Building a Secure mHealth Data Sharing Infrastructure over NDN

Saurab Dulal
University of Memphis
sdulal@memphis.edu

Nasir Ali
University of Memphis
cnali@memphis.edu

Adam Robert Thieme
University of Memphis
athieme@memphis.edu

Tianyuan Yu
UCLA
tianyuan@cs.ucla.edu

Siqi Liu
UCLA
tylerliu@g.ucla.edu

Suravi Regmi
University of Memphis
sregmi1@memphis.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

Lan Wang
University of Memphis
lanwang@memphis.edu

ABSTRACT

Exploratory efforts in mobile health (mHealth) data collection and sharing have achieved promising results. However, *fine-grained contextual access control* and *real-time data sharing* are two of the remaining challenges in enabling temporally-precise mHealth intervention. We have developed an NDN-based system called *mGuard* to address these challenges. mGuard provides a pub-sub API to let users subscribe to real-time mHealth data streams, and uses *name-based access control policies* and *key-policy attribute-based encryption* to grant fine-grained data access to authorized users based on contextual information. We evaluate mGuard's performance using sample data from the MD2K project.

CCS CONCEPTS

• **Networks → Naming and addressing**; • **Security and privacy → Access control**.

KEYWORDS

Named Data Networking (NDN), mHealth, Access Control, Real-time Data Sharing

ACM Reference Format:
Saurab Dulal, Nasir Ali, Adam Robert Thieme, Tianyuan Yu, Siqi Liu, Suravi Regmi, Lixia Zhang, and Lan Wang. 2022. Building a Secure mHealth Data Sharing Infrastructure over NDN. In *9th ACM Conference on Information-Centric Networking (ICN '22), September 19–21, 2022, Osaka, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3517212.3558091

1 INTRODUCTION

Wearable devices have seen a wide adoption in the consumer market, and are expected to grow exponentially in the near future [1]. This growth is fueled by their increasing use in health and wellness [33], which are made possible by an increasing number of mobile health (mHealth) biomarkers. The research community has been engaged in the discovery and validation of novel biomarkers and interventions using these wearables.

Our work aims to address two identified data access challenges in sharing mHealth data among researchers. First, because wearable sensor data may expose privacy-sensitive information about a user, it should only be accessible by authorized users. Currently, this access control is largely handled manually or through passwords, incurring high overhead while being subject to human error. Second, supporting real-time intervention for certain medical conditions requires retrieving and processing high-frequency sensor data in real-time, a functionality not supported by existing systems. High frequency data has proved useful to get insights from mHealth data and produce better biomarkers [29, 35]. For example, eating disorders [4] and smoking episodes [32] can be detected using data from high-frequency sensors such as accelerometer and gyro. If the data can be streamed and analyzed in real time, medical professionals can intervene as soon as the problems are detected [18].

We tackle the above challenges by utilizing the functionalities provided by NDN [2, 40], specifically *name-based access control* (NAC [43, 45]) which automates cryptographic key management for data access control and *NDN Sync* [15, 16] which enables real-time notification of newly published data. NAC encrypts data as soon as it is produced and shares the decryption key with authorized consumers. mGuard uses NAC-ABE [43] to grant contextual data access to authorized users [11]. It also provides a Sync-based pub-sub API to enable users to subscribe to mHealth data streams.

Our contributions can be summarized as follows: **First**, we designed a naming scheme for mHealth data (§3.2). This hierarchical, semantic naming structure can be applicable to other studies that collect sensing data from human subjects. **Second**, different from existing NDN access control solutions [31, 43], we developed an intuitive access control policy specification using *name-based data-centric* attributes, which allows data owners to (1) specify fine-grained policies based on time, location, or other context information, and (2) add policies for new data requesters over time (§3.4.2). **Third,** our pub-sub API publishes *manifests* that contain sets of data names, and uses Sync to notify data requesters of newly-published manifests §3.5). This design achieves real-time data dissemination while avoiding the overhead of signing, verifying, and sending notifications for individual data objects. **Lastly,** we evaluated mGuard's performance using Mini-NDN [20] with the NDN testbed topology and sample data from the MD2K project [12]. This experimentation not only demonstrated the feasibility of our design, but also revealed the design and implementation issues in NDN Python Repo and NAC-ABE (§5 and §6). Addressing these issues will improve the storage and access control support for future NDN applications.

2 BACKGROUND

Our work was motivated by challenges encountered by the MD2K center [12] in their development of an mHealth research cyberinfrastructure. In this section, we first provide a brief overview about mHealth data collection and distribution and the MD2K cyberinfrastructure. We then introduce some basic NDN concepts.

2.1 mHealth Data

Today's sensors in wearables, hand-held devices, and environments produce a massive amount of data that can be used for advanced healthcare and wellness management. A single user's wearable device may produce GBs worth of data each day. Wearable devices typically have limited onboard memory to store collected data, so they usually upload their data to an external storage device, e.g., a mobile phone, a local or a cloud server. The data is then processed to compute biomarkers to predict physiological and behavioral events. The server can also send computed results, e.g., activities performed, and/or interventions, back to the wearable or handheld devices. Once the data is uploaded to the server platform, data can be accessed by study coordinators, data scientists, or researchers.

Unlike most other mHealth systems, MD2K collects *high-frequency* data that varies from 1hz to 128hz from various sensors. *Real-time data collection and dissemination* of such high-frequency data can help detect eating disorders [4] and smoking episodes [32] and perform timely interventions. Currently, MD2K researchers can log on to a server to access and process data there, but cannot receive the data in real-time to perform timely intervention.

Data security and privacy becomes challenging [7], especially when distributing data generated at a high rate. Data collected by mHealth research, e.g., raw GPS data, and the computed biomarkers, are sensitive in nature. Researchers who conduct research involving identifiable human participant data streams are bound via federal laws and oversight by the Institutional Review Boards (IRBs) to preserve the rights and welfare of human research subjects. mHealth data is typically shared with terms of use overseen by an internal oversight body and IRBs. Moreover, study participants must be made aware and agree to the manner by which their data are disclosed and used. Currently, researchers' access to the MD2K server is authenticated via a user name and password, but there is no additional control based on the user's roles or any other restrictions once the user is inside the server. In some cases, MD2K staff members manually identify which data can be accessed by a researcher based on their access rights, then send the encrypted data to the researcher using a storage device. This process is slow and prone to human error.

The above limitations in the MD2K cyberinfrastructure are shared by other mHealth data collection systems. Most of these platforms do not support *real-time data distribution*, and their data access control do not support *fine-grained contextual access control policies* based on time, location, or other context information. More information about existing mHealth systems can be found in §7.

2.2 NDN

Today's Internet applications, by and large, are built on the web semantics of requesting data by names. Applications seek to get the named data, and do not care where they reside. However, because today's Internet protocol stack uses IP addresses to communicate, the application data names, e.g., https://mhealth.md2k.org/images/datasets/mOral_dataset.zip which identifies a dataset from an oral health study curated by MD2K, must be first translated to the IP addresses of a *specific server*, so that applications can fetch data from that server.

In contrast, Named Data Networking (NDN) [40] adopts the web's request-reply communication model and uses application data names in network layer data delivery. NDN's requests and responses work at a network packet granularity – each request, carried in an NDN *Interest*, contains the name of the requested data and fetches one NDN *Data* packet back. Each NDN forwarder uses the name in an Interest to determine where to forward the Interest.

