Securing Mobile Cloud Data with Personalized Attribute-based Meta Information

Sebastian Zickau, Felix Beierle, and Iwailo Denisow
Service-centric Networking
Technische Universität Berlin | Telekom Innovation Laboratories | Berlin, Germany
{sebastian.zickau|beierle}@tu-berlin.de, iwailodenisow@mail.tu-berlin.de

***Abstract*—With the spread of fast mobile Internet ä connections, such as 3G and LTE and the increasing processor power of mobile devices accessing cloud computing services on-the-go is common among all users. Sharing private information with friends and family members are options of popular cloud services, such as storage and social media services. But recent headlines show that the access to private information is often not sufficiently secured on the service level. The approach presented in this paper aims to use attribute-based meta-information to secure data on the level of files without relying on additional functionality of third-party services. A mobile device app is used to access and alter the meta-information. Attribute-based encryption mechanisms secure the private data and define access policies for friends and other users simultaneously.**

Keywords: Mobile Cloud Data, Attribute-based Encryption, Meta-Information, Personal Health Record

I. Introduction

Preceding the trend of small and big data in the cloud, the mobile IT revolution has already prevailed. With the use of mobile devices, such as smartphones, personal data is produced on-the-fly, e.g., photos, personal messages and notes. Often this data is only stored on the device itself. We call this data *mobile data*. There are applications, which give the user the possibility to share this data. Eventually it is stored in the *cloud*, using commercial services such as Dropbox[1] or Open Source solutions such as ownCloud[2]. We call this data *mobile cloud data* or rather *personal mobile cloud data*. Hereby the average users rely on the security mechanisms provided by the services. The loss of user control that comes with the introduction of such cloud computing solutions, is supported by the fact that companies target a large audience with their popular services. This audience needs to be addressed with an easy-to-use approach. The increasing popularity of these services underlines this requirement. The regain of control over users' personal data is cumbersome in most cases. Comparatively the encrypted e-mail, using technologies such as S/MIME [1] never really reached a wide user acceptance, even though S/MIME is a mature standard that is implemented in almost every e-mail client. These technologies are more accepted within economical or political markets. Unfortunately the introduction of additional security mechanisms in a mobile cloud data scenario seems to be a Sisyphean task. As a reaction to the census boycott protests by politicians and the German people in 1983 the German Federal Constitutional Court (BVerfG[3]) resulted in the *right to informational self-determination*[4]. The BVerfG ruled on December 15th, 1983:

> [...] in the context of modern data processing, the protection of the individual against unlimited collection, storage, use and disclosure of his/her personal data is encompassed by the general personal rights of the German constitution. This basic right warrants in this respect the capacity of the individual to determine in principle the disclosure and use of his/her personal data. Limitations to this informational self-determination are allowed only in case of overriding public interest.

In the English-speaking world the terms *data sovereignty* or *right to privacy* are often used in a similar manner, but can have varying semantics in different countries and cultures. Personal data is also protected under Article 8 of the Charter of Fundamental Rights of the European Union[5]. Simplified it means that the owners of the data/information have the right to decide who has read/write-access to their data and what are third parties allowed to do with this privilege. This right has been in place for more than 30 years and became the basis of data protection acts. It has also been a point in discussions since the rise of the Internet in the late 1990s and early 2000s, and became even more important with the rise of cloud computing services and big data applications in recent years. The sharing of personal data is based upon knowing the people you want to share your data with. This can be the relation of a friend or family member or someone in the same sports club. Either you know the person very well, e.g., you know the name of your friends or family members or you know persons by knowing that they are in certain groups and/or both of these attributes. Could there be a solution of securing personal data based on these known attributes? Another requirement is that you deal with your data on a mobile device. Is such a device able to handle the control you want to have as an owner of your personal data effectively? And the third question unfolding: Will you be in control to restrict access to your personal mobile cloud data at a later stage?

The article is structured as followed: Section II motivates the work and gives an overview of actors and roles as well as describing an example scenario. Section III presents the related work in the context of encryption techniques and applications with regards to health information. A design and architectural overview is given in Section IV. The implementation and evaluation results are presented in the two follow-up sections.

