Enabling Privacy Policies for mHealth Studies

Brian Wang
UCLA CS

Mani B. Srivastava
UCLA EE

***Abstract*—Pervasive sensing has enabled continuous monitoring of user physiological state through mobile and wearable devices, allowing for large scale user studies to be conducted, such as those found in mHealth. However, current mHealth studies are limited in their ability of allowing users to express their privacy preferences on the data they share across multiple entities involved in a research study. In this work, we present mPolicy, a privacy policy language for study participants to express the context-aware and data-handling policies needed for mHealth. In addition, we provide a privacy-adaptive policy creation mechanism for byproduct data (such as motion inferences). Lastly, we create a software library called privLib for implementing parsing, enforcement, and policy creation on byproduct data for mPolicy. We evaluate the latency overhead of these operations, and discuss future improvements for scaling to realistic mHealth scenarios.**

I. Introduction

The recent decades have seen a tremendous effort towards establishing new software frameworks for performing scientific studies [1]–[3]. We see works such as [4]–[6] proving the significant value found in studies enabled by pervasive sensing. One particular class of these studies is mHealth, the concept of using mobile and wireless technologies for scientific studies to enable medical objectives [7] - we term this category of scientific studies as *mHealth studies*. Through sensing enabled mobile devices such as smartphones and wearables, such mHealth studies have become easier to conduct and are less disruptive of user habit, with the benefit of having multiple modalities for streaming personal user data. But despite research showing that users are willing to participate in these studies to benefit science [8], there remains the possibility that participant sensor data is exposed to multiple 3rd parties who may perform unwanted inferences, thus incurring a privacy risk.

Providing a language that can describe the privacy preferences of study participants is a first step towards improving the current model for conducting mHealth studies. In this work, we present mPolicy, a privacy policy language modeled off the interaction involved in scientific studies, specifically in the context of mHealth. We adopt a model-centric view of privacy policy, where policies should express constraints on the modeled interactions between actors in an mHealth study, allowing for more intuitive and readable policies than those typically expressed in XML based languages [9], [10]. In addition, we design methods of automatic policy creation on data byproducts - particularly policy fusion mechanisms for combining data with different policies. Finally, we build a library called privLib designed around the functionalities of mPolicy. More specifically, privLib allows for enforcement of policies and attaching new policies to the various types of data byproducts. mPolicy and privLib are our first steps towards providing a unified framework for enabling privacy policies in mHealth and other scientific studies.

Our contributions are as follows:

- A model-centric language called mPolicy for expressing the context-aware and data-handling needs of mHealth studies.
- A privacy-adaptive mechanism for attaching new policies to data byproducts of mHealth studies.
- An enforcement mechanism utilizing a context data stream to enforce querying and downstream dissemination.
- privLib, a library written in Java for evaluating the overhead of parsing and enforcement of mPolicy policies, as well as the data byproduct policy creation mechanisms.

The rest of the paper is as follows: In Section II we discuss the motivation behind providing a privacy policy language for mHealth studies. In Section III we explore the interactions between the actors in an mHealth study to inform our language design. In Section IV we present the taxonomy of our language. In Section V we provide a mathematical formulation for automatically producing privacy policies on data byproducts. In Section VI we design mechanisms for receiving, querying, and disseminating data according to privacy policies, which we use to implement privLib. We evaluate the overhead of privLib in Section VII. In Section VIII we discuss the related work to this topic and differentiate our work from others. In Section IX we discuss the limitations and conclude in Section X.