NDN views a networked system as connected named entities which have various trust relations among each other. Each entity possesses security credentials and communicates with others via named, secured data: it signs every data packet it produces, so that Data packets can be cached and served by any device, and authenticatible by any data consumers. NDN enforces data access control via content encryption, and has developed name-based access control (NAC [43, 45]) to automate the key management in controlling data access. NAC differs from traditional access control schemes in two important ways: (a) NAC keeps data encrypted both in transit and in storage, achieving true end-to-end protection; and (b) it supports data access control across a spectrum of granularity from an entire data repository to a single data packet by utilizing NDN's hierarchical semantic naming. Cryptographic keys and security policies are also named contents, they can be fetched as named, secured data in the same way as any other types of data. NDN enables applications to define *security policies* and automates *key management and access control* using semantically meaningful data names, e.g. key names, device names, and user names [45]. These policies can be specified using defined turst schemas and their execution can be automated using NDN security libraries [37]. The basic components in the NDN design, i.e., hierarchical naming structure, data-centric security, and name-based data distribution, enable innovative solutions to address the challenges in mHealth data sharing in a systematic manner.

3 DESIGN

In this section, we first give an overview of mGuard, and then present its naming scheme, trust model, access control scheme, and pub-sub API.

3.1 Design Overview

Figure 1 shows the mGuard's system design. There can be multiple service instances, each serving the data from one mHealth data repository (e.g., MD2K). On each service instance, a *Data Adaptor* module[1] extracts data points from an mHealth data repository and matches them with NDN names and data attributes. It then passes the data and associated names and attributes to a *Publisher* module which creates NDN data objects, encrypts them, and stores them in an *NDN Repo*. In addition, the *Publisher* creates manifests containing the new data names, stores the manifests in the NDN Repo, and advertises their names to data requesters through a pub-sub API (§3.5). Note that each mHealth study may produce multiple

[1]This Data Adaptor is not needed if mHealth studies produce NDN data directly.

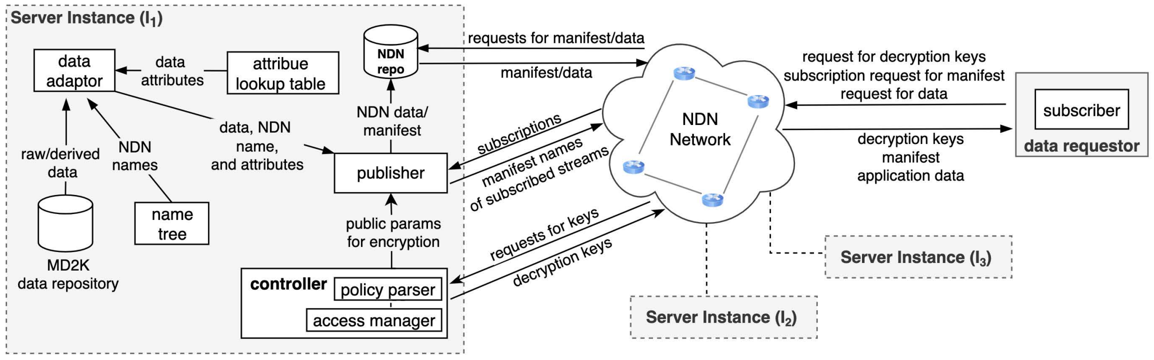


Figure 1: mGuard System Design

datasets. Each dataset is converted to a data stream identified by its name prefix, e.g., /org/md2k/mperf/dd40c/phone/accelerometer (see our naming scheme in §3.2), and all the data objects in a data stream share the same name prefix.

Each data requester runs a *Subscriber* module to subscribe to one or more data streams of interest through our pub-sub API (§3.5). When the pub-sub library gets notification about a new manifest name for one of the subscribed data streams, it fetches the manifest and verifies its authenticity. It then fetches the data objects using the data names contained in the manifest and decrypts them.

Each mGuard data repository has a trust anchor representing the organization that manages the repository. In order to authenticate the data from an mHealth repository, data requesters should be configured (either manually or through a bootstrapping process) with the corresponding trust anchor and trust schema. Every data object is encrypted using public parameters provided by the *Access Manager* and one or more attributes specific to the object (i.e., attribute-based encryption). The data object can only be decrypted by authorized requesters who are issued the corresponding decryption keys. A *Policy Parser* processes a policy configuration file to extract the access control policies for authorized data requesters and passes them to the Access Manager. Each data requester has a name and an associated public key which must be certified by trust anchor. After an *Access Manager* verifies the authenticity of a requester, it checks the access policies, and issues a decryption key to the requester if it has access right. More details about our trust model and access control mechanism can be found in §3.3 and §3.4, respectively.

3.2 Naming Scheme

Our first step in developing mGuard was to design an NDN naming scheme for mHealth data which can be used by both MD2K and other mHealth repositories. The current MD2K data comes from two sources: i) raw streams – data collected directly from phones/devices used by study participants, and ii) derived streams – data created by researchers and other users after processing the raw streams. Although the current MD2K system does not have a strict naming convention for these data streams, the existing practice loosely follows a common pattern: *<sensor>–<namespace>–<device-name>–<attachment>*, where *namespace* represents the scope of the collected data and helps organize data belonging to the same group, e.g. */org/md2k*, and *attachment* refers to where the device is attached, e.g., chest or wrist.

After examining hundreds of MD2K stream names, we developed an NDN naming scheme for MD2K's raw and derived streams following a hierarchical structure (Figure 2). A raw stream has the name prefix */<data-prefix>/<study>/<participant>/<device>/<sensor>/<attachment>* (the *attachment* component is optional), while a derived stream has the name prefix */<data-prefix>/<study>/<participant>/<algorithm>/<package>* (note that other name components can also be added to this hierarchical structure as needs arise). We ordered the name components based on their semantics and physical properties. For example, each data repository hosts datasets from multiple studies, so the study name (e.g. *mperf* and *mdot*) follows the data repository name (*/org/md2k*). Similarly, each study can have one or more participants, and the participants collect various data through different types of sensors embedded inside wearable or portable devices.

Once a data point is extracted from an MD2K stream, we assign a unique name to it. First, we take the corresponding data stream name as the prefix and add a predefined keyword "DATA" component to it. This keyword helps distinguish a data name from other types of names. Then, we add the timestamp of the data point to the end of the data name, so the final data name becomes */<datastream-prefix>/DATA/<timestamp>*, e.g., /org/md2k/mperf/dd40c/phone/gyro/DATA/1492637493876.

Naming data consistently across protocol layers and securing each piece of data independently provide a number of benefits in supporting application development, data and network management, secure data distribution, and data provenance [2, 40]. However, different layers may prefer different naming schemes. For example, NDN transport (e.g., Sync) may prefer sequential naming whereas the application may not. In fact, mGuard solves the naming inconsistency using manifests (§3.5.3) – mapping application names to sequential manifest names allows the application and Sync to use different naming schemes. We will explain other name types in our naming scheme in the relevant sections, e.g., attribute name and content key (CK) name in §3.4, and manifest name in §3.5.

