7-1-2017

ShareABEL: Secure Sharing of mHealth Data through Cryptographically-Enforced Access Control

Emily Greene
Dartmouth College

ShareABEL: Secure Sharing of mHealth Data through Cryptographically-Enforced Access Control

Thesis by Emily Greene
Advisor: David Kotz
Dartmouth Computer Science Technical Report TR2017-827

ABSTRACT

Owners of mobile-health apps and devices often want to share their mHealth data with others, such as physicians, therapists, coaches, and caregivers. For privacy reasons, however, they typically want to share a limited subset of their information with each recipient according to their preferences. In this paper, we introduce ShareABEL, a scalable, usable, and practical system that allows mHealth-data owners to specify access-control policies and to cryptographically enforce those policies so that only parties with the proper corresponding permissions are able to decrypt data. The design (and prototype implementation) of this system makes three contributions: (1) it applies cryptographically-enforced access-control measures to wearable healthcare data, which pose different challenges than Electronic Medical Records (EMRs), (2) it recognizes the temporal nature of mHealth data streams and supports revocation of access to part or all of a data stream, and (3) it departs from the vendor- and device-specific silos of mHealth data by implementing a secure end-to-end system that can be applied to data collected from a variety of mHealth apps and devices.

CCS CONCEPTS

• **Security and privacy → Access control**; *Mobile and wireless security*; *Privacy protections*;

KEYWORDS

mHealth, mobile health, access control, data sharing

1 INTRODUCTION

Mobile health (mHealth) apps and devices like smartwatches, smart scales, and insulin pumps generate a large amount of data. This data may be helpful for patient care, enabling healthcare providers to monitor patients more effectively and tailor their treatments accordingly. Unlike Electronic Medical Records (EMRs), where data is primarily collected during in-person appointments, mHealth devices collect data continuously and throughout normal life activities, giving providers a more complete picture of the patient's health. Researchers can also use mHealth technologies to measure subjects' physiology, behavior, and activities in natural free-living conditions, deepening their understanding of how behavior and environment affect health or exploring the efficacy of experimental health-related interventions. By combining data from multiple devices into one easily accessible repository, with strong controls in the hands of the health subject, we can facilitate these data-sharing use cases and more.

Many systems deployed today incorporate security as an afterthought. With security breaches and data leaks becoming more frequent, it is imperative that sensitive medical data is securely stored and shared. Medical data systems are especially attractive to cyber-criminals and those with criminal intent: a wealth of sensitive personal health information can be exfiltrated to sell on the underground market or used to extort ransoms [13]. While many security solutions currently exist to encrypt and protect data, a more granular approach is necessary in this situation. An mHealth-device owner may not want to share all of her mHealth data with each of her providers. There are data that are applicable and necessary to be viewed by her cardiologist, but she may not want her physical therapist to see the same information. Studies have shown that patients want granular privacy control over which health information their doctors can access [4]. mHealth data owners need to not only protect their data from outsiders, but also share only pertinent data with each of their specified data consumers.

Therefore, device owners need a solution that is both secure and flexible. *Owner-driven access control* [3, 19] allows owners to optionally share subsets of their mHealth-device data with others, and these access-control policies may change over time. Patients may change providers and revoke access to the data, or they may reevaluate which data they would like to share with which providers. This thesis describes a system that delivers a high level of data security and flexibility to the data owner in determining who has access to which data from his apps and devices. Our system, ShareABEL, will help healthcare professionals treat patients without sacrificing the security of patient data.

With ShareABEL, we focus solely on individuals' mHealth data. We envision that each mHealth app or device produces data on a regular basis, resulting in an ongoing *data stream* of a specific type of data. Such data streams may include heart rate, blood-glucose levels, or data associated with specific exercises (e.g., force produced on a gym weight). We assume that data flows from one or more mHealth apps or devices, through ShareApp (the data owner's smartphone app for ShareABEL), and into ShareBase (a cloud storage system). There it can be accessed by various data consumers, with ShareView, according to policies set by the data owner.

In our model, we think of access control on two axes: (1) whether access has been granted to a data type (such as heart rate), and (2) whether access has been granted for the time the data was collected. To provide access control by type and by time period, we use a combination of two different approaches: cryptographically-enforced access control (to limit access to only deserving recipients) and a pseudo-random indexing scheme (to constrain the time periods for which recipients can access the data stream). These two methods complement each other.

Our contributions. In this paper, we develop a system that seeks to accomplish three goals. First, we apply cryptographically-enforced access control measures to mHealth data. Much of the health-data literature related to access control has concentrated on EMRs, where fine-grain access control consists of restrictions on discrete resources within individual patient records (e.g., images and appointment notes). Most mHealth data, on the other hand, are collected outside the clinical context and are structured as ongoing data streams, collected and uploaded over time. Therefore, access control of this data must address the temporal nature of data streams, especially at the frequency and rate that mHealth data is produced.

Thus, our second contribution is temporal access control, a means to temporally segment each data stream using a pseudorandom indexing scheme, enabling owners to share the entire data stream or only time-specific subsets of the stream (e.g., data collected prior to a specific date).

Finally, we seek to depart from the vendor- and device-specific silos of mHealth data by implementing a scalable, usable, and secure end-to-end system that can be applied to data collected from a variety of mHealth apps and devices. In the current mHealth space, each app or device has its own vertical ecosystem, which can be inefficient for people with multiple devices, and which misses opportunities for aggregation of data across devices to provide a more-complete picture of an individual's health. We are able to break down the barriers between silos by having a system that compiles data from multiple devices on an individual's smartphone before sharing the data with others through a common cloud database.

1.1 Organization

We begin by giving necessary background information and pertinent definitions in Section 2. We continue with a discussion of related work in Section 3. We describe use cases for our system in Section 4 and the security model on which our system is based in Section 5. We then present our solution, especially centered around the access control and revocation support, in Section 6. We detail our implementation in Section 7. We follow with an evaluation of the system in Section 8. We discuss limitations to our system and interesting extensions in Section 9, and finally conclude with Section 10.

2 BACKGROUND

Cryptography has long been used to protect sensitive data from adversaries. In this system, we seek to not only use cryptography as a safeguard against data exfiltration, but also as a means to provide access control. There has been significant work on the use of cryptography as an access-control mechanism, enforcing access to content by encrypting that content and sharing keys only with permitted users [12, for example].

2.1 Definitions

For clarity we define the key terms we use in this paper. There are two types of human actors that use our system: data owners and data consumers. A *data owner* is the person who produces the mHealth data measured by the app or device. Even if the collection device itself is owned by a third party, such as a doctor or employer, our focus here is on the data. In some related literature, this person is referred to as a *patient*, but we envision a broader range of use cases such as a subject in a research study sharing the data with the researcher, a professional athlete with a trainer, an elderly parent with an adult child, and so forth.

A *data consumer* is an individual or entity with whom the data owner would like to share information (e.g., doctor, physical therapist, coach, family, researcher, laboratory, insurance company). We assume that the owner knows the consumer personally (and in the case that the consumer is an organization, a personal knowledge of a representative of the organization), and that there is an opportunity for secure exchange between data owner and data consumer (Section 6.3).

These two parties intend to share mHealth data. We consider a *data point* as one individual measurement (e.g., one weight reading, one heart-rate measurement computed over a 15-second interval, or the step-count for a 5-minute interval), and a *data stream* is a sequence of data points from a single data source, measured on a periodic basis and sent from the source as they occur, or in batches. These data streams vary in frequency and their data points vary in size. Some streams produce new data points frequently (e.g., step counts, measured from 100Hz accelerometer data) and others rarely (e.g., weight, typically measured a few times a week). Some records are small (e.g., a single blood-pressure measurement) and some are large (e.g., photos of changes in wound healing). Therefore, our solution must support fine-grained access control regardless of the size, type, or frequency of data.

Each data stream has one *source* – an mHealth app or mHealth device; but a source that produces multiple types of data (e.g., heart rate and step count) would produce multiple data streams. Each of these streams has only one *data type*, which is used for the most granular level of access control within the system (e.g., heart-rate data collected by two different devices or two different applications on the same device are all controlled under the "heart-rate" data type). We group data types into *data categories* based on similarities (e.g., heart-rate and blood-pressure data both fall within the "cardiology" category). Default categories are set by a data configuration file (see figure in Section 7), and data owners can set custom categories if desired.

For data owners to share data with various consumers, that data must be stored in a *database*, an external, untrusted (honest but curious) server that hosts data from a variety of owners. We refer to each entry in the database as a *record*. We assume each record corresponds to one or more data points in the stream, and is encrypted separately. In the case of particularly small or frequent data points, one record may hold multiple data points, grouped into temporally contiguous batches for efficiency. In the case of particularly large data points, they may either be fragmented into multiple records, or stored with a layer of indirection (the record would be equivalent to a pointer to a larger encrypted blob), for efficiency. Records are immutable – once produced, they are never changed or deleted. To retrieve a record from the database, a data consumer must provide the *index*, or unique code, corresponding to that record.

Our goal is to enable data owners to specify consumers' *access*, the permission granted by a data owner to a data consumer to view the plaintext of a specified record (read-only). An *access-control policy* is a set of permissions authored by the data owner that specifies which records each data consumer is able to read. *Revocation* of access occurs when there is an access-policy change that narrows or restricts a consumer's access to a specific data stream (i.e., there exists at least one record in the data stream (past or future) to which the consumer used to be allowed access and now is denied access). Therefore, revocation can be as narrow as a change in the access to one record, and can be as broad as the revocation to the entire data category. Our system design and underlying implementation thus support tremendous flexibility and granularity – recognizing that the design of an interface suitable for human data owners to author these policies remains an important HCI challenge worthy of future work [19, 20].