[1] https://www.dropbox.com/
[2] http://owncloud.org/
[3] German acronym for 'Bundesverfassungericht'
[4] The German legal term is 'Recht auf informelle Selbstbestimmung'
[5] http://www.europarl.europa.eu/charter/pdf/text_en.pdf

The article concludes with a summary and an outlook of future work in Section VII.

II. Motivation

In global trends studies about future developments in the digital world (e.g., [2]), the driving factor behind the rise of - especially mobile - services are made out to be social media and the growing number of information services. In more and more areas of their life, users expect to use cloud services. Ubiquitous availability of Internet access and faster broadband connection speeds, further support the prevalence of cloud computing services. Our research project follows the idea of offering a cloud service that enables the user to control who can access their information. Files should be encrypted, and in an (optional) *access history* it is tracked who accessed the file - with information about the access operation(s), if applicable. Further, we want to enable users to share only part(s) of the data that the file holds. A container file comprises meta-information, access history, and a set of files that are stored. We aim at researching and developing a generic system that works with personal files, medical records, (school, university) certificates, etc. In the following, we present the roles and actors for such a system, as well as an example scenario.

A. Actors and Roles

When considering the roles and actors, first, there is the owner of the file. In relation to the owner, we can establish several other roles, defined through private relationships:

- Legal guardian
- Partner
- Family
- Friends

Besides those roles based on private relationships, there can be professional relationships with actors out of four categories. We give example actors for each category that have a public or professional relationship with the owner of the file:

- Contractual partners
  - Landlord
  - Travel agency
- Governmental institutions
  - Police
  - Fiscal authorities
- Education
  - High school
  - University
- Health
  - Doctors
  - Health insurance

Regarding access functions to the file, we basically need CRUD (create, read, update, delete) operations. Write access enables actors to change their own records, and to append information to the file. Read access enables them to only read the file. The access history is written automatically, and it is only readable for the file owner (and legal guardian, if applicable). We follow a generic approach and provide adequate features for the listed actors. In order to discuss the proposed functionality in more detail, we present an example scenario.

B. Example Scenario: Personal Vaccination Card

For a user of the system, Alice, a container file is created. The container holds meta-information about the owner of the file, e.g., name, address, gender and DOB. In a file inside the container, Alice's vaccinations are tracked. She can access the file at any time and see an access history (who accessed when, what changes have been made). Alice can also access her son's file, because he is underage and Alice is his legal guardian. When tracking vaccinations, doctors - who administer vaccines or measure antibodies - need to have write access to the file, which Alice can grant them. Oftentimes, there are recommended - and sometimes mandatory - vaccinations when traveling abroad. For example, in Germany, the governmental Robert-Koch-Institute [6] makes recommendations for vaccinations when traveling abroad. In some countries in Africa, a yellow fever vaccination is mandatory, e.g., in Togo. While Alice is preparing for her trip through Africa, she visits a doctor and receives a vaccination against yellow fever. The doctor administers the vaccination and writes two files to Alice's container: One contains all the information about the vaccination, e.g., date, name of the doctor, or price of the vaccine. The second file contains information that is less sensitive, e.g., just name of the vaccination and the date of administration. When Alice is traveling through Africa and is on the border to Togo, a read access may be granted according to a location-based rule, e.g., in this border-region, any government personnel is granted read access to the electronic vaccination record to verify the compulsory vaccination. In this scenario, the border officer only needs to be able to read the second file with less sensitive information. To be able to ensure that the data record is about Alice, government personnel also needs to be able to access the owner information of the container file. Half a year after the trip, a scientist is conducting research about yellow fever. For a statistical analysis he asks to access information about vaccinations. Alice opts in by sharing her file. The scientist then is able to read her vaccination record with less sensitive information. For his research, he further needs information about Alice's age and gender, but not her name or address. Overall, we enable users of the proposed system to store sensitive information in the cloud while preserving privacy by providing fine-grained access control and security through attribute-based encryption.

