Smart Health 12 (2019) 49–65

Contents lists available at ScienceDirect

Smart Health

journal homepage: www.elsevier.com/locate/smhl

Secure sharing of mHealth data streams through cryptographically-enforced access control

Emily Greene, Patrick Proctor, David Kotz*

Dartmouth College, 6211 Sudikoff Lab, Hanover, NH 03755-3510, USA

ARTICLE INFO

Keywords:
Mobile health
MHealth
Security
Privacy
Encryption
Electronic health records

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 ShareHealth, 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 make three contributions: (1) they apply cryptographically-enforced access-control measures to stream-based (specifically mHealth) data, (2) they recognize the temporal nature of mHealth data streams and support revocation of access to part or all of a data stream, and (3) they depart 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.

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 collection only occurs 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. Because of the continuous nature of this data, securing mHealth data is more complex than securing EMR data. Researchers can also use mHealth technologies to measure subjects' physiology, behavior, and activities in their regular free-living conditions, deepening researchers' 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.

Unfortunately, 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; (Guo et al., 2016). While many security solutions currently exist to encrypt and protect data, none of these solutions deal with temporal granularity.

Studies have shown that some patients express the desire to restrict access to health data depending on the time it was collected; (Caine et al., 2015). Therefore, temporal access control is an important option for these patients, including control of their mHealth

* Corresponding author.
E-mail addresses: Emily.C.Greene.17@Dartmouth.edu (E. Greene), Patrick.O.Proctor@Dartmouth.edu (P. Proctor), David.F.Kotz@Dartmouth.edu (D. Kotz).

https://doi.org/10.1016/j.smhl.2018.01.003
Received 13 October 2017; Received in revised form 19 December 2017; Accepted 30 January 2018
Available online 17 April 2018
2352-6483/ © 2018 Elsevier Inc. All rights reserved.

data. An mHealth-device owner may not want to share the entirety of her mHealth data history with each of her providers, as some historical data may be sensitive and not applicable to current conditions. Patients want granular privacy control over which health information their doctors can access; (Caine & Hanania, 2013). There are data that are applicable and necessary to be viewed by her cardiologist, but the data owner may not want her physical therapist to see the same information. 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* allows owners to optionally share subsets of their mHealth-device data with others, and these access-control policies may change over time; (Benaloh, Chase, Horvitz, & Lauter, 2009; Reeder, Karat, Karat, & Brodie, 2007). 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 paper 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 her apps and devices. Our system, ShareHealth, will help healthcare professionals treat patients without sacrificing the security of patient data.

With ShareHealth, we focus solely on individuals' mHealth data. We envision that unlike other forms of health data, such as the data contained in an EMR, each mHealth app or device produces data on a continuous basis, resulting in an ongoing *data stream* of a specific type of data. These data streams vary. Some streams produce new data points frequently (e.g., step counts, measured from 100 Hz accelerometer data) and others rarely (e.g., weight, typically measured a few times a week). Some data 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. We assume that data flows from one or more mHealth apps or devices, through ShareApp (the data owner's smartphone app for ShareHealth), and into ShareBase (a cloud storage system). There it is stored, in encrypted form, until it is accessed by data consumers according to policies set by the data owner.

One must consider two dimensions when controlling access to mHealth data streams: data type and data time. That is, one must determine whether access has been granted to a given data type (such as heart rate), and whether access has been granted for the time the data was collected. ShareHealth cryptographically enforces access control by both type and time period, using a combination of two cryptographic approaches: encryption to control access by data attribute and hashing to control access by data time. ShareHealth enables both temporal and attribute-based access control functionality, with fine-grained revocation mechanisms, using a novel combination of two well-tested cryptographic primitives: attribute-based encryption and hash chaining.

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 finite 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 rate that mHealth data is produced.

Our second contribution is temporal access control, a means to temporally segment each data stream using a pseudo-random indexing scheme. This feature enables owners to share the entire data stream, or only time-specific subsets of the stream (e.g., data collected prior to a specific date). The problem of controlling access to specific time periods of stream data is an open question not addressed by previous literature, which we solve in this paper.

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.

Organization: We begin with the foundation, giving necessary background information and pertinent definitions in Section 2, and the security model in Section 3. We then present our solution, focusing on access control and revocation support, in Section 4. We then detail our implementation in Section 5, followed by an evaluation of the system in Section 6. We explore several use cases in Section 7 and discuss related work in Section 8. We discuss limitations to our system and interesting extensions in Section 9, and draw final conclusions in Section 10.

2. Background

Cryptography has long been used to protect sensitive data from adversaries. In ShareHealth, we seek to not only use cryptography as a means to safeguard data confidentiality, 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; (Gudes, 1980), for example. Instead of developing a new cryptography scheme, we leverage existing implementations of robust cryptographic schemes and focus on system design and security.

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, or an elderly parent with an adult child.

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 key exchange between data owner and data consumer (Section 4.4).

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-s interval, or the step-count for a 5-minute interval), and a *data stream* is a sequence of data points measured on a periodic basis and uploaded as they occur or in batches.

Each data stream has one *source* – an mHealth app or mHealth device; thus, 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).

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.[1] Records are immutable – once produced, their contents 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 give data owners control over *access*: to grant permission to a data consumer to view the plaintext of specified record(s) (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, based on type and temporal restrictions. (We leverage attribute-based encryption (ABE) as part of our approach; see the ABE background figure for more information.) *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; (Reeder et al., 2007, 2008).

3. 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. This must be achieved in the face of threats from adversaries with capabilities described below, building on the trust assumptions described below.

