Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 37 (2014) 370 – 375

Procedia Computer Science

The 6th International Symposium on Applications of Ad hoc and Sensor Networks (AASNET'14)

# SCSlib: Transparently Accessing Protected Sensor Data in the Cloud

Martin Henze*, Sebastian Bereda, René Hummen, Klaus Wehrle

*Communication and Distributed Systems, RWTH Aachen University, Ahornstr. 55, 52074 Aachen, Germany*

**Abstract**

As sensor networks get increasingly deployed in real-world scenarios such as home and industrial automation, there is a similarly growing demand in analyzing, consolidating, and storing the data collected by these networks. The dynamic, on-demand resources offered by today's cloud computing environments promise to satisfy this demand. However, prevalent security concerns still hinder the integration of sensor networks and cloud computing. In this paper, we show how recent progress in standardization can provide the basis for protecting data from diverse sensor devices when outsourcing data processing and storage to the cloud. To this end, we present our Sensor Cloud Security Library (SCSlib) that enables cloud service developers to transparently access cryptographically protected sensor data in the cloud. SCSlib specifically allows domain specialists who are not security experts to build secure cloud services. Our evaluation proves the feasibility and applicability of SCSlib for commodity cloud computing environments.

Peer-review under responsibility of the Program Chairs of EUSPN-2014 and ICTH 2014.
*Keywords:* Sensor Networks ; Cloud Computing ; Security ; Library

## 1. Introduction

Sensor networks, which sense information about the environment, and cloud computing, which realizes elastic storage and computing, attract a lot of attention from both industry and academia. Their combination, referred to as *sensor cloud*, allows to enhance the potential application areas of sensor networks with the strengths of cloud computing. By utilizing the cloud, sensor data can be collected, processed, and stored at large scales as well as shared world wide. Thanks to the cloud computing paradigm, the required storage and processing resources can elastically and automatically be scaled on-demand with a pay-as-you-go billing model. We consider a scenario, in which owners of sensor networks (i.e., private end users, companies, or public institutions) connect their sensor networks to the cloud. There, services selected by the owner of the sensor network operate on the outsourced sensor data [1–3].

As sensor data often contains sensitive information, a major challenge when interconnecting sensor networks with the cloud are security concerns. Importantly, traditional transport security mechanisms between data sources and the cloud do not suffice to protect sensor data in an end-to-end manner as such channel security is typically terminated at the entry point to the cloud. In contrast, *object-level* security between data sources and cloud services affords the

* Corresponding author. Tel.: +49-241-80-21425 ; fax: +49-241-80-22222.
*E-mail address:* henze@comsys.rwth-aachen.de


Peer-review under responsibility of the Program Chairs of EUSPN-2014 and ICTH 2014.
doi:10.1016/j.procs.2014.08.055

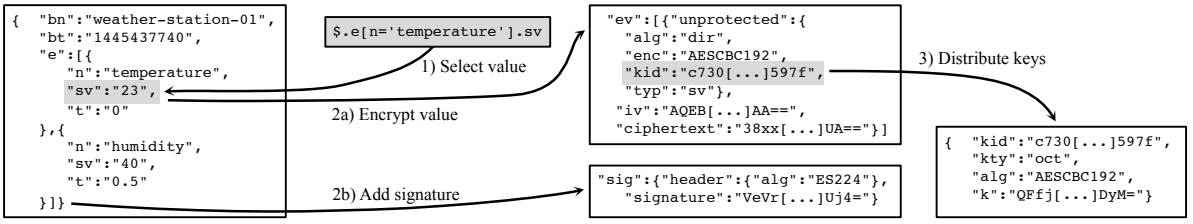



Fig. 1: Overview of the process of protecting a sensor data item for the cloud and distributing the data protection keys.

required end-to-end protection of outsourced sensor data. However, applying object security in the context of sensor clouds has two inherent challenges. First, sensor data can originate from a wide variety of sensor nodes and thus can be structured arbitrarily. This significantly complicates the application of object security mechanisms. Most notably, cloud services have to be informed how data has been protected to successfully decrypt and verify the integrity and the authenticity of the received data. Second, object security operates at the application level. Hence, contrary to transport security, object security is *not* a transparent security mechanism for cloud services. However, implementing the necessary security mechanisms is a laborious and error-prone task. It is our strong believe that developers of cloud service should not need to be responsible for realizing security functionality as they often are not security experts.