III. Related Work

A. Attribute-based Encryption

Attribute-based encryption (ABE) was first introduced in [3] and [4]. The general idea is that to decrypt a file, one must have a set of certain attributes. Two possible approaches are ciphertext-policy (CP) ABE and key-policy (KP) ABE. A policy consists of logical conjunctions and/or disjunctions of attributes. An example for a possible policy could be that a user who wants to decrypt the file has to be a friend and be in Berlin. In CP-ABE, the ciphertext contains the policy that has to be fulfilled, and the users' *private keys* have attributes. In KP-ABE, the policy is contained in the users' private keys, and the files are encrypted with attributes. The majority of both research and software implementations of ABE focus on CP-ABE. The general process of CP-ABE can be described in four phases [5]:

[6] http://www.rki.de

1) **Setup** - An authority creates a *public key* and a *secret master key*.
2) **Key Generation** - With the *secret master key*, individual *private keys* can be generated for individual users.
3) **Encryption** - The data owner encrypts data with a *policy*, requiring only a *public key*.
4) **Decryption** - A user can decrypt an encrypted file with his/her *private key*, if the attributes satisfy the *policy*.

The authority is called *Attribute Authority* (AA). The Attribute Authority holds the secret master key and is able to generate private keys for users of the system. Additionally, it can provide further attributes, especially dynamic attributes, e.g., correlating to the location of a user.

B. Personal Health Records

A patient's information is most commonly stored in an electronic health record (EHR) at the doctor's office. Patients do not have access to the information stored about them. A more recent approach is the concept of a personal health record (PHR). The medical information is maintained by the patient. With the introduction of iOS 8 in 2014 and the included Health app provided by Apple and the official Google Fit app the PHR concept has been picked up by the major players in the mobile device and app market. The Health app can provide medical information, such as allergies and medication in case of an emergency to people who have access to the mobile's Lock screen. The app also can use medical and fitness data provided by other apps on the smartphone[7]. Traditionally vaccination information are kept in a yellow paper document provided by the World Health Organization (WHO). While paper-based PHRs are possible, recent research focuses also on cloud-based solutions ([6] [7]). Some studies focus on how to provide a fine grained access control to medical documents for researchers, hospitals, physicians, insurance company, etc. [8]. Some suggested PHR systems focus on one specific provider, e.g., Indivo [9] [10], whereas we pursue a generic approach that is not bound to a specific service provider or usage scenario. Akinyele et al. suggested a PHR system utilizing attribute-based encryption in [11]. A mobile client was implemented, which can be used to decrypt files provided by a hospital. We envision a system where the users are not limited to decrypting provided data, but can also create new files. The generic approach will not only enable the electronic vaccination card scenario, but also scenarios with sharing private files. The encryption ensures that the cloud storage provider hosting a container cannot access the stored content. Further, the idea is to provide an explicit access history to the file owner, and to introduce dynamic aspects, such as the current location of a user as attributes within an ABE scheme.

IV. Design and Architecture

There are currently two goals for the system. One is the introduction of a history function within the container format of the (mobile) cloud data files. The other is the implementation of personalized attribute-based meta-information in order to secure the mobile cloud data. The former one is used in scenarios in which the CRUD operations need to be audited after they were performed on a data file. The latter one is to protect the data itself during its usage or transfer. This section is divided into four parts. A brief overview of the data structure in the container is given, after that the functions of the current mobile application are explained. Subsection C describes the Attribute Authority. An overview of the whole architecture is given at the end of this section.

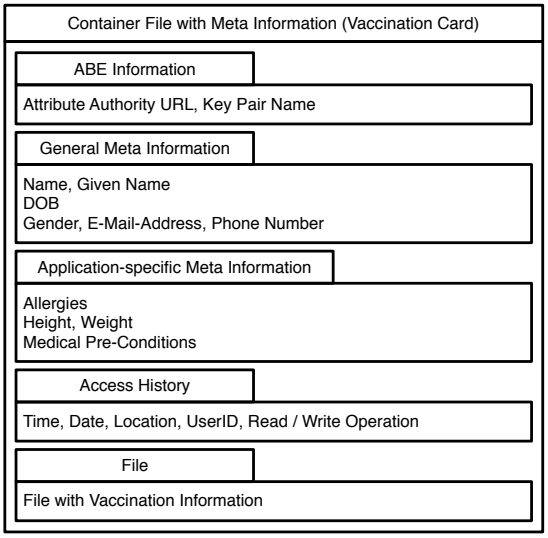

Fig. 1: Basic Overview of the Container File Structure

A. File and Metadata Structure

Figure 1 shows the different functional parts the container file is divided into. In this figure we relate to the vaccination card use-case scenario described in Section II. The file contains five different sub-parts, namely:

- **ABE Information:** The information, which is needed to perform encryption based on personalized attributes using ABE. The Attribute Authority URL and the key pair name are given (see Section V).
- **General Meta Information:** The file owner's information, such as name, DOB, and e-mail address, is stored within this part.
- **Application-specific Meta Information:** In our use-case scenario it contains information related to the owner's medical history, such as allergies, height, medication.
- **Access history:** As this example refers to a file type to which doctors, other medical stuff or border control officers need access to, this part stores information about the access history (CRUD) to the parts *general meta information* and *application-specific meta information*. This can be information, such as: time, date, user access ID, name and location.
- **File (Content):** This part consists of the data itself, in our cases the vaccination information of the file owner, i.e., patient.

[7]https://developer.apple.com/healthkit/

The container file also includes a signature to ensure integrity and user identification of the data described above.

B. Mobile Application

In our architecture, the client is a mobile device, i.e., a smartphone. In Section V the client application is realized on the Android platform. The mobile application should support the basic ABE functionalities as described in Section III. The following lists the needed functionalities of the mobile application:

- **Master Keys:** Creating and deleting master keys with a chosen name.
- **Private Keys:** Creating and deleting private keys with a chosen name. For the creation of private keys, attributes can be defined or chosen from a list.
- **Encrypt File:** A file can be chosen from a local folder or from a cloud service, such as ownCloud. To encrypt a file a policy is given, using the operators: *and*, *or*, $<$, $>$, and $=$. The attributes in the private key used to decrypt the file have to match this policy. The encrypted file is stored locally and can be transferred to its destination folder at a later time, using a separate third-party application in the prototype.
- **Decrypt File:** An encrypted file and a private key are chosen for decryption. This can be processed only if the attributes match the policy in the encrypted file.
- **Benchmark:** This function is used to evaluate the performance of different operations, e.g., *decryption* and *key generation*, as described fully in Section VI.

TABLE I: Functions of Attribute Authority

Anonymous Users	- Query list of public keys by name - Read all public keys - Create an ABE cipher to encrypt a file (given a policy) - Decrypt a file (given an ABE cipher and a private key)
Authorized Users	- Create key pairs - Delete owned key pairs - Display and download owned secret master keys - Create private keys for owned master keys

C. Attribute / Key Authority

The mobile application is connected to an ABE key authority. This authority manages public and private keys and is accessible via a REST application programming interface (Representational State Transfer API) [12]. The key authority can be accessed via this interface by authorized and anonymous users. The services the API provides differ depending on the user type, see Table I. The example scenario describes the situation in which an owner of cloud data needs to share certain information with an individual not previously known. But the file owner knows at least for which purpose the file sharing is needed. In our case the border control officer needs to verify that a traveler was given a certain vaccination before she is allowed to enter a specific region or country (yellow fever regulations).

D. Client and Server Architecture

Figure 2 shows the features of the different architectural components. The client application and the attribute authority were already discussed. The third-party services are exemplary client and server functionalities, which are provided by, e.g., Dropbox or ownCloud apps and services. These third-party applications are integrated into the prototype. The fourth box represents server capabilities within the project's boundaries and add needed functionalities such as dealing with the access history and signing and verification within the container file. The next section focuses on the implementation details of the ABE services.