2.2 Attribute-Based Encryption

To address the "all or nothing" problem of traditional encryption systems, Sahai and Waters introduced the concept of attribute-based encryption (ABE) [21]. ABE allows ciphertexts and keys to be created with specific *attributes* and *policies* attached. In an ABE system, any string or numeric value can serve as an attribute. Attributes serve as tags or descriptors of the underlying data. Policies are Boolean formulas over these attributes that specify access to data tagged with these attributes. Policies can be expressed with AND, OR, threshold gates (e.g., m-of-n), and relational operators (less-than, greater-than, etc.) [1]. There are two different types of ABE that depend on the access control desired: Key-Policy ABE provides content-based access control, and Ciphertext-Policy ABE provides role-based access control. In ShareABEL, we use Key-Policy ABE.

In *Key-Policy ABE (KP-ABE)*, ciphertexts are labeled with sets of descriptive attributes, and a user's key can only decrypt a ciphertext if the key's policy matches the attributes of the ciphertext [11]. KP-ABE provides content-based access control, as the attributes are tied to the content (the ciphertext) and the user's private key is associated with a policy over these attributes [26]. For example, consider an athlete who desires to share with her coach some subset of exercise data. The attributes on data obtained while working out could include: `type:heart-rate`, `location:gym`, `exercise-intensity:high`, and the coach could be issued the following key:

```
(type:heart-rate AND (exercise-intensity:high
     OR (exercise-intensity:medium AND location:gym)))
```

With this key, the coach can only decrypt heart-rate data collected during exercise (any high-intensity activity or medium-intensity activity that happened at the gym). (The specifics of the syntax vary by implementation, but have similar capabilities.)

On the other hand, in *Ciphertext-Policy ABE (CP-ABE)*, the ciphertext (for each individual record) has an attached policy that dictates the attributes that a user must possess to decrypt that document. CP-ABE provides role-based access control, as the policies reference a list of attributes that describe the user (and his role) and are embedded into the user's secret key. In the example above, the attribute on a coach's key would be `role:coach` and all applicable heart-rate data would need to have an associated policy allowing the coach's key to decrypt the ciphertext.

In ShareABEL, we elected to use Key-Policy Attribute-Based Encryption for several reasons: (1) because the attributes are content-based, different data consumers with similar roles could still have different access, (2) we assume ShareABEL will store a large quantity of data, but share it with relatively few data consumers (a data owner will probably have a short explicit list of people with whom he would like to share his data), so it would be more efficient to have one policy per consumer, instead of one policy per record, (3) KP-ABE supports temporary and limited access requirements more efficiently than CP-ABE, and is generally more computationally efficient than CP-ABE in key generation, encryption, and decryption [28], and (4) KP-ABE is more flexible than CP-ABE with regards to revocation and changes in access-control policy, so re-encryption costs will be minimized [1]. Because policy is tied to a data consumer's key, each consumer receives one key with the associated formula specifying which ciphertexts he will be able to decrypt. A consumer's device uses this one key to decrypt all permitted records. Multiple consumers can decrypt the same record with different keys as long as each key satisfies the attributes tied to the record.

When considering a content-based access-control policy that is cryptographically-enforced by KP-ABE, we can conceptualize the policy in two parts: the *attribute-tagging scheme* and the *key-policy generation*. The attribute-tagging scheme encompasses the definition of the attribute universe (the selection of which attributes will be used to describe the data) and the tagging of each record with these attributes. The key-policy generation part encompasses the construction of ABE private keys for each consumer, defined as a Boolean expression over attributes.

3 RELATED WORK

Other approaches have sought to provide access control for medical record systems. Rizvi et al. developed an open-source library that uses relationship-based access control, where the policy grants access based on how the access requester is related to the resource owner [25]. This method of access control works best in defined networks, such as social networks, where the relationship between two individuals or entities adheres to one of a set of identifiers. For example, in Facebook there are four built-in relationship types: 'me', 'friend', 'friend-of-friend', and 'everyone'. In our case, access policies may not fit well within this paradigm of different levels of access for different relationships. For example, if an athlete wanted to share all of his training data but only a subset of his vitals with his coach, and all of his vitals but only a subset of his training data with his doctor, this relationship-based model would not support the granular control of these content restrictions. Furthermore, relationship-based schemes require an owner to explicitly manage those relationships, adding, renaming, and deleting edges in the graph as policies change. Content-based access control gives the data owner more flexibility in defining which content is seen by which requesters and lessens the administrative burden.

Benaloh et al. provide content-based access control through hierarchical identity-based encryption (HIBE). Their system leverages HIBE by using a data category hierarchy to define different access levels. Therefore, it requires that a patient's record is organized into a hierarchical data structure, allowing patients to grant access to a category without knowing the contents of that category. This system also allows doctors to define subcategories within the categories that they have permission to access [3]. Our system seeks to provide content-based access control regardless of the relational structure of the different types of mHealth data collected from the patient. We want to allow data owners to have the flexibility to define their access-control structure in different ways and not be restricted to a default hierarchy structure. For this reason, we decided to use a close relative to identity-based encryption – attribute-based encryption – to provide flexible content-based access control.

Attribute-based encryption was created in 2005 by Sahai and Waters as a form of public-key cryptography that allows for more verbose and flexible policy definitions [21]. In ABE, the secret key and the ciphertext are dependent on attributes, and a private key can only decrypt the ciphertext if there is a match between the set of attributes linked to both the ciphertext and the key. By allowing for policies that are Boolean expressions across attributes, ABE allows fine-grained access control. Fine-grained access-control systems facilitate granting differential access rights to a set of users and allow flexibility in determining these access rights. Goyal et al. developed Key-Policy Attribute-Based Encryption (KP-ABE) [11]. Because the private keys in KP-ABE allow users to be associated with different access structures, it provides a large degree of flexibility in specifying content-based access control of data.

Akinyele et al. propose a core scheme for applying ABE to EMRs [1]. Their approach does not support the revocation of long-term keys when a provider leaves, nor changes in access-control policies or decisions. Their system uses a rigid pre-determined policy engine that does not allow for patient discretion. Akinyele et al. focus on hospital systems that struggle to manage many different patient records, whereas ShareABEL allows patients to manage providers across different networks and outside the hospital ecosystem.

Li et al. leverage attribute-based encryption in combination with a security domain scheme to ensure scalability in multi-owner personal health record (PHR) systems [16]. Their scheme fragments the system into multiple security domains, with each responsible for a subset of the users. The framework proposed relies on auxiliary attribute authorities to govern a disjoint subset of attributes. While dividing the responsibilities within the system increases scalability, it also depends on third-party authorities to manage users and attributes. In our system, we seek to decentralize further, by moving the attribute and user management to the data owner's smartphone device.

Ambrosin et al. demonstrated the feasibility of ABE on smartphone devices by creating the AndrABEn library, a C-based library that uses the Java Native Interface, and evaluating its performance on Android devices. While they concluded that using ABE on Android smartphones and similar devices is feasible, they did not apply this finding to any data-management or secure-sharing problems [2]. Zhang et al. developed a Java-based implementation of Key-Policy ABE and applied it to a specific personally-controlled health record mobile platform, Indivo X.[1] We extended their work by using their KP-ABE implementation and applying it to a more general and flexible system, focusing on different challenges posed by mHealth data.

4 USE CASES AND APPLICATIONS

We present four use cases to illustrate the value of a system that allows data owners to share specific subsets of mHealth data with various parties. These examples demonstrate applications of a secure remote-monitoring system, and the importance and desire for flexible and fine-grained access control by data owners. In each case, our system could balance patients' privacy with providers' ability to deliver effective healthcare.

Case 1: Remote patient monitoring. Individuals living in rural areas may have difficulty accessing a provider in person, due to the time or expense required to travel to visit a doctor or clinic. Telemedicine systems allow for remote consultation, including bidirectional audio/video conversations, but are not themselves sufficient for management of chronic conditions like high blood pressure. These visits may be complemented by ongoing monitoring of patient activity (such as sleep patterns and medication compliance) and vital signs (such as weight and blood pressure). Although remote patient-monitoring systems are emerging, they are often disease-specific or hospital-specific, and rarely give patients control over the collection and sharing of data. Through ShareABEL, data collected by mHealth apps and devices can be shared with the clinical team.

Case 2: Research. ShareABEL also allows for the remote monitoring of volunteer subjects by researchers. Because many health conditions are related to behavior, it is valuable to collect data about behavior and physiological parameters from subjects, over weeks or months, in free-living conditions. Examples include studies of smoking-cessation interventions, eating behavior, or student stress and its links to depression. A larger example is the US Precision Medicine Initiative, which aims to recruit one million subjects to participate in a comprehensive, long-term mHealth-based study of health, health behavior, and health environments [15]. Our system allows research subjects to selectively contribute data from their existing apps and devices, and any others given to them by the researchers, without granting researchers full or permanent access and without necessitating the construction of a separate data-collection pipeline. Confidence in their privacy and control over their data would encourage more potential subjects to agree to participate. Additionally, our system lessens the interference of the study on the subjects' lives, as they would have to carry less extra equipment and interact with the researchers less frequently.

Case 3: Preventative medicine. One increasingly common application of mHealth devices is data collection for preventative care; individuals are investing in apps and devices that monitor everything from physical activity to sleep, weight, and blood pressure. mHealth apps and devices not only help individuals self-regulate through immediate data-driven feedback, but also allow doctors to monitor their patients and detect issues earlier. Doctors, therapists, and coaches desire to monitor patients for different health issues, but they all care about temporal trends in the data. By allowing patients to provide access to mHealth data on a temporal basis, patients are able to retroactively grant access to a past temporal segment of the data stream. For example, this capability would be valuable if a patient wants to give a new doctor insight about her asthma and exposure to various allergens over the past several months. The new doctor can use the ongoing data stream to monitor the patient's response to a new treatment regimen.