To overcome these inherent challenges, we present the following two contributions: i) a widely-applicable, standards-based approach to represent and protect sensor data in the cloud (Sec. 3), and ii) the design of our generic Sensor Cloud Security Library, SCSlib (Sec. 4). We specifically develop SCSlib for cloud service developers to transparently access sensor data without the need to care for security. To motivate our work, we discuss related work in Sec. 2. In Sec. 5, we evaluate the performance of SCSlib and show its general feasibility, before we conclude this paper in Sec. 6.

## 2. Related Work

Early approaches in the area of combining sensor networks or the Internet of Things with cloud computing, especially in the area of health care and ambient assisted living, identified security as a crucial corner stone[4,5]. However, these initial solutions provided only rudimentary security measures, e.g., transport security or authentication using passwords[5]. We proposed a comprehensive security architecture for sensor data in the cloud in earlier work, enabling end-to-end security from the sensor network to an authorized cloud service[1,3,6,7]. Our initial approach did not consider flexible configuration of security mechanisms and a transparent access to protected sensor data for cloud services. However, flexible configuration of security mechanisms is required to support different application scenarios, while transparent data access allows also non-security experts to develop cloud services.

In order to enable flexible configuration of security mechanisms, Itani and Kayssi proposed SPECSA[8]. Their policy format allows to specify which parts of a message have to be protected. However, they assume messages with a fixed structure and use the same encryption key for all parts of the same security level. Hence, their approach is not suitable for the sensor cloud scenario, as this scenario requires fine-grained, flexible access control.

Several approaches for supporting the developer by abstracting from security paradigms have been proposed. GSS-API by Linn[9] provides security services in order to protect the communication between two entities. However, in our scenario we require security at the granularity of objects instead of communication channels in order to protect sensor data also during storage. JSAL by Huang et al.[10] is a security aspect library. In their approach, the developer has to apply the security measures manually, which requires expertise in the area of security.

To conclude, there is a need for a common way to represent protected sensor data and a security abstraction layer for accessing protected sensor data in the cloud that allows non-security experts to develop secure cloud services.

## 3. Protecting Sensor Data for the Cloud

Depending on the sensor device, sensed information varies considerably regarding its structure (i.e., the serialization of measured data and meta information), the number of measurements fields (e.g., a single value for a simple temperature sensor or multiple values in case of a complex industrial control unit), and the units of these fields (e.g.,

degree Celsius, hertz, joule). A promising approach to unify the representation of such diverse sensor data items is SenML[11], which has been proposed for standardization at the IETF. It supports JSON, XML, and Efficient XML Interchange (EXI) for serializing sensor data. To showcase our approach, we focus on the JSON representation of SenML. Still, our findings can also be extended to other data models or other serializations, e.g., XML. Independent from the actual representation and serialization of sensor data, we identify the following three essential tasks: i) identifying those parts in the serialized sensor data that should be covered by the protection, ii) performing the necessary cryptographic operations and augmenting protected sensor data such that an authorized cloud service can reverse these operations, as well as iii) securely distributing the employed data protection keys to authorized cloud services. In the following, we discuss these tasks in more detail. An illustration of this process can be found in Fig. 1.

*Specifying Coverage of Sensor Data Protection.* In the scope of this paper, we assume that sensor data items are readily serialized in SenML. To this end, sensor devices either natively output SenML-encoded sensor data items or a gateway at the boundary of the sensor network has to translate the employed proprietary sensor data formats to SenML. An essential part of protecting such SenML-encoded sensor data items for the cloud then is to encrypt the contained information. However, encrypting sensor data items as a whole is infeasible as certain meta information (e.g., sensor node identifier and timestamp) is required for indexing purposes in order to afford an efficient retrieval of sensor data in the cloud. Moreover, such holistic protection would restrict service access granting to an all-or-nothing approach as all information was encrypted with the same key. Especially in the context of industrial automation, however, it is necessary to break down access granting to individual data fields. This way, the manufacturer of an industrial machine can, e.g., access certain measurement values for monitoring the health of the machine without getting to know details about the product that is currently being processed on this machine. Consequently, to provide confidentiality on a fine-granular basis, we require a way to address parts of a sensor data item. We facilitate JSONPath[12] for this purpose, which allows to address arbitrary fields in a JSON object (similarly, XPath can be used for XML). This way, the parts of a sensor data item that should be encrypted can be specified (see Step 1 in Fig. 1). Notably, digital signatures that provide integrity and authenticity of the protected sensor data item cover the entire data item and thus do not require such fine-granular control.