II. Motivation

Pervasive sensing has made it possible to conduct large scale user studies with continuous monitoring of digital biomarkers, which are measurable indicators of a

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

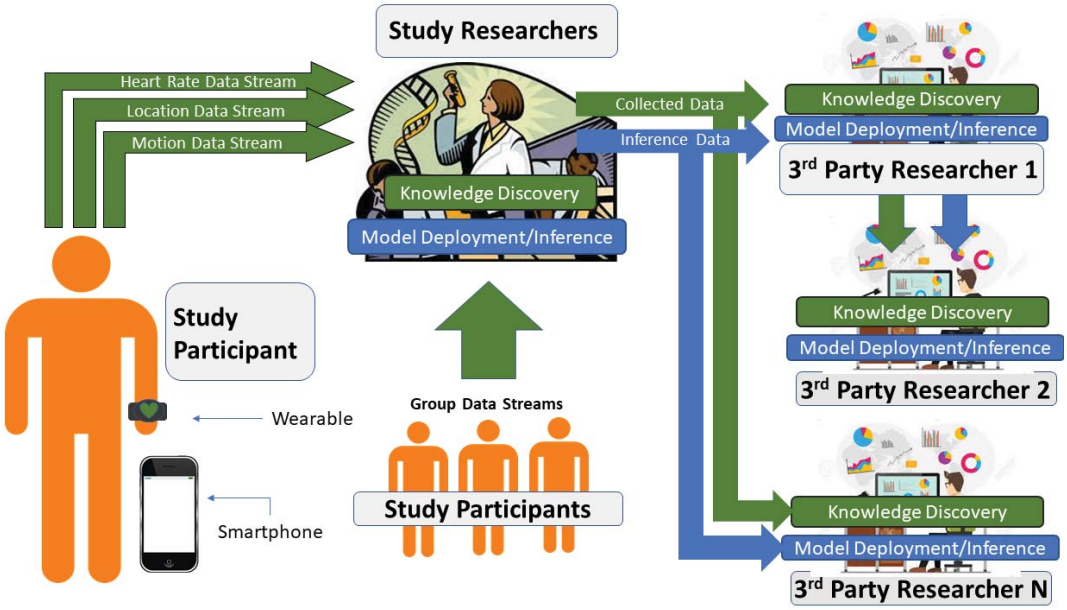

Fig. 1: Example showing the flow of sensor data from study participants to downstream entities. Components relating to knowledge discovery and model training are shown in green, while model deployment and inference is shown in blue.

user's physiological state [11], [12]. These biomarkers are potentially unlimited sequences of sensory data; for that reason, we characterize biomarkers as sensory *data streams*. Current mHealth study platforms such as the Center of Excellence for Mobile Sensor Data-to-Knowledge (MD2K) [13] allows a variety of interested organizations and other entities besides study researchers to take part in the collection, processing, and analysis of these sensory data streams from a diverse group of participants. These entities may be government organizations, university research groups, commercial companies, or individual researchers, all with their own data byproducts. Figure 1 shows an example of the flow of information from study participants to study researchers, and finally to multiple entities downstream, whom may also share information amongst each other. In addition, these studies may consist of two general phases [14], where biomarkers are developed and validated. We use the characteristics of these phases to inform the modeling of interactions in Section III. Broadly speaking, the first phase in an mHealth study is for knowledge discovery, where data is collected to train a multitude of scientific models for inferring additional information. For example, study researchers may collect Heart Rate Variability (HRV) data and Inertial Measurement Unit (IMU) data to identify average heart rates of individuals at rest. The end goal of this may be to simply draw inferences from this collected data, in which case the study ends during this first phase. However, other goals may include analyzing this collected data to produce scientific models, which are then deployed for validation on new collected data. Thus, there is a potential second phase to the study.

As the first phase scientific models are using collected data to determine new biomarkers, the second phase deploys those models to validate the biomarkers. More specifically, these models are deployed to be used on the sensor data streams, resulting in new data byproducts. These data byproducts are new data streams[1] (such as machine learning inferences). Using the prior example, study researchers collected HRV and IMU data from multiple participants. Using this data, they may aggregate these measures from many individuals to produce a machine learning model determining whether an individual's resting heart rate is within a normal threshold. In this second phase, the researchers may deploy this model to detect abnormal resting heart rate, resulting in a new data stream of detection results to validate against their model. Studies may then repeat these two phases to continuously validate and improve the model.

