Hook-In Privacy Capabilities for gRPC

Jiaao Li, Riccardo Marin, Timo Ramsdorf
TU Berlin

June 3, 2022

Abstract

gRPC is a RPC framework and it is considered a modern solution for building microservice applications. It has been adopted by several tech companies in different domains for connecting their backend services while maintaining speed and scalability.
In this technical report, first we start with the related privacy principles, then collect and organize the latest scientific and technical literature on applications and techniques related to privacy in gRPC and other similar microservices contexts. In the process, we discovered that among the existing theoretical research and technical results, there is not only a lack of a set of gRPC-native privacy capabilities, but also in fact a lack of gRPC-based frameworks that focus on privacy-preserving capabilities.
Thus, we researched and assessed current technologies in this regard, following the privacy principles of data minimization, purpose limitation and data security, and proposed a pluggable and reusable component. Such component offers several privacy capabilities and makes it easier for the data controllers to meet regulatory requirements.

1 Introduction

1.1 Background

The principles of data minimization, purpose limitation and data security, which aim to protect the fundamental human right to personal data protection and privacy, are at the heart of privacy norms such as the GDPR [1].
Conceptually, they require in three areas the following:

- Personal data should be limited to what is necessary in relation to the purpose for which it is to be processed, and the form and periodicity of its retention should be regulated and limited as necessary. The collection, processing and use of personal data shall only be permitted if it is actually necessary for the fulfillment of the purpose. Otherwise, sensitive data shall be presented by means of deleted or obscured data fields through noising or generalization.
- Data shall be collected for specific and explicit legitimate purposes. It shall not be further processed in a manner that is incompatible with those purposes. Processing for purposes other than those for which the personal data was collected requires the consent of the data subject.
- Personal data should be protected, including through the use of appropriate technical or organizational measures, against unauthorized or unlawful processing, as well as against accidental loss and destruction. Encryption, hashing, etc. should be used where necessary to ensure a level of security commensurate with the risk.

However, there are still more gaps in the relevant legislation as to the exact practical actions to be taken. In reality, personal data is constantly generated in the course of daily and business activities. The enormous amount of user data stored in service providers' back-end databases is not only private information that deserves to be protected, but there is also the need to limit the extraction, analysis and use of this user's personal data.
Not only that, but with the large number of APIs, ports and components exposed, these issues put users at significantly increased risk of having their private information exposed.
Data security is indeed another challenge for microservices. In particular, gRPC deployments in cloud environments take the complexity of identity and access control to a new level.
In this analysis, we first present the legal ground related to privacy protection, then critically evaluate strength and weaknesses of the state of art and related works to our goal. Along the assessment of the viability of other technical solutions, we also provide a rough definition of our proposal. The discussed implementation aims at filling the current gap in terms of a comprehensive gRPC framework that could provide access control policies and privacy protection features such as: erasure and obfuscation of data fields, encryption and hashing of sensitive data, etc.

1.2 gRPC

gRPC is an open source RPC (Remote Procedure Call) framework. The idea behind RPC is that code running on a client can call methods on a remote machine, making distributed application systems possible [2].
The interchange message format can be freely chosen. By default it uses *Protocol Buffers*, a serialization mechanism developed by Google. Protocol buffers can handle structured data in a language-neutral way. The structure is defined in a *IDL (Interface Description Language)*, a key-value format defined in a *proto* file (with extension `.proto`). It's a format independent from any language and this is a simple example:

```
message HelloRequest {
string greeting = 1;
}

message HelloResponse {
string reply = 1;
}

service HelloService {
rpc SayHello(HelloRequest) returns (HelloResponse);
}
```

The example shows that a .proto-file defines (among other possible elements) messages and services as its most important elements. A service describes a function that a server implements and that can be called by a client. A service function is defined by a type of message that it expects as an argument and a type of message that it returns. It is important to mention that a defined message is not necessarily a message in the true sense of the word, but something that is transmitted as a complete packet. A message can also be part of other message definitions, so can be considered as the defined structure of a data container.
Messages consist of one or more fields that have a type, a name and an id. Even if the syntax of proto-files makes it look like the id (the 1 in the example above) is a value, it is indeed the key in the key-value pairs that are serialized later. The defined name (`greeting` in `HelloRequest` and reply in `HelloResponse`) does not occur in the serialized bytestreams. It is only used to define names of the generated method stubs. [3].

