ProTego-ACC: Access control and key management for healthcare systems

Seyed Farhad Aghili, Dave Singelée
seyedfarhad.aghili@esat.kuleuven.be, dave.singelee@esat.kuleuven.be
imec-COSIC, KU Leuven, Belgium

December 15, 2021

Preserving the confidentiality of sensitive Electronic Health Records (EHRs), stored on a medical server, is an essential issue in healthcare systems. The system should have the means to avoid unauthorized users from accessing this sensitive information. This security problem has been tackled in the ProTego project, where an access control mechanism is developed as an integral part of a toolkit for data protection in healthcare. Without the appropriate access control mechanisms, it is impossible to protect the EHRs. This white paper discusses multiple technical approaches to provide access control and key management for healthcare systems.

1 Problem Statement and Background

Hospitals and medical care centers have to process and store sensitive medical data. This data is often stored in a Electronic Health Record (EHR). To ensure interoperability, one often stores medical data in a specific format: FHIR (Fast Healthcare Interoperability Resources). One of the research goals of the ProTego project was to design and develop a toolkit to protect and provide secure access to this data. One of the components in this toolkit is the access control and key management service (ProTego-ACC). In this section, first, we explain the security problem and then provide the necessary background on access control.

1.1 Problem Statement

The main application of the access control and key management service is to enhance the security of the Data Gateway (DGW). This component is the main interface for a user (denoted as data producer or data consumer) to read or write data from the system. Moreover, to protect the medical data, the DGW is also responsible for the secure storage of the data (i.e., store the data in encrypted format). The DGW first encrypts medical data that it receives from a data producer and only then stores it in external storage, for example a cloud system. When a data consumer wants to retrieve data from the DGW, the DGW will decrypt it before passing it to the data consumer. From a security point of view, the following three requirements should be fulfilled:

- Users should be able to only retrieve the plaintext data they are authorized for to receive (i.e. according to security policies defined in the system).
- All encryption/decryption keys need to be stored securely. So, an adversary that compromises the DGW would still not be able to decrypt all the encrypted data that is stored in the cloud storage.
- In scenarios where we assume the adversary can also compromise the ProTego-ACC component, (s)he should not be able to decrypt the EHR data. Same scenario when the DGW and access control component collude.

The main goal of the ProTego-ACC component is to realize the first two security requirements above. Moreover, we also investigated how the third requirement can be fulfilled as well. However, since mitigating the scenario where the DGW and ProTego-ACC increases the complexity of the access control solution, we first focus on the two first security requirements solely and only afterwards discuss access control solutions to tackle the third requirement as well.

1.2 Background

Traditionally, hospitals use either local servers or cloud systems to store EHRs data. Using some predefined access control rules, medical staff (employees in the hospital) can access this information. This access control model is static (defined once – used many times) in which a user with a specific role in the hospital is allowed to access part of the information. Such an access control mechanism typically uses a Role-Based Access Control (RBAC) model [1]. In the general RBAC model, the roles are assigned to the users and the permissions are assigned to the roles. Some constraints and role hierarchies are defined in other RBAC models such as RBAC1, RBAC2, and RBAC3. For example, the user can be a human and the role is a job function or job title like "doctor". The permission is an approval of access to one or more objects in the system (this can be the patient's EHRs). The authorized user that holds this permission is then allowed to perform some action(s) such as "read" or "write" on the object in the system. RBAC1 model is based on role hierarchies, including senior and junior roles such as doctor and nurse roles in hospital. For example, in this model, a doctor who is senior to a nurse inherits all permissions from the nurse. The doctor role in this model can also have extra permissions in addition to those inherited from the nurse. In RBAC2, the same user is not permitted to have two roles to avoid fraud attacks. Finally, RBAC3 is a combination of RBAC1 and RBAC2.
In the RBAC model, it is possible to have some dynamic rules that can be injected to the model at the decision point. However, these rules can be changed only by the administrator of the system, and it is not possible for the data owner to define his own rules(e.g., a user with role X can also access the current data).
ABAC Model:
Using smart devices introduces several new security and privacy challenges that RBAC cannot tackle in healthcare systems such as a "mobile user compromised" attack. Fortunately, the Attribute-Based Access Control (ABAC) model can be an alternative solution for RBAC. In ABAC, the user can only access the data if (s)he has all the attributes the data is associated with. In fact, in the ABAC model, the user that can satisfy some attributes such as "time," "location," "user identity," and "user role" can access the information. In the ABAC model, the data owner may define rules for his/her data consumers.

Due to the sensitivity of the healthcare records that are outsourced to the cloud, this information should remain confidential (i.e., information should be encrypted). So, only an authorized user can decrypt the data and access the information.
ABE as a Solution:
Integrating techniques such as Attribute-Based Encryption (ABE) [2] into ABAC makes it possible to encrypt the outsourced medical information and protect it from the cloud system. Even if a malicious insider gets access to the cloud system, (s)he would not retrieve the plaintext data. In ABE-based ABAC model, a data owner encrypts plaintext data using some attributes. In such a system, the decryption of encrypted data is possible only if the set of attributes of the data user's key matches the attributes of the encrypted data.

1.3 Outline

In Section 2, first, we explain the basic ProTego-ACC solution for managing and protecting access to confidential data. This basic solution is based on RBAC with static rules. Next, in Section 3, we further enhance the basic ProTego-ACC solution, and propose two additional schemes that offer protection against colluding DGW and access control components. These two extensions are briefly compared in Section 4.

