Data Querying and Access Control for Secure Multiparty Computation

Marcel von Maltitz, Dominik Bitzer, Georg Carle
Chair for Network Architectures and Services
Department of Informatics, Technical University of Munich
{lastname}@net.in.tum.de

***Abstract*—In the Internet of Things and smart environments data, collected from distributed sensors, is typically stored and processed by a central middleware. This allows applications to query the data they need for providing further services. However, centralization of data causes several privacy threats: The middleware becomes a third party which has to be trusted, linkage and correlation of data from different context becomes possible and data subject lose control over their data.**

Hence, other approaches than centralized processing should be considered. Here, Secure Multiparty Computation is a promising candidate for secure and privacy-preserving computation happening close to the sources of the data.

In order to make SMC fit for application in these contexts, we extend SMC to act as a service: We provide elements which allow third parties to query computed data from a group of peers performing SMC. Furthermore, we establish fine-granular access control on the level of individual data queries, yielding data protection of the computed results. By adding measures to inform data sources about requests and the usage of their data, we show how a fully privacy-preserving service can be built on the foundation of SMC.

***Index Terms*—Transparency, Intervenability, Access Control, Internet of Things, Secure Multiparty Computation**

I. Introduction

In the last years, the Internet of Things (IoT) has become an emerging trend, including the rise of smart environments. In these contexts, a certain structure of applications is prevalent: Data points are collected at different (spatial or logical) locations; for their utilization, they are horizontally aggregated over a set of collectors; the outcomes are statistics or higher-level information which are then used by other services. The corresponding architecture normally features sensors for data collection, a middleware for data storage and processing and services which receive the processed data for further usage. Examples for services are public displays providing statistics and controllers or actuators triggered by data change events.

However, the data which is initially collected by sensors is privacy-sensitive since it typically reflects user presence and interaction: Presence influences the temperature in single rooms, the CO_2 concentration and device utilization (e.g., lighting). Certain sensors, e.g., for power consumption tracking provide even more specific insights into user behavior (Intrusive Load Monitoring, cf. [1]). This raises a huge need for information security and privacy, which is not well addressed by centralized storage and processing solutions.

Secure Multiparty Computation is an emerging and highly promising approach for providing privacy-preserving systems. It encompasses a class of protocols which allows a group of parties to compute arbitrary functions based on confidential inputs without sharing these values with any other party. With the advent of edge computing [2] [3] it also becomes a promising alternative in the context of smart environments: Data can remain on local sensor platforms in the proximity of the sensors where it was collected. Execution of SMC protocols between these nodes then enables secure processing without sharing or centralizing raw sensor data.

However, SMC is not directly applicable in the depicted context due to an architectural mismatch: SMC uses a peer-to-peer like communication (cf. [4], [5]) while smart environments are service-oriented: Nodes provide services for other clients; in particular, clients should be able to request specific data processing and receive the result afterwards. Our goal in this paper is to extend SMC to be applicable in the architecture of smart environments. Using a state of the art notion of privacy, we yield a fully privacy-preserving service for the processing of sensor data.

The remainder of this paper is structured as follows: In Section II, we provide our notion of privacy and a background on SMC. In Section III, we elaborate the related work on practical application of SMC. Section IV provides a goal statement of our work. We present our approach in Section V. Its evaluation, with regard to security and privacy as well as performance is carried out in Section VI. Section VII concludes the paper.

II. Background

In this section, we provide the background to our notion of privacy, SMC in general and the Security and Privacy Model of SMC applied to our domain of smart environments.

A. Privacy

Initially, privacy and data protection are high-level concepts. In order to realize these properties in an information technological system, it is necessary to further refine them and break them down into more specific parts.

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.

Like security decades before, privacy and data protection also underwent the process of refinement by defining protection goals some years ago, which make the concepts more graspable. Following [6]–[9], these protection goals mainly are *data minimization*, *unlinkability*, *transparency* and *intervenability*. For the standard definition we refer to the mentioned literature, especially [6]. An interpretation in our specific context is provided in Section II-C.

We utilize this notion of privacy and data protection, since it is state of the art and has found widespread adaptation, be it on the level of individual states [10], [11] or the European Union [12].

B. Secure Multiparty Computation (SMC)

SMC formalizes a problem of controlled leakage. Assume multiple cooperating parties, each holding a confidential value. They agree on some function which takes these values as inputs. Using SMC, the function is correctly evaluated while its result is the only new information released. The input of each peer is not shared with any third party including the other cooperating peers. [13]

The seminal work of Yao [14], [15] laid the foundation of SMC; from there, several different methods for realization emerged (e.g., [4], [16], [17]). The most promising foundations for SMC currently are garbled circuits, homomorphic encryption and secret sharing schemes. These approaches enable different usage models [18], i.e., outsourced processing, outsourced services and joint processing. For a comprehensive overview see [18]. Today, research mainly focuses on the performance of general purpose computation suites, strengthening their security, identifying new fields of application and designing efficient specific purpose protocols. [5], [19]–[21]

With regard to privacy protection, SMC naturally fulfills data minimization and unlinkability. Data minimization is provided since raw data can remain where it is created. Desired results can be computed by SMC directly without creating privacy-critical intermediary data. Two types of unlinkability are given: By aggregation of multiple parties' input data, linkability between the individual input data and the result is prevented. Tracing back unique parties' inputs from a result is impossible in the general case. Furthermore, linkability among different parties, i.e., correlation of their data, is prevented, since this data is never available at the same logical location.