3.1. Adversary model

We consider an adversary who is one of the following: (1) a curious family member, friend, or colleague of either the data owner or a data consumer; (2) a malicious individual with physical access to the data owner's smartphone or the data consumer's device; (3) an attacker with access to the network communications among the parties; (4) an authorized data consumer who desires access to more data than she is permitted; or (5) the honest-but-curious database itself. 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 between 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. Additionally, if Alice has the temporal permissions to access blood-pressure data from June and Bob has access to August blood-pressure data, they might collude to discover July data.

Although the adversary may intercept all network messages between 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.

3.2. Threat model

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

[1] 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.

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 process of sharing key material between the data owner and the data consumer, and either cause the data consumer to accept the wrong material or cause the data owner to believe that she is sharing these secrets with the data consumer but is really sharing them with the adversary.

3.3. 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 (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. Additionally, if ShareBase is compromised, we trust that an adversary will be unable to dump all data from the database in order to separate the records from their corresponding indices. 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 and leveraging one of many common mechanisms for key exchange (Section 4.4). 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 ShareView may connect to the database server via an anonymizing network such as Tor, if IP-level anonymity is desired.

4. Our approach

ShareHealth consists of three components: ShareApp (the data owner's smartphone app), 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.[2]

ShareHealth allows owners to specify access-control policies and cryptographically enforce those policies, so that only parties with the corresponding attributes are able to decrypt desired data. In this paper we focus on the underlying mechanisms to enable these policies, deferring the equally challenging human-interface problem for future work; see Section 9.

To enforce access control with an untrusted (honest but curious) data server, we use complementary 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. While we use well-established cryptography methods, it is the combination of these two methods that 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 specify more granularly who gets access to what 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.

[2] One might envision an alternative in which the data is encrypted at the mHealth device and merely passes through the smartphone as ciphertext; we anticipate, however, that many people will want to view or otherwise make use of mHealth data within smartphone apps, and thus the data should be available to the phone. Furthermore, the key-management challenges multiply if every mHealth device must be provisioned with keys. For ShareHealth, therefore, we chose to focus the encryption role on the owner's trusted smartphone.

In ShareHealth, we elected to use Attribute-Based Encryption (see Fig. 1 for background on ABE), and specifically Key-Policy Attribute-Based Encryption (KP-ABE) as opposed to Ciphertext-Policy ABE (CP-ABE), for several reasons: (1) because the attributes are content-based, different data consumers with similar roles could still have different access, (2) we assume ShareHealth 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 (Zheng, 2011), 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; (Akinyele et al., 2010).

