FP7-SMARTCITIES-2013
Project number: 609132
http://www.e-balance-project.eu/

Co-funded by
the European Union

e-balance

Deliverable D5.4

Detailed specification, implementation and evaluation of security and privacy means

Editor:	Krzysztof Piotrowski (IHP)
Dissemination level: (Confidentiality)	PU
Suggested readers:	Consortium/Experts/other reader groups
Version:	1.0
Total number of pages:	22
Keywords:	Security, Privacy, Smart Grid

Abstract

This deliverable describes the security and privacy solution for the e-balance system. The focus of this document is on the modules that implement the security and privacy protocol for the Energy Management Platform that relies on the basic security mechanisms provided by the Communication Platform. The modules from both platforms cooperate with each other to achieve the goal of protecting the data according to the definition of the data owner.

Disclaimer

The information, documentation and figures available in this deliverable are written by the e-balance partners under EC co-financing (project number: 609132) and does not necessarily reflect the view of the European Commission.

Impressum

[Full project title] Balancing energy production and consumption in energy efficient smart neighbourhoods

[Short project title] e-balance

[Number and title of work-package] WP5, Energy Management Platform
[Document title] Detailed specification, implementation and evaluation of security and privacy means
[Editor: Name, company] Krzysztof Piotrowski, IHP

[Work-package leader: Name, company] Marco Gerards, UTWE

Copyright notice

Executive Summary

The security aspects of the energy grid are gathering more and more attention in the recent years. This is mainly due to the fact that the energy grid is part of the critical infrastructure and it shall be protected from any interference that can disturb its working, especially if this interference is an intentional act. The security aspects actually become critical in the era of smart grid, where the exchanged data controls the state of the grid (sometimes in real-time) and it is sometimes even easier to interfere with this data exchange due to the scale of the communication. The system level protection helps to protect the smart grid from external attacks.

In addition, guaranteeing privacy is of utmost importance when it comes to user acceptance. This deliverable focusses on user data protection as this is probably the most important aspect for the system users. In this document we will provide the detailed description of the security and privacy protocol that is used in the e-balance system to control the access to the data of the users according to their defined policy. This deliverable builds up on the basic and standard security mechanisms that can be used to protect the devices and the communication, as presented in deliverable D4.2. It is assumed that the protection on the system level is present and thus, the context for the security and privacy protocol described in this document is defined.

This document describes the way the data gathered and generated by the system is exchanged between the system users. It provides the implementation details for the blocks that are involved in the security and privacy protocol. These blocks are located in the communication platform – the data access control module, as well as in the energy management platform – the security and privacy module.

The operations involved by the protocol are also evaluated with respect to the computational overhead. In order to measure this impact we have implemented the main mechanisms in C programming language and measured the computational effort required by these operations.

List of authors

Company	Author
IHP	Krzysztof Piotrowski Ievgen Kabin Peter Langendörfer
UMA	Daniel Garrido Eduarto Canete Jaime Chen
EDP	Nuno Emanuel Pereira
EFA	Paulo Rodrigues Alberto Bernardo
ALLI	Marcel Geers
LW	Henry Schomann

Table of Contents

List of Tables

List of Figures

Abbreviations

CP	Communication Platform
EMP	Energy Management Platform
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
RSA	Rivest-Shamir-Adleman
DSA	Digital Signature Algorithm
SHA	Secure Hash Algorithm
CaMyTs	Castelluccia-Mykletun-Tsudik
PRNG	Pseudo Random Number Generator
DH	Diffie-Hellman
MU	Management Unit
TP	Trusted Party

1 Introduction

The security aspects of the energy grid are gathering more and more attention in the recent years. This is mainly due to the fact that the energy grid is part of the critical infrastructure and it shall be protected from any interference that can disturb its working, especially if this interference is an intentional act. The security aspects actually become critical in the era of smart grid, where the exchanged data controls the state of the grid (sometimes real-time) and it is sometimes even easier to interfere with this data exchange due to the scale of the communication.