Once you define the service, you can use *protoc* (the Protocol Buffers compiler) to generate the method stubs for client or server side. *protoc* takes a proto file as input and generates code in a specified language. The code contains classes, setters, getters and serialization methods from/to protocol buffers messages represented in raw bytes.
gRPC API may work in both synchronous and asynchronous mode, with different implementations according to the chosen programming language: unary RPC, server streaming RPC, client streaming RPC, bidirectional streaming RPC. The first one is similar to REST, it's based on a simple request/response architecture. Moreover, message streaming is also supported and can be configured in a bidirectional way.
For the purpose of safe use of gRPC and communication with other systems, gRPC is designed to work with a variety of authentication mechanisms, developers can use SSL/TLS with Google Token based authentication or can plug in a customized authentication system. gRPC also provides an authentication API that allows it to provide all the necessary authentication information.

1.3 Privacy law and materialization

Since 25th May 2018 companies dealing with EU citizens personal data have to comply with the GDPR regulations [1]. The abstract privacy concept has been translated to a list of principles defined by European privacy law.
The key element is personal data - anything that relates to an identifiable natural person - that belongs to a data subject and is collected and processed by a controller. The controller is accountable for managing the data and may delegate the processing to another party called processor.
The principles to consider are the following: lawfulness, purpose limitation, data minimization, transparency, accuracy, security, accountability, data portability, enforcement. Implementing these principles into information systems means designing technical solutions that can: proof that informed consent has been collected, limit access through access-control policies, delete and anonymize data, keep track of third parties, and transmit data in a machine-readable format. Some points of reference recognized by privacy researchers are the contribution of Jaap-Henk Hoepman regarding the privacy design patterns and strategies [4] and the foundational principles of Ann Cavoukian [5].

From a technical point of view, information minimization and anonymization can be implemented in a variety of ways: *data erasure* is the complete removal of certain attributes from a data point or database, which usually refers to some private data. *Generalization*, on the other hand, reduces the level of detail that an attribute contains. In our use case, for example, replacing a physical address with a distance. *Hashing* here refers to replacing a value with the result of a hash function, which is difficult to reverse and maps in a unique way data to a fixed-size value. *Data noisification* is the transformation of otherwise accurate data, into more granular information, by adding artificially created false values. *Noising* (sometimes also referred to as perturbation) herein refers to when data in non-aggregated form is required, the level of noise depends on the sensitivity of this information by increasing or decreasing the value according to a typical probability distribution. *pseudonymization* means replacing a field of personally identifiable information in a data record by one or more artificial identifiers or pseudonyms.

Without proper tools and frameworks data controllers are more prone to commit mistakes that violate this regulatory law. Our project aims to offer a simple and configurable tool that includes privacy primitives towards GDPR compliance.

2 State of the art

2.1 Discussion

In the last few years there has been an increasing popularity of microservices in the design of information systems. Monolithic systems may be faster to develop in an initial phase, but managing the codebase becomes progressively more complicated. The convenience of microservices comes from modularity, maintainability, testability and speed of deployment.
However, when choosing microservices, privacy and security aspects should be considered even more carefully. The compromise of one service may lead to security issues extended to the entire system. If data management policies are not defined, the exposure of one service could leak much more data than the strictly necessary for the functioning of the single service. Despite being a field where there is still plenty of room for research, the topic of privacy capabilities in microservices architectures have been addressed by several authors.

2.2 Technologies

The study *"Towards Application-Layer Purpose-Based Access Control"* [6] presents an architecturally new approach to implementing purpose-based access control in practice. It consists in a development-friendly framework for implementing PBAC at the application level through native integration with an object-relational mapper. It allows better integration with established architectures and practices of real-world application engineering and for database independence. They follow an approach that elevates their respective functionality to the application layer, which greatly simplifies the use of their system in practical implementations, allows fairly good integration with established implementation practices, and provides flexibility regarding the databases to be used. In particular, the application layer approach is consistent with the established practice of implementing access control mechanisms mentioned above and allows developers to retain the low-level abstractions they are used to. Regarding the viability of this approach, we can consider the proposed idea applicable to web applications with an acceptable performance overhead. The integration with ORM is not applicable to the gRPC case, however, the use of standard, developer-friendly configuration files for defining the purposes can be a reference point.