*Representation of Protected Sensor Data.* In the next step, the data fields identified with JSONPath need to be encrypted. To this end, we use standard symmetric encryption that affords efficient protection of bulk data. Still, we design our approach to be flexible regarding the employed symmetric-key primitives and key lengths in order to allow for scenario-specific security/performance trade-offs and to account for potential future advances in cryptography that may lead to certain primitives no longer being considered secure. However, due to this flexibility, simply replacing the plain value in the sensor data item with the encrypted value is not sufficient. Instead, the cloud services require additional information, e.g., the used encryption algorithm, an identifier for the used data protection key, or the initialization vector to decrypt the protected data fields. Hence, this information also has to be encoded in the sensor data item. This information can be expressed using JSON Web Encryption (JWE)[13], which is a proposed standard for representing encrypted content using JSON. Thus, the plain value in the sensor data item is replaced by a JWE object that contains the encrypted value and the additional information needed for decrypting this value (see Step 2a in Fig. 1). Additionally, the whole sensor data item should be integrity and authenticity protected. For this, the common best-practice is to use public-key signatures. Similarly to JWE, we employ JSON Web Signature (JWS)[13] to represent a public-key signature with JSON. Hence, a JWS-encoded signature is added to the protected sensor data items (see Step 2b in Fig. 1).

*Distributing Keying Material.* In the last step, the data protection keys have to be distributed to authorized services. These keys are needed by the services to decrypt the individual data fields of a sensor data item. We assume that each service is in possession of a public/private key-pair. In order to grant a service access to (parts of) a sensor data item, the corresponding data protection keys are then encrypted with the public key of the respective service and uploaded to the cloud. For this, similar to JWE and JWS, we leverage JSON Web Key (JWK)[13] to represent the encrypted key in JSON format (see Step 3 in Fig. 1). This way, only an authorized service is able to decrypt the data protection key and thus gain access to the data. Whenever the service requires the data protection key for decrypting a sensor data item, it queries the cloud for that key and decrypts it with the service's private key. We additionally periodically exchange data protection keys to increase security[14,15] and offer fine-granular access control with respect to these key change intervals.

## 4. Transparent Access to Protected Sensor Data for Cloud Services

In order to allow cloud service developers to access protected sensor data in the cloud without the need to care about decryption, signature verification, and key management, we propose the Sensor Cloud Security Library (SCSlib)[a]. We implemented SCSlib as a C library and make use of the cryptographic algorithms of the OpenSSL library[16]. In the following, we discuss our design and implementation of SCSlib with respect to the following three main functionalities: i) interfacing with the cloud, ii) processing of sensor data items, i.e., verification and decryption, as well as iii) caching of cryptographic keys for performance improvement.

*Interfacing with the Cloud.* Our design of SCSlib is driven by the goal to provide flexibility and re-usability on the one hand and transparency for cloud service developers on the other hand. Hence, we decided to develop a library that can be integrated into a cloud service SDK, such as Google Cloud SDK and Amazon Web Services SDK, *and* directly being used by the service developer. This way, the security-critical computations take place in the context of the service and no secrets (e.g., the service's private key) have to be revealed to others. Additionally, this design allows the cloud service developer to implement (parts) of the necessary cryptographic operations himself if he does not (fully) trust the provided implementation. This property is aimed at further reducing security concerns when outsourcing sensor data to the cloud by affording an increased transparency of the employed security mechanisms.