Currently, a small amount of actively developed SMC frameworks [22]–[24] exists. These provide an implementation of the basic operations and enable creation of arbitrary composed algorithms. Their application is initially restricted to the distributed execution of the created protocols, which does not encompass management and orchestration of peers, coordination of computations and infrastructural requirements enabling application of SMC in data processing scenarios.

C. Security and Privacy Model

a) Assets: The main asset to be protected is the individual raw data of sensor platforms. We assume them to be owned each by the respective data subject, i.e., the person(s) about which the platform gathers information. This is given in use cases where a single smart building is inhabited by different parties, e.g., smart hotels, smart houses with individual rental apartments and can also be given in smart office buildings, if employees have dedicated offices. The necessity for data protection is based on the possibility that sensor data gives insights about the presence and behavior of individuals [25].

b) Protection Goals: With respect to the mentioned assets we understand security to be confidentiality of this very raw data. However, confidentiality may not hinder all processing of the data. Instead, a privacy-preserving access must be designed, meaning data access which is controllable by and accountable for the data owners. Following our privacy background in Section II-A, privacy-preservation encompasses the protection goals of 1) data minimization, 2) unlinkability, 3) transparency, and 4) intervenability. They have the following meaning in our context: 1) Information is only derived from raw data if it is actually needed by any client service. The purpose is known before information is created. 2) Information made available to clients does not allow restoring contributions of individual single peers. Correlations between individual peers should not be possible by client accessible data. 3) Peers should know, what information is derived from their data and for which purpose this information is used. 4) Based on this preliminary knowledge, they should remain in control of their data by deciding which computations may be carried out.

c) Attacks: In classical architectures previously mentioned protection goals are not or only partially fulfilled. Central storage creates a high value target and a single point of attack. Furthermore, it allows arbitrary processing, using data for other purposes and correlation of available data; all this being completely intransparent for individuals and without any ability to intervene.

In our architecture, these attacks are mitigated, since raw data stays on the peers, clients only obtain the post-processed information they have been granted access beforehand. That information is generated by SMC in a privacy-preserving manner and the gateway only orchestrates data processing while not having access to raw data.

The main goal of our approach is that only permitted requests of clients are answered and only the requesting clients obtain the corresponding result. This implies the following set of premises, from which we derive each attacker by the attempt to circumvent one of them. Access requires permission (A.1). Given permissions are legitimate (A.2). If a permission is valid, it was given by an accepted authority (A.3). Only the authorized client can use a given permission (A.4). Validity of permissions is correctly checked (A.5). Data contribution by peers only happens if the validity check was positive (A.6). Results of valid requests do not leak to third parties while being transmitted to the authorized clients (A.7). The results have not been modified on the way to the clients (A.8).

Further attacks are possible but out of scope of this work: Malicious peers can try to obtain information from other peers or provide wrong results. We exclude the former since it has

to be addressed on the level of the SMC protocols and the latter since the correctness of input values provided by peers is out of scope of realizing secure computation (cf. [13, p. 11]). Lastly, clients can try to correlate information they were able to obtain. This is excluded since it depends on the exact choice of available computation queries.

d) Trust: An ultimate design goal is to reduce the amount of components which have to be trusted to handle private raw data faithfully. Our architecture has been designed to avoid single points of attack and high value targets holding private raw data from several parties. The remaining trust is diversified: We associate each sensor platform with individual users. These users trust their respective platform to faithfully collect and store their data. This assumption does not strongly differ from assumptions in classical architectures: In any case, by generating it, sensors have access to privacy-critical data. Furthermore, all trust requirements for the used SMC protocol realization apply.

III. Related Work

Several results were achieved in the last years. However, they show mere feasibility without aiming for an automated system providing SMC as a service. In [26] SMC was used to perform an auction between buyers and sellers of a specific product. Providing data and receiving results was executed manually. Another auction was performed in [27] among different airlines for implementing the EU Emission Trading Scheme. Data input and output are performed using CSV files. In [28] a comparison of key performance indicators among a group of competitive companies was performed. They provided a Javascript library enabling data collection via the browser; computation results were made available via a spreadsheet. Burkhart et at. [29] applied SMC to generate network traffic statistics and anomaly detection. Similarly, [30] reused the same framework to perform collaborative outage detection. Both do not address deployment and data access challenges. Recently, Bonawitz et al. [31] used SMC to collect private user data from smartphones in order to train a central machine learning model. For this specific use case, they provide a solution which is intended to serve as an automated service collecting the desired data.

All of these solutions fall short for the application in the IoT. In most cases, data is provided manually by user interaction. Similarly, the SMC setup is created ad-hoc for single computations. Correspondingly, computations are invoked by manual intervention. Also, the architecture does not match: Data providers and result consumers are the same entities. They cooperate in a peer-to-peer fashion processing data for themselves instead of providing a service for third parties.

A notable difference is the last mentioned work. They actually provide an automated SMC service. However, while optimizing for a certain use case, they sacrifice the ability to compute arbitrary functions and specialize on secure aggregation. Following [32], our solution is agnostic regarding the specific SMC implementation as long as it supports an arbitrary number of computing parties.

A. Previous work for SMC in the Internet of Things

In [33] von Maltitz et al. provide a vision how SMC can be applied in smart environments: The starting point are distributed sensor platforms, understood as edge devices. They represent an intermediary to low-end sensors; they collect the data created by the connected sensors, store it locally and have sufficient resources to perform local, small-sized data processing. SMC computations among these devices allow to derive processed and aggregated information from this local data, which is then made available so that services (public displays, actuators, etc.) can act on them.