[1]https://sites.google.com/a/ualr.edu/reu-project-by-liang-zhang/home

Case 4: Sensitive medical data. There are wearable healthcare data that can either imply or directly indicate a highly sensitive and private disorder. Knowledge of such data can be used to unfairly discriminate against a person or deny him opportunities that he otherwise deserves. For example, if someone uses a wearable device to track his blood-glucose levels, this indicates that he is probably a diabetic. Knowledge that he is a diabetic might adversely affect his employment opportunities, professional growth, personal relationships, and insurance costs. Therefore, while a patient may want to share some of his mHealth data, there may be some data that he wants to restrict. This demonstrates that bulk encryption of data is not sufficient [1].

For example, some women use wearables to track their menstrual cycles to facilitate conception.[2] An elite athlete would not want her coach to know that she's trying to get pregnant because she may be worried about losing sponsors due to an inevitable future break from the sport. If she's newly pregnant, she may also want to restrict telling data from her coach because she may be worried that he would not let her enter a tournament.

Additionally, in the case of the treatment of minors, there can be some data that minors are not willing to share with their parents or personal representatives (but should share with their doctors). Under US law, there are situations in which a minor can exercise his rights as an individual without parental knowledge or approval [7]. These legal protections are intended to ensure that minors will seek needed health care if they can receive it confidentially. There are some especially sensitive data types that a minor might not want to share with his parents (but should share with his doctors). One example would be apps that use Ecological Momentary Assessments (EMAs) to track substance use [22]. A teenager seeking help for substance abuse from his doctor may use these apps, but he does not want his parents to know that he uses drugs. ShareABEL provides a flexible technology foundation that allows the son to control the release of this sensitive mHealth data.

5 SECURITY MODEL

As a foundation for the presentation of our approach, and for later analysis of its security properties, we next present our adversary model, threat model, and trust model. Our system's security and privacy goals are to maintain the confidentiality, integrity, and authenticity of the data produced and consumed by the players in our system, in the face of threats from adversaries with capabilities described below, building on the trust assumptions described below.

5.1 Adversary model

We consider an adversary who is a curious family member, friend, or colleague of either the data owner or a data consumer, a malicious individual with physical access to the data owner's smartphone or the data consumer's device, or an attacker with access to the network communications among the parties. The adversary could also be an authorized data consumer who desires access to more data than she is permitted. The adversary may also be the database itself, honest but curious. When considering the adversary, we assume that the adversary has a variety of capabilities that she can employ to compromise our system. We assume the adversary can intercept all network traffic from the data owner's smartphone to the cloud database (for data upload), from the cloud database to the data consumer's device (for data download), and between the data owner's smartphone and the data consumer's device (for key/seed sharing). Adversaries can capture, insert, modify, and remove messages among the three entities.

If the adversary is a set of data consumers, we assume that those adversaries would attempt to collude to access a wider set of data than they collectively have permission to access. For example, if Alice and Bob collude, and Alice's key policy allows her to view data tagged with the `heart-rate` attribute and Bob's key has the `exercise` attribute in his key policy, they may try to combine their keys in some way to decrypt a record with both the `heart-rate` and `exercise` attributes.

Although the adversary may intercept all network messages among the system components, we leave traffic-analysis attacks for future work. We also assume that the adversary cannot compromise the data owner's smartphone or the secure key and seed storage location in the consumer's device. Finally, we assume that all of the cryptographic primitives (SHA-256, Pairing-Based Cryptography, Decisional BDH) are computationally hard and the adversary cannot break those cryptosystems.

5.2 Threat model

Given the capabilities of the adversary, we focus on the following threats.

Threat to privacy: The adversary wants to learn sensitive information about the owner, such as medical conditions (e.g., disease or treatment type), mHealth usage (e.g., types or number of apps/devices), or other personal information deemed private (e.g., location or activity). For this threat, the adversary tries to eavesdrop on the system, including all communications between data owners, data consumers, and the database, to discover sensitive information from the messages. The adversary may also try to compromise the database to determine this sensitive information. Indeed, the database itself may be adversarial regarding this threat.

Threat to data integrity and authenticity: The adversary wants to cause the database or the data consumer to accept incorrect, invalid, or duplicate data by either forging an entry that looks legitimate to the database, tampering with a legitimate entry from the smartphone, or replaying a previously submitted entry. The adversary also wants to insert himself into the key and/or seed sharing process between the data owner and the data consumer, and either cause the data consumer to accept the wrong seeds or keys or cause the data owner to believe that she is sharing keys/seeds with the data consumer but is really sharing these secrets with the adversary.

5.3 Trust model

We make certain trust assumptions about each system component.

Data owner's smartphone: We assume that all players in the system can trust the integrity of the smartphone, its operating system, and our **ShareApp** smartphone application. We assume the smartphone application is preconfigured with the public key of the cloud database and they are able to use standard protocols

[2]avawomen.com

(such as TLS) to establish a secure channel. We further assume that the smartphone may connect to the database server via an anonymizing network such as Tor, if IP-level anonymity is desired. Finally, we assume the smartphone can effectively authenticate its user before use.

Cloud database: We assume that the data server is "honest-but-curious", meaning that the server may seek to obtain plaintext mHealth data, or link data to the data owner or data consumer, but will honestly execute its role as a data-storage server: to store a given data record under a given index, and to return that record when later asked for the record corresponding to that index. **ShareBase** (our cloud database) can be trusted to never delete data, nor mis-index data, nor overwrite data at a given index if presented with an incoming message providing new data for an existing index. The server need not otherwise be trusted with the integrity or confidentiality of the data.

Data consumer's computer: We assume that all data owners and data consumers can trust the integrity of the computer used by the data consumer, its operating system, and our **ShareView** consumer application. Specifically, we trust its filesystem to secure the consumer's keys and seeds, and to protect **ShareView** from inspection or tampering by other applications. We assume that ShareApp and ShareView have previously completed a secure key exchange, likely during an in-person meeting of the respective data owner and consumer and leveraging one of many common mechanisms for key exchange (Section 6.3). We assume ShareView can effectively authenticate its user leveraging one of many common authentication mechanisms. We assume ShareView is preconfigured with the public key of ShareBase and that they are able to use standard protocols (such as TLS) to establish a secure channel. We further assume that the portal may connect to the database server via an anonymizing network such as Tor, if IP-level anonymity is desired.

6 OUR APPROACH

ShareABEL consists of three components: ShareApp (the data owner's smartphone application), ShareBase (the cloud database), and ShareView (the data consumer's desktop application). All the owner's data is encrypted on the owner's smartphone and stored in ShareBase on an untrusted cloud server.

ShareABEL allows owners to specify access-control policies and cryptographically enforce those policies, so that only parties with the proper corresponding attributes are able to decrypt desired data. To enforce access control with an untrusted (honest but curious) data server, we use two different methods: (1) content-based access control, enforced through the encryption of the data, and (2) temporal access control, enforced through the database's indexing scheme. The combination of these two methods makes our system unique: it allows for more granular access control based not only on the attributes of the data, but also on the time that data was produced.

Encrypting the records serves three purposes: (1) to provide data confidentiality, both in transit and while at rest; (2) to protect information about the contents, source, and associations of the data from exposure to the untrusted database; and (3) to enforce the access-control policies set by the data owner. Issues with traditional encryption schemes arise when the data owner wants to define more granularly whom gets access to which pieces of information. For example, if an athlete wants to share only a subset of her heart-rate data with her coach (e.g., only those readings taken when she was exercising), she would have to either manually sort through the heart-rate data and encrypt exercise readings separately or just give her coach the decryption key for all heart-rate data and trust that her coach will only look at the desired data.

Key-Policy Attribute-Based Encryption (KP-ABE) allows ShareABEL to support the desired granularity. ABE cryptography enforces data access and requires no trusted mediator [24]. Each record is tagged with pertinent attributes, defined by an attribute-tagging scheme and dependent on the data in the record. ShareABEL can support a variety of attribute assignment schemes, such as tagging each data type with a different attribute (e.g., data collected by a heart-rate application has `heart-rate` as an attribute), including additional attributes to differentiate records within a stream (e.g., heart-rate data collected while exercising would have `heart-rate` and `exercise` as attributes), or having attributes that encompass data categories (e.g., heart-rate, blood pressure, and ECG readings all have the `cardiology` attribute). Regardless of attribute-tagging scheme, data consumers can only decrypt a record if their key policy matches the record's attributes.

While data encryption using KP-ABE protects the confidentiality of the data and enforces access control, a message authentication code (MAC) ensures data integrity. By applying a keyed MAC to the plaintext and then encrypting the concatenation of the plaintext and MAC with ABE, all parties are able to authenticate the message after decryption to ensure there have not been modifications to the record [8]. Although each data consumer has a different ABE decryption key but can decrypt the same ciphertext record, only one MAC key is necessary for each data owner; it is created by ShareApp and shared with all ShareViews that receive streams from that owner.

6.1 Hash-chain indexing scheme

ShareABEL also enforces access control through a specialized database indexing scheme that allows for temporal access control. To access a record in the database, the data consumer must query the database using a specific index that corresponds with that record. ShareBase uses pseudo-random hashes, produced by randomly-seeded hash-chains, as the sequence of indices for logging sequential records. These hash chains are built from a cryptographically secure one-way hash function like SHA-256. With large hash values, the resulting index space is very sparse and indices into the database are effectively secret and collision-free. Furthermore, each owner salts each iteration of the hash function with a hash key, so an adversary with access to one hash value (including the untrusted database) cannot generate the next hash value without knowledge of this key. All consumers with some temporal access to a data stream are provided its unique hash key. (Each data owner creates a unique hash key for each data type – see Section 6.3).

A consumer can only gain access to a specific temporal segment of the data stream if her ShareView received the key from that owner and the seed for that time period. Once a data consumer's device stores one hash in the chain, it can easily generate future hashes using the keyed hash function until a new random seed is generated. New seeds are generated by the owner's ShareApp regularly, over a system-defined seed refresh interval (SRI). For each data owner, every data type has a unique set of seeds.