4.1. Background on attribute-based encryption (ABE)

To address the "all or nothing" problem of traditional encryption systems, attribute-based encryption (ABE) was introduced by Sahai and Waters (2005). 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.); (Akinyele et al., 2010). 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 ShareHealth, 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; (Goyal, Pandey, Sahai, & Waters, 2006). 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; (Zeutro, 2016). For example, attributes on data obtained while working out could include: `type:heart-rate`, `location:gym`, `intensity:high`, and the data owner's coach could be issued the following key:

```
(type:heart-rate
    AND (intensity:high
      OR (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 the 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.

Both CP-ABE and KP-ABE can be used for access control on data. CP-ABE is a more favorable option when the number of data consumers is high, and the access-control policy will rarely change. In our use case, we elected KP-ABE because the number of users that will access the data is low in comparison to the quantity of data. As such, it is more efficient to have one policy per consumer than it is to have one policy per data record.

Key-Policy Attribute-Based Encryption (KP-ABE) allows ShareHealth to support the desired granularity. 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. It is easy for a data owner to add a new data consumer; the owner's SmartApp simply generates a new ABE key for that consumer, and shares it with that consumer as described in Section 4.4. 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.

While we chose KP-ABE as the existing cryptographic scheme to leverage to provide granular access control, one could use any cryptographic scheme with the following properties: flexible policies, the ability for multiple consumers to have overlapping access without encrypting multiple copies of the data, ease of re-encryption and revocation, and computationally efficient. KP-ABE is only one example of a cryptographic scheme with these desired properties, and this paper focuses not on improving existing cryptography, but on the novel application of a content-based access-control system to mHealth data.

When considering a content-based access-control policy, 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 attributes. The key-policy generation encompasses the construction of ABE private keys for each consumer, defined as a boolean expression applied to attributes.

Each record is tagged with pertinent attributes, defined by an attribute-tagging scheme and dependent on the data in the record. ShareHealth 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 tagging similar data with the same encompassing attribute (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. (Specifically, the key policy must match a sufficient subset (k of n) of the record's attributes, which are expressed as a boolean predicate; readers are referred to Fig. 1 for more about ABE.).

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 verify the message after decryption to ensure there have not been modifications to the record.[3] 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.

4.2. Hash-chain indexing scheme

ShareHealth also enforces access control through a specialized database indexing scheme that allows for temporal access control. This temporal access control is what makes ShareHealth the only existing solution that can adequately support access control on data streams. We use hashes as indices into the database to granularly express the beginning and end times of a specific subsequence of the data stream that the data owner wishes to share. The data owner can share overlapping temporal segments with different data consumers, can share multiple sequences of time ranges to which a consumer has access, and can change their access retroactively.

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, and each data type has its own hash chain. 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 unique key, so an adversary with access to one hash value (e.g., the untrusted database) cannot generate the next hash value without knowledge of this key. All consumers with some temporal access to a data stream also have access to its unique hash key.

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), in order to preserve freshness and allow for narrow temporal restrictions. 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 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 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 continue to have 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 any time by updating the seed for the hash chain without affecting others' access to the chain (see Fig. 1). 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. While hash chains are a well-known technology, hash chains have not been used previously to provide this sort of temporal access control.

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 temporal 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. Hash chains have security properties useful for stream access control (one-way, difficult to guess, collision resistant, easy to compute, etc.). Therefore, changing the seed for the hash indices of a data stream is equivalent to any traditional content revocation.

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 corresponds. 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 multiple hashes in each chain, it cannot determine which hashes are related nor associate hashes with their owner. We assume the hash-chain indexing scheme 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.

[3] We chose MAC-then-Encrypt as recommended in Ferguson, Schneier, and Kohno (2010), because in the alternative Encrypt-then-MAC approach, the MAC input and MAC value are both visible to an attacker. In the MAC-then-Encrypt configuration, the attacker only gets the ciphertext and the encrypted MAC value; the MAC input (i.e., the plaintext) and actual MAC value are hidden. This makes it much harder to attack the MAC than in the encrypt-first situation. Overall, the decision of whether to MAC first or encrypt first depends on whether authentication or confidentiality is more valued and whether to expose the MAC function or encryption function to the attacker. We value authentication in this system and prefer to expose the encryption function to direct attacks and protect the MAC as much as possible; (Ferguson et al., 2010).

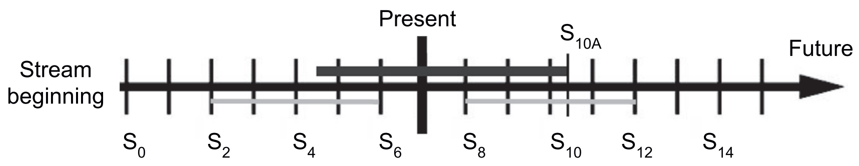


