-world problems like interrupted connections, vanishing participants, and NAT-shielded devices. The corresponding talk [17] implies that the system will be productively used for privately creating auto-suggestion models for smartphone input trained by the typing history of a multitude of smartphones. Abstractly, Bonawitz et al. address *Secure Aggregation*. This facilitates their use case: They only need a single server finally holding the global model while being oblivious to the input data. Furthermore, only integer summation must be supported.

Thoma et al. [18] present a secure smart metering solution. They aim for protecting the individual, temporally fine-grained consumption data of households, providing correct and verifiable billing of each individual household and fine-grained (non-individual) consumption feedback for load management. Here, SMC is utilized to spatially aggregate temporally fine-grained consumption data over several households before sending them to the energy provider. Thereby, they also address a secure aggregation in a real-world setting and provide a framework which enables practical application. However, their solution is rather on the level of a concept. The paper does not discuss how infrastructural problems like discovery of available households and interconnection between them are handled.

VIII. Conclusion

Smart environments and the Internet of Things are emerging technologies which require the privacy-protection of distributedly collected data. Secure multiparty computation (SMC) is a valuable candidate for this purpose. Its practical feasibility has already been shown in several studies in the last decade. However, most of them have been carried out in highly controlled environments and with a lot of manual intervention. Relevant questions of productive application in dynamic environments have therefore not yet been answered.

We examine the problems arising for SMC when considering these contexts: Due to their volatility preliminary information about possible participants and their identities are not necessarily known. Similarly, secure channels for SMC require previously performed trust exchanges between the participants. Lastly, failing hosts and connections must be taken into consideration, with which SMC realizations cannot trivally cope themselves. Our contribution is FlexSMC, a management and orchestration framework which turns SMC into a dependable service for these scenarios. Its main paradigm is *Virtual Centrality*: A selected node among the data sources provides management and orchestration functions for enabling computations while remaining untrusted with regard to data handling. During setup, it supports node discovery and trust establishment. It serves as single point of contact for data requests and translates them into executable SMC sessions. When performing computations, it monitors them and enables recovery when node or communication failures occurred.

Our performance evaluation shows that SMC in general is feasible in dynamic environments and FlexSMC adds only negligible overhead to it.

In this work we addressed the peer-faced side of the gateway. Extending the client-faced side is vital future work: This comprises that the current request API needs access control for clients. In order to enable peers to stay in control, access control decisions could be gathered from them and then be carried out by the gateway. Furthermore, further privacy protection goals like transparency and intervenability should be addressed. SMC provides a promising foundation since data processing and computation always has to happen in cooperation with the data sources. This enables them to stay informed about how their data is used. Similarly, veto rights could be introduced which allows peers to deliberately refrain from certain computations.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 1645–1660, 2013.

[2] D. Bogdanov, R. Talviste, and J. Willemson, "Deploying secure multiparty computation for financial data analysis," *Financial Cryptography*, pp. 57 – 64, 2012.

[3] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "SEPIA: Privacy-preserving Aggregation of Multi-domain Network Events and Statistics," *Proceedings of the 19th USENIX Conference on Security*, p. 15, 2010.

[4] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft, "Secure multiparty computation goes live," in *Lecture Notes in Computer Science*, vol. 5628 LNCS, 2009, pp. 325–343.

[5] M. Djatmiko, D. Schatzmann, X. Dimitropoulos, A. Friedman, and R. Boreli, "Collaborative Network Outage Troubleshooting with Secure Multiparty Computation," *IEEE Communications Magazine*, no. November, pp. 78–84, 2013.

[6] M. Zanin, T. T. Delibasi, J. C. Triana, V. Mirchandani, E. Álvarez Pereira, A. Enrich, D. Perez, C. Paaolu, M. Fidanoglu, E. Koyuncu, G. Guner, I. Ozkol, and G. Inalhan, "Towards a secure trading of aviation CO2 allowance," *Journal of Air Transport Management*, vol. 56, pp. 3–11, 2016.

[7] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical Secure Aggregation for Privacy Preserving Machine Learning." *IACR Cryptology ePrint Archive*, vol. 2017, p. 281, 2017.

[8] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed internet of things," *COMPUTER NETWORKS*, vol. 57, pp. 2266—-2279, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2012.12.018

[9] A. C. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE, 1982, pp. 1–5.

[10] ——, "How to generate and exchange secrets," in *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 1986, pp. 162–167.

[11] R. Canetti, "Security and Composition of Multi-party Cryptographic Protocols," 1999.

[12] S. Cheshire and M. Krochmal, "Multicast DNS," RFC 6762 (Proposed Standard), Internet Engineering Task Force, Feb. 2013.

[13] ——, "DNS-Based Service Discovery," RFC 6763 (Proposed Standard), Internet Engineering Task Force, Feb. 2013.

[14] "A FRamework for Efficient Secure COmputation." [Online]. Available: https://github.com/aicis/fresco

[15] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault Tolerant Distributed Computation," *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 1–10, 1988.

[16] D. Bogdanov, "Sharemind: programmable secure computations with practical applications," Ph.D. dissertation, 2013.

[17] B. Kreuter, "Secure Multiparty Computation at Google," in *Real World Crypto Symposium*, 2017. [Online]. Available: https://www.youtube.com/watch?v=ee7oRsDnNNc

[18] C. Thoma, T. Cui, and F. Franchetti, "Secure Multiparty Computation Based Privacy Preserving Smart Metering System," *44th North American Power Symposium (NAPS)*, pp. 1–6, 2012.