For example, if seeds refresh weekly, the heart-rate data type has one seed per week, initialized on a Sunday. In this case, the first record (the first data point collected after the midnight when Sunday begins) uses this seed as the index into the database. The seed is then shared with all consumers that have been granted access to the stream during this time period. The owner's ShareApp uses a hash function (and hash key) to generate the next hash in the chain, which will be the index for the subsequent record. The consumer's ShareView uses the same hash function and key to generate hashes in the chain to query future records. When the new week begins, ShareApp generates a new seed and shares it with those consumers that deserve access to ongoing records in that data stream.

Access can be granted beginning at any time (by sharing the current hash in the chain) and can expire at the end of any SRI-defined time period. Therefore, the temporal granularity depends on the chosen SRI (in our examples, we use an SRI of one week). The temporal controls have three purposes: (1) to allow data owners to grant access to specific temporal segments of the data stream, (2) to provide a form of key freshness by requiring explicit sharing of each new seed, and (3) to limit damage from loss (or unauthorized sharing) of the seed or any of the hashes of the sequence (indices), since knowledge of any such hash is useless without the corresponding hash key.

New seeds can also be generated at any time by the data owner to support revocation. If the data owner decides in the middle of a seed-refresh period to change the access-control policies, a new random seed is generated for each affected data type. ShareApp then distributes the new random seeds to those who now have access to the corresponding streams.

We also use hash chains as an indexing scheme to ensure no information is leaked to the untrusted database. Hashes as indices allow a consumer to access encrypted records of a specific data type for a specific individual without ShareBase knowing the person or data type to which the hash refers. Because we use a keyed hash function to generate each hash in the chain, and the key is not shared with the database, the database is unable to generate the chain. Although ShareBase has access to past hashes in the chain, it cannot determine which hashes are related nor associate hashes with their owner. We assume the hash function is collision-resistant, so if the database ever receives a request to add another record with the same index, it assumes that the message may be a replay attack and ignores the message.

6.2 Access control and revocation model

In this paper, access is defined as the permission granted by a data owner to a data consumer to view the plaintext of a specified record (read-only). To access plaintext for a record, the consumer must possess both the hash (and hash key) that corresponds to that record (to query the database and receive the proper encrypted record) and a cryptographic key with attribute tags allowing it to successfully decrypt that record. Therefore, each consumer must possess two different secrets to access any singular record.

For example, to access a patient's heart-rate data from September 1, 2016, the doctor must have the random seed for the week of September 1st, and the patient's hash key. For each desired record, the doctor's ShareView computes the corresponding hash-chain index to query ShareBase for the encrypted version of that record. His ShareView then uses his KP-ABE key to decrypt the record.

We support owner-driven access control – the data owner is the party setting all access-control policies. An access-control policy is a set of permissions authored by the data owner that specifies which records each data consumer is able to read. We use a default-deny system (i.e., explicit permission must be given to allow access to the record, and if permission has not been specified in the access policy, a "deny" is assumed). In this model, access control is enforced on a record-by-record basis (i.e., each consumer must be explicitly granted access to each record). Although access control is record-specific, ShareApp's UI allows the owner to grant access to a temporal segment of the data stream (multiple contiguous records) in one policy, due to the combination of the hash-chain indexing scheme and attribute-based encryption.

Because all access-control policies are set in ShareApp, it is responsible for all key and seed generation. When an access-control policy is first defined by the owner, the determined attribute set is used to generate the public key that will be used by the owner to encrypt all records. Private keys for each of the data consumers are then generated from a master secret key possessed only by the data owner based on the access granted to them by the policy. This approach is different than the external key authority used by other KP-ABE systems to generate and disseminate private keys.

For each data type and each data consumer, an access-control policy can begin at any record (by sharing the current hash value), and can end at the end of any SRI. Generally, access policies can be divided into two groups: those that provide restrictions on future data (data that has yet to be generated and/or inserted into the database) and those that provide restrictions on past data (data that has already been inserted into the database).

Access to future data can be granted to begin immediately, or at a specific future time, and ends on a defined expiration date. In all cases, this temporal access is determined by the sharing of seeds. From the beginning of the period where access to future records is permitted, the consumer receives the ABE key with the new policy attached and the seed for the next temporal segment (e.g., next week) only. Every time the seeds are refreshed, the owner sends the consumer the new seed for that data type if the consumer has access for some or all of the next period. Once the explicit future segment has ended or once a consumer's access has expired or been revoked, she will no longer receive fresh seeds (and will no longer receive fresh keys if her access is revoked across all of the owner's records). For policies specifying ongoing future access, the data owner will be asked to review and renew the consumer's access after the policy's expiration date, so that the data consumer can continue to receive new seeds.

Access to past data can either be granted for a range of time or for specific existing records. For a continuous range of time, data consumers are granted the temporal seeds that correspond to that period of time, as well as a key that satisfies the attributes of the encrypted records. The data consumer will initially be granted a subset (e.g., four seeds for one month of past data), and the consumer can request more seeds if access to older data is required. This enhancement is for both security and performance reasons. If fewer seeds are shared with the data consumer, fewer records would need to be re-indexed in the event of a revocation, and less strain is placed on the system for sending a large number of seeds at one time.

To restrict access to specific record(s), as opposed to a range of records, the data owner must include an attribute in the consumer's key so that the consumer can only decrypt records with a combination of attributes. For example, if a data owner only wanted to share with his coach the heart-rate data that had been collected while he was at the gym (and no other heart-rate data), an additional attribute `location:gym` would be added to those heart-rate entries. The coach would then be given a key with the policy of `(type:heart-rate AND location:gym)`, meaning the key could only decrypt records with both of those attribute tags, and the coach could only see heart-rate data collected at the gym.

Another example of granular control is a "summary" attribute. To allow data consumers to see the trend of data over time without granting them access to all records, a subset of records in the stream can have a `mode:summary` attribute. Consumers with the policy of `(type:step-count AND mode:summary)` would only see these summary records (e.g., the mean number of steps over the past week). If a consumer wants more detailed data, she must request broader access from the data owner.

Revocation. Individuals should be able to change their minds about data sharing and express those new changes in policy. An owner may decide to add new data consumers, expand a consumer's access to his data, or narrow the scope of a consumer's access. ShareABEL supports several forms of revocation, all by narrowing the scope of access: changing an access policy to a narrower set of attributes, reducing the temporal range accessible, or removing all access by a given consumer. The effect is to prevent the affected consumer(s) from retrieving and decrypting data from the database.[3] In our system, *revocation* means that an access-policy change narrows or restricts a consumer's access to a data type (i.e., there exists at least one record (past or future) of that type to which the consumer used to be allowed access and now is denied access). Revocation can be as narrow as a change in the access to one record, and can be as broad as the revocation to the entire data category (all data tagged with the same attribute). We can break all revocation down into two categories: revocation of past data (data that was added to the database prior to the owner's decision to change the access policy) and revocation of future data (data that has not yet been added to the database). Revocation to an entire data stream can be considered a union of these two categories.

To revoke data that has already been added to the database, the data owner can leverage the database indexing scheme to remove access to records. By starting at the first affected record and generating a new seed, the data owner can re-index each of the records with the freshly-seeded hash chain. The new seed for the hash chain is then shared with those consumers who, according to the new policy, should still have access to the record(s). (As above, the owner may take a "lazy" approach by sharing only a subset of the seeds, depending on the consumers to request seeds as-needed if/when they desire to download the old data.) When the revoked consumer attempts to query the database for the revoked record(s), using the old hash chain, she will not be able to find records at those indices. Attribute-based encryption is expensive, so re-indexing the records is more efficient than re-encrypting the data [9]. It also obviates the need for the owner to cache (or download) all the data for re-encryption; the owner need only archive the sequence of prior seeds for each data type.

Consider a data owner who wants to revoke access to some or all previously granted future records; e.g., Bob gave month-long access to the data to Alice, and partway through the month decides that he no longer wants Alice to have access to the remainder of the month. If the stream is currently in the middle of an SRI (e.g., mid-week), he generates a new seed and starts a new hash chain to use for indexing future records (and shares that new seed with all consumers that still deserve access). Alice would be able to generate future hashes from the old hash chain, but the database would not return anything to her when queried with those hashes.

6.3 Key management

To meet our security goals, key management is paramount. By *key management* we mean the storing and sharing of attribute-based encryption keys, MAC keys, hash-chain seeds, and hash-chain keys. As keys and hash-chain seeds are generated on the data owner's smartphone, the data owner must have a mechanism to securely store and send these keys and seeds to the permitted consumers. Due to the structure of ABE, each data consumer possesses one key that expresses a Boolean policy over the attributes of the data. Regardless of the number and variety of data records that the consumer can access, she only receives one ABE key from each owner, with the entire access-control policy incorporated into the key. If the data owner changes the policy, that key may be replaced with a new policy across old or new attributes. Similarly, each consumer receives one MAC key from each data owner; this key is the same for all consumers of a given owner's data, but are unknown to the database and other possible adversaries.

Regardless of the attribute-based tagging of data, each data type determined by the owner's data configuration file (see figure in Section 7) has its own set of temporal seeds and its own hash-chain key. ShareApp must share a new seed at the beginning of each SRI to each data consumer for each data type she can access. While the seed for the hash chain changes at the beginning of each SRI, the hash-chain key remains unchanged. Therefore, any data consumer that has access to some temporal segment(s) of that data type will be granted the hash-chain key for that data type in order to generate future hashes. If the data owner changes the policy and a data consumer gains access to a new data type, the respective hash-chain key is shared with that consumer.