This current approach for performing mHealth studies is greatly beneficial to the various researchers and entities involved. There are also studies showing that users are interested in providing their information to benefit science [8], and the success of data sharing

[1]It is important to note here that the models themselves are byproducts as well, although we do not consider them in this work.

platforms such as OpenHumans [15] illustrates this point further. However, there is still a lack of control for study participants to express their privacy needs, which remains a significant concern. This is further exacerbated with the current consent model for participating in such mobile app-based studies. Consent is still a binary decision, where users may join a study by allowing all the requested data streams to flow freely downstream, or be forced to leave the study. Study participants are unable to express their own privacy policies on their data, which in turn harms the study by turning away participants with diverse biomarkers who do not agree to the policies fixed by the study. What is needed is a language that can express the diverse privacy needs of study participants and ensure that each research entity can adhere to it before using it. More specifically, the language should enable expression of *context-aware* policies with *data handling*. Context-aware privacy policies enable dynamic altering of policies according to context - examples of context include time, geographical location, device in motion, and user defined events such as button presses. Data handling capabilities allow for expressing how the data is processed, how it can be stored, and how the data may be further disseminated.

As mentioned previously, there is a secondary phase to mHealth studies where models are deployed, producing new data byproducts using the sensor data streams. These data byproducts are themselves streams of data (i.e. an activity classification based on IMU data), which we term *byproduct data streams*. Since these byproduct data streams do not have their own privacy policies, there is a challenge to automatically determine what the privacy policies of these streams should be. In other words, we seek to generate the *byproduct policies*.

Motivated by the limitations of the current consent model in mHealth, we create a language that is capable of expressing privacy policies that are both context aware and capable of data handling. This language allows for a convergence of participant privacy policies with a study's outlined policies, allowing studies to benefit from a more diverse set of participants who would have otherwise quit the study. In addition, we design the mechanisms for producing privacy policies of data byproducts, motivated by the second phase of current mHealth studies.

III. Modeling Interactions

Before creating a privacy policy language for mHealth studies, it is first necessary to be able to model the interactions among actors involved. More specifically, these interactions describe how the entity interacts with a participant's data. We characterize these interactions using *entities, data windows, data stream types, operations* and *intents*. *Entities* are the identifiable actors at different points of the flow of data - examples include Bob's mobile device, University Health researchers, and government groups. *Data Windows* are the segments or time windows of a data stream that these entities might perform processing on. *Data stream types* describe the type of sensor or inference stream producing the data (i.e. GPS, IMU, Motion Detections). *Operations* are specific functions or classes of functions performed on a data stream, such as group statistical aggregations, machine learning inference, and so on. Lastly, *Intents* refer to the purposes that this stream could be used for, such as personalized fitness tracking or inferring heart conditions.

Using these objects, we can formulate how these interactions work. More specifically, we want to characterize how entities may process data, how they produce data byproducts, and how they disseminate data further downstream:

- Interaction A: "Entity E_1 processes data window DW_1 from data stream DS_1 for operation O_1 and intent I_1"
- Interaction B: "Entity E_1 produces new output data stream DS_2 using operation O_2 for intent I_2 using processing interactions A_1, A_2..."
- Interaction C: "Entity E_1 disseminates data window DW_1 from data stream DS_1 to Entity E_2 for operation O_3 and intent I_3"

Using how each study entity interacts with the participant's data, we introduce how we express privacy policies for participants. In our approach, each study entity must be able to express the interactions they desire with the study participants. The participants may then express additional constraints on these interactions, resulting in a set of privacy policies. As a result, all interactions between study entities and participant data must now satisfy the set of privacy policies - if not, then the interaction is not valid, and therefore not permitted according to the participant's privacy policy. We discuss how these interactions can be expressed in Section IV.