One paper that is facing a similar problem like this project is *"Per-Query Data Minimization for Privacy-Compliant Web APIs"* [7]. It also provides a hook-in privacy framework for interfaces of distributed architectures, which applies different privacy features to configured fields of the interface depending on the authorized user and role of the requesting client. They decided to realize the privacy capabilities within the server architecture, so that the behavior is fully transparent from the client-perspective.
The paper defines several requirements for a hook-in privacy solution: for example such a framework should support privacy on attribute-level, should be configurable and therefore reusable and should be usable with low integration and performance overhead. Since these requirements match ours quite well, it may be helpful to reuse some principles, which have shown themselves to be suitable. For example we may also use the principle of extracting the privacy features as directives separated from the part that controls the application of these directives. On the other hand the main difference is probably that our project has a more generic environment, i.e. we can not assume a web server existing and the gRPC server may sometimes not even have a database. The authentication mechanism works differently as well.

When it comes to specific gRPC environments, the amount of scientific literature is more limited. A framework for tracking of data transferred across subject, controller and processor

was proposed in the report *"Framework for Data Tracking across Data Controllers and Processors"* [8]. It produces graphs that trace the data flow outgoing from the data subject. A version control system for the data is included and it is based on interceptor modules stacked on the application layer of the services. The important advantage is that the user is being provided a tool to track the usage of its personal data. This means an increased awareness of what and where the data flowed and it may be beneficial also from a data controller perspective. Indeed that system works in the direction of transparency and accountability. The paper claims that such a framework is a "plug-and-play solution", whereas it is not clear how the integration with an existing system would work.
Furthermore, the evaluation of the impact on the performance is missing. Nevertheless, the research and the developed prototype bring some progress in a still relatively unexplored field.

The topic of data noising in microservices has been faced by the paper *"Privacy-Preserving Microservices in Industrial Internet of Things Driven Smart Applications"* [9]. In the context of distributed machine learning the authors researched if it was possible to apply differential privacy to the dataset and without affecting heavily the results produced by the machine learning models. gRPC was the chosen framework for the proposed distributed architecture. The research project ended up implementing a microservice layer for privacy preservation. The layer is based on perturbation of the weights and the center of an unsupervised algorithm along with the laplacian noise applied to the data points. In spite of having a different scope from ours, the authors provided a significant proof of concept of the integration of privacy functionalities in distributed settings. Similarly to the data tracking study [8], the effect of the privacy layer on the performance is not discussed and details about the implementation are not provided. The latter is a particularly remarkable aspect, as the generality and efficiency - and more importantly the right balance between them - are the limitations of all hook-in privacy components.

As the access to sensitive data is verified as permitted at runtime, this can be addressed by preventing unwanted access to the data. There is a paper, *"Implementing Data Flow Assertions in gRPC and Protobufs – Final Report"* [10], which propose to automatically generate and embed such data flow assertions into gRPC. Data flow assertions allow organizations to secure data even as their architecture evolves or degrades with minimal changes to their codebase. One of their main contributions is a Go library that allows gRPC, protocol buffers and other packages to perform runtime assertions about messages within and between service boundaries. They modified gRPC and protocol buffers in certain places to ensure the messages generated and transmitted by these frameworks is protected by the privacy library.

To sum up, the above-mentioned solutions contribute to the privacy aspects and related law requirements. However, we reckon that a comprehensive framework for the gRPC technology is missing. The scientific literature already faced the topic of access control in gRPC. Nonetheless, we could not find a project that supports privacy primitives related to data minimization together with access control, in a viable and re-usable technical fashion. Another clear limit of many implementations on this topic is that they are often language-dependent. Ideally, a language-agnostic alternative is more inclined to diffusion and does not require the maintenance of several codebases in the 11 different languages that support gRPC. For this reason we investigated different approaches and eventually proposed a new technical solution.