To share all of the keys and seeds with each consumer, initially and for these periodic updates, ShareApp must send a secure message to each data consumer (e.g., an encrypted and signed email message). To establish the cryptographic keys to send this secure message, an initial secure key exchange between the data owner

[3] Of course, if a data consumer had cached the plaintext of data downloaded and decrypted earlier, or cached both the ciphertext and the associated key, we cannot prevent that consumer from continued local access to the cached data.

and consumer is necessary. In our approach we expect that the owner (a patient or research subject) has a personal relationship with the consumer (a doctor, coach, family member, researcher, etc.). Ultimately, the root of trust between the data owner and consumer is this interpersonal relationship. The owner can visually verify the identity of the consumer during an in-person meeting, then exchange secret(s) as the foundation for future secure messaging. Many approaches are possible; here is one example inspired by McCune et al. [17]. The owner specifies the consumer by email address or phone number through the smartphone application. The ShareApp uses this address to contact the consumer's ShareView, which, with her assent, displays a QR code containing a one-time symmetric key. The consumer can then show the code to the owner, who scans it with ShareApp on his smartphone. The owner's ShareApp is then able to use this shared secret to establish an immediate (but short-lived) secure network connection to the consumer's ShareView for the exchange of other address and key material needed for future secure messaging.

Once the key exchange is complete, the data owner's ShareApp and the consumer's ShareView can use this shared key material to securely send messages. Key and seed sharing can happen via push or pull, depending on the context: the ShareApp will *push* new ABE private keys due to access policy changes and new seeds at the beginning of each SRI; the ShareView may *pull* any missing hash-chain seeds necessary to obtain historical records; the latter sends a request to the owner's ShareApp, which may check with the owner if the access-control policy does not already allow that consumer access to the requested data.

7 IMPLEMENTATION

To create a working prototype, we implemented all three major components of ShareABEL: ShareApp, the data owner's smartphone app; ShareBase, the cloud database; and ShareView, the data consumer's desktop app.

7.1 ShareApp: data owner's smartphone app

We implemented ShareApp by writing an application for Android 5.0 (Lollipop).[4] This smartphone application is the main driver of our system, as it enables the data owner to set policy preferences and applies those preferences to cryptographically-enforce access to data. Through our app's interface the data owner is able to add, modify, and delete data consumers with whom the owner wants to share his mHealth data. For each consumer, the data owner is required to input the consumer's name, title (e.g., cardiologist), and an email address that can be used for the initial introduction between owner and consumer devices. Within the system, each consumer is differentiated using UUIDs assigned to each consumer upon creation, so that even if fields change, there is a defined ID for each individual consumer.

ShareApp plays the role of the intermediary between the mHealth apps/devices and the back-end database, by receiving data from those apps and devices, encrypting that data based on the access policy set by the owner, and uploading the encrypted records to the database. The smartphone is responsible for all seed and key management, and is the data owner's means of contact with data consumers.

[4]We wrote and tested the app on a Nexus 9 tablet and Nexus 6 smartphones.

Listing 1: Example Data Configuration

```
Heartrate : Cardiology
Blood_pressure : Cardiology
ECG: Cardiology
R-R_Interval : Cardiology
Sleep : Activity
Location : Activity
Blood_Glucose : Diabetes
Insulin : Diabetes
Weight : Wellness
Eating : Wellness
Smoking : Wellness
GSR_Stress : Wellness
Step_count : Fitness
Theraband : Fitness
Weights : Fitness
Cycling : Fitness
```

In our current implementation, we connected ShareApp to the *Amulet*, a wearable healthcare device that collects a variety of data about physical activity, stress, and use of exercise equipment [14]. We envision other devices being connected to ShareApp in a similar way. The Amulet interfaces with its own Android application, the *Amulet companion* app, which receives data from the Amulet device and forwards it to ShareApp through secure intents. Intents are the preferred mechanism for asynchronous inter-process communication in Android, and explicit intents between the two applications guarantees that no other application can eavesdrop on the communication, because only the specified application has the permission to receive the intent [10]. (We rest here on our assumption that the smartphone and its operating system have not been compromised, as noted in the security model. Securing Android or other smartphone operating systems is outside the scope of this paper.)

In addition to forwarding data through intents, the only other requirement that we ask of mHealth developers is to format the data in JSON with a *data_type* tag that can be cross-referenced with a *data configuration file* (Listing 1) that includes all data types supported by ShareApp. Once ShareApp receives data, it encrypts the data with Key-Policy Attribute-Based Encryption under the current access-control policy, which specifies an attribute-tagging scheme that matches the data with its respective attribute(s) for encryption. Once the data is encrypted, it computes an index for this new record using the appropriate hash chain and uploads the (index, ciphertext) pair to the cloud database as a new entry. If the device is not currently connected to Internet service, the entries will be queued for later upload; a feature we plan to add in future iterations of the implementation (Section 7.4).

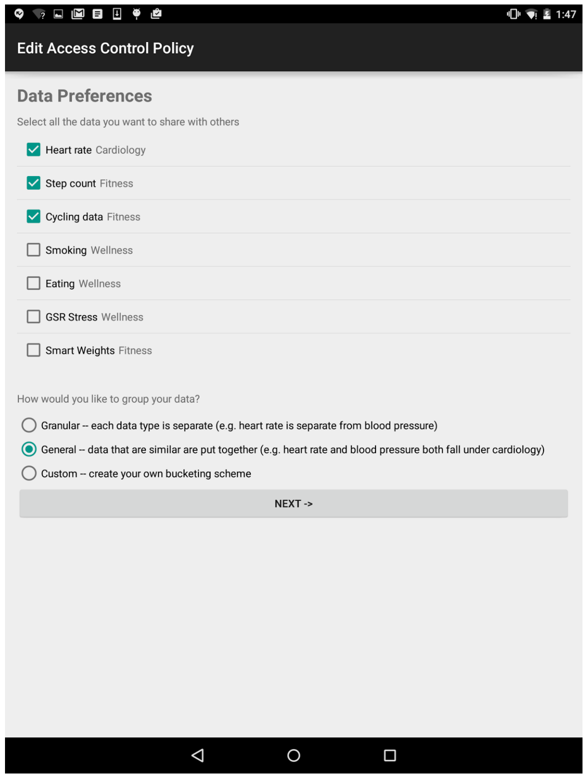
Key-Policy ABE incorporation. While there are many ABE libraries written in C, Java, and Python, there are few KP-ABE libraries supported on Android [29]. Because CP-ABE has been explored more thoroughly in the literature, many of the available libraries only support CP-ABE. We found two previous implementations of KP-ABE for an Android platform. *AndrABEn* is a C-based library with GMP, Pairing-Based Cryptography (PBC), and OpenSSL as dependencies [2]. AndrABEn then uses Java Native Interface (JNI) to link the C implementation of KP-ABE with Java classes in Android. JNI is known to be a faster solution for computationally expensive operations on Android due to its native implementation. However, AndrABEn is not tested or supported on recent versions of Android. The *kpabe toolkit*, implemented entirely in Java, offers a slightly less efficient solution [27]. It depends on Java Pairing-Based Cryptography (JPBC) [6].

We leveraged the kpabe toolkit to generate the KP-ABE master secret key for the data owner to generate the KP-ABE private keys for each consumer, and to encrypt each record with the KP-ABE public key and the attributes defined by the owner's access-control policy. These keys are stored in Android's internal storage, in a directory hierarchy that can only be accessed by the application (according to Linux file permissions). As noted in our security model, we assume that no entity has root access on the device and thus no entity has access to these keys.

Defining policies. For the purposes of this prototype, ShareApp provides a simple graphical interface to allow the data owner to define an access-control policy (Figures **??**–**??**). (The development of a rich interface for defining such policies – and a proper study of that interface's usability – is an important HCI challenge worthy of future work and a paper in its own right [19, 20].) Owners add consumers and select which data to share with which consumers, and ShareApp translates these selections into an access-control policy composed of an attribute-tagging scheme (through which the data is tagged with attributes) and a key-policy generation step (through which key-policy ABE keys are developed for each consumer and seed sharing is determined for each consumer).

The data types supported by ShareABEL can be configured using a configuration file. For each data type, the configuration defines a unique name for that data type, and a default category in which that data type belongs (Listing 1). This configuration file allows our system to simplify policy definitions for the data owner by providing two preset attribute-tagging schemes: a granular and a general scheme. The granular scheme keeps each data type separate by having an attribute for each data type (e.g., for heart-rate and blood-pressure data, heart-rate data would have a `heart-rate` attribute and blood-pressure data would have a `blood-pressure` attribute), allowing for each consumer's key-policy to include any combination of data types.

The other preset scheme is a general scheme. This scheme relies on the categories defined in the configuration file. Each attribute is equivalent to a category (e.g., both heart-rate and blood pressure fall within the cardiology category, so both types of data would be tagged with the `cardiology` attribute), allowing for grouping similar data without putting the onus on the data owner to devise his own groupings. Using the general scheme, access-control policies for each consumer would refer to data categories, instead of individual data types. While this scheme is less granular, it makes the definition of access policies less complicated for the data owner. Due to these two preset scheme options, our prototype interface currently only supports disjunctions ("or") in key-policy expressions over the attributes, although our underlying ABE and system support more complex expressions.



Data Preferences

Select all the data you want to share with others

- [x] Heart rate Cardiology
- [x] Step count Fitness
- [x] Cycling data Fitness
- [ ] Smoking Wellness
- [ ] Eating Wellness
- [ ] GSR Stress Wellness
- [ ] Smart Weights Fitness

How would you like to group your data?