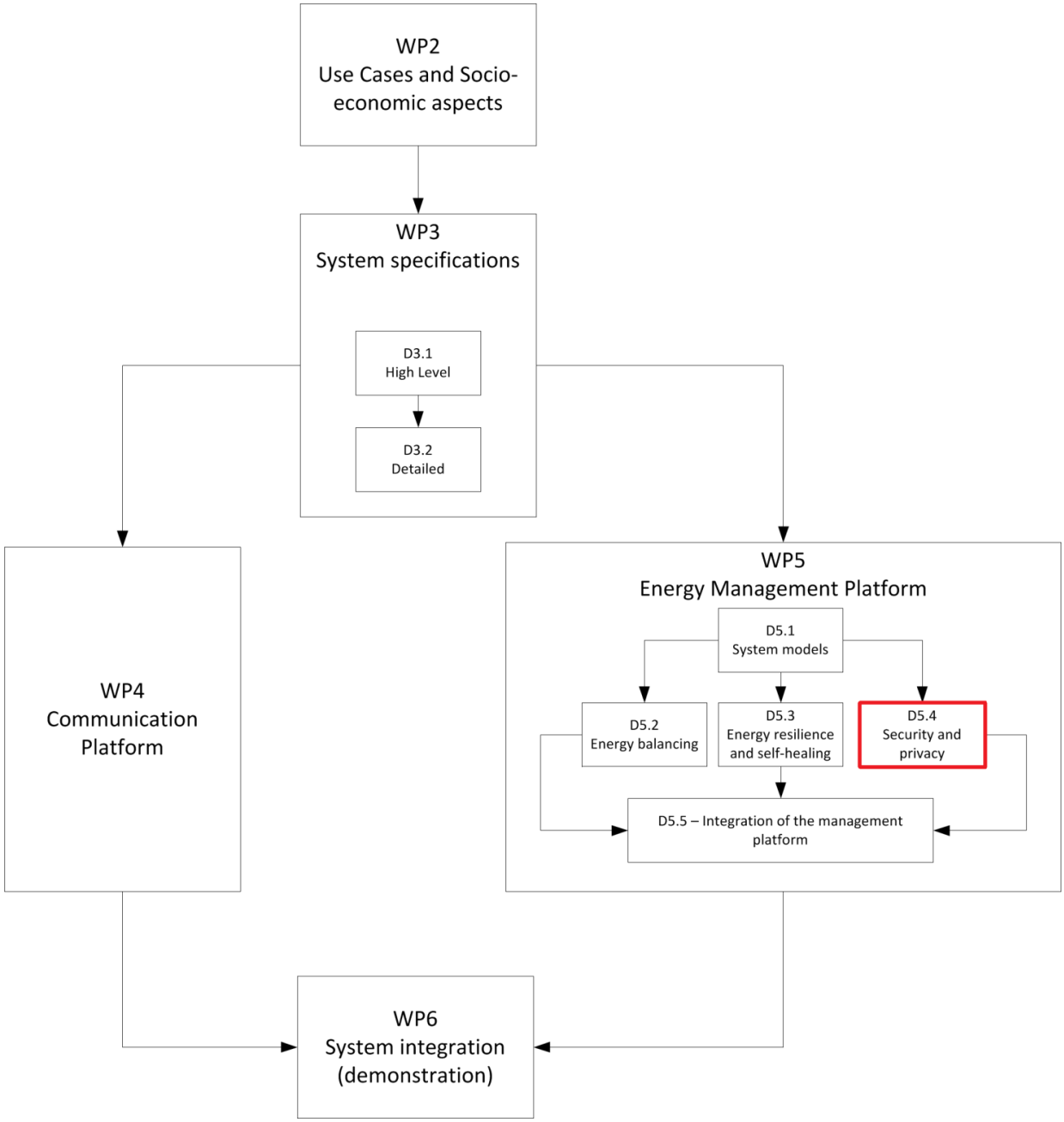

Figure 1: The position of the deliverable D5.4 within the e-balance project

Due to the importance of the subject we address it in the e-balance project. In this document we will provide the detailed description of the security and privacy protocol (and its implementation) that is used in the e-balance system to protect the data of the users according to their defined policy. This deliverable builds up on the basic and standard mechanisms that can be used to protect the devices and the data, presented in deliverable D4.2 [1]. It is assumed that the basic protection on the system level is present and thus, the context for the security and privacy protocol is defined. This document describes the way the data gathered and generated by the system is exchanged.

In the following paragraphs we will describe the coverage of each of the security related documents, based on the security architecture, defined in deliverable D3.2 [2] and depicted in Figure 2.

The D4.2 covers all the security and privacy modules (blue boxes in the figure) except the data access control module within the request processor and the security and privacy modules within the energy management platform. Those blocks are implementing the security and privacy protocol and are covered by this document – deliverable D5.4.