3.3 Trust Model and Supporting Mechanisms

Trust in the mHealth data distributed by mGuard is derived from the trust anchor of each mHealth repository. The trust anchor verifies

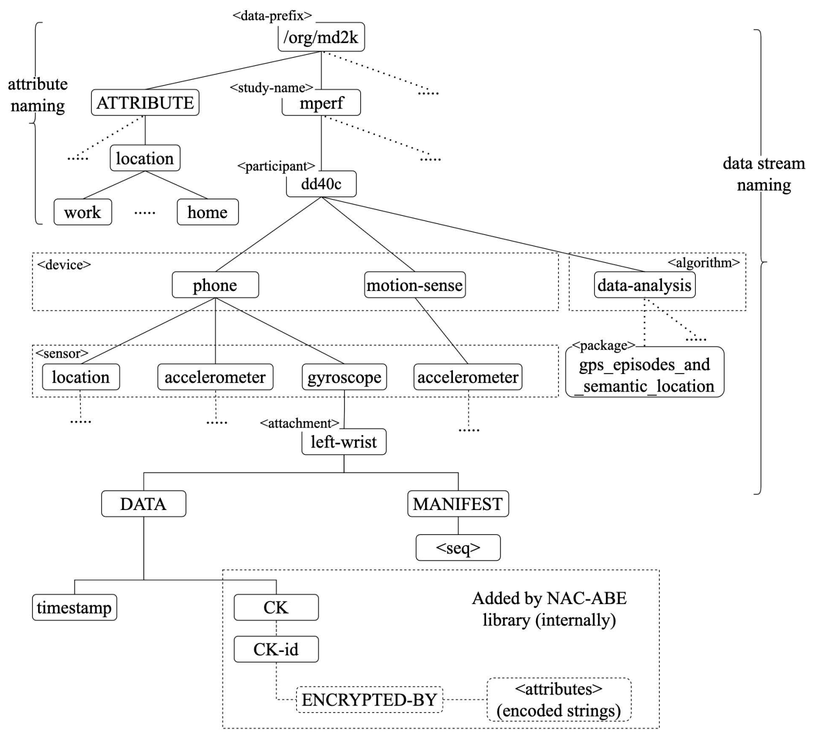


Figure 2: mHealth Data Naming Scheme

the identity of the entities in the system and certifies them. For example, the MD2K repository's trust anchor is the self-signed public key of the MD2K system operator, i.e., /org/md2k/KEY . This trust anchor certifies the public key of the *Publisher* (/org/md2k/publisher/KEY) and the public key of the *Access Manager* (/org/md2k/manager/KEY). The *Publisher* further signs the manifests of the data streams it produces (see §3.5), while the *Access Manager* signs the encryption material distributed to the *Publisher* and the decryption keys distributed to data requesters.

Data requesters in mGuard are named under participating organizations' prefixes (e.g., /edu/campus). During security bootstrapping, a data requester securely obtains the trust anchor and trust schema from the system operator. It then obtains a public-key certificate from the trust anchor following the NDNCERT [42] protocol. Afterward, the data requester provides its public-key certificate to the *Access Manager* which accepts the key after verifying that it is signed by the trust anchor. The *Access Manager* then generates a decryption key following the access control policy as defined in §3.4.1. The data requester then verifies that the decryption key is signed by the *Access Manager*, and received manifests are signed by the *Publisher*.

3.4 Access Control

The main components of access control are specifying access control policies (§ 3.4.1) and enforcing the policies to limit data access to only authorized users (§ 3.4.2).

3.4.1 Access Control Policies. mGuard supports defining fine-grained access control policies for study participants, study coordinators, and data administrators based on the terms of use and an understanding of privacy risks.

By default, no one should have access to the data except its owner. This will allow data owners to explore their data, evaluate privacy concerns, and define policies accordingly. Whenever a new data access request is approved (e.g., through the signing of a Non-Disclosure Agreement), the data owner can define a new policy by specifying (a) the data requesters' NDN names; and (b) data streams' names which those requesters should or should not be allowed access. Optionally, policies may include context-based attributes such as location, activity, and time for fine-grained access control.[2]

As shown in Figure 3, policies are structured in two sections. The first section specifies the unique policy ID and the NDN names of the data requesters. If all the members of an organization, e.g., a research lab, are allowed to access the data, their name prefix can be used here. The second section, attribute-filters[3], has two fields ("allow" and "deny") to provide the policy writer the ability to specify attributes of the data that the requesters can/cannot access. Requesters are allowed access to data defined in the "allow" field excluding any data defined in the "deny" field.

We can use both stream names and derived attributes such as location and activity in the "allow" field and the "deny" field. A stream name can be a prefix which includes all the streams under that prefix. We assume that the policy parser is configured with all the legitimate stream names so it can calculate the

[2]If there is an existing policy that has the same terms, then the new data requesters' names can be added to that policy.

[3]Policies may include more than one of these sections and may have names that differ from "attribute-filters".

Policy Description

```
policy-id        <unique-id>
requester-names  <names>
attribute-filters
{
  allow
  {
    <attribute 1>
    <attribute 2>
    ...
  }
  deny
  {
    <attribute 1>
    <attribute 2>
    ...
  }
}
```

Examples

```
policy-id         A
requester-names   /edu/mit/alice
attribute-filters
{
  allow
  {
    /org/md2k/
  }
  deny
  {
    /org/md2k/mperf/dd40c/phone/gps
  }
}

  /* brief: alice may access all
   streams under /org/md2k except
          dd40c's gps */
```

```
policy-id    B
requester-names   /edu/harvard/bob
attribute-filters
{
  allow
  {
    /org/md2k/mperf/dd40c/
    /org/md2k/ATTRIBUTE/location/home
    /org/md2k/ATTRIBUTE/location/gym
    /org/md2k/ATTRIBUTE/date > 20210901
  }
  deny
  {
    /org/md2k/ATTRIBUTE/activity/sleeping
  }
}
  /* brief: bob may access all streams
 under /org/md2k/mperf/dd40c if the data
  was collected at home or at the gym,
  after Sept 1, 2021, but not when the
        participant was sleeping */
```

Figure 3: Access Control Policy Structure

set of data streams that a requester is allowed to access using the streams listed in the "allow" field and "deny" fields. The attributes are split into groups based on their type. For example, a data requester may want access to the mPerf study data generated by the participant dd40c (/org/md2k/mperf/dd40c) either at home (/org/md2k/ATTRIBUTE/location/home) or in the gym (/org/md2k/ATTRIBUTE/location/gym). Here, the data stream name /org/md2k/mperf/dd40c belongs to one group, while the two location attributes, /org/md2k/ATTRIBUTE/location/home and /org/md2k/ATTRIBUTE/location/gym belong to a second group having the same type (location). Time-based attributes can also be used, e.g., one can add the attribute /org/md2k/ATTRIBUTE/date>20210901 to further restrict access to only data produced after Sept. 1, 2021. If we want to exclude data generated when the participant is sleeping, then we can add the attribute /org/md2k/ATTRIBUTE/activity/sleeping to the "deny" field.

Each policy is converted to a logic expression composed of stream names and derived attributes, which is used by NAC-ABE [17, 43] to generate a decryption key for each data requester listed in the policy (see the next section). These keys are distributed to the data requesters and are used to decrypt any data encrypted with attributes that satisfy the logic expression (i.e., the policy).

3.4.2 NAC-ABE Policy Support. As application data is semantically named and signed, it is natural to perform Named-based Access Control (NAC) [44] over the dataset. However, the traditional NAC's public-key cryptography has high complexity in key management. If the access manager is granting M granularities of access rights to N users each owning a public/private key pair, it has to manage M KEK/KDK (Key Encryption Key/Key Decryption Key) pairs for the M access granularities and produce $M \times N$ packets to distribute the KDKs to the N users. The overhead makes the system unscalable in MD2K's scenario where a large user group may request data access for a highly diverse dataset. Therefore, we decide to use Attribute-Based Encryption (ABE) techniques to aggregate the granularities.

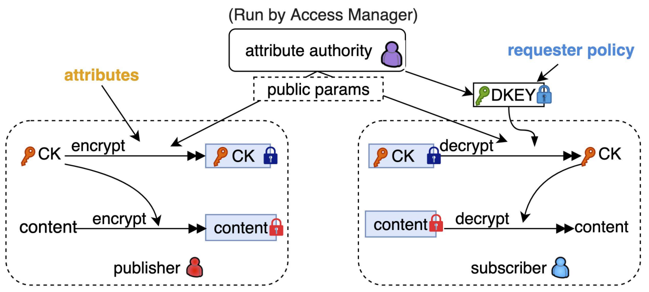


Figure 4: Use of NAC-ABE in mGuard

With ABE, each user has a set of attributes representing their access rights which are encoded into the user's decryption key. This key needs to be distributed only once to each user, so the initial key distribution overhead is only N packets.