As middleware, an *SMC gateway* is deployed. It realizes the link between the sensor platforms and the data consuming services but without having access to or storing the sensor data. Regarding the sensor platforms – called *peers* –, it coordinates SMC sessions for data processing. Towards the services – called *clients* –, the SMC gateway acts as a middleware which allows querying of data in the standard client server paradigm while abstracting from application of SMC.

In [32] von Maltitz et al. focused on the interaction between the peers and the SMC gateway of the aforementioned vision. This work addressed natural technical mismatches between the premises of secret sharing based SMC [4], [34] and the characteristics of dynamic environments. The result is a management and orchestration framework for SMC which enables stable and automated execution of SMC sessions in dynamic environments.

However, since it only provides the first purpose of the SMC gateway, the remaining challenges for extending SMC, i.e., enabling data querying and access control on data processed by SMC, are addressed in this work.

IV. Open Problems and Goal Statement

The first goal of this work is to allow clients to query data from a coordinated SMC group. While doing so, the fact that SMC is used for data processing should be abstracted away for the requesting clients. For this, we need queries which describe the results to be obtained in a non-SMC specific manner (R.1). Protection of original sensor data is achieved by SMC. They never leave the sensor platforms and are not shared with any third party. However, access control has to be carried out on the derived results, ensuring that no client obtains more or other data than intended (R.2).

The second goal is a provide a fully privacy-preserving service based on SMC. This means in addition to fulfilling unlinkability using SMC, transparency (R.3), including accountability (R.4), and intervenability (R.5) should be achieved according to their definitions in Section II-C.

V. Approach

We extend the gateway solution of [32]. First, we define a format for *queries*. A query is a declarative data structure which stands for a computation the gateway offers and clients can ask for. Corresponding *authorization grants* are specified that state the permission which queries a client is allowed to post. Lastly, three types of requests are defined: *Metadata*

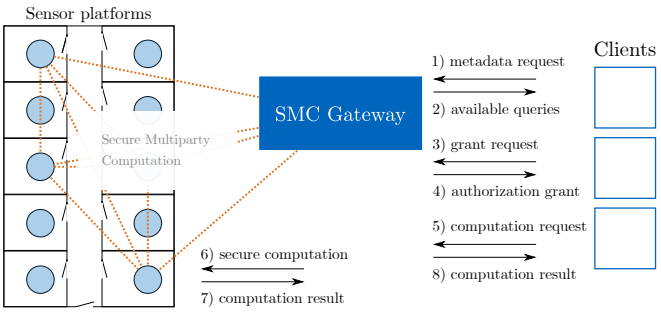

Fig. 1. Interactions between the clients, the SMC gateway and the SMC peers.

requests request the set of available queries from the gateway. *Grant requests* are sent from the client to the gateway in order to obtain an authorization grant, permitting a certain query. *Computation requests* are actual requests, specifying the data to be obtained using a query and providing a corresponding grant authorizing them to obtain this very data.

The subsequent protocols enable the following interaction (cf. Figure 1): Clients perform a metadata request to the gateway (1), obtaining meta information about all available queries via this gateway (2). From this set clients select desirable queries and ask for permission to issue them by performing a corresponding grant request (3). On success, they obtain the respective authorization grant (4). Data is demanded by a computation request (5). When the gateway obtains this type of request, permission is checked using the authorization grant which is sent along with the request. On success, the affected peers are informed, enabling them to also check and verify the request using the provided authorization grant. Additionally, they can perform arbitrary further privacy checks. If all peers agree, the SMC session is setup and carried out (6). When the gateway receives the result (7), it is forwarded to the client (8).

A. Requests and Authorization

a) Cryptographic identity for clients: Given the previous work of [32] we can already assume the availability of a public key infrastructure. Clients have to be equipped with a cryptographic identity, e.g., X.509 certificates, for providing a secure reference point for communication and authorizations to be bound. In order to support transparency later on, the identity is enhanced with metadata about the clients, especially a short description which states the usage purpose of data obtained by this client.

b) Query format: For performing a secure computation the following properties of the computation must be known: 1) The group of peers to participate in the computation 2) the input data to use and 3) the protocol to execute.

Regarding 1) we refrain from letting clients specifically select single peers to form a group. In our use case, the knowledge which is of interest for clients is on a higher abstraction level like a department, a floor, a specific room type etc. Hence, this abstraction is made on the side of the gateway: Newly added peers provide metadata about themselves, including labels describing peers in a domain-specific way. We hence define $labels(p)$ to be the set of key-value pairs (k, v) of peer p. As an example, a peer can have the label set {`roomtype: kitchen, level:3, buildingpart:A,...`}.

When a set P_g of multiple peers is connected to a gateway g, it can build the superset $L_{P_g} \equiv \bigcup_{p \in P_g} labels(p)$ of all provided attributes. This information can then be used to create predefined logical predicates forming groups, which can then be queried by clients. A predicate ``roomtype = kitchen'' would hence select all peers which communicated this label upon pairing with the gateway. Clients can then choose from the finite set of predicates for each of their requests. In order to guarantees privacy towards the peers, clients should not be allowed to craft queries themselves instead of selecting from a predefined set: Making more general queries than provided could enable them to gain information about peers which they are not allowed to obtain. Making them more specific could allow derivation of input data of single or a small group of peers.