3 Implementation

3.1 Use case

In this context, we considered use cases that involve microservices backend systems based on gRPC and involving flow of personal data. The chosen use case is a food delivery app. Such a system includes several challenges. A broad range of types of personal data are exchanged between parties, including the position and need to comply with data protection regulations. We supposed the existence of the following microservices:

- Order service, handles customer orders reception and communicates with the microservices in order to process and deliver the order
- Driver service, manages the drivers' list and interacts with other services for meal delivery and collection
- Restaurant service, receives the orders, provides driver identity check and meal collection confirmation
- Routing service, handles the calculation of the route and the selection of the driver for the requested delivery

The general architecture of the system is shown in figure 1.
In our use case we are going to address the following categories of personal data:

- Customer name
- Customer favorite addresses
- Customer current position
- Customer orders history
- Meal
- Driver position
- Driver name

For each service and field of personal data, we determined whether a privacy capability is necessary and which one would best fit the use case. The result is summarized in table 1.

Figure 2 contains a sequence diagram representing interaction between components in our use case. At each point of interaction between microservices our tool will apply a specific privacy technique to the transmitted data.

3.2 Functional requirements

We expect to implement the following data minimization techniques:

- Erasure of data field
- Generalization
- Noising
- Hashing

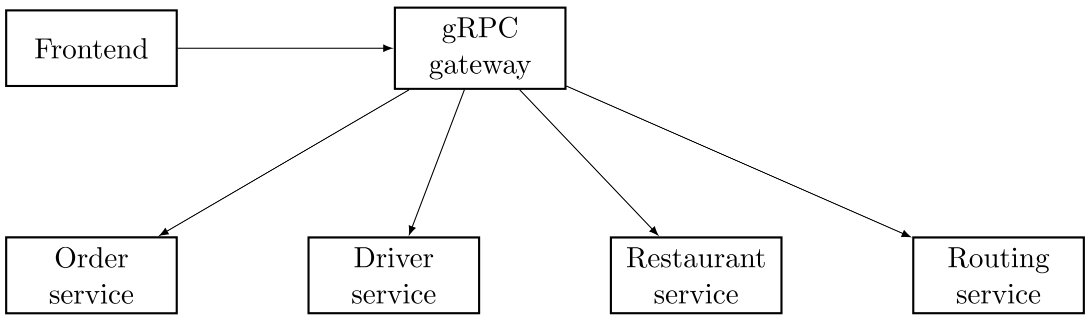

Figure 1: System architecture

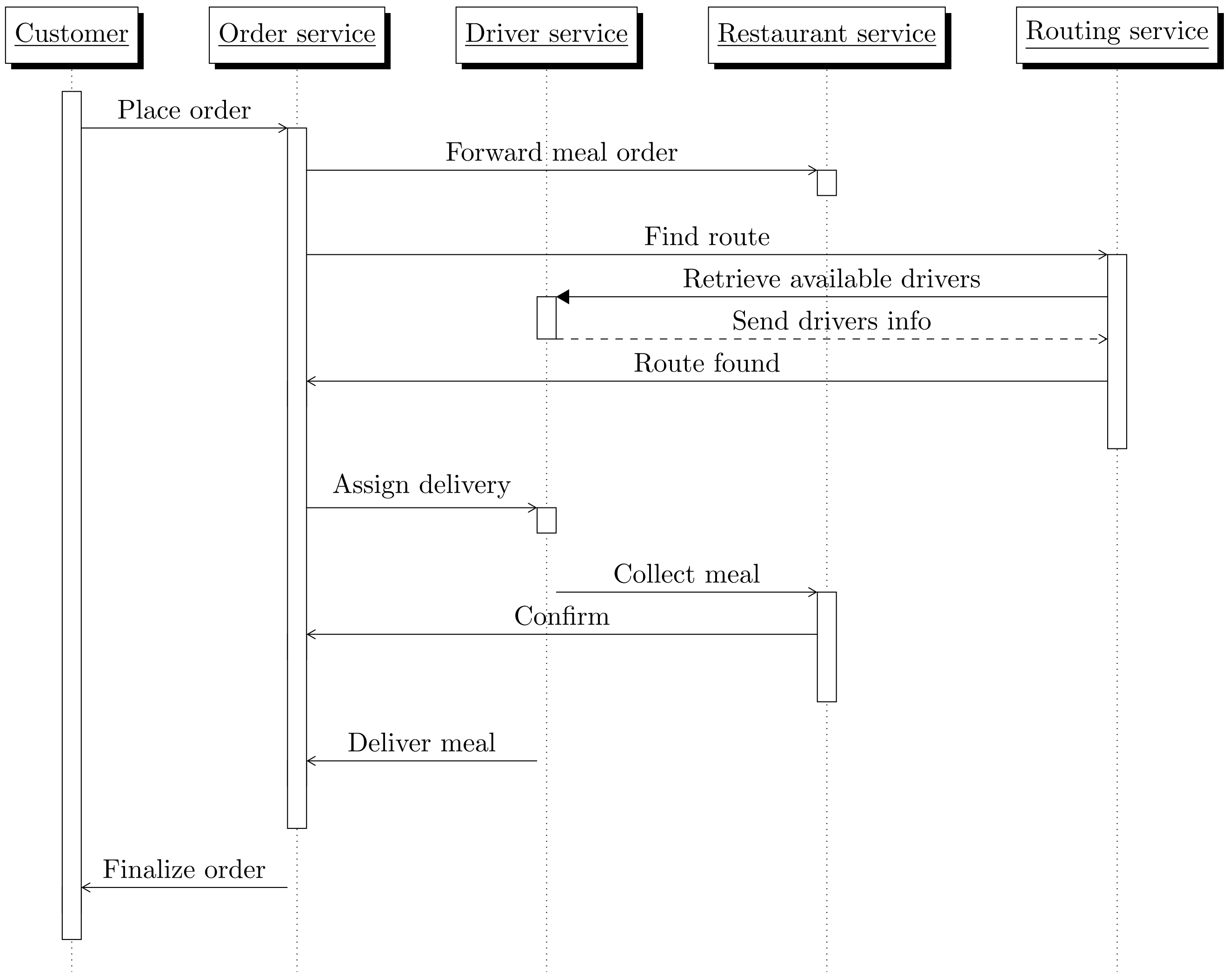

Figure 2: Sequence diagram of the system

Service	**Personal data**	**Privacy tools**
Order service	Name	
	Favorite addresses	
	Current position	
	Orders history	
Driver service	Meal	Data erasure (he doesn't know about the meal)
	Customer data (except delivery address)	Data erasure (no info about customer except for address)
	Delivery address	
Restaurant service	Customer data (except delivery address)	Data erasure (no info about customer except for address)
	Delivery address	Generalization (convert to distance)
	Meal	
	Driver name	Hashing (restaurant can see only a pseudonymized ID)
Routing service	Driver position	Noising (return an approximate position)