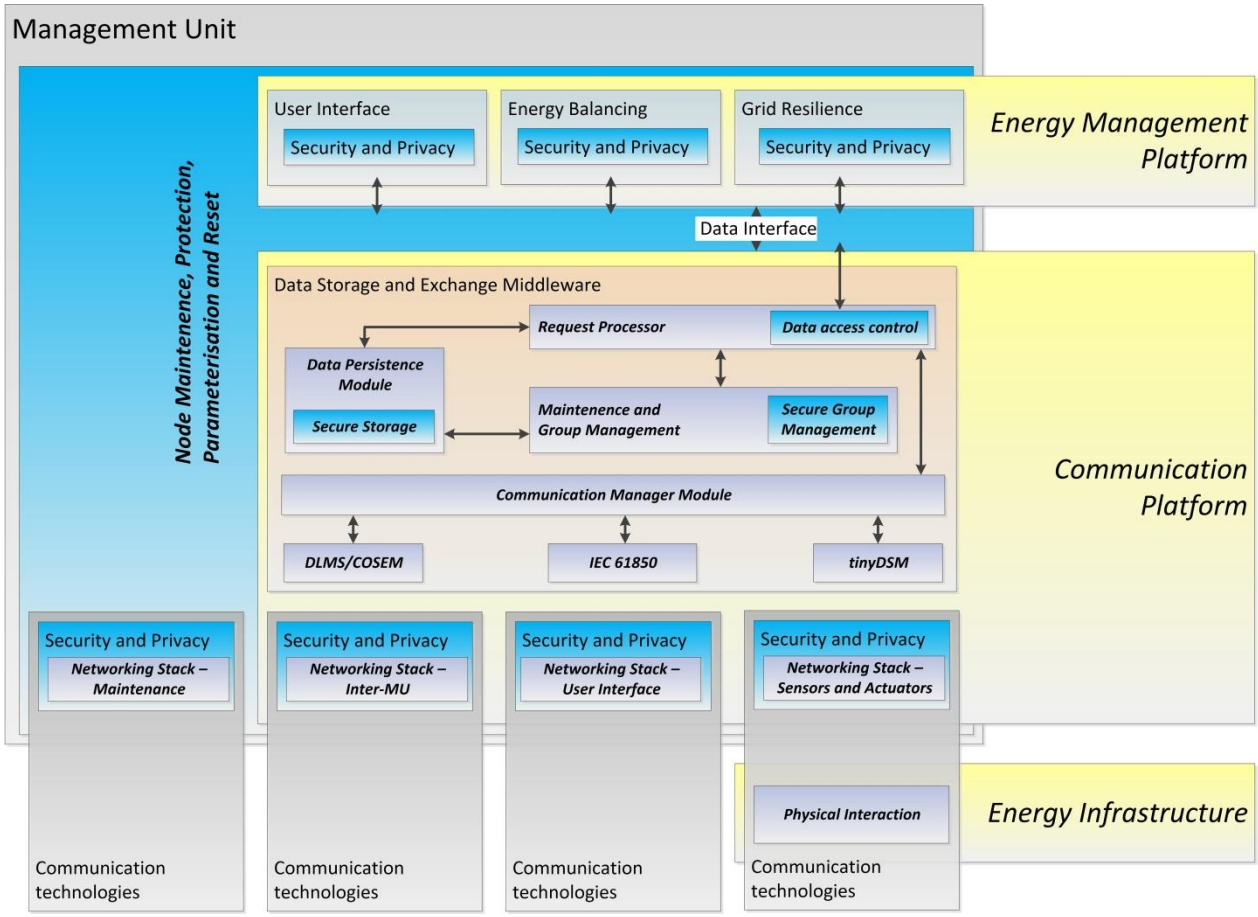

Figure 2: The security architecture of the e-balance management unit

There are two major groups of security and privacy modules within the energy management platform that are covered by this document. They can be defined as follows:

- Energy management platform service security and privacy modules
- Data access control (will be covered also by the deliverable D4.2)

The specific instantiations of these modules and their very specific low level functions depend very much on the hardware and software implementation of the individual management unit. However, these modules should fulfil a specific set of requirements in order to provide their function properly. Additionally, the standard IT security solutions as well as the level of data protection regarded as secure, change with time and need to be updated with newer and stronger algorithms or parameters and fixes. Thus, the implementation described in this document supports updating the security approach with up-to-date basic means.

It is suggested to read the deliverable D4.2 document prior to this document.

2 The security and privacy protocol

As presented in Figure 2 and explained in the previous section, the security and privacy protocol for the energy management is implemented around the Data Interface connecting the Communication Platform (CP) and the Energy Management Platform (EMP). The role of this protocol is to define the rules for providing the access to the data exchanged between the management units within the e-balance system.

The Access Control Module is implemented in the communication platform and the data is accessed via the data interface by the services located in the energy management platform. A service runs on behalf of a specific stakeholder. The data owner may define which services from which stakeholders can access her data. The access control is driven by that definition. Further, the data owner may also restrict the granularity of the data in the time dimension, i.e., defining the minimum delay between two read requests.

The privacy is further protected by means of an end-to-end encryption supporting a diversity of mechanisms, including standard encryption of individual values, but also privacy homomorphic encryption that allows performing aggregation of multiple encrypted values.

The modules involved in the implementation of the protocol are presented in Figure 3. The functionality provided by the security and privacy module for the energy management platform services allows them to generate a valid request for the data interface – signed, in order to authenticate the data access. Further, the data (to be) stored in the middleware may be additionally protected by means of encryption. In order to prepare the data for storing and also to use the encrypted data the service uses the respective functions provided by the security and privacy module. If the values are encrypted using a privacy homomorphism, then the module also provides respective functions allowing performing the defined aggregation operations on the encrypted data, e.g., addition.

In order to perform the above mentioned security related functions, the security and privacy module provides also the storage for the security credentials of the respective stakeholder used by these security operations. These credentials include passwords and key materials with certificates.