Similar to the labels, for 2) each peer provides metadata about the inputs it can provide. We denote this as $inputs(p)$ for peer p. It corresponds with the sensors the peer has available. Besides the type of input data, clients are given the ability to select not only the latest data point but also a list of the points of a given time window reaching into the past. I.e., as $preselector$ the gateway allows to choose from window sizes like *last value*, *last hour*, *last 6 hours*, ...

When selecting a window of values, they have to be merged into a single value as input for the SMC session. For this, we define a $preprocessor$ function that can be selected by the client and request a corresponding aggregation of the values before performing the actual computation. These encompass typical aggregation functions like $min, max, sum, average, \ldots$ but can also be extended.

Regarding 3) we depend on the protocols being available on the peers. These can be provided as labels and clients can again select from a finite set of options.

In summary, the three requirements for SMC sessions are transformed into five attributes a request has to provide. An example is given in Listing 1. Request translation then conceptually consists of two steps: 1) The selected group label is evaluated and the corresponding peers are chosen. All further request attributes are evaluated at the peers, selecting the right (preprocessed) input and the protocol to execute. The session is then carried out according to [32].

```
{
 predicate: type = heater ∧
     roomtype ∈ [kitchen, meetingroom]
 preselector: last 6 hours
 preprocessor: avg
 protocol: sum
 input: power_consumption
}
```

Listing 1. Computation request query

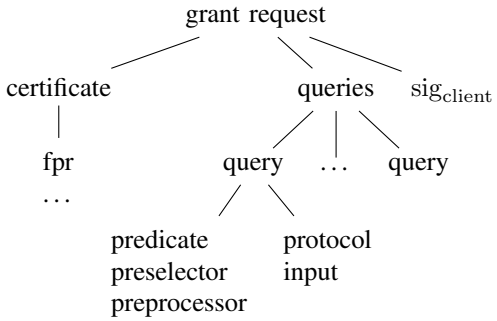

Fig. 2. Structure of a grant request

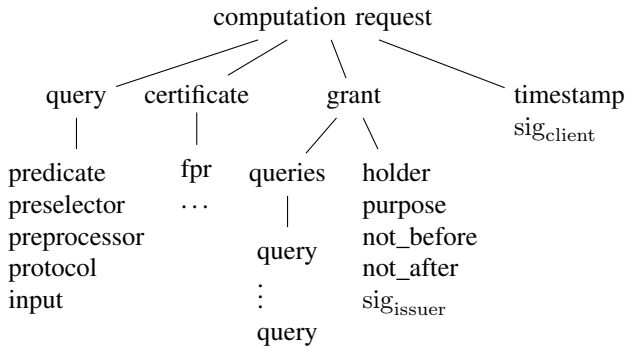

Fig. 3. Structure of a computation request

c) Authorization grants: In a dynamic environment, it is imaginable that multiple gateways are deployed for different purposes. These gateways should be able to verify access requests without time or communication overhead. I.e., no contact to other entities should be necessary nor should extensive state on the gateway itself be necessary for verification. Due to these reasons, we decide for a serialized representation of permission which is transferable and verifiable with a low amount of state information. The purpose of such an *authorization grant* is to state whether a given request is legitimate or not. The client can obtain these grants from an access authority (e.g., the gateway or an external entity). These grants encode the requests allowed for this client. Upon request, the clients also send the grant along in order to prove legitimacy of the request. Forwarding the grant to the peers enables individual revalidation by the data sources themselves.

Technically, the grant mirrors the attributes of the request. I.e., it also contains a predicate which can be matched against the request predicate. Furthermore, preselector, preprocessor and the input type must be of permitted value. Besides that, it is bound to the identity of the corresponding client by adding a cryptographic identifier bound to the client's certificate. To avoid complications regarding revocation, we suggest a short lifetime of the grants and renewal on demand.

d) Request formats: There are two request types: The purpose of the *grant request* from the client is to obtain an authorization grant stating access permission. A request $r_{c,grant}$ of client c consists of the following components (cf. Figure 2): The *certificate* states the identity of the client. *Queries* contains all queries the clients demands access to, characterized by its attributes. sig_{client} serves authentication and integrity-protection of the request. The purpose of the second request, the *computation request*, is to actually obtain computed data. A request $r_{c,comp}$ of client c consists of the following components (cf. Figure 3): The *query* contains the *predicate* and the other characterizing attributes (cf. Section V-A0b). The *certificate* is as described in the previous request type. The *grant* is the authorization grant which states permission to obtain the data in question. It is the answer of the gateway answer to the previous request type. The *holder* is the owner identifier of the *grant*. *not_before* and *not_after* specify the time frame of validity. The *queries* of the *grant* mirror the queries to be allowed for the holder. *Timestamp* reflects the time when the computation request has been created. sig_{issuer} states the permission given by the issuing entity.

e) Accountability of requests: Based on the transparency of computation requests which are forwarded to peers, accountability is achieved by persisting the request data structure. This is extended by a signature of the accepting gateway and optionally the result of the computation.

B. Protocols

We dissect the client/gateway interaction into three independent protocols: 1) *Metadata request*, 2) *grant request*, 3) *computation request*.

Here, we model the state of the gateway to be $(\mathcal{Q}, \Phi)$ where $\mathcal{Q}$ is the finite set of queries made available by the gateway. Each query has the structure as shown in Figure 3. $\Phi : (query, client, context) \rightarrow \{true, false\}$ is an access control structure; it takes a tuple of a query, a requesting client and a context and returns whether or not access is permitted.

a) Metadata request: This is the first interaction between the client and the gateway that requests meta information about the data being available via the gateway. The gateway answers with a list of all $q \in \mathcal{Q}$.

b) Grant request: Afterwards, clients can demand access to certain information by requesting a corresponding grant (cf. Figure 4): They send a grant request $r_{c,grant}$ to the gateway where the checks described below are performed. If they are successful, the gateway creates and signs a corresponding authorization grant. This is sent back to the client.