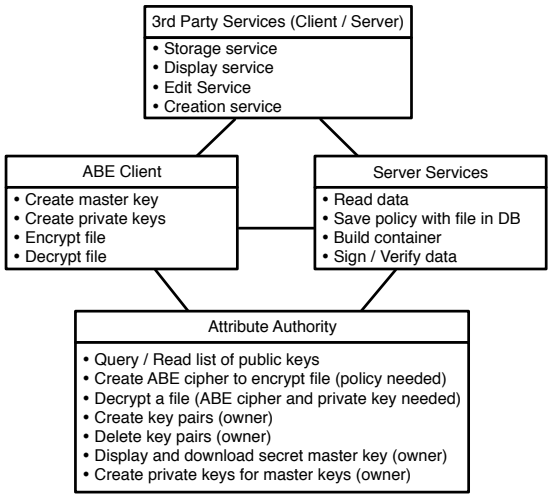

Fig. 2: Basic Client Server Functionalities

V. Implementation

The prototypes of the Android application and the server component have both been fully developed in Java. This was done, so that they could easily be deployed on most systems. The CP-ABE implementation relies on JPBC [13], which is a Java implementation of the Pairing-Based Cryptography library (PBC) [14]. On certain system architectures, it is possible to use JPBC as a wrapper for PBC, accelerating the most computationally expensive parts of the encryption scheme by running them natively. An already existing CP-ABE implementation [15] has been modified and extended to suit the requirements. In this implementation, a hybrid encryption scheme is used, where ABE acts as the key encapsulation scheme. This is a typical way to construct systems using asymmetric encryption, since the asymmetric part is normally very slow and should be used sparingly. The actual data is encrypted using a symmetric encryption algorithm. For this purpose, AES is used. The primary target platform for the client application has been Android, but we've identified tools (e.g. RoboVM[8]) with which it should be possible to use the existing code on iOS devices.

A. Android Application

The application has to be able to fulfill the four basic functions required by ABE and additionally make it as easy

[8] http://www.robovm.com/

as possible to create and share files as well as keys. As such, it is possible to create new key pairs, encrypt files, decrypt files, and generate private keys. These operations can be very time consuming, because they are computationally expensive. Specifically the ABE part of the encryption and decryption as well as the key generation need a lot of CPU power. Since JPBC has no support for running the pairing-based computations natively on ARM devices (and computation on ARM devices would most likely still be too slow to actually use the application), the load of the mobile device is reduced by performing these parts on the server component. The assumption is that the time required on the server to execute the various operations is considerably lower, due to the server being generally more powerful and being able to use JPBC as a wrapper for PBC. With the server doing the bulk of the computation, the only remaining part for the mobile device is to encrypt/decrypt the file with the AES key. AES is reasonably fast, and runs natively on Android devices. The assumption is that the server component is trusted. While this is true for the architecture described, it is not necessarily the case in other applications. Even if the server component cannot be trusted it is possible to reduce the load on the mobile device when outsourcing the decryption [16][17]. Each private key saves the information on which server it was produced. When a user is trying to decrypt a file, the application will offload the ABE decryption to the server where the private key was produced (and where consequently the master key is held). This is a sensible approach, since the server is already trusted. It is also possible to specify other servers to use for offloading parts of the encryption and decryption, but since the private key is transmitted to them, it is required for them to be trusted. Data usage for the remote en- or decryption is very low. For the encryption, only the policy and the public key need to be transmitted, whereas for the decryption, only the ABE cipher and the private key are needed. The actual file never has to leave the device for either operation. In the case of having no Internet connection or a very unstable connection, there is a fallback option to do the en-/decryption locally, even if this might take a while. The private keys are stored on the device itself. When the user wishes to encrypt a file, he must first choose a public key to use. In the next step he needs to specify a policy. The Android application features an interface for building these policies, to make that process easier. With this, it is possible to specify policies that can later be reused as sub-policies. The file can then be encrypted. Depending on whether an Internet connection is available, the ABE part can be offloaded.

B. Server Component

The server component was written as a Java servlet and has been tested to work with Apache Tomcat. It manages the public and master keys and regulates the access to them. This is done with a simple user management system. Only two groups of users are distinguished in the system: an authenticated user and an anonymous user. Due to the nature of ABE, everything not concerning the secret master key can be done by an anonymous user. The abilities that anonymous users and authorized users are able to perform were described in Section IV. All these operations can be accessed via a REST interface. The serialization/deserialization of the various ABE/jPBC related classes has been implemented using custom serializers for Gson. The authentication has been implemented as a servlet filter to allow for custom authentication schemes. The key pairs and all other user data are being stored in an SQLite database. SQLite has been chosen since it is easy to deploy, lightweight, and has all the features needed. During development, the question arose whether or not to use stored procedures for performance gains due to decreased communication, but since the database performance only had a negligent impact on the overall time needed to execute the REST functions, it was ultimately decided to keep using SQLite. However, this is something that may change when the database is not running on the same machine. Due to the experimentation with different RDBMSs some changes have already been made that would make a transition to another RDBMS easier.