2 High-level overview of ProTego-ACC

ProTego-ACC consists of several components that jointly realize this access control functionality. Moreover, these components interact with several other components in the ProTego toolkit, in particular with the DGW and an external Identity and Access Management (IAM) component. The latter is needed to know which user is requesting to upload or retrieve data from the DGW (through the application that connects to the DGW). A high-level view on the access control and key management architecture has been depicted in Fig.1

2.1 ProTego-ACC framework

Below, we first briefly discuss the basic ProTego-ACC framework. In the rest of this section, we then discuss the limitations of this basic solution. In the next section, we discuss our improvements to this solution.

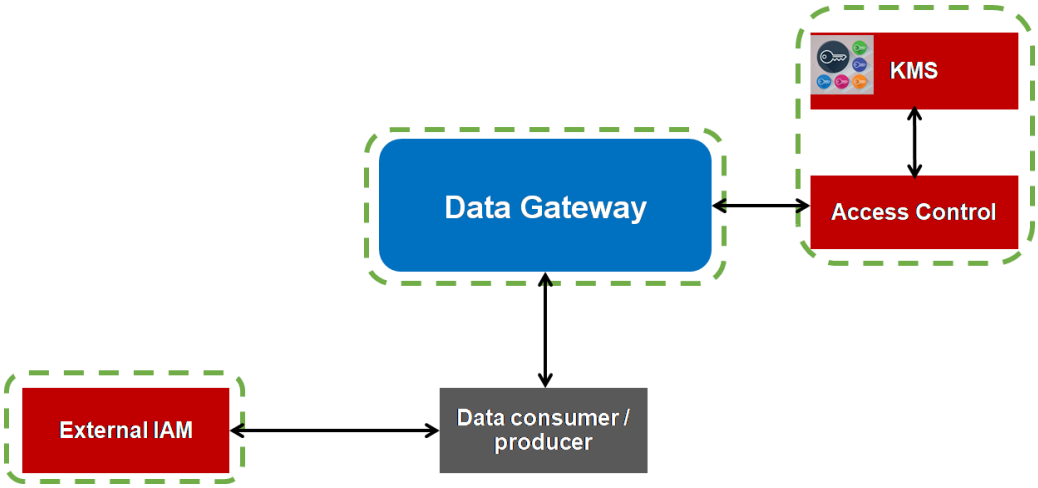

Figure 1: *High level view of the Access Control and Key Management architecture*

2.2 Functionality of the ProTego-ACC scheme

The ProTego-ACC framework is responsible for the authorization and Key Management Service (KMS) to enhance the DGW security. The main purpose of ProTego-ACC framework is to decide whether a data consumer is allowed to retrieve the plaintext data or not (authorization service). Let's look at how a data producer/consumer can upload/access (resp. read) the medical data.

As shown in Fig. 2, once the data producer wants to send the data to the DGW, it first authenticates itself to the external IAM component and obtains an authorization token (the IAM is responsible for managing the attributes of all system users and generating the authorization token based on these attributes and the expiration time). The data producer then sends the token to the DGW and all the medical data it wants to upload. The DGW now encrypts this data and sends the key encryption key (KEK) to the access control component. The KEK is then forwarded via the access control interface sub-component of ProTego-ACC to a key management service (KMS) where all the keys will be securely stored. Within the ProTego project, we have opted to use the open-source Vault [3] key management service (KMS). The KMS will use an internal key which is the master encryption key (MEK) to encrypt the KEK and the data producer's parameters (i.e. the content of the authorization token). This encrypted data is called a wrapped KEK. This wrapped KEK is then sent back to the DGW for storage along with the encrypted medical data.

As depicted in Fig. 2, when a data consumer wants to access the encrypted medical data, it first authenticates itself to the external IAM and obtains an authorization token. Then, it sends the authorization token to the DGW. The DGW relays this token to the ProTego-ACC component. It is important to note that the DGW, in addition to the authorization token, also sends the master key identity and the wrapped KEK associated with the medical data that the data consumer wants to retrieve to the ProTego-ACC component. At this point, the access control component checks the validity of the token (i.e., being a legitimate token that is not yet expired) and for which the medical data access is requested. Next, the control interface sub-component of ProTego-ACC forwards the outcome of this evaluation (token valid or not) to the Policy Decision Point (PDP) sub-component. The Policy Decision Point will need additional information (i.e. the data producer's parameters associated with the KEK) to make a proper security decision (i.e. to grant access or not). This input is processed in two stages. First, the Control Interface sub-component relays the wrapped KEK and the master key identity to the KMS to fetch the unwrapped KEK and the corresponding data producer's parameters (i.e. the owner's identity, the owner's role and the group to which the owner belongs). Second, the access control interface sub-component forwards these parameters to the Policy Decision Point. Based on all this information and applicable security policies, the Policy Decision Point will now be able to decide if access should be granted to the data consumer. This decision is based on pre-defined security policies that are stored in the