NAC-ABE [17] is a name-based access control library that supports confidentiality and access control in NDN. It uses attribute-based encryption (ABE) for data encryption, leverages specially crafted NDN naming conventions to define and enforce access control policies, and provides automated key management [43]. The original implementation of NAC-ABE supports only CP-ABE (Ciphertext-Policy ABE) [5] which assigns user-centric attributes to each data requester, e.g., the requester's organization and role, and encrypts data with the associated access control policy (i.e., a logic expression of attributes). This approach requires knowing the exact access control policy when the data is encrypted. Whenever the policy changes, the data needs to be re-encrypted with the new policy. In our use case, the data owner may give access to data requesters from new organizations with different roles over time, so it is infeasible to determine the exact access control policy at data encryption time. Unlike CP-ABE, KP-ABE (Key-Policy ABE) [9] does not require the data owner to know the exact policy at data encryption time, so we extended NAC-ABE to support KP-ABE.

In KP-ABE, each piece of data is encrypted with some public parameters and a set of data-centric attributes, and the data can be decrypted by a data requester only if the requester has the correct policy in his/her decryption key. For example, if Alice has the policy "/org/md2k/mperf/dd40c and /org/md2k/ATTRIBUTE/location/gym" in her decryption key, then she can access only data encrypted with both attributes. Note that, in NAC-ABE, attribute-based encryption scheme is used to encrypt a *content key (CK)*, which is used to symmetrically encrypt and decrypt the data object (see Figure 4). This design increases the lifetime of the decryption key since attribute-based encryption is used on the content keys, which are a smaller set than the data objects, thus reducing the attack surface. This also improves the performance of encrypting and decrypting larger data objects, since the data is encrypted with a fast symmetric encryption scheme. Furthermore, this design increases the flexibility of the decryption keys, since any update on decryption keys will only require re-encryption of the content keys, not the data objects.

In mGuard, the *Access Manager* runs the *Attribute Authority* provided by NAC-ABE to distribute the public parameters used for encryption to the Publisher (Figure 4). The *Attribute Authority* also generates decryption keys according to requesters' access rights and distributes them to the requesters. Each decryption key is encrypted with the data requester's public key to ensure its confidentiality.

A data requester's decryption key should ideally be bound to a time window. This can be achieved by creating time-based policies so that each decryption key can only access encrypted data produced within a time window. This approach limits the power of each key and enhances the overall security. It also provides a default way of key revocation: a revoked key cannot be used to decrypt data generated outside its time window. If a data requester no longer has access permissions, they cannot obtain a new key anymore. There are scenarios where a key is compromised and must be revoked immediately without waiting for the key renewal. We can change the public parameters so that the revoked key can no longer decrypt data encrypted using the new public parameters. However, the Attribute Authority needs to generate a new key for every legitimate data requester and they each need to fetch a new decryption key, which can be expensive when there is a large number of data requesters. This is an open issue for future research.

3.5 Pub-Sub API and Library

The goal of our Pub-Sub API and library is to support applications to publish data in NDN data streams and subscribe to data streams as permitted by access control policies. This library helps to achieve real-time[4] data dissemination by guaranteeing the reception of the latest, verified data for each subscribed data stream.

3.5.1 API Design. An application can use the `publish()` function to publish data by specifying the data name, attributes, and content (Figure 5). The module implementing this function hides all the network-level abstractions such as Interest, Data, and security details, and exposes a very high-level API to publish data. The `subscribe()` and `unsubscribe()` functions allow an application to subscribe to or unsubscribe from one or more data streams by specifying the data stream name(s) and a callback function. The module implementing these functions is configured with the data requester's access control policy and trust schema. Through the callback function, it delivers decrypted and verified data belonging to the subscribed data streams to the application (Figure 5). All data fetching, decryption, and verification functions are implemented internally in the module, so as to keep the API simple and intuitive.

[4]Note that, in mGuard, real-time data transfer happens only after the data is received by an mHealth repository. NDN can also help reduce the delay in transferring data from sensors to the repository, but this is out of scope for our current project.

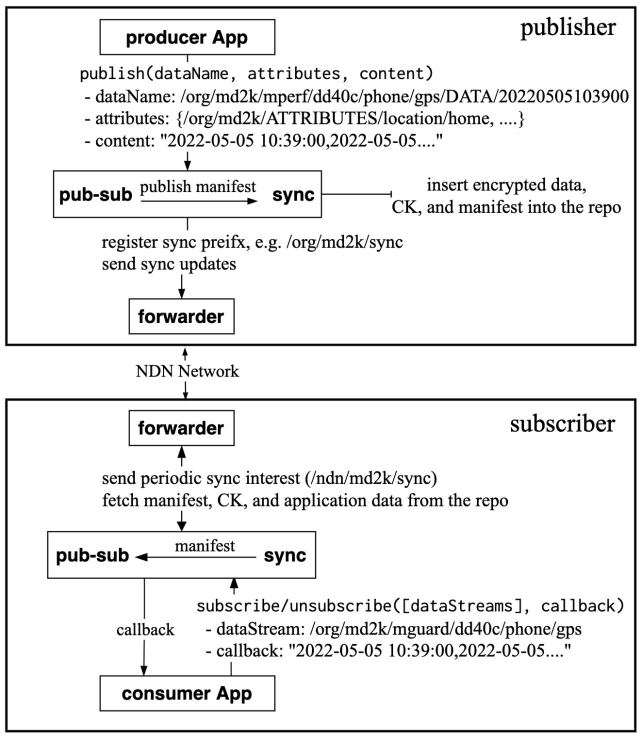

publisher
producer App
publish(dataName, attributes, content)
- dataName: /org/md2k/mperf/dd40c/phone/gps/DATA/20220505103900
- attributes: {/org/md2k/ATTRIBUTES/location/home, ...}
- content: "2022-05-05 10:39:00,2022-05-05...."
pub-sub
publish manifest
sync
insert encrypted data, CK, and manifest into the repo
register sync preifx, e.g. /org/md2k/sync
send sync updates
forwarder
NDN Network
forwarder
subscriber
send periodic sync interest (/ndn/md2k/sync)
fetch manifest, CK, and application data from the repo
pub-sub
manifest
sync
subscribe/unsubscribe([dataStreams], callback)
- dataStream: /org/md2k/mguard/dd40c/phone/gps
- callback: "2022-05-05 10:39:00,2022-05-05...."
callback
consumer App

Figure 5: Application Workflow Showing All Layers Involved in the Pub/Sub Process

3.5.2 Use of PSync. We built the pub-sub library on top of the *PSync* protocol [41] which uses Sync Interests (NDN Interests) for subscription and Sync Reply messages (NDN Data) for notification. Every subscriber sends a Sync Interest towards the publisher both periodically and after it receives a Sync Reply. When new data is generated, the publisher sends a Sync Reply message containing the new data name back to the subscriber. Note that PSync does not fetch the new data. It is up to the application (the pub-sub library) to decide whether to fetch the new data after getting a notification from PSync. For efficiency and name mapping purposes, instead of publishing individual data objects through PSync, our pub-sub library uses PSync to publish Manifests each containing a set of data names belonging to a data stream (see the next section).

3.5.3 Use of Manifest. Every NDN data object carries the publisher's signature which ensures both integrity and authenticity. However, when the data production rate is high, signing and verifying every data object may incur significant computation overhead. Publishing all the data object names through PSync may also lead to excessive Sync message overhead as well as high delay and losses.