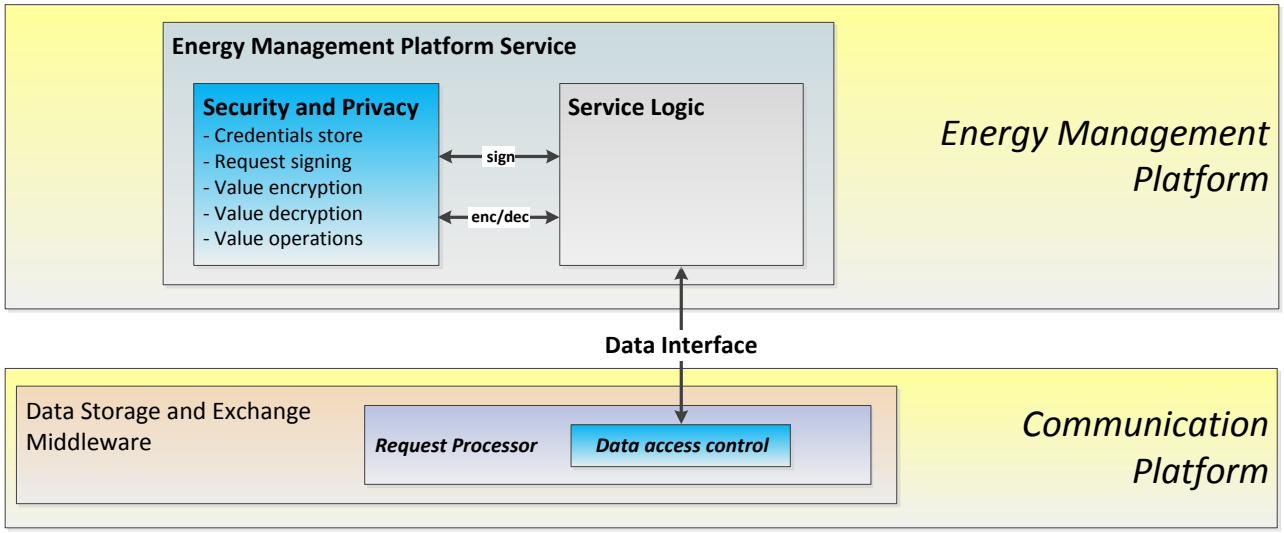

Figure 3: The modules involved in the security and privacy protocol for energy management

Figure 3 shows also the features provided by the security and privacy module in the energy management platform. While preparing the data access request, the service can use different functions provided by the security and privacy module. A service instance (process) usually works for a single stakeholder, thus it is connected to one security and privacy module. This allows separating the service logic and the credentials of the stakeholders and by that allows replicating the same service implementation to work for different stakeholders.

Further, it is also possible to extract the security and privacy module and realize it as a hardware module that is connected to the respective management unit and provides its functions via a defined API. Within e-balance we focus on the software realisation.

Additionally, if a case where one service is working for multiple stakeholders is justified, it must use individual security and privacy module for each stakeholder, to separate their credentials.

The security and privacy block of the energy management platform provides the following functions.

- sign (message, *out* signature)
- encrypt (plaintext, *out* ciphertext)
- decrypt (ciphertext, *out* plaintext)
- add (a, b, *out* sum)

The accesses to the security credentials are embedded in the functions listed above. The service executes its logic and each time the access to the data stored in the communication platform is necessary the request is prepared, i.e., the content is prepared, collected and the whole request is signed.

On the other hand, the data access control module has the following security related function available.

- verify (message, signature, key, *out* result)

Here the specific key material from the certificate is accessed and hiding of the accesses to the credentials is not necessary. The certificate is either provided with each request, or it can also be stored in the communication platform. There are advantages and disadvantages of each solution. The provision of the certificate with the request may cause an increase in the data exchanged, but simplifies the implementation and no special storage (and management) for the certificates is necessary.

Depending on the kind of the data access request, different security related actions can be performed and there are different contents of the request data structure. Common for all the four kinds of requests, are the parameters of the functional interface and the signature of the request issuer over the whole request. Additionally, if a write request is performed by the data owner the request may also contain the privacy and security policy that defines the allowed operations on the written data item. If a write request is done by other stakeholder, then, depending on the owner's preferences either the latest policy defined by the data owner or a default policy is applied.

The security and privacy policy is defined for each written data value. It is a structured set of the following items:

- identifier of the authorized stakeholder
- identifier of the authorized service of the stakeholder
- allowed access rights, and (optionally)
- the minimum delay between two accesses (in seconds)

Such a data set is provided for each authorized stakeholder and service combination. Different stakeholders and services with exactly the same access rights may be grouped together.

The protocol running on the data interface between the communication platform and the energy management platform services can be shortly described by the following steps.

1. Service logic (preparation of the request)
    - identification of the data to be accessed (variable and data owner)
    - definition of the access type (write, read, event, periodic)
    - for the write access:
        - for own data (optional) definition of the security and privacy policy (or use of the default)
        - (optional) encryption of the value with *encrypt()*
    - for the event and periodic access, definition of the condition and period, respectively
    - performing the signature on the request with *sign()*
    - transferring the request (and optionally the certificate) over the data interface

2. Data access control (processing of the request)
    - for incoming certificate: verifying the certificate with *verify()*
    - verifying the request signature with *verify()*
    - verifying the access rights (comparing the stored policy with the actual access)
    - if the above steps are successful: processing the request by the request processor
    - (the processing and data exchange is protected by the D4.2 security mechanisms)
    - transferring the results back to the service logic

3. Service logic (receiving and processing the reply)
    - (optional) processing (aggregation) of encrypted data with *add()*
    - (optional) decryption of the value with *decrypt()*
    - further processing by the service logic

The main aim of the privacy protocol is to allow the data source to define the allowed use of the data it generates. Of course, there may be contradicting interests in the sense that the data source may be willing to restrict the accesses to its data very much, while the data consumers may be willing to have the access on very detailed basis, since they need the data to perform their operations. The border line that specifies the default (and minimum) level of access shall be defined by the contract between the data source and the data consumers, like the contract between the customer and the energy retailer. The complete set of contracts that the data source has with all the service providers defines the base for the privacy and security policy. Due to the contract, the data source is obligated to provide specific data to specific stakeholders at specific rates (temporal resolution). Anything above the contracted level (additional access rights, extensions of present access rights) has to be negotiated between the parties and can be added to the basic policy. In any case a change in the privacy policy restricting the data access for a party the data source has a contract with may cause a contract violation.