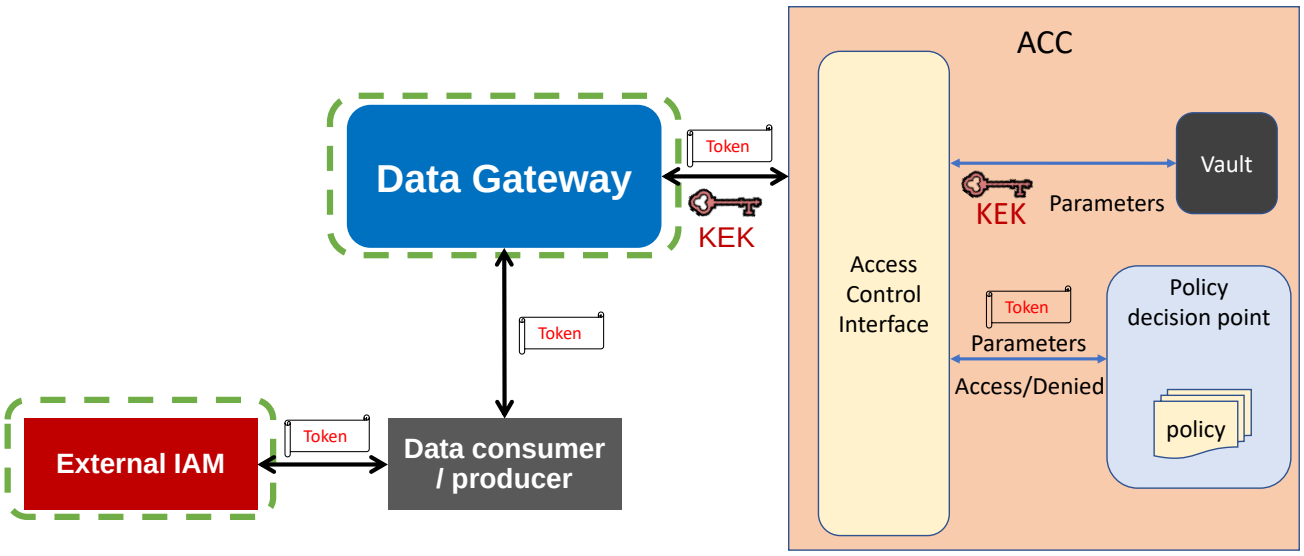

Figure 2: *ProTego-ACC architecture*

Policy Decision Point sub-component.

2.3 Limitations of the ProTego-ACC scheme

Although the basic ProTego-ACC framework provides the necessary security functionality, it also has some limitations. First of all, as mentioned above, the current authorization service is based on static security policies. Since the access policy is stored in the Policy Decision Point and also constant parameters are statically linked to the KEK, the policies cannot be automatically changed when the system evolves. This means that the current version of the access control framework (basic ProTego-ACC) does not meet the requirements for dynamic security policies (e.g. when the administrator of the system wants to change the policies). Therefore, we propose a new representation of a policy, called dynamic policy access control, which could be generated and bound dynamically to the wrapped KEK.

Moreover, if an adversary would compromise both the DGW and ProTego-ACC components, then it would be able to decrypt all the encrypted data. To improve upon this, the encryption/decryption keys need to be constructed based on not only secret parameters stored in the ProTego-ACC component but also secret parameters (e.g. secret key) from a non-colluding third party (for example the data producer itself). To this aim, we proposed an enhanced access control solution that relies on Ciphertext-policy attribute-based encryption (CP-ABE). In CP-ABE based scheme, the medical data first gets protected based on security policies defined by the data producer. Afterwards, the medical data gets an additional round of protection, based on the system's security policies. We also propose an alternative solution, the Cryptographic attribute-based access control (C-ABAC) scheme, that can provide user revocation functionality. An important property of the proposed revocation mechanism, as will be discussed later, is that all the non-revoked data consumers' secret keys will not need to be updated when a revocation event occurs. It is worth noting that in conventional revocation mechanisms for Attribute-Based Encryption, all encrypted data needs to be re-encrypted to prevent a revoked user from using its secret key for decryption. Thus, all other non-revoked data consumers will have to receive updated secret keys, which is not efficient at all, particularly in the context of a large-scale hospital ecosystem.

In the next section, we describe all our proposed improvements and novel access control solutions in detail.

3 Enhancements of the ProTego-ACC Scheme

This section presents an overview of our three proposed improvements: i) Dynamic policy attribute-based access control (D-ABAC) protocol, ii) Ciphertext-policy attribute-based access control (CP-ABAC) scheme and iii) Cryptographic attribute-based access control (C-ABAC) scheme. The D-ABAC protocol replaces our basic ProTego-ACC solution explained before. The D-ABAC scheme can then be further extended with either the CP-ABAC

protocol or the C-ABAC protocol.

3.1 Dynamic policy attribute-based access control (D-ABAC) protocol

To propose a new representation of a security policy, called dynamic policy access control, we enhanced our access control architecture by relying on the Open Policy Agent (OPA) [4] framework. OPA is an open-source engine that unifies policy enforcement across the stack. OPA provides a high-level declarative language (Rego) that lets us specify policy as code and simple APIs to offload policy decision-making from our access control component. When our component needs to make policy decisions it queries OPA and the supplies structured data (e.g., JSON) as input. Then OPA outputs policy decisions by evaluating the query input against the security policies and data. The policies that the admin of the system defines can be inspected and transformed using Rego. It is important to note that policies from several existing policy systems can be implemented with the OPA framework using the Rego language. Fig.3 shows an example of an Attribute-based access control (ABAC) realisation. With attribute-based access control, one makes policy decisions using the attributes of the users (e.g. a user being a doctor) involved in the access control request. In this example, "user_attributes" is an attribute set of the user. We could also have some dynamic attributes such as "tenure" (e.g., doctor joined the hospital more than 10 years ago) for the users. According to the policies defined in this example, the OPA will return "true" in the cases where i) the data consumer has the role of doctor, ii) the data consumer is the data producer, or iii) their identity was assigned to the KEK at the time of wrapping. In the latter case, the data producer must send these assigned identities (the 'input' shown in Fig. 2) to the access control component through the DGW.

```
package abac1
user_attributes :=  input.u_a
user_ids :=  input.u_i
default allow = false
# all doctors can access patient files
allow {
    # lookup the user's attributes
    user := user_attributes[input.user]
    # check that the user is a doctor
    input.role == "/Doctors"
    user.dp_role == "/patient"
    # check the action
    input.action == "read"
}
# doctor can access its own files
allow {
    # lookup the user's attributes
    user := user_attributes[input.user]
    # check that the user is a doctor
    input.role == "/Doctors"
    user.dp_role == "/Doctors"
    user.dp_id == input.user
    # check the action
    input.action == "read"
}
# expert access file
allow {
    # lookup the user's attributes
    user := user_attributes[input.user]
    # check that the user is an expert (more than 10 years)
    user.tenure > 10
    # check that the point is more than 5M
    input.point > 500
    # check the action
    input.action == "read"
}
# user assigned id
allow {
    # check ids
    user_ids[_].Pid == input.user
    # check the action
    input.action == "read"
}
```