Fig. 1. Example of temporal granularity supported by ShareHealth. S_0 represents the first random seed generated for this data stream. Each black tick represents the beginning of a new SRI, with S_x being the random seed generated for the x^{th} SRI. Data currently generated by Alice, the data owner, is hashed using the 7^{th} hash chain. Bob (top thick dark gray) and Charlie (bottom thin light gray) are two data consumers with whom Alice would like to share different temporal subsets of this data stream. To grant Bob access, Alice shares the hash within the 4^{th} hash chain (seeded by S_4) that corresponds to the first record she would like Bob to access in addition to the hash key for that chain. Bob will continue to receive the new hash chain seeds until Alice terminates his access in the future by generating seed S_{10A} in the middle of the 10^{th} SRI and sharing the new seed with all data consumers except Bob. Alice desires to grant Charlie access to two different subsets of temporal data. She retroactively shares seeds S_2 through S_5 with Charlie in addition to the hash key for the hash chain, but does not share seed S_6 (thereby restricting his access to only the time period contained by SRI 2-5). She will also share seeds S_8 through S_{11} (including S_{10A}) as they are generated to share future stream data with him.

4.3. Access control and revocation model

In this paper, 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). To access plaintext for a record, the consumer must possess both the hash 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. It is the combination of these secrets that makes our system unique.

For example, to access a patient's heart-rate data from September 1, 2016, the doctor must have the random seed for the SRI 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.

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 master key that will be used by the owner to encrypt all records. Private keys for each of the data consumers are then generated based on the access granted to them by the policy.

For each data type and each data consumer, a 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 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, the hash key, and the seed for the next temporal segment (e.g., next week) only. Every time the seeds are refreshed, the owner's ShareApp sends the consumer's ShareView the new seed for that data type if and only 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 appropriate data consumers 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, less data may be at risk for exposure in the result of a revocation, and less strain is placed on the system for sending a large number of seeds at one time. For specific past records, data consumers are granted individual hashes without their corresponding hash keys, so they can only access the records to which those hashes correspond and no other hashes in the chain.

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. ShareHealth 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.[4] As our system hinges on temporal access control of continuous data streams, 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.

[4] 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 the cached data.

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; (Garrison, Shull, Myers, & Lee, 2016). 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.

4.4. 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 is unknown to the database and other possible adversaries. Even if data consumers collude, they cannot see more than the union of what they could see individually; collusion does not enable them to access data that none of them deserve, nor discover what (or how much) data they are unable to retrieve.

In addition to the KP-ABE policies, each data type 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 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 for key exchange; here is one example inspired by McCune, Perrig, and Reiter (2005). 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 own 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. If the owner approves the request, the consumer will get a notification on her ShareView detailing her new access. For a high-level overview of the communication protocols between ShareApp, ShareBase, and ShareView, refer to Fig. 2, which details both the sharing of cryptographic secrets and encrypted data.

It's worth noting that the protocols for key and seed distribution may be asynchronous; thus, the data consumers' computing devices need not be online and reachable whenever the data owner's ShareApp chooses to push new keys or seeds. The updated information could be carried over asynchronous secure channels (such as encrypted email messages), or could be held by the server for later retrieval by the data consumer when it comes back online. All communications are over secure channels, and are encrypted using standard protocols, such as TLS. Therefore, no data consumer or adversary could spy on the channel and determine the new hash-chain seeds or ABE keys. Similarly, the data owner's ShareApp can receive data from mHealth devices periodically (whenever a connection is established), queue data received, and send data to ShareBase in batches depending on Internet connectivity.

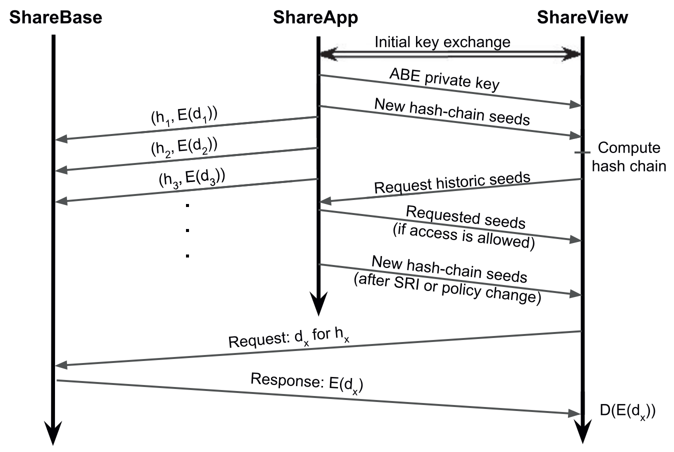


Fig. 2. An overview of the protocol between ShareHealth components. Seed and key sharing between ShareApp and ShareView occurs periodically after the initial key exchange. ShareView uses seeds to generate hashes (where the x^{th} value in a hash chain is denoted h_x) and requests the corresponding data for that hash (denoted d_x) from ShareBase. ShareBase returns the encrypted data (denoted $E(d_x)$) for ShareView to decrypt ($D(E(d_x))$). All communications between ShareApp, ShareBase, and ShareView are encrypted using standard protocols, such as TLS.

4.5. End-to-end system summary

Our approach applies a unique combination of cryptographic techniques to address the yet-unsolved problem of securely sharing mHealth data while allowing the device owner to decide what is shared with whom, and when. Fig. 3 provides an overview of our system components and their interaction; each component is described more thoroughly in the following section. When Bob, a data owner, shares his mHealth data with Alice, a data consumer, he relies on his personal relationship with Alice to complete the secure introduction (as described in Section 4.4). Once the introduction is complete, Bob specifies what subset of mHealth data he would like to share with Alice. By defining the desired data to share, Bob creates an access-control policy enforced by KP-ABE and a collection of keyed hash chains, one per data type, per week.

If, for example, Bob decided to share all heart-rate data from December with Alice, he would send Alice a KP-ABE key that included the heart-rate attribute in its associated policy, and the hash-chain key and seed for the first week of December. As the month of December continues, Bob will send Alice new hash-chain seeds every week, so that she can continue to query Bob's heart-rate data. Bob's ShareApp seamlessly uploads data through a secure connection to ShareBase, so Alice's ShareView can securely query ShareBase for the data – using the hashes generated from the provided hash-chain key and seed. At any time, Bob can choose to revoke access, and his ShareApp will send a re-index request to ShareBase, changing the hashes that correspond to the heart-rate data records. When Alice queries ShareBase with the old hashes associated with the revoked data, she will receive no results. Additionally, if Bob wants to provide Alice access to different data types, he can generate a new KP-ABE key for Alice and share that key and the related hash-chain seeds.

5. Implementation

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