- ○ Granular -- each data type is separate (e.g. heart rate is separate from blood pressure)
- ● General -- data that are similar are put together (e.g. heart rate and blood pressure both fall under cardiology)
- ○ Custom -- create your own bucketing scheme

Figure 1: Defining the attribute universe and attribute-tagging scheme

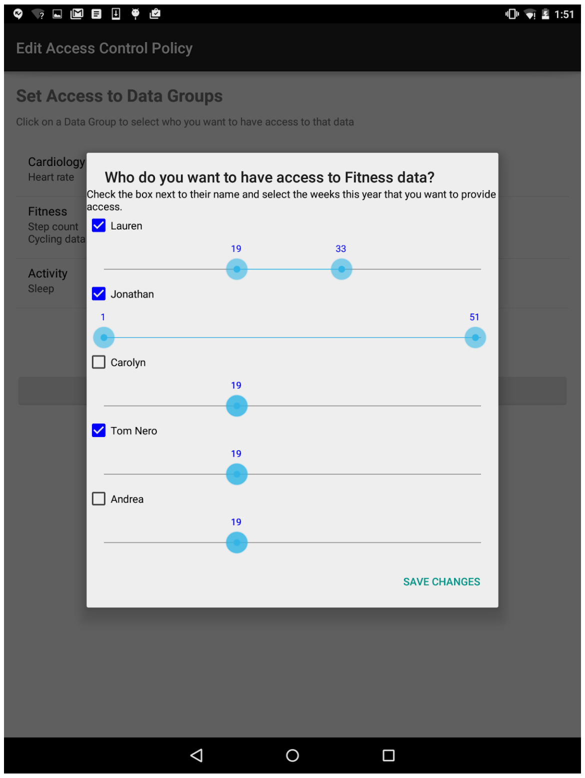
For advanced data owners or those with specific needs, we also offer a custom scheme setting that allows data owners to create their own categories and place data types into these categories as they see fit. The categories created by the data owner then become the attributes used in tagging the data types within each category. In this custom setting, data types can belong to multiple categories, giving their respective records multiple attributes, and resulting in more complicated key-policy expressions. For example, if the data owner is an elite athlete, a lot of her training is data-driven. Based on how she specifies her attribute-tagging scheme, her head coach could have a key with the policy:

```
(category:cardiology AND location:gym) OR (category:fitness)
    OR (type:weight AND (mode:pre-race OR mode:weekly-weigh-in))
```

but her cardiologist could have a key with the policy:

```
(category:cardiology) OR (type:respiratory-rate)
    OR (category:fitness AND effort:high)
```

Once the attribute-tagging scheme has been defined, the data owner selects which consumers can access which attributes (and therefore the data tagged with those attributes), thus creating a bipartite mapping from data consumers to the attributes to which they have access (Figure 3). This mapping is then used to generate each consumer's private KP-ABE key. For example, based on the edges in the graph in Figure 3, the cardiologist Dr. Nero's key policy would be `category:cardiology OR category:fitness`. As the data owner selects which consumer can access which data, he also



Edit Access Control Policy

Set Access to Data Groups

Click on a Data Group to select who you want to have access to that data

Cardiology
Heart rate

Fitness
Step count
Cycling data

Activity
Sleep

Who do you want to have access to Fitness data?

Check the box next to their name and select the weeks this year that you want to provide access.

[✓] Lauren 19 33

[✓] Jonathan 1 51

[ ] Carolyn 19

[✓] Tom Nero 19

[ ] Andrea 19

SAVE CHANGES

Figure 2: Defining the key-policy generation and seeds for each consumer

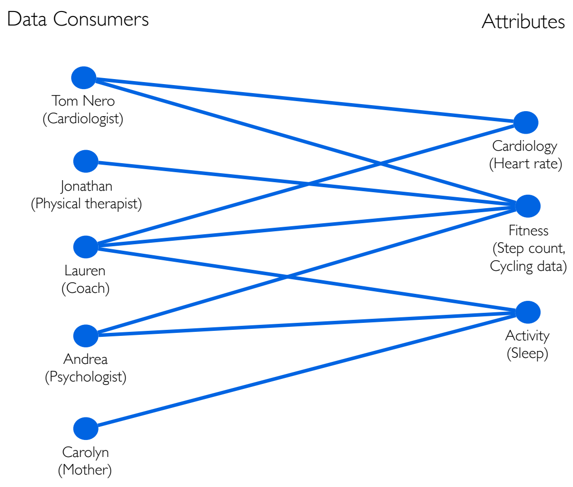


Figure 3: Translation of UI Access Control Decisions to Bipartite Access Graph

provides temporal access-control restrictions by selecting the time range that each consumer can access each data type within possibly broader data categories.

This selection determines which of the temporal seeds will be shared with which consumers.

Local persistent and secure storage. To generate the seed of each hash chain, we used Java's built-in KeyGenerator to generate an `HmacSHA-256` key, which functions as the seed. We then used SHA-256 to generate each subsequent hash in the hash chain. Although our system design uses a keyed hash function, so the database cannot determine which hashes are related, for simplicity of the prototype we used an unkeyed approach. We chose a weekly SRI, leveraging Java's Calendar class with the built-in `WEEK_OF_YEAR` property.[5] We similarly used the KeyGenerator `HmacSHA-256` to generate the HMAC for the plaintext prior to encrypting.

To send the right seeds to the right ShareViews and to support policies that allow the sharing of past data, seed storage on the data owner's smartphone is especially important. We used SharedPreferences, a built-in Android tool that allows for saving state variables between runs of an application. We store seeds in the SharedPreferences map, indexed by data type, SRI number (in our case, the week of the year), and year. The result is an efficient way to store all seeds from all hash-chain periods for each data type. The map also stores the most recently generated hash, to enable fast generation of the subsequent hash in the chain.

To enforce KP-ABE, ShareApp also needs to store the attribute-tagging scheme, the access-control policy, and the KP-ABE private keys of each of the data consumers (as the smartphone generates these keys, it must store them for distribution to their respective consumers). The final set of secrets that ShareApp must store is the set of public keys (e.g., RSA) for each of the data consumers so ShareApp can securely communicate with each consumer's ShareView. All of these secrets are stored in different sections of SharedPreferences. SharedPreferences are stored as files in the filesystem on the Android device, and are secured by filesystem permissions. The permissions state that only applications with the UID of the creating application can access the file [23]. While anyone with root access to the phone would be able to access SharedPreferences, as stated in the trust model, we assume that the smartphone has not been compromised.

7.2 ShareBase: cloud database

The purpose of the database is to store data from multiple owners without revealing the contents of the data, the owner of the data, or which records belong to the same owner. This database is not trusted with the confidentiality or integrity of the data, but the database is trusted with the integrity of indices (that is, the data owner can insert a record at a given index and trust the database to later return that record (albeit with the contents possibly modified) when queried with the same index).

In our prototype implementation, the database server is an Express Node.js server and the storage is a Linux filesystem. The database supports three operations: *Add* (add a new record at a specific index), *Query* (given an index, return the corresponding record), and *Re-index* (given a current index and a new index, re-index the corresponding record at the current index to be at the new index). To add a new record to ShareBase, the data owner provides the hash for that record and the encrypted record itself. The hash is used as the name of the file, and the encrypted record is the body

[5]The `WEEK_OF_YEAR` variable is initialized to 1 on January 1st, and increments every Sunday at midnight.

of the file. Querying the database requires providing the database with the hash associated with the desired data record; that hash is used to find the filename in the filesystem, and the body of the file is returned. Re-indexing the record just requires providing the current hash and the new hash; the file with the name of the current hash is renamed to the new hash. While in our prototype all files live in the same directory, this organization could be optimized into a directory tree, or the content could be stored in a different database structure. All communications between ShareBase and the client applications (both ShareApp and ShareView) occur over secure HTTPS connections, and the clients can verify the identity of the database server.

7.3 ShareView: data consumer application

There are two major functions of ShareView, which we implemented as a command-line application for laptop or desktop computers. Although one could also imagine ShareView as a mobile or web-based app (as long as the keys are stored securely), we selected a computer-based application as a typical use case for providers and researchers. First, ShareView can send and receive secure messages from each data owner's ShareApp to obtain the respective seeds and keys from that data owner. Second, ShareView provides an interface by which the data consumer can query ShareBase. Because one data consumer could have many patients or subjects who use ShareABEL, ShareView maintains a separate *data-configuration file* and *seed-storage file* for each data owner sharing with this consumer.

When Bob, a data owner, first decides to share his mHealth data with Alice, a data consumer, his ShareApp sends her ShareView his data-configuration file. This file includes all of the data types provided by his mobile apps and devices and the attributes that are linked to those data types. Alice's computer caches this file locally. Every time Bob's ShareApp sends Alice's ShareView new hash-chain seeds to access his data, Alice's ShareView saves them in Bob's seed-storage file. Alice's ShareView maintains a *user configuration file* mapping each owner's name/identity to the names of the data-configuration and seed-storage files.

To query ShareBase, Alice selects the data owner then selects the the type and time period of the desired data. ShareView examines its cached copy of that owner's data-configuration file to obtain the list of available attributes and data types, and the seed-storage file to retrieve the seeds for the relevant time period. ShareView computes the necessary hash indices, connects to the database, and asks the database for the records corresponding to the given indices. It decrypts the response, and displays the plaintext to Alice.

Although our prototype implementation has a simple command-line user interface where the data consumer must explicitly type in the names and values that she desires (Listing 2), we envision future iterations of the portal with a robust graphical user interface.

Listing 2: Example Data Query

```
Welcome to ShareABEL. Connecting to the database
    now.
You are now connected to ShareBase. Please enter
    the person whose data you want to query:
Bob Smith
Now enter what type of data you want to query.
Heartrate
Do you want current or past data?  Type "c" for
    current and "p" for past
c
Bob Smith, Heartrate data for May 17, 2017 at
    12:30pm: 102
Do you want to continue querying current Heartrate
     data? Type "y" for yes and "n" for no
y
Bob Smith, Heartrate data for May 17, 2017 at
    12:33pm: 96
Do you want to continue querying current Heartrate
     data? Type "y" for yes and "n" for no
y
There are no further entries in this data stream.
     If this is a current stream, there may be
    more later.
```

7.4 Future Implementation Goals

Due to the technical breadth of this thesis, some features have been relegated to future work. Some features, such as the inclusion of MACs, secure communication, and key and seed sharing were not implemented because these problems have been investigated thoroughly by prior research. We were confident that their implementation would be routine, and that our energy was better spent on innovating new solutions to our research questions. Others, such as secure introductions and HCI questions of how best to support owner-defined access policies, were not considered within the scope of this research and were considered full research papers in their own right. A few components of the design would be beneficial additions to future iterations of the implementation, detailed below.