Figure 3: *Rego example for ABAC*

High-level overview of the D-ABAC architecture: First of all, it is important to stress that our proposed D-ABAC solution replaces our initial basic access control framework. In the D-ABAC scheme, we rely on a new sub-component, denoted as the ABAC Policy Decision Point, to realize dynamic ABAC. As depicted in Fig. 4, the ABAC Policy Decision Point sub-components include the Token-based Policy Agent (TPA) and Open Policy Agent (OPA). It also supports updating the security policies.

Token-based Policy Agent (TPA): The TPA is based on a simple access control rule that was already defined in the Policy Decision Point sub-component of the basic access control framework. This simple access control rule states that the user that creates the data (i.e. data producer) always can access their own data (it means that the TPA evaluates the identity of the data consumer and will grant access to this user if the data consumer is the data producer). By using this simple access control rule, already many access control requests (i.e. the data producer's requests to access their produced data) can be handled. Therefore, TPA is a fast decision point that improves the efficiency of the overall access control system.

Open Policy Agent (OPA): In the cases that the requester (data consumer) is not the data producer, the ABAC Policy Decision Point sends the request (Query) to the OPA engine using specific Command Line Interfaces (CLIs). The Query which is the file with the JSON format includes the Token parameters, data producer's parameters, and data producer's input (note that parameters and input of the data producer are fetched from Vault, similarly as in the basic ProTego-ACC framework). OPA generates policy decisions by evaluating the query input and security policies stored in policy files.

Updating the security policies: When OPA

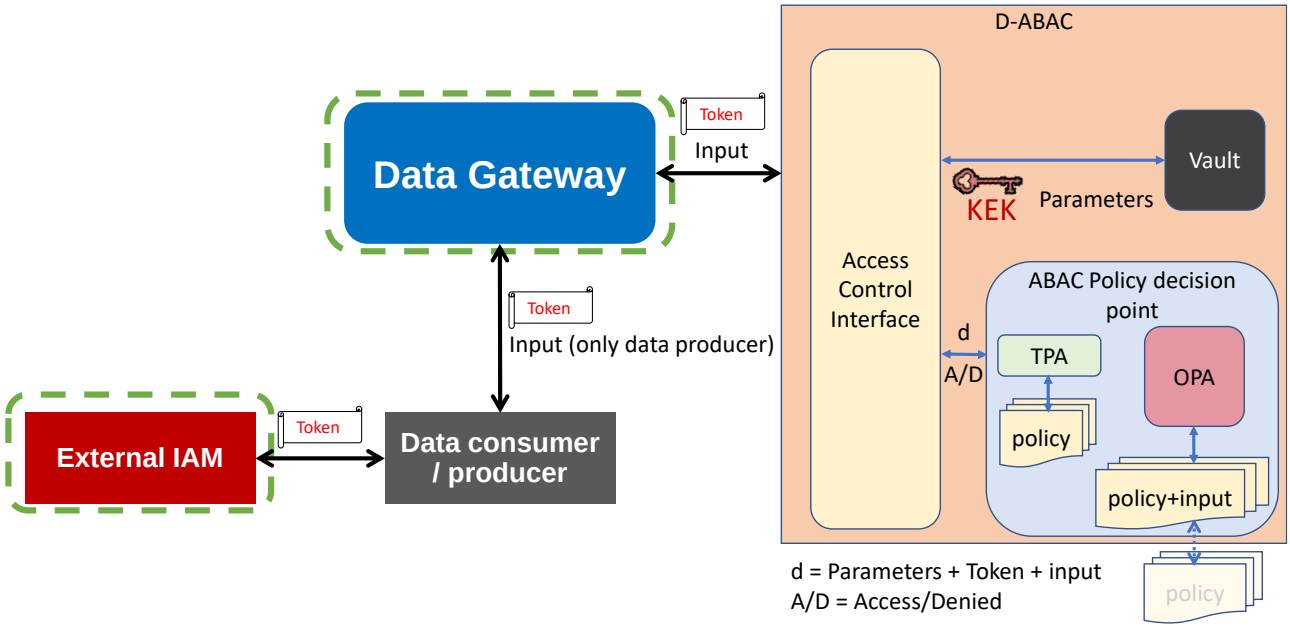

Figure 4: *Dynamic ABAC Protocol to access medical data*


```
{
    "user": "c7d84712-8720-46ff-862f-615262690db5",
    "role": "/Doctors",
    "action": "read",
    "point": 1000,
    "u_i": [
            {"Pid": "c34166f0-71ec-4c1d-88a6-65caa696daa6"},
            {"Pid": "dc32bbf7-5611-44ab-925e-3c84d25173b1"}
            ],
    "u_a": {
            "c7d84712-8720-46ff-862f-615262690db5":{
                "tenure": 5,
                "dp_id": "c34166f0-71ec-4c1d-88a6-65caa696daa6",
                "dp_role": "/patient"}
            }
}
```


Figure 5: *Example of input data (JSON format)*

starts for the first time, it will not have any security policies. Policies can be added, removed, and modified at any time. Since OPA accepts external data, the system will support dynamic policy access control. This dynamic property is based on external input and policy files.