Table 1: Summary of data accessed by services and related data minimization measures

The services are linked to one or more purposes:

Order service $\rightarrow \{Meal\ purchase\}$
Driver service $\rightarrow \{Meal\ delivery\}$
Restaurant service $\rightarrow \{Meal\ cooking, Meal\ collection\}$
Routing service $\rightarrow \{Route\ computation\}$

Purpose	**Personal data**
Meal purchase	Name
	Favorite addresses
	Current position
	Orders history
Meal delivery	Delivery address
Meal cooking	Meal
	Distance
Meal collection	Driver id
Route computation	Driver position

Table 2: Purpose based access control and linked data

Regarding access control schemes, we are going to support purpose-based access control.
In this way, instead of directly requesting access to a set of information, every service is linked to purposes. Purposes can be configured to include only strictly necessary data for fulfilling that goal. An overview for the chosen use case is highlighted in table 2.

In general these functions will be used in an open interface, which supports custom data processing functions and unique combinations of the above mentioned techniques.
We intend to design a technical solution that can be configured by anyone without any specific knowledge of the middleware.

3.3 Non functional requirements

Along the core features of the system, we identified 3 non functional requirements: performance, ease of use and reusability.
First of all, we are addressing a well-known part of the trade-off between privacy and speed in information systems: performance. We should benchmark the system with the privacy

component and we expect the system to still have an acceptable speed.
Based on our related work analysis in section 2, we reckon that performance is the main challenge of our project.
Secondly, in order to not limit the potential use of our system, we shall care about ease of use. The proposed frameworks should be well-documented, easy to install and configure, without specific prerequisites for the programmer. Along this feature, we aim also at flexibility: the component should ideally be language-agnostic, it can be used in every application, without depending on a specific environment. As we recognize the importance of creating a component that can be customized, we plan to develop an extendable system. Namely, for the application developer it should be possible to include its own custom privacy features, completely new or combined with pre-existing capabilities.