Verification Given a request $r_{c,grant}$ of client c, the validity of the client certificate (Equation 1) is checked and whether the requesting client possesses the corresponding private key.[1] Similarly, validity of the request signature (Equation 2) is verified. Then the semantics of the queries are checked (Equation 3) using Φ. The parameters are set as follows: The *query* reflects the demanded data in a form as described above. The *client* is represented by its certificate. The *context* is the current

[1]When using TLS, this is already handled during TLS session establishment.

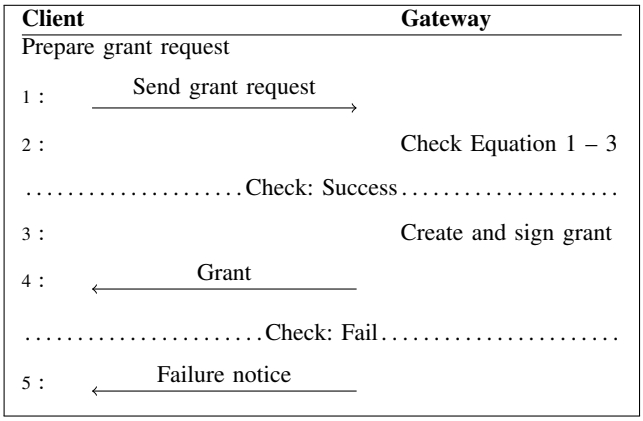

Fig. 4. Grant Request Protocol. In a grant request clients demand an authorization to execute a specified set of queries. The gateway checks, whether or not the requesting client is allowed to access the corresponding data. If yes, an authorization grant is created and handed over to the client.

state of the gateway, this encompasses the information about currently connected peers and environment information like the current time.

$$verify(r_{c,grant}.cert) \quad (1)$$

$$verify(r_{c,grant}.sig_{client}, r_{c,grant}.cert) \quad (2)$$

$$\forall q \in r_{c,grant}.queries : \Phi(q, r_{c,grant}.cert, context) \quad (3)$$

c) Computation request: These requests are performed repeatedly during productive use in order to obtain aggregated data from the peers. The protocol is shown in Figure 5. The client sends a computation request (cf. Figure 3) to the gateway. If the checks as described below are successful, the request is accepted and transformed into an SMC session. The result of this session is the requested information. It is signed by the peers and, since the certificate of the client is available to the peers, encrypted for the client. The encrypted result can then be forwarded by the gateway to the requesting client.

Verification The gateway and the peers play different roles regarding access control and intervenability; hence they validate different aspects of the request.

Given a request $r_{c,comp}$ of client c, the gateway first checks whether the holder matches the requesting client and the client certificate is valid (Equation 4), it checks the authenticity of the request (Equation 5), then it verifies whether the authorization grant itself is valid (Equation 6 – 8).

$$r_{c,comp}.grant.holder = c.cert.fpr \wedge verify(c.cert) \quad (4)$$

$$verify(sig_{client}, c.cert) \quad (5)$$

$$verify(issuer.cert) \wedge verify(sig_{issuer}, issuer.cert) \quad (6)$$

$$r_{c,comp}.grant.not\_before \leq now \quad (7)$$

$$now \leq r_{c,comp}.grant.not\_after \quad (8)$$

After checking formal validity, the validity of the request itself is verified, i.e., whether the grant supports the stated query.

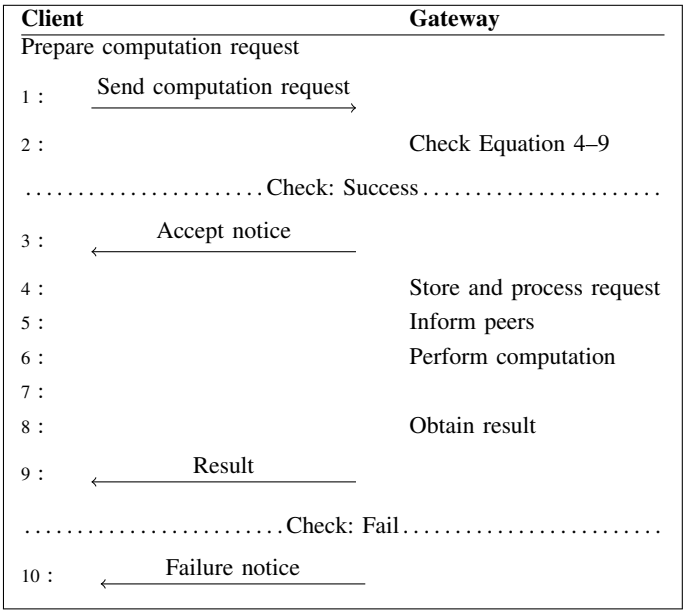

Fig. 5. Computation Request Protocol. A client demands for the results of a computation specified in its request. If the request is valid, the client is notified about acceptance and processing of the request is performed in lines 4 – 6 as described in [32]. The result, coming back in line 8 is then forwarded to the client in line 9.