Input file: Often policies require external data that is not available to the OPA. The query can include external data (necessitating of course that the policy is written accordingly). The Policy Decision Point acts as below:

- It sends a query to OPA including the external input data (Fig. 5 shows an example of the input data) and token parameters.
- OPA makes a decision based on the query and the external policy file.

Policy file: The policy file is expressed in the Rego language (see Fig. 3). This file is under the control of the security administrator of the system. The administrator can add or remove one or more policies inside this file at any time. The OPA will make its access control decision based on this file and the access control query. Fig. 6 demonstrates an example of a query including the name of external policy and input files.

```
./opa eval -i input.json -d abac1.rego "data.abac1.allow"
```

Figure 6: *Example of a query to OPA*

3.2 Ciphertext-policy attribute-based access control (CP-ABAC) scheme

3.2.1 Description of the protocol

Since both DGW and access control components are executed and managed in the same cloud environment, and therefore most likely managed by the same entity, an adversary who compromises this cloud environment would be able to obtain all necessary secret keys to decrypt all the encrypted medical data that is stored externally.

To mitigate this problem, one should only be able to retrieve the encryption/decryption keys by using secrets from the access control component and secrets from another external entity that does not collude with the DGW or access control framework.

For the latter, we will rely on the data producer itself. In summary, one can only retrieve the necessary keys to encrypt/decrypt the medical data when both the DGW/access control framework and the data producer contribute. None of these parties in isolation is able to get the encryption/decryption keys. To this end, we propose an enhanced access control scheme called CP-ABE. In CP-ABE, a ciphertext is encrypted with access policies over attributes called access structure. Any data consumer whose attributes satisfy the access policy can decrypt the ciphertext; otherwise, the decryption fails. Associating the access policy with the ciphertext means that the ciphertext chooses which key can recover the plaintext, giving the data producer more control of its outsourced data [5].

Generally speaking, three entities are responsible for running the CP-ABE scheme: (i) attribute authority, (ii) data producer, and (iii) data consumer. The role of the attribute authority is to generate system public and master secret keys and issue attribute secret keys to each data consumer based on their corresponding attribute list. A data producer chooses an access policy themselves and integrates that into the ciphertext. Then, the data consumer has the means for decrypting the ciphertext to retrieve the original data based on their attribute secret keys.

It is important to stress that the CP-ABE protocol is a security enhancement of ProTego access control framework. Therefore, it can be combined with either the basic ProTego-ACC solution discussed before in this white paper, or the D-ABAC solution (see Section 3.1). In the ProTego project, we have opted for the latter, as this provides the most functionality.

In our enhanced scheme, the medical data first gets encrypted based on security policies defined by the data producer. After this encryption, the result is then forwarded to the DGW. The rest of the process is similar as in the D-ABAC solution described in Section 3.1. When requesting access to the data, the DGW will now send an encrypted medical file to the data consumer (assuming that the access control framework granted access). The data consumer can then decrypt this data and retrieve the plaintext medical data when it has all the necessary attributes, as was defined by the security policies of the data producer. Note: In the enhanced scheme there are two types of polices; i) policies defined by the data producer, and ii) policies defined by the organization (e.g., hospital). The data consumer can access to the data iff they can satisfy both of these policies. Let us now look more into detail in this scheme.

High-level overview of the CP-ABAC architecture: As demonstrated in Fig. 7, our proposed CP-ABAC relies on attribute-based encryption (ABE) [2]. Integrating ABAC into the ABE technique provides us a possibility to encrypt the outsourced information (e.g. FHIR data) and protect it from attacks when both the DGW and access control framework are compromised. Using ABE, even if an adversary compromises both DGW and access control components, they would not be able to read the sensitive information. The algorithms and steps of the proposed CP-ABAC scheme are as follows.

CP-ABAC Algorithms: The scheme consists of six algorithms: Setup, AES-Enc, CP-ABE-Enc, ABE-KeyGen, CP-ABE-Dec, and AES-Dec.

Setup Algorithm: This algorithm generates a pair of public parameters (pp) and a master secret key based on the system security parameter and an attribute space. The IAM runs this algorithm, stores the master secret key, and publishes *pp*. Note: In the CP-ABAC scheme, one attribute authority administrates all system attributes. This authority has the master secret key that is used to derive all users' decryption secret keys. When CP-ABAC is combined with D-ABAC, the IAM plays the role of the attribute authority.

AES-Enc Algorithm: This algorithm picks a random key K_{AES}. It then uses this key and the AES encryption algorithm to encrypt the FHIR data. The output of this algorithm is the ciphertext CT_{AES}. The data producer runs this algorithm.