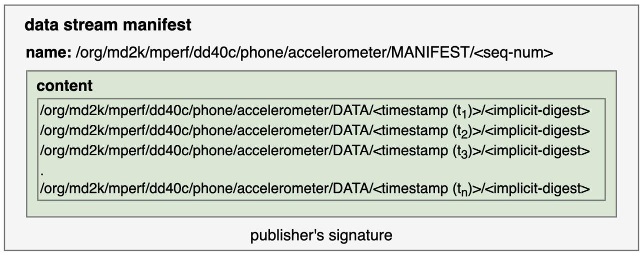

data stream manifest
name: /org/md2k/mperf/dd40c/phone/accelerometer/MANIFEST/<seq-num>
content
/org/md2k/mperf/dd40c/phone/accelerometer/DATA/<timestamp (t1)>/<implicit-digest>
/org/md2k/mperf/dd40c/phone/accelerometer/DATA/<timestamp (t2)>/<implicit-digest>
/org/md2k/mperf/dd40c/phone/accelerometer/DATA/<timestamp (t3)>/<implicit-digest>
/org/md2k/mperf/dd40c/phone/accelerometer/DATA/<timestamp (tn)>/<implicit-digest>
publisher's signature

Figure 6: Manifest Data Format

To solve this problem, we use Manifests to carry meta information about the data objects and verify the signatures of the manifests only. The exact meta information may vary depending upon the specific use cases [14, 34]. Our manifest design is inspired by [14], as it is simple to use and closely fits our use case. As shown in Figure 6, each manifest carries a list of full names (including the Implicit Digest) [23] of the data objects in a data stream.[5] The manifest name is the data stream name followed by the keyword MANIFEST and a sequence number. We use a sequence number here because a large set of data names need to be carried by multiple manifests with unique names and PSync requires sequential names.

When packetizing mHealth data, the *Data Adaptor* does not directly sign each data object using the data publisher's public key. Instead, it puts the digest (e.g., SHA-256) of the data into the signature field and passes the data object to the *Publisher* which uses the pub-sub API internally to publish the data. The pub-sub library does the following: (a) add the full name of each data object to a manifest, (b) insert the data object and its corresponding Content Key (CK) into the repo, (c) after accumulating a certain number of data object names in the manifest, or reaching a time limit since the last data object was received, create and sign a manifest data packet, (d) insert the manifest into the repo, and (e) publish the manifest's name through PSync. On the Subscriber side, the pub-sub library does the following: (a) learn the name of the new manifest through PSync, (b) send an Interest to fetch this manifest, and verify its signature, (c) for each data name in the manifest, fetch the corresponding data object, and verify its digest (this is much faster than verifying a public-key signature), (d) fetch the CK for the data object, (e) decrypt the data object, and (f) return it to the application through the callback function. Figure 7 shows the message sequence that takes place during the data publication and fetching.

4 IMPLEMENTATION

We implemented mGuard in C++. It uses the master branch of ndn-cxx [21], NFD [25], ndn-tools [22], PSync [26], and NDN Python Repo [19] as of August 20, 2022. In the original NDN Python Repo implementation, when data objects are inserted via the TCP bulk insertion protocol, the name of every data object is registered with NFD. This can cause massive overhead if the number of data objects is large. If multiple data objects share the same name prefix, registering the prefix should be sufficient. Thus, we made changes to the Repo implementation to support registering name prefixes with NFD. The prefixes and their granularity are specified via the repo configuration file.

[5]This is similar to the torrent file in BitTorrent.

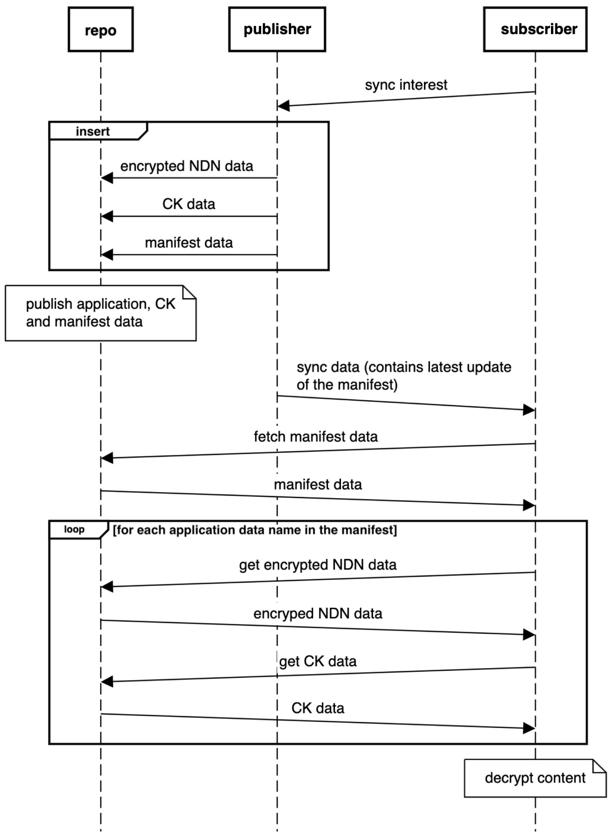


Figure 7: Message Sequence Diagram

As mentioned previously, we use the NAC-ABE library [17] for access control. Since the original implementation of NAC-ABE supports only CP-ABE, we added KP-ABE key generation and distribution to NAC-ABE using the OpenABE library [28] which provides KP-ABE functionality. In addition, the original NAC-ABE library generates a new Content Key (CK) for every data object. We changed the library to use the same CK for multiple data objects with the same attributes up to a configured number of objects. This reduces the overhead of generating and fetching CKs while maintaining data security. More specifically, if a user is allowed to access one data object, they should be able to access another data object with the same attributes, therefore using the same CK for both data objects will not allow unintended access. Additionally, restricting the number of data objects encrypted with the same CK limits exposure of the CK. In the data encryption function of NAC-ABE, we use a cache to store the CKs that may be used later.

Our policy parser converts access control policies into logic expressions of attributes (ABE Policies) corresponding to the data that the requester(s) should be allowed to access. Attributes of the same type are concatenated by OR, while those of different type are concatenated by AND. For example, in Figure 3, the attributes in Policy B's "allow" field are converted to the expression "(/org/md2k/mperf/dd40c) and (/org/md2k/ATTRIBUTE/location/home or /org/md2k/ATTRIBUTE/location/gym) and (/org/md2k/ATTRIBUTE/date>20210901)". If a policy includes more than one "attribute-filters" section, ABE policies will be calculated individually for each section and then concatenated with OR in the full ABE policy. The current implementation supports context-based attributes such as location and activity. Functionality for time-based attributes is a work-in-progress.

Due to the absence of support for the NOT operator in OpenABE, we have to divide the derived attributes listed in the "deny" field into sets according to their type (such as location, or activity) and add the compliment of each set, with respect to the set of all possible attributes for that type, to the ABE policy. For example, if the available location-based attributes are /org/md2k/ATTRIBUTE/location/gym, /org/md2k/ATTRIBUTE/location/home, and /org/md2k/ATTRIBUTE/location/work, and the access policy denies the /org/md2k/ATTRIBUTE/location/home attribute, the result is to allow access to data encrypted with either the /org/md2k/ATTRIBUTE/location/gym attribute or the /org/md2k/ATTRIBUTE/location/work attribute. While the implementation for denied stream names is different, the logic is the same. Given the policy in Figure 3 where /org/md2k/mperf/dd40c/ is an allowed stream name prefix and /org/md2k/mperf/dd40c/phone/gps is denied, every stream name under /org/md2k/mperf/dd40c/ will be allowed except for /org/md2k/mperf/dd40c/phone/gps.

5 EVALUATION

We evaluated the performance of mGuard via emulation using Mini-NDN [13], configured with the NDN testbed topology [24] of 37 nodes and 94 links.