VI. Evaluation

In this section, we present our evaluation results for the current prototype. In the previous section, we established that there are cases in which the en- and decryption - and for a fully mobile application even the key generation - need to be performed on the mobile device. ABE is normally used in a hybrid encryption scheme. We use AES as the symmetric part of it, as described in section V, but in the tests only the asymmetric part of the decryption process has been measured. In the following, we evaluate the decryption process in dependency of the complexity of the policy, as well as the key generation. We ran tests with different devices, two mobile smartphones running Android, Nexus 4 (using a Qualcomm Snapdragon S4 Pro APQ8064 at 1.5 GHz) and Nexus 5 (using a Qualcomm Snapdragon 800 at 2.26 Ghz), and, for comparison, a laptop PC running an Intel i5-2410M CPU at 2.30 Ghz. Each test was run five times and each data point in the following figures indicates the average of these five runs. The performance results for the decryption process and the key generation can give an indication of the usability of the application.

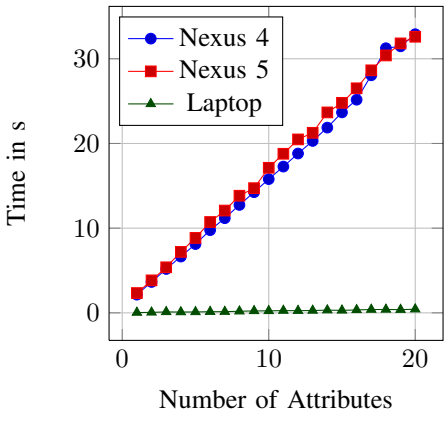

Fig. 3: ABE Decryption (policies with *and*)

Figure 3 shows the decryption times in relation to the number of attributes when using a policy with only logical conjunctions of attributes. Figure 4 shows the decryption times in relation to the number of attributes when using a policy with only logical disjunctions of attributes. For the policy with only conjunctions, the complexity seems to rise in a linear fashion.

The mobile devices took almost twice the time for twice the attributes. For policies with attributes connected by *or*, the time seems to be constant for the Nexus 4. The varying times for the Nexus 5 suggest that there might still have been background tasks running at the same time. This result matches with the expectation, that when only one attribute is required to satisfy the policy, then the length of the policy does not seem to have an impact on the decryption time. For both results, the laptop is much faster and only takes an insignificant amount of time, because it can run the ABE code natively.

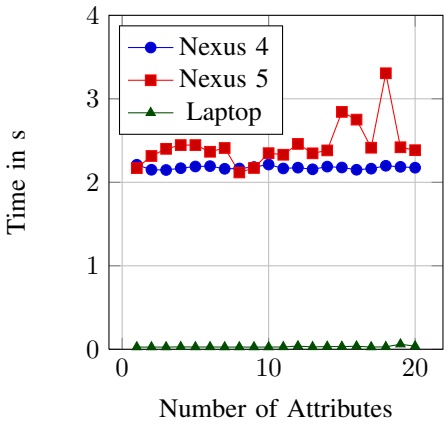

Fig. 4: ABE Decryption (policies with *or*)

As for the key generation, the mobile devices perform almost equally well (no figure). A good rule of thumb seems to be that the generation takes a second for each attribute in a key. The laptop again outperforms the mobile devices. Overall, the performance of the ABE system seems to be usable for mobile devices, although outsourcing parts of the decryption should alleviate most of the problems that would be encountered with more attributes. Further research will show, how many attributes will be necessary in a common use case.

VII. Conclusion and Outlook

The work shows that the prototype ABE implementation of the project provides usable results within the described scenario. As processor power increases on mobile devices and implementations of cryptographic functions might be included on the hardware layer, the evaluated results can certainly be improved in the future. The introduction of server functionality can also support the usage of ABE in mobile cloud data scenarios. A solution to the question on how to handle key distribution within an attribute-based encryption scenario needs to be included in future versions of the client and server side software. The basic scenario presented also works within an only-mobile-clients setup. The inclusion of storage services, such as Dropbox, and added additional components in open distributions, e.g., ownCloud, will be addressed in the next prototype. The described access history functions and the fully developed container is the next step to be integrated, as well as the introduction of dynamic ABE attributes, such as location information.

Acknowledgment

The work presented in this paper was performed in the context of the *Curcuma*[9] project within the *Software Campus*[10] program. The project is funded by the German Federal Ministry of Education and Research (*BMBF*)[11].