CP-ABE-Enc Algorithm: This algorithm is the encryption algorithm that takes as input the public parameters, K_{AES} as a message, and an access structure A over the universe of attributes (i.e. a security policy based on the attributes). The algorithm will encrypt K_{AES} and produce a ciphertext CT_{ABE} such that only a data consumer that possesses a set of attributes that satisfies the access structure will be able to decrypt the message (and hence retrieve K_{AES}). We will assume that the ciphertext implicitly contains A. The data producer also runs this algorithm. Note that we use CP-ABE-Enc algorithm to encrypt the encryption key rather than the FHIR data, for efficiency reasons. Note: access structure A will be defined by the data producer.

ABE-KeyGen Algorithm: The key generation algorithm takes as input the master secret key and a set of user attributes DC_{Att} that describe the key. It outputs the data consumer's private key K_{ABE}. The IAM runs this algorithm for each data consumer.

CP-ABE-Dec Algorithm: The decryption algorithm takes as input the public parameters *pp*, a

ciphertext CT_{ABE}, which contains an access structure A, and a data consumer private key K_{ABE}, which is a private key for the attribute set DC_{Att}. If the set DC_{Att} of attributes satisfies the access structure A, then the algorithm will decrypt the ciphertext and return K_{AES} as an output message. Using this output (i.e. this key), the data consumer can then decrypt the encrypted FHIR data (see algorithm below). The data consumer runs this algorithm.

AES-Dec Algorithm: This algorithm takes the key K_{AES} and the ciphertext CT_{AES} as input. It then uses this key and the AES decryption algorithm to decrypt the CT_{AES}. The output of this algorithm is the FHIR data. The data consumer also runs this algorithm.

Different steps within CP-ABAC: The steps of our proposed CP-ABAC scheme are described as follows. Note that steps 1 and 4 are only used for initialization, steps 2 and 3 for data storage (by the data producer), and steps 5 and 6 for data retrieval (by the data consumer).

Step 1 The IAM takes the security parameter and the attribute space and runs the Setup algorithm. The Setup algorithm returns the master secret key and public parameters *pp* of the system. At this point, IAM stores the master secret key and publishes the public parameters *pp*.

Step 2 Once the data producer generates the FHIR data, it picks the random key K_{AES}. And runs the AES-Enc algorithm to encrypt the FHIR data. The output of this algorithm is the ciphertext CT_{AES}.

Step 3 At this point, the data producer defines the access structure A over the universe of attributes and runs the CP-ABE-Enc algorithm to encrypt the key K_{AES}. The output of this algorithm is the ciphertext CT_{ABE}. After running this algorithm, the data producer determines the access rights to the encrypted message in access sets and grants access to the ciphertext only to a specific group of data consumers. Finally, data producer uploads the tuple (CT_{AES} , CT_{ABE}) to the DGW. From this stage onwards, the rest of the encryption and access control works exactly as before (see D-ABAC solution).

Step 4 During initialization, each data consumer needs to get the secret key corresponding to its attributes. Therefore, it will query the IAM. The IAM then runs the ABE-KeyGen algorithm and produces the secret decryption key K_{ABE} under the attribute set of the data consumer DC_{Att}. Finally, the IAM sends the secret decryption key K_{ABE} to the data consumer. It is worth mentioning that this step of the scheme can be performed offline and once.

Step 5 When the data consumer with a valid attribute set wants to access the FHIR data, it will communicate to the DGW and the access control framework, similarly as in the D-ABAC solution. When access was granted, the DGW will provide the data consumer with the tuple (CT_{AES} , CT_{ABE}). Using its secret key K_{ABE}, the data consumer performs the CP-ABE-Dec algorithm on CT_{ABE}. If the collection of data consumer attributes satisfies the access structure A associated with the ciphertext, then the algorithm outputs a decrypted message K_{AES}, else, decryption fails.

Step 6 The final decryption algorithm (AES-Dec) uses the key K_{AES}, retrieved in step 5, to securely decrypt FHIR data. This step is also performed by the data consumer.

3.2.2 Key Revocation in CP-ABAC

Since the data producer encrypts the data based on a chosen access structure and does not obtain the data consumer's certificate online, the data producer is not able to check whether the data consumer is revoked. The most popular solution to handle revocation in ABE-based access control systems is to re-encrypt the data each time there is a new access structure or to update the key. However, this type of method has several shortcomings. Several different data consumers might match the decryption policy. Updating the key might therefore force the IAM to maintain a large amount of private key storage, i.e. a key for every time period. Therefore, to provide the revocation functionality, we propose another attribute-based scheme called cryptographic attribute-based access control (C-ABAC), which we will discuss below.

3.3 Cryptographic attribute-based access control (C-ABAC) scheme

3.3.1 Description of the protocol

In this section, we propose an alternative solution for the CP-ABAC scheme described above. As mentioned, CP-ABE schemes have the limitation that they only have limited support for dynamic key revocation. Therefore, to provide key revocation, we propose a novel scheme denoted as cryptographic attribute-based access control (C-ABAC). In this scheme, we rely on the IAM to decrypt the encryption key that was chosen by the data producer. Note that similarly to the CP-ABAC solution, also the C-ABAC scheme is a security extension of the current access control framework (so either the basic ProTego-ACC framework or the D-ABAC solution).