5.1 Performance Metrics

We used the following metrics in our evaluation: (a) **encryption time** is time spent by the publisher encrypting data in a data object; (b) **decryption time** is time spent by the subscriber decrypting an encrypted data object; (c) **manifest propagation delay** is time between when the publisher starts to create a new manifest and when the manifest name is received by the subscriber via sync; (d) **manifest, data,** and **CK retrieval delay** are the times spent by the subscriber fetching the respective data from the repo; (e) **end-to-end delay** is the time between when a data object was published by the publisher and when it was received by the subscriber; (f) **maximum packet overhead** is the maximum number of packets sent and received by a node assuming there is no Interest aggregation or data caching; and (g) **measured packet overhead** the actual number of packets sent and received by a node. The overhead consists of Interest and Data packets for Sync, manifests, mHealth data, and keys.

All delays except for encryption and decryption are normalized by the round-trip time between the subscriber and the publisher to account for different network delays from node to node. To get an accurate representation of the data, we compute the average values for packet overhead and median values for all other metrics across three runs for each experiment.

5.2 Emulation Setup

In all experiments, we used one server instance and four data consumers. The server instance, consisting of the MD2K data repository, NDN repo, Controller, and Publisher, was connected to the Memphis node, while the consumers were connected to the UCLA, Washington University, Osaka, and Michigan nodes. For each experiment, the server processes application data received from the MD2K repository and the subscribers fetch this application data in real time. Additionally, we publish a new manifest after 50 data object names have been accumulated in it, or 100 milliseconds has passed since the last data object name was received. For our access control configuration, we used three streams under the name prefix /org/md2k/mperf/dd40c, i.e., /org/md2k/mperf/dd40c/phone/battery, /org/md2k/mperf/dd40c/phone/gps, and /org/md2k/mperf/dd40c/data_analysis/gps_episodes_and_semantic_location, along with Policy A and Policy B as shown in Figure 8.

```
policy-id      A
requester-names /edu/tu/alice, /edu/mit/bob, /edu/tu/
      hira, /edu/mit/ganga
attribute-filters
{
 allow
 {
  /ndn/org/md2k/mperf/dd40c
 }
}
/*brief: Alice, Bob, Hira, and Ganga shall access the
      streams under /ndn/org/md2k/mperf/dd40c */
```

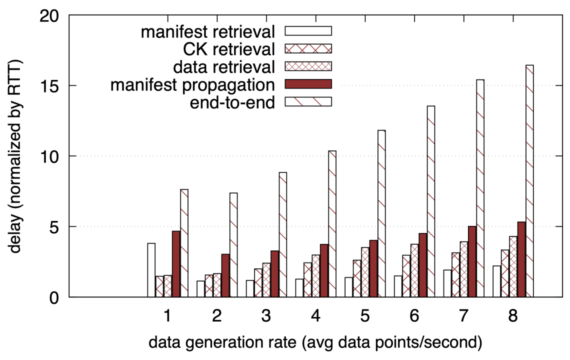
```
policy-id      B
requester-names /edu/tu/alice, /edu/mit/bob, /edu/tu/
      hira, /edu/mit/ganga
attribute-filters
{
 allow
 {
  /ndn/org/md2k/mperf/dd40c/phone/battery
 }
}
/*brief: Alice, Bob, Hira and Ganga shall only access
      battery stream */
```

Figure 8: Policies Used in Experiments

5.3 Rate Experiments

In this set of experiments, each data stream takes turns to publish a batch of data points and the aggregated data rate from the three streams ranges from 1 to 8 data points per second in different experiments. The publisher creates an NDN packet for each data point. Requesters use Policy A to access all the streams under /org/md2k/mperf/dd40c.

The results of the experiments in Figure 9 show that *the normalized delays increase with data generation rate.* When processing individual data points as single packets, repo insertion times increase as the data generation rate increases, causing an increase in the delay to retrieve data and CK packets. A higher manifest publication rate is also problematic for PSync to handle, as it requires a round-trip time to publish a new manifest (each publication consumes a pending Sync Interest at the producer and the next Sync Interest is sent by the consumer only after receiving the Sync Data packet carrying the name of the recently published manifest). Both of these factors contributed to a higher end-to-end delay as the data generation rate increased. Since packet sizes are unchanged, the cryptographic delays are mostly unchanged. We also analyze the packet overhead using the data from the rate experiment. Note that, in our definition, packet overhead includes both sent and received packets. The result, Figure 10, shows that *the measured packet count*



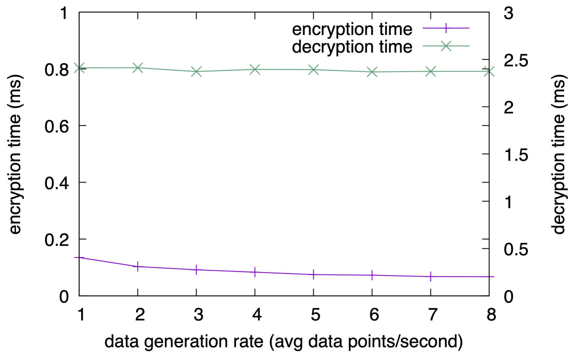


Figure 9: Network, Processing, and Cryptographic Delays for Rate Experiments

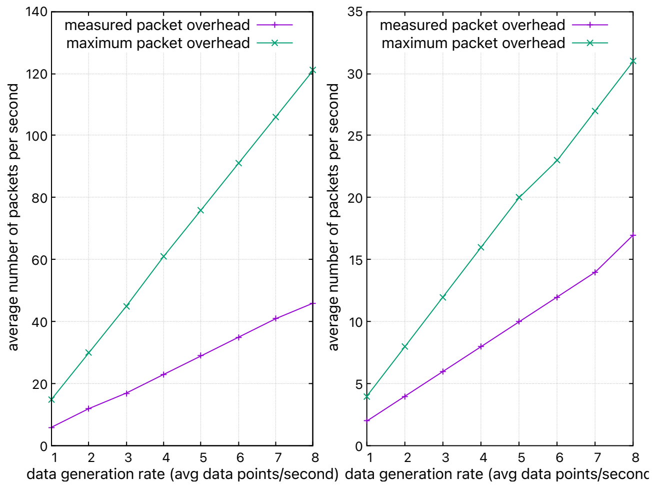


Figure 10: Maximum vs Measured Packet Overhead at Producer (left) and Consumer (right)

is about 61% and 47% lower than the maximum packet count at the producer and each consumer, respectively. This is a result of the Interest aggregation and data caching features offered by NDN.

5.4 Packet Size Experiments

We publish data from the three data streams at a constant rate of 300 data points per minute. The publisher combines data points with the same attributes into a single NDN packet. The maximum amount of data points per packet ranges from 10 to 70, as 70 data points almost exceeds the maximum NDN packet size (8KB). All consumers in this experiment use Policy B, so they can access only the battery stream /org/md2k/mperf/dd40c/phone/battery.

As we can see from Figure 11, *the normalized delays are constant or decreasing and are much improved compared to the rate experiments.* This is because a larger number of data points in a single packet will decrease the overall rate of data and manifest generation, thereby reducing the overall processing and propagation delays and reducing the number of packets consumers will fetch. However, *larger packet sizes resulted in slightly greater encryption and decryption times.*

6 LESSONS AND REMAINING ISSUES

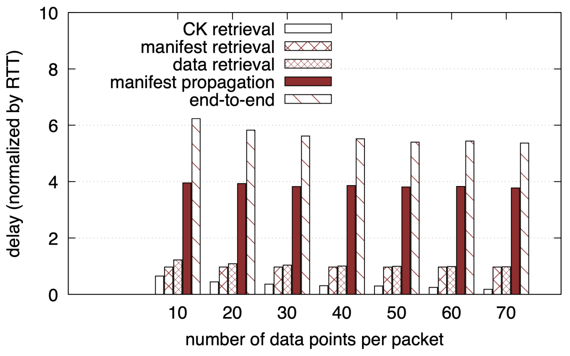
We initially designed the access control policies using syntax similar to database queries and firewall rules, as the MD2K data is stored in a database and we are familiar with access control in firewall configuration. After many revisions, we started using attribute names directly in the "allow" and "deny" fields, which simplified the policy specification and made the policies much more intuitive.