3.4 Component

Developing a component that affects data flow between client and server leads to technical choices such as: middleware, proxy service.
The choice between these solutions is a trade-off between speed and reusability that needs to be evaluated.
Therefore we implement and benchmark 3 different technical components: interceptors, classical proxy and binary stream proxy.
Interceptors are a gRPC native component that allows a proto message before it is sent or received by the client or the server. In that case we can proceed with a add-on that adds the interceptors logic to the pre-existing application. A relevant drawback of choosing an interceptor is that it should be written in the same language of the IDL. As a consequence, it is very unlikely to obtain a general, universally valid system: for each language we shall implement the respective interface. On the other hand, by using this component we expect the best results speed-wise.
In comparison to the above mentioned solution, a classic proxy has more flexibility, as it can be used along any platform or language. However, since this means an additional step of serialization and deserialization, we can expect a relevant increase in the latency.
Alternatively, it is possible to work at a lower level, using a binary stream proxy. It consists of an intermediate component placed between the services. The connection is based on web sockets (i.e. it is working directly above the TCP layer). The proxy receives the binary stream from one end of the communication, decodes only some of the bytes, does some processing to the decoded data and re-encodes it before transmitting the patched data to the final node. Due to less workload in terms of serialization and deserialization of the flowing data, the expected impact on performance should be lower than the classical proxy.

3.5 Benchmark

As already pointed out in the section 3.3 ”Non functional requirements” it is one important goal, that the resulting framework should be performant, i.e. the use of the framework should cause as little effects on throughput and latency of the gRPC connection as possible. It is necessary to evaluate whether this goal is achieved and to quantify the effects on throughput and latency, so there is a need to build some benchmark setups.
There are several approaches of architectures for the hook-in Privacy framework like different proxy technologies and interceptors. At the current state of the art we already implemented benchmarks for the latency of different proxies. We compared the direct gRPC connection between a client and a server hosted on the same system, a proxy working on gRPC level and a proxy working on socket/byte stream level. However, we are not sure how to interpret

the first (unexpected) results, so we have to point out that this is just a starting point at the moment and needs further investigation during the project.

References

[1] *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)*. https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[2] *gRPC documentation*. https://grpc.io/docs.

[3] Google. *Protocol Buffers documentation*. https://developers.google.com/protocol-buffers/docs/.

[4] J. Hoepman. *Privacy Design Strategies (The Little Blue Book)*. https://www.cs.ru.nl/~jhh/publications/pds-booklet.pdf.

[5] A. Cavoukian. *Privacy by Design - The 7 Foundational Principles*. https://www.ipc.on.ca/wp-content/uploads/resources/7foundationalprinciples.pdf, 2009.

[6] S. Tai T. Peikert M. Reppenhagen D. Wenzel P. Wille K. Wolf F. Pallas, M. Ulbricht. *Towards Application-Layer Purpose-Based Access Controls*. https://www.ise.tu-berlin.de/fileadmin/fg308/publications/2020/ACM_SAC_Application_Layer_PBAC_preprint.pdf.

[7] P. Heinrich J. Kipke E. Grünewald F. Pallas, D. Hartmann. *Configurable Per-Query Data Minimization for Privacy-Compliant Web APIs*. https://arxiv.org/pdf/2203.09903.pdf, 2022.

[8] A. Yu Z. Lai, Y. Xin. *Framework for Data Tracking across Data Controllers and Processors*. http://cs.brown.edu/courses/csci2390/2020/assign/project/report/2020/gdpr-tracking-framework.pdf.

[9] N. Moustafa M. S. Rahman N. Bugshan, I. Khalil. *Privacy-Preserving Microservices in Industrial Internet of Things Driven Smart Applications*. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9492287, 2021.

[10] J. Weisskoff A. Mahajan, Y. Xue. *Implementing Data Flow Assertions in gRPC and Protobufs – Final Report*. http://cs.brown.edu/courses/csci2390/2020/assign/project/report/2020/grpc-df-asserts.pdf, 2020.