IV. mPolicy Language

We present mPolicy, a language designed to express privacy policies over the interactions involved in mHealth studies. As defined in Section II, mPolicy must express constraints over the existing interactions, while being both context aware and capable of data handling. More specifically, we introduce the following general classes of constraints:

- **data-window**, used to determine what aspects of the data window is relevant to a policy. These depend on the time conditions (i.e. 9am to 9pm), location conditions (i.e. can't access data from this location), and other contextual values (i.e. if a button is pressed, motion is detected, and so on). We can use these rules to create context-aware policies.

- **operations**, used to determine what set of operations are valid for the data segment.
- **intents**, used to determine what purposes this data segment can be used for.
- **data-stream**, implicitly involved in the interaction, and represent the actual stream values themselves. For example, the actual stream values from a GPS sensor are the latitude and longtudinal values. This constraint allows us to apply operations on the stream itself, such as perturbing data values to add noise or creating more coarse representations such as rounding up of values, and aggregating data (i.e. average all values in this data segment).

Our general language structure for expressing a privacy policy shown Figure 2. The language structure has a number of variable fields, such as PROPOSITION_FUNCTION or OPERATIONS, which can be used to express privacy policies. Note that many of these fields take on a potentially infinite set of values to provide freedom in expressing constraints. In our implementation of mPolicy, we restrict ourselves to a smaller set of values, which we discuss in Section IX. Here we describe each of these classes of constraints in detail and then explore how policies are created using this language. More specific example policies are included in our GitHub [2].

As discussed earlier, we seek to produce a language can express context aware policies with data handling capabilities. Context-aware privacy policies enable dynamic altering of policies according to context, while data handling expresses how the data is processed, how it can be stored, and how the data may be further disseminated. We can create these context aware policies by using the *data-window* constraint. This constraint allows us to specify what policies should be evaluated given a particular context. This case is expressed in Fig 2 in lines 6-9. In addition, the *data-window* field also allows us to constrain access with respect to *time*: one of the aspects of data-handling is *data retention*, which determines how long the data can be stored for. Looking at the syntax guide in Figure 2, the TimeGreaterThan proposition function can be used to express when this policy expires, which will prevent further access to the data. As discussed in Section III, any interaction that does not satisfy the privacy policies is not permitted.

To further improve data-handling abilities, we can use the *data-stream* to express how the data must be processed before access, such as reducing the granularity of the data values. Line 12 shows how one can apply optional methods to the data stream, with possibilities such as reducing the resolution of GPS values, perturbing data, etc. The *operations* and *intents* constraints (line 15) allow us to further tune the conditions that must be satisfied for any interaction a study entity would like to have with the participant's data.

Although we have described how this language can achieve our conceptual goals of expressing privacy policies that are context aware and capable of data handling, we have yet to prove this language working in a real system. To that end, we created a library privLib for supporting mPolicy in terms of attaching policies to the context stream, as well as enforcing access to participant data streams. We discuss the functions of privLib in Section VII.

It is important to note how participants are expected to use this language. As we discussed in Section III, study entities must be able to express their desired interactions with the participant's data. In our approach, these entities are expected to create an initial policy using mPolicy, where most of these fields are empty or always true (i.e. setting a time range of 24 hours every day). The entities will then submit these policies to the participant, who may then express the additional constraints they desire to create their own privacy policies, or have another entity set these policies on behalf of the user (i.e. an automated privacy policy agent). In this work we focus on the designing and developing language itself, leaving this envisioned communication for future work.

This language uses the interactions discussed in Section III to create constraints on them. However, the assumption here is that study participants have sufficient knowledge of the interaction elements to express their privacy policies. For example, users are likely aware that a research organization would like to use their HRV data at any time window to track average heart rate - such information is already present in the current mHealth study consent model [16]. Thus, for processing and dissemination of data streams, we can apply constraints on each of the interaction elements because we assume most elements are likely known to the user. However this is not the case for the creation of data byproducts: study participants are probably unaware that there even exists a data product from their data streams. For example, study participants are likely not aware that their IMU and HRV data is producing an additional activity recognition stream of data, unless it is explicitly mentioned by the study. Since participants are not able to express those privacy preferences on these unknown interactions, we provide some insights as to how policies can be automatically attached to these data byproducts.