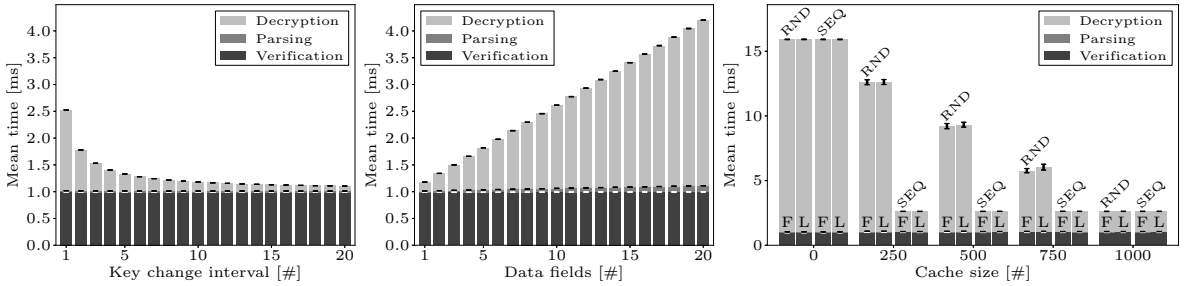
For the decryption and verification of sensor data items, SCSlib requires the public key of the data source as well as the keys used for encrypting the individual data fields. We designed SCSlib to use callback functions, i.e., functionality provided by the cloud service SDK, for retrieving the necessary keying material. This allows each cloud service SDK to implement the communication with the cloud specific to the individual scenario.

*Processing of Sensor Data Items.* To invoke the processing of a sensor data item, SCSlib provides a notably slim API that consists of three public methods: `sc_verify_data_item()` for verifying the integrity of a data item, `sc_decrypt_data_item()` for decrypting a data item, and `sc_process_ data_item()` which combines the previous two methods. Data items are passed to the library as JSON-encoded strings, which yields a simple and portable interface. First, integrity and authenticity of the sensor data item should be checked. For this, SCSlib looks up the public key of the data source using the corresponding callback function (see above) and then verifies the digital signature using the retrieved public key. In order to decrypt the sensor data item, SCSlib iterates recursively over the JSON-serialized object to search for JWE objects representing an encrypted sensor value. For each JWE object, the data protection key needed to decrypt this data field is requested using the above described callback function. Then, the encrypted sensor value can be decrypted, which allows to restore the original value in the JSON-serialized sensor data item. If a cloud service is not permitted access to a specific data field, the corresponding data protection key is not available to this service. Conceptually, there are two possibilities for handling this exception. Either the encrypted sensor value can remain in the data item (which allows the cloud service to notice that it cannot access this specific field) or it can be removed (which yields performance). Our implementation of SCSlib supports both approaches and the actual behavior can be set via a configuration flag.

*Caching of Cryptographic Keys.* For processing data items in SCSlib, different keys are needed for decrypting sensor values and verifying integrity and authenticity of sensor data items. However, which keys are actually needed cannot be determined before a specific data item is processed, especially when considering random access to sensor data items (i.e., data is not accessed in chronological order). Data protection keys needed to access sensor values are encrypted with the public key of the service. Thus, they have to be decrypted before being used. Additionally, one data protection key is typically used more than once during a key change interval (see Sec. 3). To prevent unnecessary overhead, it should be avoided that the same key has to be requested from the cloud and/or decrypted more than once.

To account for this issue, we introduce internal caches in SCSlib for decrypted data protection keys as well as public keys of data sources. As long as a key is present in the cache, it does not have to be requested from the cloud and, in the case of data protection keys, decrypted, again. We will show in our evaluation that this has a huge impact on the overall performance of processing protected sensor data in a cloud service. The cache size as well as the caching algorithm can be configured. We currently support FIFO and LRU as cache management schemes.

[a] Our implementation of SCSlib is available at `https://code.comsys.rwth-aachen.de/redmine/projects/scslib/`




(a) Varying length of key change interval, one data field, "unlimited" cache.

(b) Varying number of data fields, key change interval of ten, "unlimited" cache.

(c) Varying cache size for **F**IFO and **L**RU as well as random (RND) and sequential (SEQ) processing.

Fig. 2: Average time for processing one protected sensor data item in a cloud service for different parameters.