During the privacy protocol design process we had multiple discussions on how and where to define the privacy policy, as well as on where and how to enforce it. There are several options for both these points and we have decided to provide a design that, on one hand allows different solutions and is thus flexible, but on the other hand we focus on implementing the one that is the most meaningful, from our point of view.

For the definition of the privacy policy there are two main options. The first is to define the policy as a rather static construct while the variable is created, while the device is installed and configured for the first time or while it is reconfigured. This policy is mainly based on the contracts the data source has with the parties interested in the data. Thus, any time the contract changes the policy needs to be updated. This can be done using the middleware tool for managing the variables. This allows the writing request to be simpler, i.e., it does not include the policy definition for each written value.

The second option is to allow the service running on the management unit to define the policy for each value that it writes on behalf of the data owner. This solution allows more flexibility as each value can have a different access right definition and as a result, for instance, a different allowed reader group. However, this solution requires storing the policy for each written value, what can cause the required data storage size to grow. In case the data is replicated within the middleware on several management units this problem may become an issue.

With respect to enforcing of the privacy policy there are two options that actually depend on the realisation of the data storage and the request processor within the middleware. Enforcing of the policy can happen on the unit that stores the value or at the one which received the request. In the latter case it is necessary to transmit the policy together with the value to be able to check the access rights before answering the request. If data replication is allowed then it should also include the replication of the privacy and security policy, otherwise it is necessary to check the access rights at the origin management unit.

In our solution we decided to support the static policy that can be overridden by the one defined by the service writing the data on behalf of the owner. The enforcing of the policy happens on the unit that stores the data.

The following section will describe the implementation details of the blocks involved in the above mentioned protocol.

3 Implementation details

This section provides the details on the implementation of the individual blocks involved in the privacy and security protocol. The following sub-section describes the part of the implementation present in the energy management platform, while the latter one describes the data access control module present in the communication platform.

3.1 Energy management platform service security and privacy module

The implementation of the security and privacy module in the energy management platform can be divided into three blocks. The first one covers the functionality and storage related to the stakeholder and security module identification. The second one is related to the public key security needed for authenticating (signing) the requests and the third block includes the functionality and storage related to the privacy homomorphism mechanism used to encrypt the data. Figure 4 shows the internal structure of the security and privacy module, with the separated functional blocks.

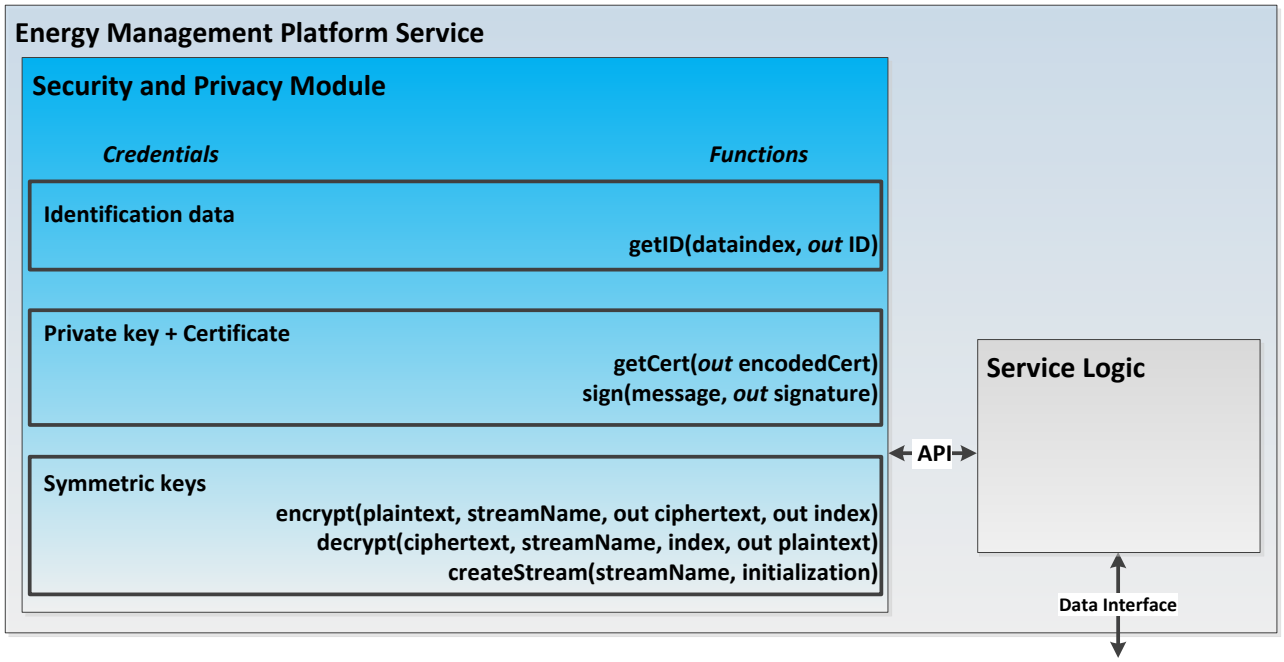

Figure 4: Internal structure and interactions of the Security and Privacy Module

We have implemented this module in Java and in C. The former implementation can be used for integration, while the latter was mainly used for performance checks on embedded devices. The following subsections present the implementation details of each of the above mentioned blocks.