V. Privacy Adaptive Output Policy Streams

Although our language can express a variety of privacy preferences on sensor data streams, finding the appropriate policies to attach to byproduct data streams and models is still a significant challenge. One method of

[2]https://github.com/nesl/mPolicy-and-privLib

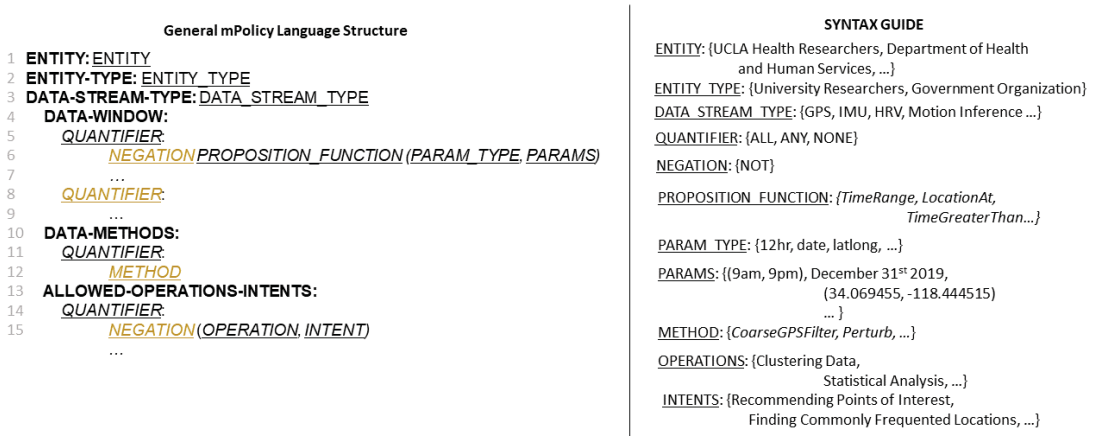

Fig. 2: The general structure of mPolicy for expressing a single privacy policy. Fields that take on multiple values are underlined, with optional fields highlighted in yellow. Each field is discussed in Section IV.

creating byproduct data policies is by applying general rule combining algorithms used in XACML [9], [17], such as decision by majority voting. Other possibilities include calculating a *residual policy* [18] within the restrictions of the input policy, or using logical entailment of policies to validate combinations of privacy policies [19]. However, these algorithms consider output policies strictly as a function of the input policies, resulting in general approaches to computing output policies without explicit regard to the entities, operations, and data at play. What is missing in these approaches is a notion of the future - more specifically, we need a method of creating data byproduct policies parameterized by the entities, operations, and data involved in an interaction *later on*, not based solely on the prior policies involved. In our approach, we use a notion of trust, which we will describe in Section V-B. More generally, if there does not exist sufficient *trust* in the entities to perform the operation on such data, such that there is a risk of accidental or purposeful misuse of the data, then the data byproduct policies need to reflect this lack of trust.

A. Privacy Policies for Data Byproduct Streams

We first consider the input policies of the data stream to create an initial privacy policy for the data byproduct stream. However, we note that because our language is semantically separated into classes of constraints, we consider each constraint class separately to produce a data byproduct stream.

First, to properly fuse the *data window* constraints from multiple policies, we first have to find the set of minimally restrictive data windows that satisfy all data window constraints for those policies. For example, we may be faced with constraints from two different policies such as ”Time Range from 9am to 9pm” and ”Time Range from 8:30am to 8:30pm”, requiring us to create a new data window constraint for ”Time Range from 9am to 8:30pm”. To solve this problem, we define a restrictive ordering on propositional functions in privLib for comparing contexts (i.e. ”Time Range of 9am to 10am is more restrictive than Time Range of 9am to 11am”. This requires a custom function to be created *for each* comparison between unique constraints - privLib currently implements a only few of these comparisons, which we discuss further in Section IX. For example, a custom function would be need to compare time ranges, while another custom function would be required to compare GPS locations.