Re-indexing as revocation. While the server-side component of re-indexing is trivial, simply renaming a file from one hash to another, the smartphone-side component is more complicated due to the user interface. Currently, the UI only supports setting new access-control policies, as opposed to making minor changes to existing policies (e.g., if a data owner only wants to revoke one time segment of a data type from one data consumer, the owner would have to define an entirely new policy embodying that change). Therefore, the smartphone is unable to determine which small components of the policy have been changed, and instead behaves as if there is an entirely new policy by generating new seeds and keys. While this behavior is adequate for revocation of new data (data that has not been stored in the database), it does not support revocation of past data through re-indexing, because the system cannot recognize which past access has changed. Future implementations must have UI inputs that allow the smartphone to recognize when minor changes to the policy are made and determine which records must be re-indexed. Once the list of records for re-index has been established, issuing the re-index command to the database is trivial.

Handling data offline. When ShareApp receives data from mHealth apps or devices, it must have a mechanism to store and queue that data in the event that it cannot connect to ShareBase immediately.

Because ShareApp is housed on a smartphone, which does not necessarily have a reliable Internet connection all of the time, this offline feature is especially important. Future implementations should include a queue that will receive incoming data, store it, and send it to the database upon a successful connection to the server.

8 EVALUATION

In this section we analyze the security properties of our approach, then describe the results of some performance experiments.

8.1 Security analysis

Given the assumptions of the security model outlined in Section 5, and the design of our system, we address how we mitigate the threats of a capable adversary.

Scenario 1: Cloud database compromise. We assume that the cloud database is not trusted with respect to the confidentiality, integrity, or correctness of the data. A passive attacker (i.e., with read-only access to all database data and inbound/outbound messages) would be unable to view the plaintext data for two reasons: (1) at no time does the cloud database have access to any plaintext or to any key/seed material, and (2) during content adding and query processing the data (and metadata) remains encrypted. By observing a deposit to (or query of) the database, the attacker cannot discern the ownership of the corresponding data record (because no owner-identifying information is shared with the database, even in encrypted form), cannot link records deposited by the same owner (because record indices are hashes produced by a keyed hash function for which neither the adversary nor the server has the key), and cannot view the data or which attributes the data possess (because the records are encrypted).

An active attacker that has read and write access to the database server would be able to modify the indices as well as the records themselves. If the attacker modifies the index of a record, it is akin to deleting that record, because the data consumers would no longer be able to retrieve the record. The data consumers would quickly become aware of this change as the hash-chain would have gaps or end prematurely. The adversary cannot insert false data, because (without access to the hash-chain seed and key) the adversary has no way to generate an index value any consumer would likely query. If an attacker modifies an existing record's ciphertext, the MAC on the plaintext would fail to verify.

Although an active attacker cannot modify the ciphertext in a way that will decrypt to valid plaintext, the attacker might instead attempt to replay previous entries or entries from another data owner. The database server would reject a replay attack in which the adversary attempts to add a fresh record with a previously-seen index value, because it never allows a record to be deposited over an existing record at a given index. If the adversary has the ability to write into the underlying database, it could replace the contents of the entry at a specific index with another entry (from either the same or a different data owner). Upon decryption, the data consumers would discover one of two outcomes. If the record is from another owner, or from the same owner but not a record the consumer deserves (according to the policy), the decryption would fail (the MAC verification will fail). If the decryption and MAC happen to succeed, because this record is indeed one deserved by this consumer, the JSON fields (timestamp, data type) would not align with the other entries in the hash-chain. Because indices are created by a randomly-seeded keyed-hash chain, the attacker could not modify multiple entries in the data stream to fool data consumers. Therefore, data confidentiality and integrity at the database are maintained.

Scenario 2a: Data consumer computer compromise. ShareView (a desktop app) is responsible for the request and decryption of the data that the consumer has been granted access to view. Because of this role, the consumer has secret keys and seeds stored on the computer. Attacks on this device, its operating system, or the ShareView software itself, could result in data exposure or data integrity concerns. In our security model we trust these components to behave correctly and to remain un-compromised (securing the underlying platform is outside the scope of this work and, indeed, a compromised OS or device would expose any and all data viewed in the desktop app regardless of our system). If an adversary gains physical access to the consumer's device, after the data consumer had authenticated (logged in) to ShareView, the attacker would be able to expose the sensitive data to which the data consumer has access. Physical security, user authentication, and defenses like screensaver locks are standard practice and outside the scope of this paper. The data consumer can also notify the data owner that her ShareView was compromised, and he can re-index all records that the consumer had been able to access.

Scenario 2b: Consumer(s) as attacker(s). If the data consumer misuses the decrypted data – saving the plaintext on an insecure medium or sharing the plaintext with an unauthorized party, the data is no longer protected by ShareABEL and is thus disclosed or vulnerable to disclosure. We have to trust the data consumer not to be malicious or careless, and should engineer ShareView to make it difficult to export the plaintext.

If data consumers decide to collude, none of them would be able to discover the random seeds for the hash chains for data types to which they do not collectively have access. Suppose one of them has the seeds and hash keys to enable him to retrieve some records, but does not have an ABE key with the policy to allow him to decrypt those records. Even if they all share their secret keys to decrypt entries to which none of the consumers individually have access, they will not be able to successfully decrypt any other entries. ABE systems are collusion resistant (individuals cannot combine attributes on their secret keys to satisfy a given policy); because each secret key is generated with a random seed, combining keys cannot create a new meaningful key [1].

Scenario 3: Data owner smartphone compromise. If the attacker physically obtains the data owner's smartphone, she is unable to gather information if the device is locked. If the device is unlocked, the attacker would be able to open ShareApp. We anticipate that strong implementations of our app would require the entry of a password, fingerprint, or other second-factor authentication before allowing the user to modify the access-control policies or consumer list, and to view recent data points. The attacker would never be able to access the stored keys and seeds, because those are not exposed to the phone user (and as we assume a secure smartphone platform, he cannot access them through other means).

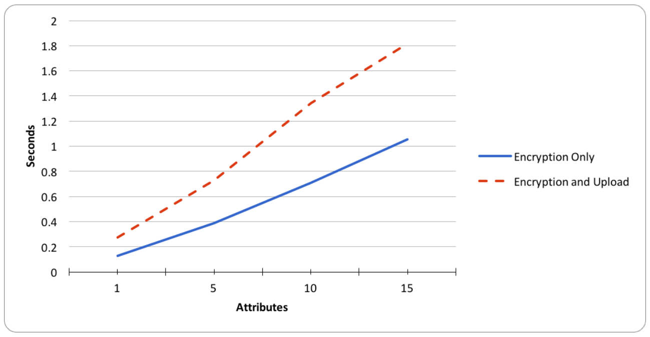


Figure 4: Execution time of encryption and upload

8.2 Performance analysis

Regarding the performance of the system, the data owner's smartphone is the most resource-constrained device in the system, so we focus on the execution time of the data encryption and upload. It is reasonable to presume that the computation on the consumer's computer will take equal or less time than the comparable operations on the owner's smartphone. We tested the SHA-256 indexing scheme relative to a simple counter indexing scheme and the KP-ABE encryption relative to plaintext (for different text lengths and attribute quantities) for both the upload and query processes. Each SHA-256 hash generation in the chain took approximately 0.103 milliseconds (0.0001 seconds). Re-indexing records is similarly efficient, as the process comprises hash generation and a trivial call to `mv` on the server.

KP-ABE uses ABE to encrypt an AES symmetric key and AES to encrypt the plaintext message. KP-ABE encryption time is independent of plaintext length (as AES runs approximately 1000 times faster than ABE, the ABE encryption of the AES key is the limiting factor of the encryption algorithm's runtime). Therefore, KP-ABE runtime only depends on the number of attributes. Figure 4 demonstrates the average time per run (in seconds) of KP-ABE encryption (solid blue) and overall data upload (dashed red) relative to the number of attributes. Plaintext data upload takes approximately 0.05 seconds, with no significant correlation to message length (over a message size range of 10B to 10MB). KP-ABE runtime is linearly related to the number of attributes.

It is important to note that no data encryption occurs while the smartphone is in interactive mode. All encryption happens in the background, and is done without impacting the user experience on the smartphone. If need be, ShareApp's background task could also run as a low-priority task so as to not affect the runtime of foreground apps.

9 DISCUSSION AND FUTURE WORK

Although our performance evaluation has established that the execution time of KP-ABE is not restrictive, pairing-based cryptography (the cryptographic root of KP-ABE) is an order of magnitude slower than traditional public-key encryption [9]. Because ShareABEL hinges on KP-ABE, the computationally-intensive nature of pairing-based cryptography will remain a limitation (although newer smartphones continue to be more powerful). We anticipate that ShareABEL's encryption time (about 0.1–1.5 seconds per record, for policies with 1–15 attributes) would be acceptable for data streams that produce new data points less frequently than once per minute – which we expect is true for most interesting mHealth data streams. Data streams with higher data rates can batch data points – for example, by batching a minute's worth of data into every record – and easily maintain acceptable throughput.

Our security model excludes traffic-analysis and denial-of-service attacks, and it would be worth investigating extensions to ShareABEL that could allow relaxation of those assumptions. Furthermore, our security model assumes that the data owner's smartphone and data consumer's computer are not compromised.

It would be worth exploring mechanisms to secure sensitive key material even in the event of operating-system compromise. For example, emerging trusted hardware mechanisms establish secure containers for computation and storage [18]; our system could use these mechanisms to store and manipulate the secret keys and seeds, and the mHealth data itself. Such hardware exists and is becoming more common; modern smartphones have trusted-hardware capabilities and currently use these containers to secure sensitive information such as credit-card information for Apple or Google Pay. Similarly, computers with Intel SGX technology could protect keys and data in the ShareView [5].