This is realized by checking for the inclusion of the query in the permitted set (Equation 9).

$$\exists q \in r_{c,comp}.grant.queries : q = r_{c,comp}.query \quad (9)$$

Verification by the peers happens when the session is communicated to them (Step 5 in Figure 5). Besides rechecking abovementioned checks, each peer $p \in P$ can have a set Φ_p of local policies which defines how their data may be used and the corresponding privacy constraints. Satisfaction of these policies can also be checked (Equation 10). This e.g., can include a check for recency of the computation request in order to prevent replay attacks.

$$\Phi_p \vdash (c, r_{c,comp}.query) \quad (10)$$

VI. Evaluation

A. Security and Privacy

In the previous sections we presented an approach to fulfill the security and privacy requirements stated in Section IV. We leave data processing of the underlying SMC intact without modification. I.e., the corresponding security properties still apply without constraints: The state of the art [5], [19]–[21] is already secure against $n - 1$ maliciously colluding peers. Since the data owners' device always participates in the computation when using their data, security of the own raw data is already achieved by guaranteeing that the own device is not compromized. The computation request and the corresponding protocol allow third parties to query for data computed by SMC (R.1). By doing so, securely computed information becomes accessible to outside clients. Access

on the computed results can be controlled by authorization grants required for computation requests (R.2). This enables controlled data flow when serving a heterogeneous set of deployed services. Making the request verification independent from the access control structure Φ and its *context* parameter, i.e., not relying on complex state of the gateway for access control but on transferable documents, allows to use the same data structures to fulfill desired privacy-protection goals: Peers are informed about upcoming computations and their context if they are involved, since the requests include the query and the corresponding authorization grant of the requesting client (R.3). An authenticated history of data access and usage can be built (R.4), since authenticated requests can also be persisted by each peer. The signatures of the client, the gateway and the peer ensure that integrity of request and corresponding grant is protected. Giving peers the ability to verify this request using their own local policies Φ_p and allowing them to veto against requested computations enables intervenability (R.5). As a consequence, data minimization is supported: Data sources can make sure that there are not more or different computations performed than they expect to happen.

Regarding possible attacks from Section II-C, our solution performs as follows: The grant represents given permission. Consequently, A.1 is mitigated since the gateway checks for the presence of a grant, and the peers are also able to perform this check. Without a grant, a request is not accepted. A.2 is addressed by having the access authority as trust anchor. It is assumed to only issue legitimate grants. This is complemented by enabling the peers to perform semantic checks on the requests. Forging permissions (A.3) is prevented by requiring a valid signature of the issuer on the grant (cf. Figure 3). Similarly, an interval of validity is included in the grants to ensure their currency. Furthermore, the possessor of the grant is included in the grant itself, ensuring that only this very client can use the given permission, mitigating A.4. Permission checks performed by the gateway and the peers include all necessary steps to ensure abovementioned assumptions. Additionally, they verify whether the query of the current request is permitted by the attached grant. This renders A.5 infeasible. The request is only forwarded to the peers, if the grant is found valid in the abovementioned sense by the gateway. To avoid requiring trust of the peers in the gateway regarding these checks, all information are also forwarded to the peers. They are able to recheck them themselves. This ensures that data is only made available if the peers had sufficient proof of the validity of the request, mitigating A.6. A.7 is prevented, since the obtained result (cf. Figure 5, step 8) is encrypted for the receiving client. Similarly, the result cannot be changed by the gateway (A.8), since the result is signed by the peers.

B. Performance

For performance evaluation of our approach we implemented a prototype in python and performed measurements of the protocols presented (cf. Figure 4 and 5).

1) Setup:

a) Scenario: We consider a single floor in a smart building with around 10 to 30 peers being connected to a single gateway. The gateway is assumed to be decent commodity hardware. The network is an intranet with low latency, a typical throughput and no packet loss.

b) Hardware and System: We used 4 hosts in our setup. These are equipped with eight cores at 2.50 GHz and a main memory of 15.780 MB. They have 1 Gbit networking interfaces and are arranged in a star topology. The default link latency is around 0.18 ms. As operating system we use Debian Stretch (9.4) based on a 4.9.0 Linux kernel.

The roles of the gateway, the client and the peers have to be reflected in the setup. We deployed the gateway and the client on individual hosts. Furthermore, a single peer was deployed on a dedicated host; all other peers were started as processes on a single further machine.

c) Implementation: The prototype is implemented in python as a flask 1.0.2 application and executed using python 3.5.3. Data is stored in a mongodb version 3.2.11 and as authorization backend authzforce 8.0.1 [35] is used which is located on the gateway host. The flask application is served by uwsgi 2.0.17.1. Uwsgi is executed with a single process and eight threads, if not told otherwise. The queue for unanswered requests has a limit of 100 entries. When contacting the peers, the gateway spawns a thread for each peer in order to allow simultaneous waiting for all responses.

For testing purposes no actual SMC component was connected to our querying framework. This allows to measure the overhead of our components without depending on the performance characteristics of a chosen SMC implementation. We decided for this omission, since we assessed the performance of a state of the art SMC implementation elsewhere [36].

d) Method: We measured the duration of client requests handled by the gateway (*latency*). The measured time begins when the request is handed to our custom code; it ends when the final response is handed back to uwsgi. In the results, we show the median, the 0.25 and the 0.75 quantile. For each frequency, we captured the amount of requests the gateway was able to handle successfully per second (*throughput*). Lastly, we recorded the state of the request queue.