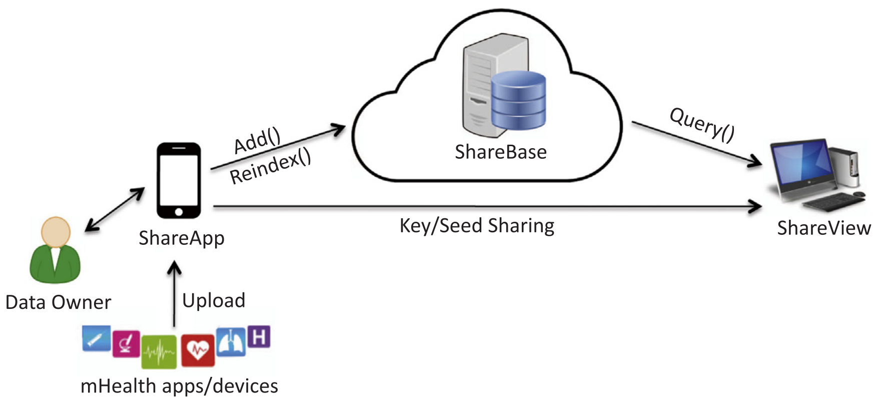


Fig. 3. The components of ShareHealth.

implementation we used Android and Linux platforms, but the ShareHealth architecture could be implemented for any common smartphone and desktop platform.

5.1. ShareApp: data owner's smartphone app

We implemented ShareApp by writing an application for Android 5.0 (Lollipop).[5] 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 her data.

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. By combining data from various mHealth apps/devices to forward to ShareBase, ShareApp makes the mHealth data upload process more streamlined and allows for aggregation of data across data sources. The smartphone is responsible for all seed and key management, and is the data owner's means of contact with data consumers.

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; (Hester et al., 2016). 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; (Google, 2017). (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 *datatype* tag that can be cross-referenced with a *data configuration file* 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 are queued for later upload.

ABE incorporation. We did not need to develop our own implementation of ABE. While there are many ABE libraries written in C, Java, and Python, there are few KP-ABE libraries supported on Android; (Zickau, Thatmann, Butyrtschik, Denisow, & Küpper, 2016). 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. One, *AndrABEn*, is a C-based library that is not supported on recent versions of Android. We used the other, a Java-based library by Zhang et al.[6] with corrections so it would properly generate keys, encrypt data, and decrypt data. We leveraged this library to generate the KP-ABE master key for the data owner, to generate the KP-ABE private keys for each consumer, and to encrypt each record with attributes based on the owner-defined 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.

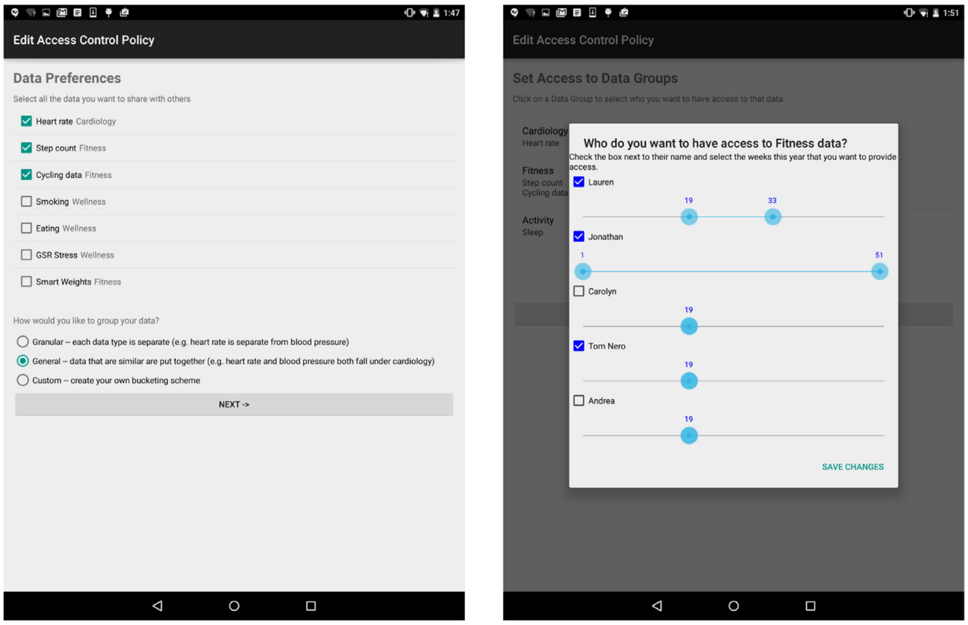
Defining policies. For the purposes of this prototype, ShareApp provides a simple graphical interface to allow the data owner to define an access-control policy (Fig. 4). (The development of a rich interface for defining such policies – and a proper study of that interface's usability – is an important HCI challenge worthy of future work and a paper in its own right; (Reeder et al., 2007, 2008).) After the data owner has added consumers with which she wishes to share some subset of her data, she can define an access-control policy for those consumers. There are two complementary components to creating an access-control policy.

First, the owner must define which types of data she would like to share and which attributes she would like to use to describe this data. This preliminary step will define the attribute universe that can be used for ABE key-policy generation.

Second, the data owner must define the temporal range of access for each data type (this range determines which data types' hash-chain seeds are shared with which consumers). Our implementation provides a preset SRI of one week, and each SRI is numbered according to the number of the week in the year in which the data was produced. Once a data owner has decided to share a type of data with a specific consumer, she must define a fixed range to share. The selection range defaults to only share the current week's seed, but the data owner can widen the range to any week in the past or in the future, with policies expiring at the end of each calendar year. If, during the definition of this policy, a data consumer has been granted access to data at the end of the year, ShareApp will automatically prompt the data consumer during week 52 to define access for the following year (thereby refreshing the access-control policy as the old one expires and ensuring that the data owner continues to permit access).

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

[5] We wrote and tested the app on a Nexus 9 tablet and Nexus 6 smartphones.
[6] https://sites.google.com/a/ualr.edu/reu-project-by-liang-zhang/home


1:47

Edit Access Control Policy

Data Preferences

Select all the data you want to share with others

- [✓] Heart rate Cardiology
- [✓] Step count Fitness
- [✓] 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

NEXT ->

1:51

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

Fig. 4. Defining the attribute universe and attribute-tagging scheme (left) and the temporal range of access for each consumer (right).

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.[7] 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 consumer 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.

SharedPreferences are stored as a file 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; (Six, 2011). ShareApp also needs to store the attribute-tagging scheme, the access-control policy, and the private keys of each of the data consumers (for distribution). These secrets are stored in a different section of SharedPreferences. 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.

5.2. *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 even 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 server runs on the Express Node.js web framework to connect with ShareApp and ShareView, and the data 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 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. To re-index the record, ShareApp generates a new random hash seed and hash key. For each hash in the current chain (generated using the old hash seed and hash key), ShareApp provides ShareBase the current hash and the new hash (generated by the same number of chaining iterations as the current hash), and the file with the name of the current hash is renamed to the new hash. Because ShareBase does not have access to the old hash key (to recreate the old hash chain), ShareBase is unable to arbitrarily re-index hash chains, ensuring the re-index operation cannot be leveraged as a denial-of-service attack. While in our prototype all files live in the same directory, this organization could be optimized

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

into a directory tree, or the content could be stored in a database. All communications between the database and the client applications (both ShareApp and ShareView) occur over secure HTTPS connections, and the clients can verify the identity of the database server.

5.3. ShareView: data consumer application

There are two major functions of ShareView, which we implemented as a command-line application for laptop or desktop computers.[8] First, ShareView can send and receive secure messages (after the initial key exchange to establish the secure channel – see Section 4.4) 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 is able to query the database. Because one data consumer could have many patients or subjects who use ShareHealth, ShareView maintains in a secure location 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 then 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 the database, Alice selects the data owner then selects 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 1), we envision future iterations of the portal with a robust graphical user interface.

Listing 1. Example Data Query.