3.1.1 Identification

The identification block contains a data structure that contains data that can be used for identification purposes, but may also be used as service configuration data. The data is set while the module is initialised (or installed). For the service logic this block is like a read-only memory.

The block provides a function to access the fields of the data structure. The data can be addressed by indices or by field names. The following method implements the function in Java.

```
public Object getID(String dataindex);
```

The function returns the object from the storage identified by the given identifier (*dataindex*).

3.1.2 Request authentication

The request authentication block contains the credentials necessary to prove the identification of the stakeholder, i.e., her private key used to sign the requests, as well as the certificate that contains the corresponding public key of the stakeholder. The certificate is signed by a party that is trusted in the system (the certification party) and it is used to verify the signatures generated by the block at the access control module.

The request authentication block provides two functions, one used to sign the request (as a byte stream) and the second that provides the certificate, so that it can be forwarded together with the request and the signature over the request to the verifying party – the access control module.

```
public byte[] sign(byte[] message);
```

The byte stream *message* is signed using the private key stored in the credential store and returned as a byte stream.

```
public byte[] getCertBytes();
public Certificate getCert();
```

The above two Java methods return the certificate of the stakeholder stored in the credential store. The first one returns the encoded certificate that is ready to be transmitted (and serialized) while the second one returns it as a Java object.

In our implementation we have focused on signature algorithms based on Elliptic Curve Cryptography (ECC), like the ECDSA (Elliptic Curve Digital Signature Algorithm), since they generate much smaller signatures, while providing the same level of protection, compared to much larger signatures, like RSA (Rivest-Shamir-Adleman) or DSA (Digital Signature Algorithm). However, the Java implementation supports all the three above named signature algorithms. The key material (and the certificate) is initialized while the module is installed and is read-only for the service logic.

We use certificates generated using the Openssl Toolkit, which are then installed while the security and privacy module is installed and initialized. These are standard X509 v3 certificates.

3.1.3 Data encryption

The data encryption is based on the additively homomorphic encryption scheme proposed by Claude Castelluccia, Einar Mykletun and Gene Tsudik (CaMyTs) [4]. It provides the feature that the encrypted data may be added and the sum can be decrypted, if the correct key material is available.

The following operations using a defined modulus *M* are defined in this scheme. The modulus must be common for all data sources, whose data should be aggregated and must be known to the data consumer.

Encryption:

1. Represent the message (plaintext data) *m* as integer $m \in [0, M-1]$ where M is a large integer

2. Let k be a randomly generated key, where $k \in [0, M-1]$

3. Compute: $c = Enc(m, k, M) = (m + k) \bmod M$, where c is the ciphertext

Decryption:

1. Compute: $m = Dec(c, k, M) = (c - k) \bmod M$, where *k* is the same key as used for encryption

Addition of Ciphertexts:

1. Let $c_1 = Enc(m_1, k_1, M)$ and $c_2 = Enc(m_2, k_2, M)$

2. Compute: $c = (c_1 + c_2) \bmod M$

3. The ciphertext can be decrypted using the aggregated key $k = (k_1 + k_2) \bmod M$: $Dec(c, k, M) => m_1 + m_2$

The main disadvantage of the CaMyTs approach is that it requires a keystream for the keys to encrypt the individual values. The maintenance of the key stream and synchronizing it between the data sources and the data consumers is not a trivial task. Thus, we have combined the approach with the secure Pseudo Random Number Generator (PRNG) – the lmRNG, developed at the IHP [3].

The lmRNG approach is a set of equations that use start values and operate on these changing its internal state and generating a bit-stream that can be broken down into individual values of an arbitrary length. Two instances of the generator that are using the same initial values (seed) will generate exactly the same bit-stream.

The lmRNG is a secure PRNG, what also means that having one (or more) of the values it generated you are not able to deduct neither the following nor the previous values, the generator generated or will generate, without knowing its seed or internal parameters. This feature makes the lmRNG a perfect source of keys. Further, it also simplifies the synchronisation between the data source and data consumer. Once the key generators on both sides are initialized with the same seed values, both communicating sides can use the key indices to identify the correct key, i.e., the data and the keys are provided as two streams, where the data stream is transmitted from data source to data consumer and the key stream is generated independent on both sides.

Additional data consumers may get access to the data by getting the key generator parameters allowing generating the key stream from a given position in the original stream only. This is a form to restrict access to the data, i.e., the joining party is not allowed to decrypt data that was created prior to the obtaining of the data access.

Further, it is possible to restrict access to the data stream by distributing keys that allow decryption of only some chosen aggregated values.

In order to optimize the encryption and decryption operations we have decided that the modulus is equal to 2^n, where n is the bit-length of the used integer – in our case n it is 64, but we support also 32 bit long integers. Thus, the modulo operation simply causes discarding the data type overflow.

Thus, the block related to the data encryption stores the following data set for each key stream: initialisation data (two integers: iv_1 and iv_2), current state of the internal equations (t_1 and t_2), the current key and its index. A key stream can be used for encrypting one variable, thus for each encrypted variable an individual set of the above mentioned data items is necessary.

Further, if a data consumer joined the data stream, the fact is stored as well, together with the key index and the internal key stream generator parameters.