## 5. Performance Evaluation

In order to prove the feasibility of our approach and quantify the performance of SCSlib, we conducted an extensive performance evaluation. To this end, we implemented a simple cloud service SDK using the C programming language that returns requested sensor data items, data protection keys, and data source's public keys from a static database as well as a cloud service that triggers the decryption and verification of sensor data items. In this paper, we focus on the performance impact at a cloud service. For our evaluation, we used the cryptographic primitives AES with 128 bit keys in CBC mode for encrypting the data field, RSA with 2 048-bit keys for encrypting the corresponding data protection keys, and ECDSA with NIST curve P-256 for digital signatures. In order to allow others to reproduce our results, we use Amazon Web Services third generation EC2 64-bit instances of type large (m3.large)[17] running Ubuntu 12.04 LTS. For each measurement point, we conducted 50 measurement runs, each consisting of the processing of 1 000 data items. We depict the average processing time per data item and the standard deviation for these measurements.

First, we examine the general minimal costs of processing protected sensor data in the cloud by considering only the absolutely necessary operations. For this, we chose cache sizes (see Sec. 4) such that each key has to be requested only once (this emulates an infinite cache size). The main influence factors on the costs of processing protected sensor data are the size of the key change interval and the number of data fields. Fig. 2a depicts the average time for processing one protected sensor data item with one data field *with respect to the key change interval* (i.e., after how many items the data protection key is exchanged). Especially for larger key change intervals, the processing time is dominated by the verification of the digital signature, which constantly requires about 0.99 ms. In addition, parsing a data item and processing keys only requires 0.02 ms. The time needed for decrypting the data item rapidly decreases from about 1.51 ms to 0.09 ms for increasing key change intervals. In the following, we use a key change interval of 10, as this constitutes a good trade-off between flexibility and performance.

Similarly, Fig. 2b illustrates the processing time of a data item with an *increasing number of data fields* for a key change interval of 10. Again, signature verification constantly requires roughly 0.99 ms. The time needed for parsing and decrypting the data item increases linearly with the number of data fields from 0.19 ms (0.02 ms for parsing and 0.17 ms for decrypting) for one data field to 3.21 ms (0.12 ms for parsing and 3.09 ms for decrypting) for 20 data fields. Based on these results, we use data items with 10 data fields for our remaining evaluation. These results constitute a lower bound for the performance, to which we compare our caching optimizations in the following.

In Fig. 2c, we evaluate the performance based on *cache size* and *cache management schemes* for data items with 10 data fields and a key change interval of 10. We differentiate between the two cache management schemes **F**IFO and **L**RU and vary the cache size between 0 (no caching) and 1 000 (all keys cached). Furthermore, we consider both, sequential and random processing of sensor data. In the sequential case, sensor data is processed in temporal order (which is often observed in real-world scenarios), while in the random case, sensor data is processed in an arbitrary, non-deterministic order (which is the most challenging scenario with respect to caching). Our results show that caching indeed has an enormous impact on performance. With an appropriate cache size, we achieve a 6-fold reduction in processing time from 15.91 ms to 2.62 ms, which corresponds to the lower bound (see Fig. 2b). As expected, for sequential processing, we are able to achieve the best possible performance as soon as the cache size

equals or exceeds the number of simultaneously required data protection keys (in our example this is ten, as we have ten data fields per data item). Similarly, the processing time for random processing decreases linearly with the cache size, as the likelihood that a key is still in the cache increases with the cache sizes. Our evaluation also shows that the performance difference between FIFO and LRU is negligible in our scenario.

To conclude, our performance evaluation showed that it is feasible to process protected sensor data items in a cloud service. SCSlib enables interoperability, allowing for an open environment and integration with different cloud offers[18]. Furthermore, due to SCSlib's caching, we can significantly improve the per-data item processing time. Most importantly, SCSlib enables non-security experts to develop secure cloud services. These advantages come with higher transmission and storage overheads than tailor-made solutions for individual scenarios. However, these overheads can be minimized by employing data item compression, e.g., using CBOR[19].

## 6. Conclusion

In this paper, we presented a best practice approach for representing and protecting sensor data in the context of cloud computing. Based on this approach, we proposed SCSlib, a library that enables cloud service developers to transparently access protected sensor data. Notably, SCSlib does not require any security expertise from cloud service developers and, at the same time, is sufficiently flexible to satisfy a wide range of performance and security requirements. Our evaluation results not only show the feasibility of SCSlib but also demonstrate a significant performance gain for sequential and random access to sensor data in the cloud. Hence, with SCSlib we contribute an important corner stone for the secure incorporation of the two technologies sensor networks and cloud computing.