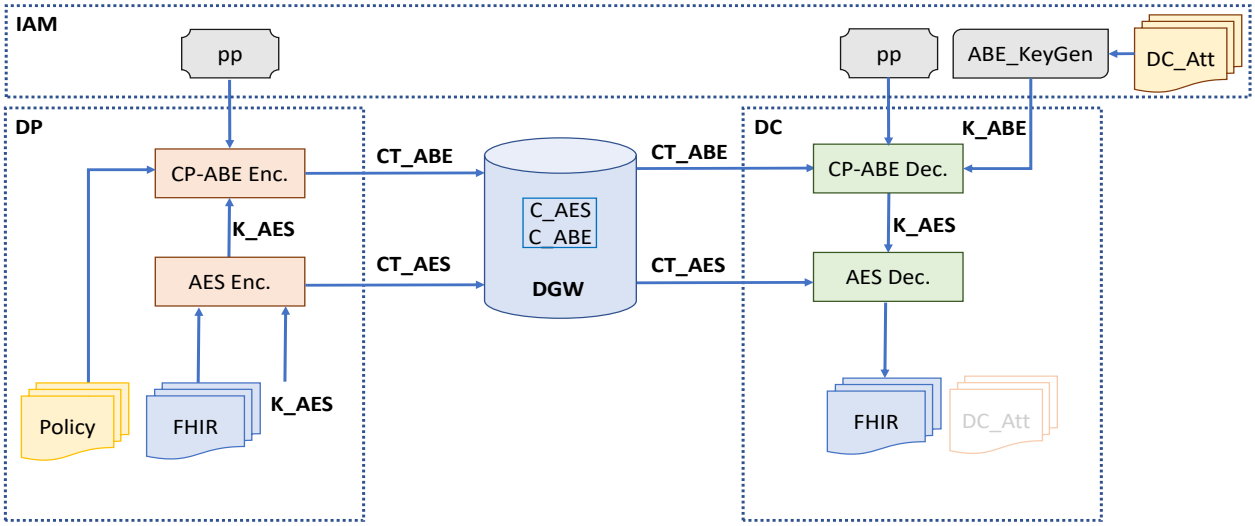

Figure 7: *Ciphertext-policy ABAC scheme to access medical data*

In the ProTego project, we opted to combine C-ABAC with D-ABAC.

High-level overview of the C-ABAC architecture: As shown in Fig. 8, our proposed C-ABAC scheme relies on encrypting the K_{AES} using the public key pk of the IAM. Thus, the decryption of the K_{AES} is now carried out in the IAM. The algorithms and steps of the proposed CP-ABAC scheme are as follows.

C-ABAC Algorithms: The scheme consists of six algorithms: Setup, AES-Enc, Public-key-Enc, IAM-PDP, Public-key-Dec, and AES-Dec.

Setup Algorithm: This algorithm generates the pair of a public key (pk) and a secret key based on the system security parameter. The IAM runs this algorithm, stores the secret key, and publishes pk.

AES-Enc Algorithm: This algorithm picks the random key K_{AES}. It then uses this key and the AES encryption algorithm to encrypt the FHIR data. The output of this algorithm is the ciphertext CT_{AES}. The data producer runs this algorithm.

Public-key-Enc Algorithm: This algorithm is the asymmetric encryption algorithm that takes as input the IAM public key pk, K_{AES} as the input message, and an access structure A over the universe of attributes (i.e. the security policy). The algorithm will encrypt K_{AES} and produce a ciphertext $CT_{pk} = Epk(K_{AES}, A)$. The data producer also runs this algorithm. Note: access structure A will be defined by the data producer.

Public-key-Dec Algorithm: The Public-key-Dec algorithm takes as input the master secret key and a message $CT_{pk} = Epk(K_{AES}, A)$. It decrypts the message and obtains the tuple (K_{AES}, A). Then it forwards the tuple to the IAM Policy Decision Point (IAM-PDP) algorithm. The IAM runs this algorithm.

IAM-PDP Algorithm: The IAM-PDP algorithm takes as input the tuple (K_{AES}, A) and outputs the private key K_{AES} to the data consumer iff the data consumer's attributes satisfy the access structure A. If these security policies are met, K_{AES} is given to the data consumer. The IAM also runs this algorithm.

AES-Dec Algorithm: This algorithm takes the key K_{AES} and the ciphertext CT_{AES} as input. It then uses this key and the AES decryption algorithm to decrypt the CT_{AES}. The output of this algorithm is the FHIR data. The data consumer runs this algorithm.

Different steps within C-ABAC: The steps of our proposed C-ABAC scheme are described as follows. Note that step 1 is only used for initialization steps 2 and 3 for data storage (by the data producer), and steps 4 to 6 for data retrieval (by the data consumer).

Step 1 The IAM takes the security parameter and runs the Setup algorithm. The Setup algorithm returns the secret key and public key pk of the IAM. At this point, IAM stores the secret key and publishes the public key pk.

Step 2 Once the data producer generates the FHIR data, it selects a random key K_{AES}. and runs the AES-Enc algorithm to encrypt the FHIR data using this key. The output of this algorithm is the ciphertext CT_{AES}.