References

[1] B. Ramsdell, "S/mime version 3 message specification; cryptographic message syntax," RFC 2633, June 1999. Housley, R.," Cryptographic Message Syntax", RFC 2630, Tech. Rep., 1999.

[2] Münchner Kreis e.V., "Innovationsfelder der digitalen Welt. Bedürfnisse von übermorgen." 2013.

[3] A. Sahai and B. Waters, "Fuzzy Identity-Based Encryption," in *Advances in Cryptology EUROCRYPT 2005*. Springer, 2005, pp. 457–473.

[4] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 89–98.

[5] C.-C. Lee, P.-S. Chung, and M.-S. Hwang, "A Survey on Attribute-based Encryption Schemes of Access Control in Cloud Environments." *IJ Network Security*, vol. 15, no. 4, pp. 231–240, 2013.

[6] M. Jafari, R. Safavi-Naini, and N. P. Sheppard, "A Rights Management Approach to Protection of Privacy in a Cloud of Electronic Health Records," in *Proceedings of the 11th Annual ACM Workshop on Digital Rights Management*, ser. DRM '11. New York, NY, USA: ACM, 2011, pp. 23–30.

[7] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou, "Scalable and Secure Sharing of Personal Health Records in Cloud Computing Using Attribute-Based Encryption," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 131–143, Jan. 2013.

[8] M. Li, S. Yu, K. Ren, and W. Lou, "Securing Personal Health Records in Cloud Computing: Patient-Centric and Fine-Grained Data Access Control in Multi-owner Settings," in *Security and Privacy in Communication Networks*. Springer, 2010, pp. 89–106.

[9] The Indivo Personally Controlled Health Record. Accessed: 2015-01-31. [Online]. Available: http://indivohealth.org

[10] C. Wang, X. Liu, and W. Li, "Implementing a Personal Health Record Cloud Platform Using Ciphertext-Policy Attribute-Based Encryption," in *4th International Conference on Intelligent Networking and Collaborative Systems (INCoS) 2012*, Sep. 2012, pp. 8–14.

[11] J. A. Akinyele, C. U. Lehmann, M. D. Green, M. W. Pagano, Z. N. J. Peterson, and A. D. Rubin, "Self-Protecting Electronic Medical Records Using Attribute-Based Encryption," *IACR Cryptology ePrint Archive*, vol. 2010, pp. 565–584, 2010.

[12] R. Fielding, "Representational state transfer," *Architectural Styles and the Design of Netowork-based Software Architecture*, pp. 76–85, 2000.

[13] A. De Caro and V. Iovino, "jPBC: Java pairing based cryptography," in *2011 IEEE Symposium on Computers and Communications (ISCC)*, Jun. 2011, pp. 850–855.

[14] B. Lynn, "On the implementation of pairing-based cryptosystems," Ph.D. dissertation, Stanford University, 2007.

[15] J. Wang. (2013) CPABE. [Online]. Available: https://github.com/junwei-wang/cpabe

[16] M. Green, S. Hohenberger, and B. Waters, "Outsourcing the decryption of ABE ciphertexts," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 34–34.

[17] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption," in *Proceedings of the 9th International Conference on Theory of Cryptography*, ser. TCC'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 422–439.

[9] http://curcuma-project.net/

[10] http://www.softwarecampus.de/, (Förderkennzeichen 01IS12056), The authors are responsible for the content of this paper.

[11] http://www.bmbf.de

Additional Information

Bibliographic Data S. Zickau, F. Beierle, and I. Denisow, “Securing Mobile Cloud Data with Personalized Attribute-based Meta Information,” in *Proceedings of the 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. IEEE, 2015, pp. 205-210.

Pre-print from https://beierle.de

Online at http://dx.doi.org/10.1109/MobileCloud.2015.14

Authors Sebastian Zickau

Felix Beierle

Iwailo Denisow

BibTeX

```
@inproceedings{Zickau2015MobileCloud,
title = {{Securing Mobile Cloud Data with Personalized Attribute-based Meta Information}},
author = {Zickau, Sebastian and Beierle, Felix and Denisow, Iwailo},
booktitle = {{Proceedings of the 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)}},
publisher = {IEEE},
year = {2015},
pages = {205-210},
doi = {10.1109/MobileCloud.2015.14}
}
```