2) Results:

a) Grant Request Protocol: The grant request protocol is carried out when a client aims to obtain further access rights to data queries offered by the gateway. This only happens when new clients are deployed or permissions change over time. We assume that a small number of requests per second is only exceeded during peak times. This amount of requests is well supported: Even under a load of ≥ 100 requests per second, answer time stays below 20 ms (Figure 6). The queue of the gateway becomes saturated only after 170 requests/s. Correspondingly, the throughput stagnates at the same point (Figure 7). Since no peer interaction happens, performance is independent of their number.

b) Computation Request Protocol: The computation request protocol is always carried out when a computation on actual sensor data is queried. With polling every second per

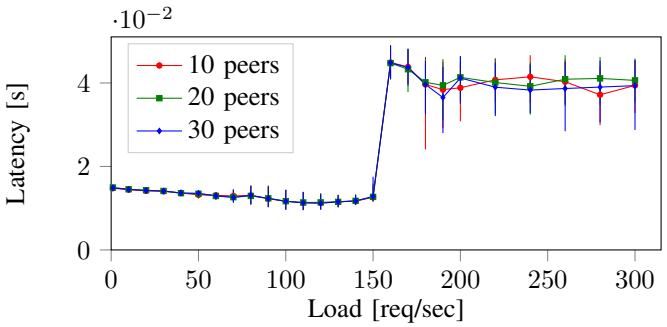

Fig. 6. Grant Request Protocol: The duration of handling a single request inside the gateway component depending on the amount of requests performed by the client.

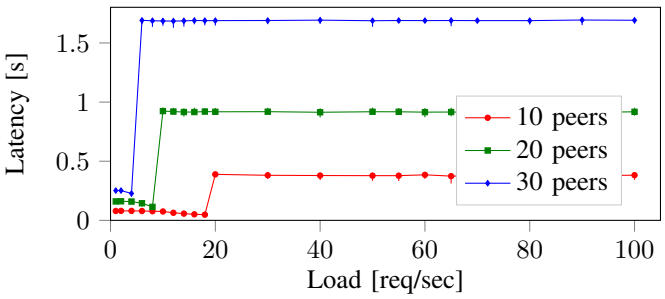

Fig. 8. Computation Request Protocol: The duration of handling a single request inside the gateway component depending on the amount of requests performed by the client. This includes forwarding the request to all concerned peers and waiting for their request acceptance.

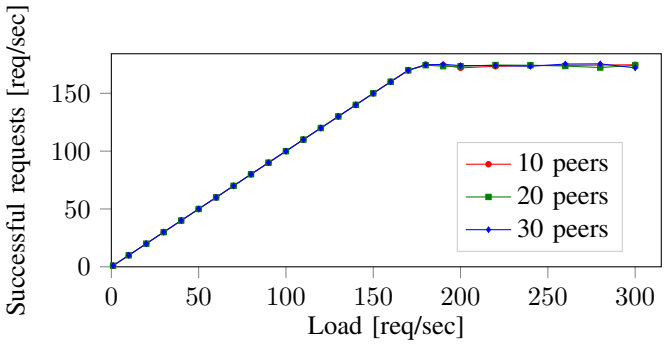

Fig. 7. Grant Request Protocol: The amount of successfully answered requests depending on the amount of requests performed by the client. Up to a load of 170 requests/s, throughput increases proportionally and no drops occur. Afterwards, the queue is filled and throughput stagnates on this level.

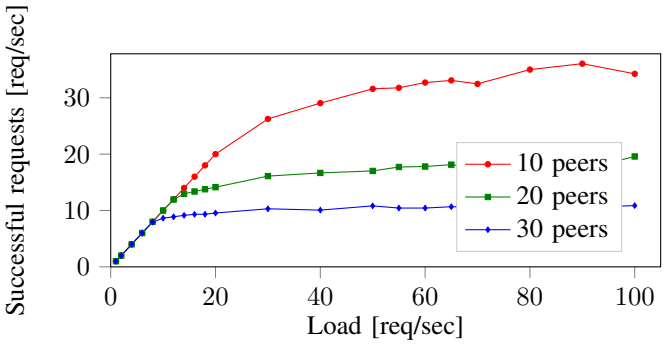

Fig. 9. Computation Request Protocol: The amount of successfully answered requests depending on the amount of requests performed by the client. Latency of single requests restricts the amount of successful requests, since the request queue is already filled between 5 – 10 requests/s.

client, and multiple clients being connected, multiple requests per second can be expected. With 30 peers connected, a single request per second yields a latency of ~250ms. With increasing load this converges to ~1.7 seconds per request (Figure 8). Each added peer approximately contributes further 50 ms. The reason is computational overhead per connection – mainly signing outgoing messages and verifying the signatures of incoming messages – which cannot be handled in parallel due to the global interpreter lock in python. A programming language with real parallelization would not exhibit a delay in such a fashion. Concomitant with the increase in latency, the request queue is exhausted between 5 – 20 requests/s and a high throughput is inhibited (Figure 9). Providing 4 uwsgi processes shows that parallelization doubles the throughput. Due to these limitations, we understand our approach to be a feasibility result with further potential for optimizations.

VII. Summary

In the Internet of Things and smart environments, services need data about the environment and its inhabitants for performing informed action. This data is personal data and, hence, privacy critical. Current systems mostly handle this data using a middleware for storage and processing. Gained results are forwarded to services and applications. However, this centralization enables several privacy threats.