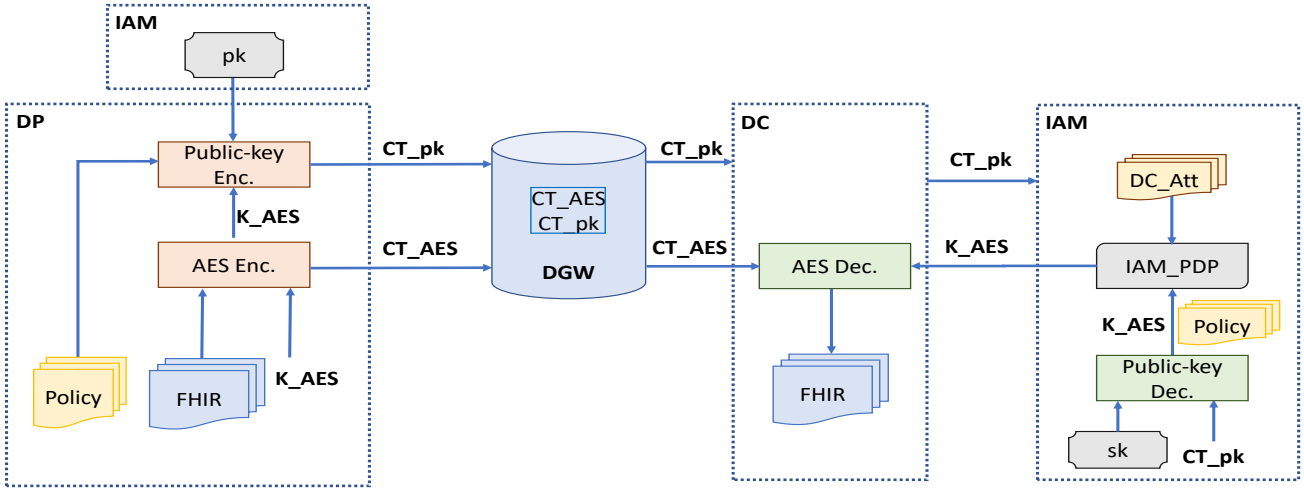

Figure 8: *Cryptographic ABAC scheme to access medical data*

Step 3 At the time of sending data to the DGW, the data producer defines the access structure A over the universe of attributes and runs the Public-key-Enc algorithm to encrypt both K_{AES} and A as $CT_{pk} = Epk(K_{AES}, A)$. Finally,data producer uploads the tuple (CT_{AES}, CT_{pk}) to the DGW. From this stage onwards, the rest of the data storage process is similar as in the D-ABAC solution.

Step 4 When the data consumer wants to access the FHIR data, it first communicates to the DGW and access control framework, similarly as in the D-ABAC solution. If access is granted, the DGW will send the tuple (CT_{AES}, CT_{pk}) to the data consumer. After receiving this tuple, the data consumer sends the ciphertext CT_{pk} to the IAM for decryption.

Step 5 Upon receiving the request, the IAM runs the Public-key-Dec algorithm to decrypt the message $CT_{pk} = Epk(K_{AES}, A)$ using the IAM secret key. Then it forwards the tuple (K_{AES}, A) to its internal IAM Policy Decision Point (IAM-PDP) algorithm. At this point, the IAM-PDP algorithm outputs the private key K_{AES} to the data consumer iff the data consumer's attributes satisfy the access structure A. Then, K_{AES} is given to the data consumer.

Step 6 The data consumer uses the decryption algorithm AES-Dec, using the key K_{AES}, to securely decrypt the FHIR data.

3.3.2 Key Revocation in C-ABAC

The IAM runs the IAM-PDP algorithm for each transaction. Since the IAM-PDP algorithm checks at runtime the dynamic access structure A and validates the data consumer's credentials online, the IAM will automatically detect when a data consumer is revoked, and will not provide a revoked data consumer the encryption key K_{AES}. In other words, the IAM performs an online revocation check on behalf of the data producer.

4 Conclusion

This white paper proposes four technical solutions to provide access control and key management scheme for healthcare systems used in the ProTego toolkit. The first scheme is the basic ProTego-ACC based on the traditional RBAC model. Although this model efficiently protects unauthorized users from accessing EHR data, the access control component decides based on some pre-defined static rules and is not robust against compromised attack. Three Enhanced ProTego-ACC schemes (i.e., D-ABAC, CP-ABAC and C-ABAC) are described in this white paper to mitigate these issues. D-ABAC is still based on RBAC but makes use of dynamic access control rules. The D-ABAC scheme can then be further extended with either the CP-ABAC scheme or the C-ABAC scheme. Both schemes are based on ABAC model and robust against an attack where the DGW and access control component would collude or be compromised. The strong points and shortcomings of both schemes are summarized in Table 1.

Acknowledgements

This work was financed in part by the European Union's Horizon 2020 Research and innovation program, under grant agreement No. 826284 (ProTego).

Table 1: *CP-ABAC extension vs. C-ABAC extension*

Scheme	Strong points	Shortcomings
CP-ABAC	• no need for extra computation on the IAM side (the IAM only runs the Setup phase that can be offline and once). • no need to add extra flow to the current scheme.	• does not support key revocation. • computational costs compared to C-ABAC on both data producer and data consumer sides (1 AES and 1 ABE encryption/decryption on both data producer data consumer sides).
C-ABAC	• efficient key revocation. • low computation costs on both data producer and data consumer sides (1 AES and 1 public-key encryption on the data producer side and only 1 AES decryption on the data consumer side).	• needs extra computations in the IAM-side (i.e., the IAM runs the Public-key-Dec and IAM-PDP algorithms). • needs an extra flow between data consumer and IAM (higher communication cost compared to CP-ABAC).

References

[1] R. S. Sandhu, Role-based access control, in: Advances in computers, Vol. 46, Elsevier, 1998, pp. 237–286.

[2] A. Sahai, B. Waters, Fuzzy identity-based encryption, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2005, pp. 457–473.

[3] Vault: open-source key management service, `https://www.vaultproject.io/`.

[4] OPA: Open Policy Agent, `https://www.openpolicyagent.org/`.

[5] J. Bethencourt, et al., Ciphertext-policy attribute-based encryption, in: 2007 IEEE symposium on security and privacy (SP'07), IEEE, 2007, pp. 321–334.