Our experimentation also revealed some implementation issues in the NDN Python Repo and NAC-ABE library. In the Repo's bulk insertion client, data insertion was synchronous, which caused our Publisher to wait for the insertion to complete. Moreover, a new TCP connection was established and closed for each insertion, resulting in unnecessary delay. These problems were fixed, considerably reducing the repo insertion time. We also found that the OpenABE library is not actively maintained. While we were able to make several changes to OpenABE to make it work, NAC-ABE cannot rely on it in the long term.

Moreover, the KP-ABE approach has a scalability issue when policies are composed of a large number of attributes, as the corresponding decryption keys may become very large. This can happen when a user is allowed to access many data streams. We currently use individual data stream names as attributes, but we plan to use stream name prefixes as attributes to solve this problem. Finally, our work does not address how study participants can verify how their data is shared. We will address this issue in future work.

7 RELATED WORK

Due to space constraint, we cannot provide a comprehensive review of mHealth systems. Instead, we point out the problems in a few state-of-the-art representatives to show our proposed enhancements can benefit them as well. The Fast Healthcare Interoperability Resources (FHIR) Server for Microsoft Azure provides a foundation to handle mHealth data [10]. FHIR is an Electronic Health Record (EHR) system and may not be able to support high-frequency mobile sensor data. To support mental health, the Non-Intrusive Individual Monitoring Architecture (Niima) prototype was developed [3] to handle data collection, storage, and privacy challenges related to mHealth data. Initially, data is stored as is and data privacy rules are only applied when exact data needs are defined. This may delay the availability of the data. RADAR-base supports data aggregation,



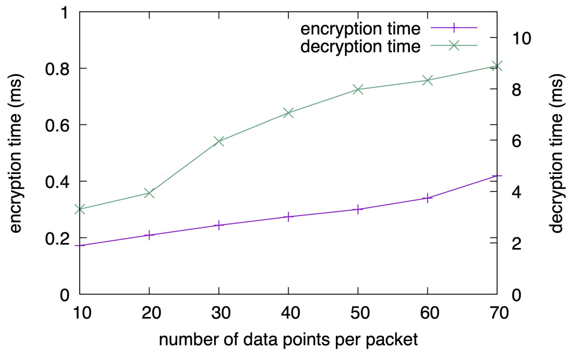


Figure 11: Network, Processing, and Cryptographic Delays for Packet Size Experiments

Table 1: Comparison of mHealth Systems Based on Key Features (* = *partially supported*)

System	High-Frequency Data	Real-time Data Distribution	Access Control	Contextual Access Control [11]
mGuard	Y	Y	Y	Y
Microsoft Azure FHIR [10]	N	Y*	Y	N
NIIMA [3]	N	N	Y	N
RADAR [30]	N	Y*	Y	N
Adaptive MapReduce [38]	Y	Y	N	N

management of studies, and real-time visualizations of data collected from wearable sensors [30]. Some of these platforms do not support high-frequency data or real-time data distribution. Moreover, their data access control schemes do not support fine-grained policies based on context [11]. Table 1 compares mGuard with some existing mHealth systems in terms of key features.

Realizing that standardized interfaces and shared components are critical for healthcare delivery and research, Estrin and Sim proposed the Open mHealth architecture that uses data exchange as the common layer of interoperability [6]. A crucial part of their vision is data exchange being user-controlled and privacy-aware across users, devices, applications, and vendor boundaries. However, this vision is challenging to achieve over TCP/IP's host-centric communication paradigm. As we have demonstrated, NDN provides the building blocks to realize the Open mHealth vision.

NDNFit [39] is an experimental NDN application for tracking and sharing personal fitness activity. It served as a use case for the initial development of NAC. In this work, we have developed intuitive access control policies that use semantic attribute names. We have also extended NAC to support context-based policies that can change over time.

Reddick et. al. proposed an ABE-based access control scheme for sharing genomics data over NDN [31]. There are two major differences between our approach and theirs. First, they use CP-ABE and user-centric attributes. In contrast, we use KP-ABE and data-centric attributes so new policies can be added over time more easily as explained in Section 3.4.2. Second, they directly encrypt data using ABE. However, we use symmetric content keys to encrypt data and then use ABE to encrypt content keys for efficiency and security reasons, as explained in Section 4.

PSIRP/PURSUIT [8], an early ICN architecture, offers native support for network-layer pub-sub mechanisms. In this architecture, publishers create an RID and a scope of the publication, which are forwarded to the rendezvous node. When subscribers first receive information about RID and SID, they send subscription requests to rendezvous nodes. Once the subscription succeeds, a forwarding path is established between the subscriber and publisher. mGuard is fundamentally different from this architecture. Our pub-sub API is built on an application-layer data synchronization mechanism (Sync) and it implements access control, data authentication, and trust management so that the consumer and producer applications do not need to perform these security functions on a packet-by-packet basis. Nichols [27] also proposed a lightweight pub-sub API based on the Sync protocol syncps. However, this API does not support access control. In addition, syncps uses a time window for synchronizing publications so data outside the time window cannot be retrieved by the subscribers. Our API was inspired by Yu et. al.'s work [36] which proposed a pub-sub API for NDN-Lite with built-in security. However, the latter is not built on a Sync protocol and does not support contextual access control.

8 CONCLUSION AND FUTURE WORK

We have built mGuard, a prototype system for secure mHealth data sharing over NDN, and demonstrated its feasibility using Mini-NDN. While our design was motivated by sharing mHealth data in realtime with automated access control, we believe that the building blocks we developed for mHealth, e.g., access control policies and pub-sub API, can potentially be used by a variety of other applications. Next, we will address various issues identified in NDN Python Repo, NAC-ABE, and mGuard, and conduct performance evaluation over the NDN testbed. We will also perform a user study to assess the usability of our access control policy design. In the longer term, we plan to explore mechanisms to support automated security bootstrapping, automated key management with revocation support, scalable attribute-based access control, and real-time data transfer from sensors to mHealth repositories.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation award CNS-2019085. We thank the anonymous reviewers and our shepherd Karen Sollins for their insightful feedback.

REFERENCES

[1] Wearables Market to Be Worth $25 Billion by 2019 (2015, September 1). https://www.ccsinsight.com/, 2015.

[2] Afanasyev, A., Refaei, T., Wang, L., and Zhang, L. A brief introduction to Named Data Networking. In *IEEE MILCOM* (2018).

[3] Aledavood, T., Hoyos, A. M. T., Alakörkkö, T., Kaski, K., Saramäki, J., Isometsä, E., and Darst, R. K. Data collection for mental health studies through digital platforms: requirements and design of a prototype. *JMIR research protocols 6*, 6 (2017), e110.

[4] Anastasiadou, D., Folkvord, F., Brugnera, A., Cañas Vinader, L., SerranoTroncoso, E., Carretero Jardi, C., Linares Bertolin, R., Muñoz Rodríguez, R., Martínez Nuñez, B., Graell Berna, M., et al. An mhealth intervention for the treatment of patients with an eating disorder: a multicenter randomized controlled trial. *International Journal of Eating Disorders 53*, 7 (2020), 1120–1131.

[5] Bethencourt, J., Sahai, A., and Waters, B. Ciphertext-policy attribute-based encryption. In *2007 IEEE symposium on security and privacy (SP'07)* (2007), IEEE, pp. 321–334.

[6] Chen, C., Haddad, D., Selsky, J., Hoffman, J. E., Kravitz, R. L., Estrin, D. E., and Sim, I. Making sense of mobile health data: An open architecture to improve individual-and population-level health. *Journal of medical Internet research 14*, 4 (2012).