The encryption block implementation in Java provides the following methods.

```
public boolean registerKeyStream(String name, long iv1, long iv2);
```

Registers a new key stream generator identified by *name* and initializes it with the given initialisation data (*iv1* and *iv2*).

```
public PRNGInit getKeyStreamFork(String name, String clientName);
```

Provides the initialization data taken from a defined local key stream (identified by *name*) to be used by a remote key stream generator at data consumer identified by *clientName.* The *PRNGInit* object contains the

initialisation data and the current key index. The current key index is used at the remote client to compute its local key for decryption.

```
public EncData encrypt(long plaintext, String name);
```

This method encrypts a new value (*plaintext*) within the defined key stream identified by *name*. It generates a new key in the key stream and encrypts the data. It returns the encrypted data together with the index of the used key – both contained in the returned *EncData* object.

```
public long decrypt(long ciphertext, String name, long index);
```

This method decrypts the encrypted data (*ciphertext*) using the key identified by *index* from the local key stream identified by *name*. It returns the decrypted value.

The key stream generators are created on both ends of the communication channel, i.e., at the data source and at the data consumer. The exchange of the parameters to configure the key stream generator can be either done manually, while the device (and the security and privacy module) is installed and configured, or it may be done on-line. In the latter case the two services have to securely exchange the initialisation data using a protocol similar to Diffie-Hellman (DH).

3.2 Data access control module

The Access Control Module is located in the Communication Platform, in the Data Storage and Exchange Middleware (see Figure 5). It is implemented in Java programming language and the main blocks of the algorithms have also been implemented in C programming language.

The data access requests are provided to the Request Processor via the Data Interface or from other Management Units (MU). Before being processed, these requests are verified by the Access Control Module.

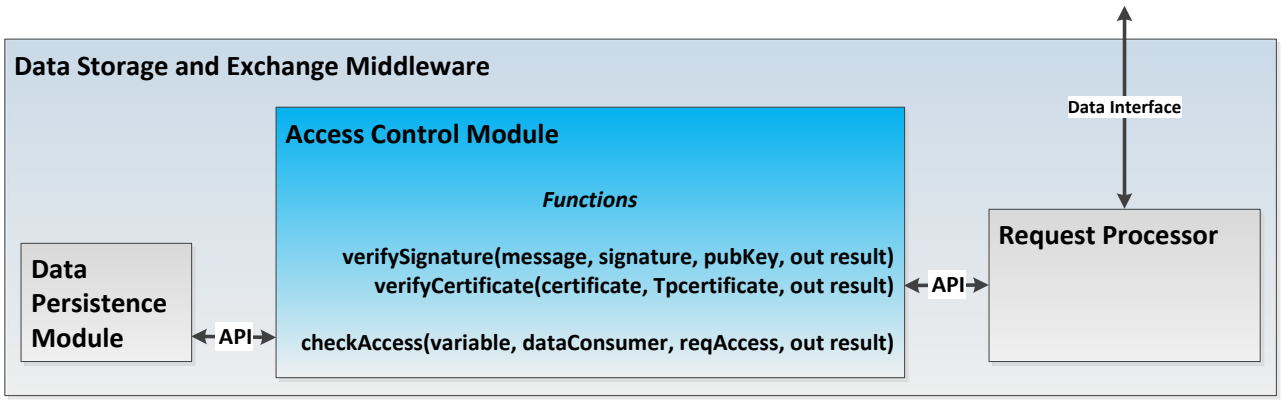

Figure 5: Internal structure and interactions of the Access Control Module

Table 1: Record of the table in the Data Persistence Module storing the data owner policies

No (primary key)	*variable*	*stakeholder*	*service*	*allowed_access*	*delay*	*last_access*

The access control module provides an API that allows the request processor to verify the request source and verify that the requested access is allowed. The verification of the certificate (the authentication of the request) is based on the signature verification, while the access control is realized based on the privacy and security policy of the data owner, stored in the Data Persistence Module. Table 1 presents the fields of the database table in the data persistence module that stores the policy of the data owner. It defines who

(*stakeholder* and *service*) can access and how (*allowed_access*) the owner's data (*variable*). It also specifies if there is a defined minimum delay between accessed data (*delay*). The latter defines the temporal resolution of the accessed (read) data. The last field (*last_access*) stores the timestamp of the last performed access of the defined kind. The table stores different access types (read, write, etc.) for the same request source in separate records.

Thus, the data access control module provides the following methods.

```
public boolean verify(byte[] msg, byte[] sign, PublicKey pubkey);
```

This function verifies, if the provided signature (*sign*) is a correct one for the provided message (*msg*) and the public key (*pubkey*). It returns ***true*** of ***false***, depending on the result of the check. It allows to verify the signature of the data access request source and thus, to authenticate the request.

```
public boolean verifyCert(Certificate cert, Certificate TPcert);
```

This function allows verifying, if the provided certificate (*cert*) of the request source is correctly signed by the Trusted Party (TP), whose certificate is also provided (*TPcert*). Thus, it allows verifying the trust chain. The function returns ***true*** of ***false***, depending on the result of the check.

```
public boolean checkAccess(String variable, String owner, String
                    stakeholder, String service, String reqAccess);
```

This function accesses the database table with the data owner's security and privacy policy and verifies, if the request source (*stakeholder* and *service*) is allowed to access (*reqAccess*) the data (*variable*) from the specified data owner (*owner*). The function also performs the internal check if the previous data access by the request source was performed following the minimum delay parameter. The function returns ***true*** of ***false***, depending on the result of all the checks.