```
Welcome to ShareHealth. 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 4:30pm: 68
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 4:33pm: 72
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.
```

6. Evaluation

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

6.1. Security analysis

Given the assumptions of the security model outlined in Section 3, 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 messages) would be unable to view the 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 the addition of content 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),

[8] Although one could also imagine ShareView as a mobile or web-based app (as long as the keys are stored securely), we selected a stand-alone application as a typical use case for providers and researchers.

and cannot view the data or which attributes the data possess (because the records are encrypted).[9]

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 be verified and the consumer would discover the tampering.

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 2: 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 3: 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 our system and is thus disclosed or vulnerable to disclosure. As in any such system, 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, they will not be able to successfully decrypt entries to which none of the consumers individually have access. 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; (Akinyele et al., 2010).

Scenario 4: Data-owner smartphone theft. If the attacker physically obtains the data owner's smartphone, he 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).

6.2. 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. Overall encryption time is effectively 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. Fig. 5 demonstrates the average time per run (in seconds) of KP-ABE encryption (solid blue) and overall data upload (dashed red)

[9] As noted above, we assume that owners and consumers concerned about IP-level anonymity can contact the server through an anonymizing network such as Tor.

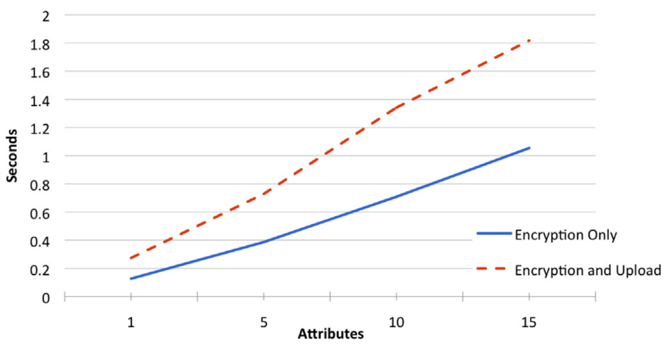


Fig. 5. Execution time of encryption and upload.

relative to number of attributes. Plaintext data upload takes approximately 0.05 seconds, with no significant correlation to message length. 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 minimize the impact on foreground apps.

Energy consumption is also a concern on smartphone devices. We measured the average energy consumption for uploading data from the data owner's smartphone. After charging two identical smartphones fully, we measured the battery life of the phone running ShareApp continuously (continuously uploading data) in comparison to the battery life of a control phone without our app. The control's battery died after 7 days 12 hrs and 45 mins, whereas the smartphone running our app lasted for 33 hrs and 57 mins. While the difference in battery life was substantial, it is not surprising; constant data upload to a remote server has a high energy cost. To extend the battery life, the app could be programmed to upload data in periodic batches, as opposed to continuous upload.

7. 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). When a patient is introduced to a new doctor or clinic, she may be hesitant to share all of her past medical data with the provider. Using ShareHealth's temporal access control, the patient can solely share the current data relevant to the treatment for her current health issue. 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 ShareHealth, data collected by mHealth apps and devices can be shared with the clinical team.

Case 2: Research. ShareHealth 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; (Intille, 2016). 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 participate in monitoring 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 temporal segment of the data stream from the past. For example, if a patient wants to give a new doctor insight about her asthma flares, she can share her respiratory and allergen exposure data for only those times in the past that she had breathing difficulties (e.g., the last two spring seasons). The new doctor can then compare data from the past to the ongoing data stream to monitor the patient's response to a new treatment regimen.

Case 4: Relevant medical history. While some medical conditions involve full body systems, other issues solely affect one part of the body. In these cases, patients may not want to share their entire medical history with a provider, but instead only share the data associated with that condition. For example, if a professional skier has a history of leg injuries, she will want to share her training and rehabilitation data from previous injuries to give her new physical therapist and coach context about her current injury. She would share mHealth data collected over her previous injured periods (e.g., the recovery from her last ACL tear) to show what treatments were effective, and what approaches should be modified to treat her current knee injury. She may also want to share mHealth data from her peak health to show the physical therapist her post-rehab goal abilities and vitals. While the athlete may trust her support team to help rehabilitate an injury, she may not want to share remaining mHealth data because this data may give insight into her proprietary elite training techniques that give her a competitive advantage. By empowering the skier to select only data from specific time periods to share with her physical therapist, ShareHealth gives her more control over her treatment.

8. 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 requestor is related to the resource owner; (Zain, Fong, Crampton, & Sellwood, 2015). 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 her training data with her coach, but only a subset of her vitals, and all of her vitals with her doctor, but only a subset of her training data, this relationship-based model would not support the granular control of the 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, they must assume that a patient's record is organized into a hierarchical data structure, allowing patients to grant access to a category without knowing all of the types of data that might eventually be included in that category. This system also allows doctors to define subcategories within the categories that they have been given permission to access; (Benaloh et al., 2009). 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.