A promising alternative is Secure Multiparty Computation (SMC). Data can remain on individual distributed sensor platforms. Desired results can be derived on-the-fly by secure computation without intermediate storage. Here, we provide an approach which enables third party clients to execute queries on privacy-sensitive data built on SMC without taking part in the computations themselves. On this foundation, we build fully privacy-preserving service: Protection of results is achieved by access control on the level of individual queries. Requests and processing becomes fully transparent for the cooperating sensor platforms. This supports intervenability, i.e., enabling them to stay in control of their data and allowing them to reject processing if their privacy requirements are not fulfilled by an incoming data requests.

Future Work: In combination with [32], we realized the full architecture of a privacy-preserving service for smart environments. However, several further challenges are open: 1) We enabled checking of requests by peers using local policies Φ_p. The design of these semantic checks could be further investigated. 2) Clients are only allowed to select predefined predicates. If arbitrary predicates could be checked against privacy rules, clients could be allowed to define predicates themselves. 3) The gateway could create its access control structure Φ by collecting and merging the distributed set of Φ_p, directly guaranteeing compatibility between them.

REFERENCES

[1] A. Ridi, C. Gisler, and J. Hennebert, "A survey on intrusive load monitoring for appliance recognition," *Proceedings - International Conference on Pattern Recognition*, pp. 3702–3707, 2014.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, pp. 30–39, 2017.

[4] 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.

[5] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Lecture Notes in Computer Science*, vol. 7417, 2012, pp. 643–662.

[6] M. Rost and A. Pfitzmann, "Datenschutz-Schutzziele revisited," *Datenschutz und Datensicherheit - DuD*, vol. 33, no. 6, pp. 353–358, 2009. [Online]. Available: http://link.springer.com/10.1007/s11623-009-0072-9

[7] A. Pfitzmann and M. Hansen, "A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management," *Identity*, pp. 1–98, 2010.

[8] M. Hansen, M. Jensen, and M. Rost, "Protection Goals for Privacy Engineering," in *2015 IEEE Security and Privacy Workshops*, 2015, pp. 159–166.

[9] M. Rost, "Die Ordnung der Schutzziele," *Datenschutz und Datensicherheit - DuD*, vol. 1, pp. 13–17, 2018.

[10] "Das Standard-Datenschutzmodell," Konferenz der unabhängigen Datenschutzbehörden des Bundes und der Länder, Darmstadt, Tech. Rep., 2015. [Online]. Available: https://www.datenschutzzentrum.de/uploads/sdm/SDM-Handbuch.pdf

[11] White House Report, "Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy," The White House, Washington, DC, USA, Tech. Rep., 2012. [Online]. Available: https://obamawhitehouse.archives.gov/sites/default/files/privacy-final.pdf

[12] European Union Agency for Network and Information Security, *Privacy and Data Protection by Design from policy to engineering*, 2014, no. December.

[13] R. Cramer, I. B. Damgard, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. New York, NY, USA: Cambridge University Press, 2015.

[14] 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.

[15] ——, "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.

[16] D. Chaum, I. B. Damgård, and J. van de Graaf, "Multiparty Computations Ensuring Privacy of Each Party's Input and Correctness of the Result," in *Advances in Cryptology*, C. Pomerance, Ed. Berlin Heidelberg: Springer-Verlag, 1987, pp. 87–119.

[17] O. Goldreich, S. Micali, and A. Wigderson, "How to play ANY mental game," in *Proceedings of the Nineteenth Annual ACM Conference on Theory of Computing – STOC '87*. New York, NY, USA: ACM, 1987, pp. 218–229.

[18] D. W. Archer, D. Bogdanov, B. Pinkas, and P. Pullonen, "Maturity and Performance of Programmable Secure Computation," *IEEE Security and Privacy*, vol. 14, no. 5, pp. 48–56, 2016.

[19] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority - Or: Breaking the SPDZ limits," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8134 LNCS, 2013, pp. 1–18.

[20] M. Keller, E. Orsini, and P. Scholl, "MASCOT," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 830–842.

[21] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ great again," *Lecture Notes in Computer Science*, vol. 10822 LNCS, pp. 158–189, 2018.

[22] "A FRamework for Efficient Secure COmputation," 2018. [Online]. Available: https://github.com/aicis/fresco

[23] "BristolMPC - SPDZ," 2018. [Online]. Available: https://github.com/bristolcrypto/SPDZ-2

[24] "SCALE-MAMBA," 2018. [Online]. Available: https://github.com/KULeuven-COSIC/SCALE-MAMBA

[25] 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.

[26] 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.

[27] 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.

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

[29] 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*, 2010.

[30] 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.

[31] 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." in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, vol. 2017, 2017, pp. 1175–1191.

[32] M. von Maltitz, S. Smarzly, H. Kinkelin, and G. Carle, "A management framework for secure multiparty computation in dynamic environments," in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. Taipei, Taiwan: IEEE, apr 2018, pp. 1–7.

[33] M. von Maltitz and G. Carle, "Leveraging Secure Multiparty Computation in the Internet of Things," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys '18*. New York, New York, USA: ACM Press, 2018, pp. 508–510. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3210240.3223569

[34] A. Shamir, "How To Share a Secret," *Communications of the ACM (CACM)*, vol. 22, no. 11, pp. 612–613, 1979.

[35] "AuthZForce," 2018. [Online]. Available: https://authzforce.ow2.org/bin/view/Main/

[36] M. von Maltitz and G. Carle, "A Performance and Resource Consumption Assessment of Secret Sharing based Secure Multiparty Computation," in *Data Privacy Management*, J. Garcia-Alfaro, J. Herrera-Joancomarti, G. Livraga, and R. Rios, Eds. Cham: Springer International Publishing, 2018, pp. 357–372.