*Acknowledgments.* This work has been funded by the German Federal Ministry for Economic Affairs and Energy (project funding reference 01MD11049). The responsibility for the content of this publication lies with the authors.

## References

1. Hummen, R., Henze, M., Catrein, D., Wehrle, K.. A Cloud Design for User-controlled Storage and Processing of Sensor Data. In: *2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*. 2012, p. 232–240.
2. Eggert, M., Häußling, R., Henze, M., Hermerschmidt, L., Hummen, R., Kerpen, D., et al. SensorCloud: Towards the Interdisciplinary Development of a Trustworthy Platform for Globally Interconnected Sensors and Actuators. Tech. Rep.; RWTH Aachen University; 2013.
3. Henze, M., Hummen, R., Matzutt, R., Catrein, D., Wehrle, K.. Maintaining User Control While Storing and Processing Sensor Data in the Cloud. *International Journal of Grid and High Performance Computing (IJGHPC)* 2013;**5**(4):97–112.
4. Rolim, C., Koch, F., Westphall, C., Werner, J., Fracalossi, A., Salvador, G.. A Cloud Computing Solution for Patient's Data Collection in Health Care Institutions. In: *2nd International Conference on eHealth, Telemedicine, and Social Medicine (ETELEMED)*. 2010, p. 95–99.
5. Zhang, X.M., Zhang, N.. An Open, Secure and Flexible Platform Based on Internet of Things and Cloud Computing for Ambient Aiding Living and Telemedicine. In: *International Conference on Computer and Management (CAMAN)*. 2011, p. 1–4.
6. Henze, M., Hermerschmidt, L., Kerpen, D., Häußling, R., Rumpe, B., Wehrle, K.. User-driven Privacy Enforcement for Cloud-based Services in the Internet of Things. In: *2014 International Conference on Future Internet of Things and Cloud (FiCloud)*. 2014.
7. Henze, M., Hummen, R., Matzutt, R., Wehrle, K.. A Trust Point-based Security Architecture for Sensor Data in the Cloud. In: *Trusted Cloud Computing*. Springer; 2014.
8. Itani, W., Kayssi, A.. SPECSA: a scalable, policy-driven, extensible, and customizable security architecture for wireless enterprise applications. *Computer Communications* 2004;**27**(18):1825–1839.
9. Linn, J.. Generic Security Service Application Program Interface Version 2, Update 1. IETF RFC 2743 (Proposed Standard); 2000.
10. Huang, M., Wang, C., Zhang, L.. Toward a reusable and generic security aspect library. In: *AOSD Technology for Application-level Security (AOSDSEC)*. 2004, p. 1–6.
11. Jennings, C., Shelby, Z., Arkko, J.. Media Types for Sensor Markup Language (SENML). IETF Internet-Draft draft-jennings-senml-10; 2013. Expired.
12. Gössner, S.. JSONPath – XPath for JSON. 2007. URL: `http://goessner.net/articles/JsonPath/`.
13. Javascript Object Signing and Encryption (jose) Working Group at IETF. 2014. URL: `http://datatracker.ietf.org/wg/jose/`.
14. Krawczyk, H.. SKEME: A Versatile Secure Key Exchange Mechanism for Internet. In: *Proceedings of the Symposium on Network and Distributed System Security (SNDSS)*. 1996, p. 114–127.
15. Eltoweissy, M., Moharrum, M., Mukkamala, R.. Dynamic Key Management in Sensor Networks. *IEEE Commun Mag* 2006;**44**(4):122–130.
16. Viega, J., Messier, M., Chandra, P.. *Network Security with OpenSSL: Cryptography for Secure Communications*. O'Reilly; 2002.
17. Amazon Web Services, Inc.. Amazon EC2 Instances. 2014. URL: `http://aws.amazon.com/en/ec2/instance-types/`.
18. Henze, M., Großfengels, M., Koprowski, M., Wehrle, K.. Towards Data Handling Requirements-aware Cloud Computing. In: *2013 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*; vol. 2. 2013, p. 266–269.
19. Bormann, C., Hoffman, P.. Concise Binary Object Representation (CBOR). IETF RFC 7049 (Proposed Standard); 2013.