Akinyele et al. propose a core scheme for applying ABE to EMRs; (Akinyele et al., 2010). 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 ShareHealth 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; (Li, Yu, Ren, & Lou, 2010). 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; (Ambrosin, Conti, & Dargahi, 2015).

Although ShareHealth uses the well-known (and well-tested) ABE cryptographic techniques, this paper contributes to the body of knowledge by recognizing the value of KP-ABE for the sharing of large data streams with a relatively small set of data consumers, whereas most of the prior literature has focused on CP-ABE; and furthermore, by extending ABE to support fine-grained temporal access control by combining KP-ABE with a novel use of hash chains.

In some ways, our temporal access-control scheme is reminiscent of the challenges posed by Digital Rights Management (DRM). Existing DRM techniques focus on per-object access control (such as a consumer's purchase of a given movie or music album), or a per-subscription access control (such as a consumer's subscription to a given radio or television channel), both of which can be accomplished with certificates and revocation lists. ShareHealth tackles a much finer-grained challenge, in which the data owner can give each consumer access to specific temporal ranges of a data stream, including historical data.

Our scheme could be extended with additional functionality like that proposed by Haber and Stornetta (1991), in which records published through ShareHealth are given irrefutable timestamps. Using their method (with our terminology), a data source would send a hash of each new data record to a trusted time server, which would sign a statement binding together the record hash, the current time, and some bits carried forward from earlier statements it has signed. This approach allows later consumers to verify the signature (and thus the timestamp) on each record, and furthermore to ensure that the data source and time server have not colluded to fake the timestamp. We could easily incorporate this approach into ShareHealth at several possible points: the timestamp could be requested by the data source (the mHealth device or app) itself, by ShareApp, or by ShareBase, with increasing generality. Their methods (and other methods that seek to provide a secure timestamp function) are complementary and orthogonal to the contributions of this paper, which leverage hash chains to add efficient temporal access-control features to the traditional ABE-based access control methods.

9. Discussion and future work

Our current approach pushes all mHealth data into data streams that flow through the ShareApp, into ShareBase, and then to the ShareView consumers. In some applications, owners may wish to delete data points; although owners could easily delete data records before they leave their device, or the ShareApp could provide owners an interface to selectively delete records before they are pushed to ShareBase, it is a policy decision as to whether records should be removed from ShareBase. (In most electronic health record systems, for example, data is never deleted; instead, it is marked as obsolete or incorrect; a similar approach could be taken for ShareHealth.) The details are left for exploration in 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; (Garrison et al., 2016). Because ShareHealth currently uses KP-ABE, the computationally intensive nature of pairing-based cryptography will remain a limitation (although newer smartphones continue to be more powerful). Additionally, the size of ciphertext produced by KP-ABE scales linearly with the number of attributes tied to the record. We expect that the number of attributes on each ciphertext will remain low, and tested policies with 1-15 attributes. Therefore, the linear scaling of ciphertext produced by KP-ABE should not be an issue. We also do not anticipate an issue with to the size of KP-ABE keys, as they are only used to encrypt an AES key after it has been used to encrypt the data. We anticipate that ShareHealth'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 exploring extensions to ShareHealth 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, emphasizing the importance of continued efforts to design secure smartphone operating systems.

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 a secure containers for computation and storage, as in Peters (2017); 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 Pay or Google Pay. Similarly, computers with Intel SGX technology could protect keys and data in the ShareView; (Costan & Devadas, 2016).

Finally, the development and evaluation of rich (yet usable) interfaces for ShareApp and ShareView are important areas of future work. One idea is to push some of the complexity onto the data consumer, e.g., for the data consumer to request access to a certain type of data, over a certain temporal range, through some interface designed for data consumers; the formatted request would be presented to the data owner in an understandable representation, and if the owner agrees, the request would be translated into directives for sharing data with the consumer.

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 paper, we propose ShareHealth, a novel framework that supports owner-driven cryptographically-enforced access control of mHealth data. This system leverages the combination of established cryptographic schemes, 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 system allows data owners to define flexible access-control policies, and is unique in the combination of content-based and temporal controls. By using hash chains as indices into the database, we support revocation without the need for expensive re-encryption of the data. Our evaluation of a preliminary prototype shows that this approach can be implemented efficiently on current common hardware platforms.

Acknowledgements