10 CONCLUSION

The development and use of mHealth apps and devices are a growing trend that promises great opportunities to improve health and wellness, increase access to healthcare, and reduce the cost of health services. We need systems to leverage mHealth data without compromising the privacy of the data owners or the integrity of the data. The design of a secure and privacy-preserving system that is practical for deployment remains a research challenge.

In this thesis, we proposed ShareABEL, a novel framework that supports owner-driven cryptographically-enforced access control of mHealth data. This system leverages hash chains and attribute-based encryption to cryptographically-enforce access to an owner's data by a variety of data consumers, with little trust required on a shared database server. By creating an end-to-end system from a data owner's smartphone to a data consumer's device, we are able to support a wide variety of applications from preventative medicine to remote monitoring of research subjects. Our evaluation of a preliminary prototype shows that this approach can be implemented efficiently on current common hardware platforms.

11 ACKNOWLEDGEMENTS

This research results from a research program at the Institute for Security, Technology, and Society, supported by the National Science Foundation under award numbers CNS-1314281, CNS-1314342, CNS-1619970, and CNS-1619950. Emily Greene's research has also benefited from the following Dartmouth awards and programs: Women in Science Project, Sophomore Science Scholars, Presidential Scholars, and the James O. Freedman Research Assistant grant for full-time work during Winter 2016. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors.

Emily would also like to thank her thesis advisor, Professor David Kotz, for his help and guidance throughout the many parts of this process, from solidifying the initial research questions to reviewing draft papers and presentations. She is also grateful for his support as her research advisor in the Kotzgroup over the past three years. Emily would also like to thank the two other members of her thesis committee, Professors Sean Smith and Charles Palmer, for their time and evaluation of her thesis. Emily thanks all of the members of the Kotzgroup and Amulet team for their advice and support, especially David Harmon, whose thesis is closely related to hers and helped to define the interface between ShareABEL and mHealth devices. Professor Matt Green (Johns Hopkins University) also contributed guidance, based on his expertise in Attribute-Based Encryption. Finally, Emily thanks her family and friends for their support throughout this process.

REFERENCES

[1] Joseph A. Akinyele, Christoph U. Lehmann, Matthew D. Green, Matthew W. Pagano, Zachary N. J. Peterson, and Aviel D. Rubin. 2010. *Self-Protecting Electronic Medical Records Using Attribute-Based Encryption.* Technical Report 2010/565. Cryptology ePrint Archive. http://eprint.iacr.org/2010/565

[2] Moreno Ambrosin, Mauro Conti, and Tooska Dargahi. 2015. On the Feasibility of Attribute-Based Encryption on Smartphone Devices. In *Proceedings of the Workshop on IoT Challenges in Mobile and Industrial Systems (IoT-Sys)*. ACM, 49–54. https://doi.org/10.1145/2753476.2753482

[3] Josh Benaloh, Melissa Chase, Eric Horvitz, and Kristin Lauter. 2009. Patient controlled encryption: ensuring privacy of electronic medical records. In *Proceedings of the ACM Workshop on Cloud Computing Security (CCSW)*. ACM, 103–114. https://doi.org/10.1145/1655008.1655024

[4] Kelly Caine and Rima Hanania. 2013. Patients want granular privacy control over health information in electronic medical records. *Journal of the American Medical Informatics Association* 20, 1 (01 Jan. 2013), 7–15. https://doi.org/10.1136/amiajnl-2012-001023

[5] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086. (Jan. 2016). http://eprint.iacr.org/2016/086

[6] Angelo De Caro and Vincenzo Iovino. 2011. jPBC: Java pairing based cryptography. In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 850–855. http://gas.dia.unisa.it/projects/jpbc/

[7] Abigail English and Carol A. Ford. 2004. The HIPAA privacy rule and adolescents: legal questions and clinical challenges. *Perspectives on sexual and reproductive health* 36, 2 (2004), 80–86. http://view.ncbi.nlm.nih.gov/pubmed/15136211

[8] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. 2010. *Cryptography Engineering: Design Principles and Practical Applications* (1st ed.). Wiley. http://www.worldcat.org/isbn/0470474246

[9] William C. Garrison, Adam Shull, Steven Myers, and Adam J. Lee. 2016. On the Practicality of Cryptographically Enforcing Dynamic Access Control Policies in the Cloud (Extended Version). (26 April 2016). arXiv:1602.09069 http://arxiv.org/abs/1602.09069

[10] Google. 2017. Security Tips. Android Developers. (May 2017). https://developer.android.com/training/articles/security-tips.html

[11] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. 2006. Attribute-based Encryption for Fine-grained Access Control of Encrypted Data. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. ACM, 89–98. https://doi.org/10.1145/1180405.1180418

[12] E. Gudes. 1980. The Design of a Cryptography Based Secure File System. *IEEE Transactions on Software Engineering* SE-6, 5 (Sept. 1980), 411–420. https://doi.org/10.1109/tse.1980.230489

[13] Cheng Guo, Ruhan Zhuang, Yingmo Jie, Yizhi Ren, Ting Wu, and Kim-Kwang R. Choo. 2016. Fine-grained Database Field Search Using Attribute-Based Encryption for E-Healthcare Clouds. *Journal of Medical Systems* 40, 11 (Nov. 2016), 1–8. https://doi.org/10.1007/s10916-016-0588-0

[14] Josiah Hester, Travis Peters, Tianlong Yun, Ronald Peterson, Joseph Skinner, Bhargav Golla, Kevin Storer, Steven Hearndon, Kevin Freeman, Sarah Lord, Ryan Halter, David Kotz, and Jacob Sorber. 2016. Amulet: An Energy-Efficient, Multi-Application Wearable Platform. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, 216–229. https://doi.org/10.1145/2994551.2994554

[15] Stephen Intille. 2016. The Precision Medicine Initiative and Pervasive Health Research. *IEEE Pervasive Computing* 15, 1 (Jan. 2016), 88–91. https://doi.org/10.1109/mprv.2016.2

[16] Ming Li, Shucheng Yu, Kui Ren, and Wenjing Lou. 2010. 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*, Ozgur Akan, Paolo Bellavista, Jiannong Cao, Falko Dressler, Domenico Ferrari, Mario Gerla, Hisashi Kobayashi, Sergio Palazzo, Sartaj Sahni, Xuemin S. Shen, Mircea Stan, Jia Xiaohua, Albert Zomaya, Geoffrey Coulson, Sushil Jajodia, and Jianying Zhou (Eds.). Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Vol. 50. Springer Berlin Heidelberg, Chapter 6, 89–106. https://doi.org/10.1007/978-3-642-16161-2_6

[17] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. 2005. Seeing-is-believing: using camera phones for human-verifiable authentication. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, 110–124. https://doi.org/10.1109/sp.2005.19

[18] Travis Peters. 2017. *A Survey of Trustworthy Computing on Mobile & Wearable Systems.* Technical Report TR2017-823. Dartmouth College. http://www.cs.dartmouth.edu/reports/abstracts/TR2017-823/

[19] Robert Reeder, Clare-Marie Karat, John Karat, and Carolyn Brodie. 2007. Usability Challenges in Security and Privacy Policy-Authoring Interfaces. In *Human-Computer Interaction âĂŞ INTERACT 2007*, Cécilia Baranauskas, Philippe Palanque, Julio Abascal, and Simone Barbosa (Eds.). Lecture Notes in Computer Science, Vol. 4663. Springer Berlin / Heidelberg, Chapter 11, 141–155. https://doi.org/10.1007/978-3-540-74800-7_11

[20] Robert W. Reeder, Lujo Bauer, Lorrie F. Cranor, Michael K. Reiter, Kelli Bacon, Keisha How, and Heather Strong. 2008. Expandable grids for visualizing and authoring computer security policies. In *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI)*. ACM, 1473–1482. https://doi.org/10.1145/1357054.1357285

[21] Amit Sahai and Brent Waters. 2005. Fuzzy Identity-Based Encryption. In *Proceedings of Advances in Cryptology (EUROCRYPT)*. Lecture Notes in Computer Science, Vol. 3494. Springer-Verlag, 457–473. https://doi.org/10.1007/11426639_27

[22] Saul Shiffman. 2009. Ecological momentary assessment (EMA) in studies of substance use. *Psychological Assessment* 21, 4 (2009), 486–497. https://doi.org/10.1037/a0017074

[23] Jeff Six. 2011. *Application Security for the Android Platform.* O'Reilly Media. http://shop.oreilly.com/product/0636920022596.do

[24] Changji Wang and Jianfa Luo. 2013. An Efficient Key-Policy Attribute-Based Encryption Scheme with Constant Ciphertext Length. *Mathematical Problems in Engineering* 2013 (2013), 1–7. https://doi.org/10.1155/2013/810969

[25] Syed Zain, Philip W. L. Fong, Jason Crampton, and James Sellwood. 2015. Relationship-Based Access Control for an Open-Source Medical Records System. In *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM, 113–124. https://doi.org/10.1145/2752952.2752962

[26] Zeutro LLC. 2016. An Introduction to Attribute-Based Encryption. White paper. (May 2016). http://www.zeutro.com/docs/zeutro_abe_whitepaper.pdf

[27] Liang Zhang, Josiah Brann, Michael Peyton, and Shucheng Yu. 2014. Enhancing Data Security in Cloud Computing: Build an Encrypted Personally Controlled Health Records Platform on Indivo X. Online poster. (June 2014). https://sites.google.com/a/ualr.edu/reu-project-by-liang-zhang/home

[28] Yao Zheng. 2011. *Privacy-Preserving Personal Health Record System Using Attribute-Based Encryption.* Master's thesis. Worcester Polytechnic Institute. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.469.3312

[29] Sebastian Zickau, Dirk Thatmann, Artjom Butyrtschik, Iwailo Denisow, and Axel Küpper. 2016. Applied Attribute-based Encryption Schemes. In *Proceedings of the International Conference on Innovations in Clouds, Internet and Networks (ICIN)*. IFIP Open Digital Library, 88–95. http://dl.ifip.org/db/conf/icin/icin2016/1570228068.pdf