[7] De Michele, R., and Furini, M. Iot healthcare: Benefits, issues and challenges. In *Proceedings of the 5th EAI international conference on smart objects and technologies for social good* (2019), pp. 160–164.

[8] Fotiou, N., Nikander, P., Trossen, D., and Polyzos, G. C. Developing information networking further: From psirp to pursuit. In *International Conference on Broadband Communications, Networks and Systems* (2010), Springer, pp. 1–13.

[9] Goyal, V., Pandey, O., Sahai, A., and Waters, B. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security* (2006), pp. 89–98.

[10] Heather Jordan Cartwright. FHIR Server for Azure: An open source project for cloud-based health solutions. https://cloudblogs.microsoft.com/industry-blog/health/2018/11/12/fhir-server-for-azure-an-open-source-project-for-cloud-based-health-solutions.

[11] Kayes, A., Kalaria, R., Sarker, I. H., Islam, M., Watters, P. A., Ng, A., Hammoudeh, M., Badsha, S., Kumara, I., et al. A survey of context-aware access control mechanisms for cloud and fog networks: Taxonomy and open research issues. *Sensors 20*, 9 (2020), 2464.

[12] MD2K: Center of excellence for mobile sensor data-to-knowledge. MD2K website, http://md2k.org.

[13] Mini-NDN Authors. Mini-NDN: A Mininet-based NDN emulator, 2021. accessed: 2021-05-10.

[14] Moiseenko, I. Fetching content in named data networking with embedded manifests. *NDN, Tech. Rep. NDN-0025* (2014).

[15] Moll, P., Patil, V., Wang, L., and Zhang, L. SoK: The Evolution of Distributed Dataset Synchronization Solutions in NDN. In *Proceedings of the 9th ACM Conference on Information-Centric Networking* (2022).

[16] Moll, P., Shang, W., Yu, Y., Afanasyev, A., and Zhang, L. A survey of distributed dataset synchronization in named data networking. *Tech. Rep. NDN-0053, Revision 2, Named Data Networking* (2021).

[17] NAC-ABE Library GitHub Site. https://github.com/UCLA-IRL/NAC-ABE.

[18] Nakajima, M., Lemieux, A. M., Fiecas, M., Chatterjee, S., Sarker, H., Saleheen, N., Ertin, E., Kumar, S., and al'Absi, M. Using novel mobile sensors to assess stress and smoking lapse. *International Journal of Psychophysiology 158* (2020), 411–418.

[19] NDN Project Team. A Named Data Networking (NDN) Repo implementation using python-ndn. https://github.com/UCLA-IRL/ndn-python-repo. (Accessed on 06/10/2022).

[20] NDN Project Team. Mini-NDN GitHub. https://github.com/named-data/mini-ndn. (Accessed on Accessed on 06/10/2022).

[21] NDN Project Team. ndn-cxx: NDN C++ library with eXperimental eXtensions. https://github.com/named-data/ndn-cxx. (Accessed on Accessed on 06/10/2022).

[22] NDN Project Team. NDN Essential Tools. https://github.com/named-data/ndn-tools. (Accessed on Accessed on 06/10/2022).

[23] NDN Project Team. NDN Name Format. https://named-data.net/doc/NDN-packet-spec/current/name.html. (Accessed on 06/10/2022).

[24] NDN Project Team. NDN Testbed Topology. http://ndndemo.arl.wustl.edu/. (Accessed on 06/10/2022).

[25] NDN Project Team. NFD: Named Data Networking Forwarding Daemon. https://github.com/named-data/nfd. (Accessed on Accessed on 06/10/2022).

[26] NDN Project Team. PSync: Partial and Full Synchronization Library for NDN. https://github.com/named-data/psync. (Accessed on 06/10/2022).

[27] Nichols, K. Lessons learned building a secure network measurement framework using basic NDN. In *Proceedings of the 6th ACM Conference on Information-Centric Networking* (2019), pp. 112–122.

[28] OpenABE Library GitHub Site. https://github.com/zeutro/openabe.

[29] Park, L. G., Beatty, A., Stafford, Z., and Whooley, M. A. Mobile phone interventions for the secondary prevention of cardiovascular disease. *Progress in cardiovascular diseases 58*, 6 (2016), 639–650.

[30] Ranjan, Y., Rashid, Z., Stewart, C., Conde, P., Begale, M., Verbeeck, D., Boettcher, S., Dobson, R., Folarin, A., Consortium, R.-C., et al. Radar-base: Open source mobile health platform for collecting, monitoring, and analyzing data using sensors, wearables, and mobile devices. *JMIR mHealth and uHealth 7*, 8 (2019), e11734.

[31] Reddick, D., Feltus, F. A., and Shannigrahi, S. Case study of attribute based access control for genomics data using named data networking. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)* (2022), IEEE, pp. 715–716.

[32] Saleheen, N., Ali, A. A., Hossain, S. M., Sarker, H., Chatterjee, S., Marlin, B., Ertin, E., Al'Absi, M., and Kumar, S. puffmarker: a multi-sensor approach for pinpointing the timing of first lapse in smoking cessation. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (2015), pp. 999–1010.

[33] Sim, I. Mobile devices and health. *New England Journal of Medicine 381* (2019), 956–968.

[34] Tschudin, C., Wood, C. A., Mosko, M., and Oran, D. R. File-Like ICN Collections (FLIC). Internet-Draft draft-irtf-icnrg-flic-03, Internet Engineering Task Force, Nov. 2021. Work in Progress.

[35] Wang, Y., Kung, L., and Byrd, T. A. Big data analytics: Understanding its capabilities and potential benefits for healthcare organizations. *Technological forecasting and social change 126* (2018), 3–13.

[36] Yu, T., Zhang, Z., Ma, X., Moll, P., and Zhang, L. A pub/sub API for NDN-Lite with built-in security. *Named Data Networking, Tech. Rep. NDN-0071, Revision 1* (2021).

[37] Yu, Y., Afanasyev, A., Clark, D., Jacobson, V., Zhang, L., et al. Schematizing Trust in Named Data Networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking* (2015), ACM, pp. 177–186.

[38] Zhang, F., Cao, J., Khan, S. U., Li, K., and Hwang, K. A task-level adaptive mapreduce framework for real-time streaming data in healthcare applications. *Future generation computer systems 43* (2015), 149–160.

[39] Zhang, H., Wang, Z., et al. Sharing mHealth Data via Named Data Networking. In *ICN* (September 2016), pp. 142–147.

[40] Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., claffy, k., Crowley, P., Papadopoulos, C., Wang, L., and Zhang, B. Named Data Networking. *ACM SIGCOMM Computer Communication Review (CCR) 44*, 3 (Jul 2014), 66–73.

[41] Zhang, M., Lehman, V., and Wang, L. Scalable name-based data synchronization for named data networking. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications* (2017).

[42] Zhang, Z., Yu, Y., Afanasyev, A., and Zhang, L. Ndn certificate management protocol (ndncert). *NDN, Technical Report NDN-0054* (2017).

[43] Zhang, Z., Yu, Y., Ramani, S. K., Afanasyev, A., and Zhang, L. NAC: Automating access control via Named Data. In *Proceedings of IEEE MILCOM 2018* (2018).

[44] Zhang, Z., Yu, Y., Ramani, S. K., Afanasyev, A., and Zhang, L. Nac: Automating access control via named data. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)* (2018), IEEE, pp. 626–633.

[45] Zhang, Z., Yu, Y., Zhang, H., Newberry, E., Mastorakis, S., Li, Y., Afanasyev, A., and Zhang, L. An overview of security support in Named Data Networking. *IEEE Communications Magazine 56*, 11 (2018), 62–68.