The certificate of the stakeholder requesting the data access is currently provided together with the request, but it may also be stored locally in the database. In the latter case the first access from a given stakeholder and service combination requires providing the certificate. The certificate of the trusted party is embedded in the module and is defined while the module is being installed or reinitialized.

4 Evaluation

The evaluation of the proposed approach was mainly focused on the complexity it induces. The aim of this evaluation was to define the overhead caused by the security and privacy functionality.

The complexity evaluation was realized as follows. In order to distinguish between the computational effort caused by the evaluated algorithms and the background processing we did the evaluation in a single threaded environment – an embedded system that was developed at IHP [5]. The test platform is based on the 16-bit MSP430 microcontroller. Further, the algorithms were implemented in C programming language. The complexity of the developed solutions is expressed in the number of clock cycles required to execute each operation. This measure allows estimating the overhead for more complex environments, but it also allows evaluating, if the proposed approach can be used on low power and computationally limited embedded devices.

The evaluated mechanisms are divided in two classes; the first contains the public key cryptography operations (ECDSA signature generation and verification), while the second contains the operations related to the data encryption (privacy homomorphism CaMyTs encryption, decryption, generating of a new key using the lmRNG).

The microcontroller on the test platform runs with 25MHz, meaning that an operation requiring 25 million of clock cycles causes the microcontroller to be busy for one second. Further, knowing the frequency of these operations the duty cycle of the microcontroller can be estimated.

4.1 Public key operations

The public key operations include the generation of the signature and the verification of a signature. The signature algorithm is the ECDSA P160 with SHA-1 (a variant of the SHA). The size of the message is 100 bytes to reflect the projected size of a request. The values are an average of 1000 measurements.

Table 2: The computational complexity of public key operations

Operation	Complexity [clock cycles / s]
Sign (ECDSA with SHA)	1’262’189’020 / 50.49
Verify (ECDSA with SHA)	2’522’699’881 / 100.91

4.2 Data encryption and key generation operations

The operations related to the encryption of the data include data encryption, data decryption, as well as the generation of a new key in the key generator. The measurements were done for two data size options, i.e. for 32-bit and 64-bit long values. The data size defines the maximum value of the sum of the aggregated values. It means that for a large number of aggregated values it is advisable to use 64-bit values. Additionally, the used data size depends on the individual values. In any case the 64-bit values provide more flexibility in both. The values given in table 3 are an average of 1000 measurements.

Table 3: The computational complexity of operations related to data encryption

Operation	Complexity for 32-bit numbers [clock cycles]	Complexity for 64-bit numbers [clock cycles]
Encryption	25	54
Decryption	26	55
Key generation	2’117	3’956

5 Conclusions

This document presents the technical details of the implementation of the security and privacy protocol for the energy management platform. It provides the API consisting of the basic set of functions (Java programming language methods) provided by the module to the services implemented in the energy management platform, as well as the functions (Java methods) available to the request processor in the communication platform. The first module allows preparing the data access request, to encrypt the data and to authenticate the request using a digital signature. The second block allows verifying that the source of the request is as declared and that the request processor is allowed to execute the requested access on behalf of the request issuer.

The implementation was evaluated with respect to the computational effort it induces. We have chosen a microcontroller based platform to investigate two aspects. First is the general computational overhead caused by the operations performed by the protocol, and the second aspect is the check of the ability to use the proposed approach on low power and energy constrained devices.

The evaluation led to the following observations. The proposed data encryption is very lightweight and does not cause too much computational effort. On the other hand, the authentication of the requests using the standard security approaches causes the protocol to be computationally too expensive to be used on low power devices – the generation of the signature takes on the microcontroller about 50 seconds. Here other approaches based for instance on short key solutions like the one proposed in [6], have to be used, but in that case the standard certificates cannot be used anymore. Anyway, the test platform was an extreme case and the target system will be providing by far better performance, it will be most probably from the BeagleBone Black or Raspberry PI class, where the signature generation takes about 50 milliseconds is thus, still in the acceptable range.

References

[1] K. Piotrowski, et al., “Deliverable D4.2 – Detailed security and privacy specification and implementation”, Public deliverable of e-balance project, FP7-Smartcities-2013, Project number 609132, 2015.

[2] K. Piotrowski, et al., “Deliverable D3.2 – Detailed System Architecture Specification”, Public deliverable of e-balance project, FP7-Smartcities-2013, Project number 609132, 2015.

[3] A. Sojka, K. Piotrowski, “lmRNG: A Lightweight Pseudorandom Number Generator for Wireless Sensor Networks.” in Proc. Of SECRYPT 2012, pp. 358-363.

[4] C. Castelluccia, E. Mykletun and G. Tsudik, “Efficient Aggregation of encrypted data in Wireless Sensor Networks”, in IEEE Mobiquitous, 2005.

[5] K. Piotrowski, St. Ortmann, P. Langendörfer, “Multi-radio wireless sensor node for mobile biomedical monitoring”, in Proc. BMT 2012 – 46th DGBMT Annual conference, pp. 725, Jena, Germany, September, 2012.

[6] A. Sojka, K. Piotrowski, P. Langendoerfer, "ShortECC-A Lightweight Security Approach for Wireless Sensor Networks," in Proc. *SECRYPT*, 2010.