We appreciate the feedback of colleagues and anonymous reviewers. 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 and CNS-1619970. 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 sponsors had no role in study design; in the collection, analysis and interpretation of data; in the writing of the report; nor in the decision to submit the article for publication. 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.

Conflict of interest

None.

Appendix A. Supplementary data

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.smhl.2018.01.003.

References

Akinyele, J.A., Lehmann, C.U., Green, M.D., Pagano, M.W., Peterson, Z.N. J., & Rubin, A.D. (2010). Self-protecting electronic medical records using attribute-based encryption. Tech. Rep. 2010/565, Cryptology ePrint Archive. URL ⟨http://eprint.iacr.org/2010/565⟩.

Ambrosin, M., Conti, M., & Dargahi, T. (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), pp. 49–54. ACM. URL ⟨http://dx.doi.org/10.1145/2753476.2753482⟩.

Benaloh, J., Chase, M., Horvitz, E., & Lauter, K. (2009). Patient controlled encryption: ensuring privacy of electronic medical records. In *Proceedings of the ACM Workshop on Cloud Computing Security (CCSW)*, pp. 103–114. ACM. URL ⟨http://dx.doi.org/10.1145/1655008.1655024⟩.

Caine, K., & Hanania, R. (2013). Patients want granular privacy control over health information in electronic medical records. *Journal of the American Medical Informatics Association, 20*(1), 7–15 (URL ⟨http://dx.doi.org/10.1136/amiajnl-2012-001023⟩).

Caine, K., Kohn, S., Lawrence, C., Hanania, R., Meslin, E., & Tierney, W. (2015). Designing a patient-centered user interface for access decisions about ehr data: Implications from patient interviews. 30 (1), 7-16. URL ⟨http://dx.doi.org/10.1007/s11606-014-3049-9⟩.

Costan, V., & Devadas, S. (2016). Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086. URL ⟨http://eprint.iacr.org/2016/086⟩.

Ferguson, N., Schneier, B., & Kohno, T. (2010). *Cryptography engineering: Design principles and practical applications* (1st ed.). Wiley (URL ⟨http://www.worldcat.org/isbn/0470474246⟩).

Garrison, W.C., Shull, A., Myers, S., & Lee, A.J. (2016). On the practicality of cryptographically enforcing dynamic access control policies in the cloud (extended version). URL ⟨http://arxiv.org/abs/1602.09069⟩.

Google (2017). Security tips. Android Developers. URL ⟨https://developer.android.com/training/articles/security-tips.html⟩.

Goyal, V., Pandey, O., Sahai, A., & Waters, B. (2006). Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pp. 89–98. ACM. URL ⟨http://dx.doi.org/10.1145/1180405.1180418⟩.

Gudes, E. (1980). The design of a cryptography based secure file system. *IEEE Transactions on Software Engineering, SE-6,* (5), 411–420 (URL ⟨http://dx.doi.org/10.1109/tse.1980.230489⟩).

Guo, C., Zhuang, R., Jie, Y., Ren, Y., Wu, T., & Choo, K.-K. R. (2016). Fine-grained database field search using attribute-based encryption for e-healthcare clouds. *Journal of Medical Systems, 40*(11), 1–8 (URL ⟨http://dx.doi.org/10.1007/s10916-016-0588-0⟩).

Haber, S., & Stornetta, W. S. (1991). How to time-stamp a digital document. *Journal of Cryptology, 3*(2), 99–111 (URL ⟨http://dx.doi.org/10.1007/bf00196791⟩).

Hester, J., Peters, T., Yun, T., Peterson, R., Skinner, J., Golla, B., Storer, K., Hearndon, S., Freeman, K., Lord, S., Halter, R., Kotz, D., & Sorber, J. (2016). Amulet: An energy-efficient, multi-application wearable platform. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 216–229. ACM Press. URL ⟨http://dx.doi.org/10.1145/2994551.2994554⟩.

Intille, S. (2016). The precision medicine initiative and pervasive health research. *IEEE Pervasive Computing, 15*(1), 88–91 (URL ⟨http://dx.doi.org/10.1109/mprv.2016.2⟩).

Li, M., Yu, S., Ren, K., & Lou, W. (2010). Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings. In O. Akan, P. Bellavista, J Cao, F. Dressler, D. Ferrari, M. Gerla, H. Kobayashi, S. Palazzo, S. Sahni, X. S. Shen, M. Stan, J Xiaohua, A. Zomaya, G. Coulson, S. Jajodia, & J. Zhou (Vol. Eds.), *Security and privacy in communication networks: 50,* (pp. 89–106). Berlin Heidelberg: Springer (chap. 6. URL ⟨http://dx.doi.org/10.1007/978-3-642-16161-2_6⟩).

McCune, J.M., Perrig, A., & Reiter, M.K. (2005). Seeing-is-believing: using camera phones for human-verifiable authentication. In *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 110–124. IEEE Computer Society. URL ⟨http://dx.doi.org/10.1109/sp.2005.19⟩.

Peters, T. (2017). A survey of trustworthy computing on mobile & wearable systems. Tech. Rep. TR2017-823, Dartmouth College. URL ⟨http://www.cs.dartmouth.edu/reports/abstracts/TR2017-823/⟩.

Reeder, R., Karat, C.-M., Karat, J., & Brodie, C. (2007). Usability challenges in security and privacy policy-authoring interfaces. In C. Baranauskas, P. Palanque, J. Abascal, & S. Barbosa (Vol. Eds.), *Human-Computer Interaction (INTERACT): 4663*, (pp. 141–155). Springer Berlin / Heidelberg (URL ⟨http://dx.doi.org/10.1007/978-3-540-74800-7_11⟩).

Reeder, R.W., Bauer, L., Cranor, L.F., Reiter, M.K., Bacon, K., How, K., & Strong, H. (2008). Expandable grids for visualizing and authoring computer security policies. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, pp. 1473–1482. ACM. URL ⟨http://www.robreeder.com/pubs/xGridsCHI2008.pdf⟩.

Sahai, A., & Waters, B. (2005). Fuzzy identity-based encryption. In *Proceedings of Advances in Cryptology (EUROCRYPT)*, vol. 3494, pp. 457–473. Springer-Verlag. URL ⟨http://dx.doi.org/10.1007/11426639_27⟩.

Six, J. (2011). *Application Security for the Android Platform.* O'Reilly Media. URL ⟨http://shop.oreilly.com/product/0636920022596.do⟩.

Zain, S., Fong, P.W.L., Crampton, J., & Sellwood, J. (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)*, pp. 113–124. ACM. URL ⟨http://dx.doi.org/10.1145/2752952.2752962⟩.

Zeutro LLC (2016). An introduction to attribute-based encryption. White paper. URL ⟨http://www.zeutro.com/docs/zeutro_abe_whitepaper.pdf⟩.

Zheng, Y. (2011). *Privacy-Preserving Personal Health Record System Using Attribute-Based Encryption*. Master's thesis, Worcester Polytechnic Institute. URL ⟨http://citeseerx.ist.psu.edu/viewdoc/summary?Doi = 10.1.1.469.3312⟩.

Zickau, S., Thatmann, D., Butyrtschik, A., Denisow, I., & Küpper, A. (2016). Applied attribute-based encryption schemes. In *Proceedings of the International Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pp. 88–95. IFIP Open Digital Library. URL ⟨http://dl.ifip.org/db/conf/icin/icin2016/1570228068.pdf⟩.