To compare *intents* and *operations* of multiple different streams, privLib provides these comparisons through matching - in our implementation we take a more liberal approach by allowing for output policies that include all unique intents and operations from the combining policies. Lastly, we do not consider the input policies when creating the *data-stream* constraints - instead we use the notion of trust to inform these byproduct data stream constraints. We discuss this further in Section V-B.

B. Enabling Privacy Policies for Byproduct Streams Based on Trust

We refer to trust as a risk of misuse: if participants do not trust an entity, it refers to how unlikely the entity will adhere to their privacy policies; not trusting their own sensor data refers to how easily it can be misused. Depending on the entity, entity type (whether it be a government organization or independent researcher), operation, or data type used, the potential and consequence of such misuse changes. To enable privacy policies informed by trust, we propose a method of adding operations on the *data-stream* constraints through the mPolicy language using a measure of trust, which we

call the *Entity Trust Measure* (ETM), with each object having a corresponding weight. More formally,

$$ETM = w_{entity} * t_{entity} + w_{entity-class} * t_{entity-class} + w_{operation} * t_{operation} + w_{stream-type} * t_{stream-type} \quad (1)$$

where w_X is the weight associated with object X, and t_X is a binary trust value for object X. In our experiments we assume uniform weights of 1.

Using this measurement, we propose a simple thresholding function that determines what policies to be added, shown in Algorithm 1. This algorithm simply seeks to measure the ETM, and based on the value of ETM append to the data-methods constraint in an mPolicy policy. The three possibilities shown are to either deny access to the stream entirely, apply a uniform perturbation to the data, and finally reducing the granularity of the values by rounding up.

Algorithm 1 Thresholding Function

```
1: w_e, t_e ← weight(w) of entity, trust value(t) of entity
2: w_ec, t_ec ← entity class weight and trust value
3: w_op, t_op ← operation weight and trust value
4: w_dst, t_dst ← data stream type weight and trust value
5: P ← Byproduct Policy
6: val ← ETM(w_e, t_e, w_ec, t_ec w_op, t_op w_dst, t_dst)
7: if val < α then
8:     P.append(DENY-ALL())
9: else if val < β then
10:    P.append(PERTURB())
11: else if val < γ then
12:    P.append(REDUCE-GRANULARITY())
```

Although all of these values are arbitrary, we use this simple formalization to illustrate the idea that policies on byproduct data should adhere to trust regarding whom the data is meant for, what the data is, and how the data is used. Having defined both the language, as well as mechanisms for creating byproduct policies for sensor data streams, we now turn to enforcing the language.

VI. Enforcing mPolicy

To show how mPolicy can be used and enforced, we have created a Java library called privLib consisting of ∼1500 lines of code for the major functionalities described below. privLib is designed to be used by entities involved in the study who want to parse the mPolicy policies that are created by the participants and ensure that their actions adhere to the policies. More specifically, it is designed for determining relevant policies, determining whether or not a query on user data satisfies the relevant policies, and finally combining policies together to produce new output policies. This section shows the concepts and algorithms we use to develop privLib.

There are several steps in our implementation for enforcing mPolicy. The first point is upon an entity receiving the data stream, to determine what policies are relevant to this stream for enforcement. This requires the addition of a *context stream* allowing entities to attach the correct policies to portions of the stream. The second point is when the entity seeks to analyze or process a portion of the data, requiring them to access a portion of the stream. This access must be controlled according to the query they offer. Lastly, the entity may provide additional enforcement on their output data and byproduct stream(s), determining which ones they may disseminate further and which ones they may not.

A. Building the Context Stream

To produce context-aware policies, we use the *data-window* constraints coupled with an external *context stream*. The context stream entails the information to correlate with the *data-window*. More specifically, this stream allows us to enforce context aware policies on downstream entities. Such a stream allows for dynamically changing policy rules over time for a variety of downstream entities, which is not captured in current languages.

The context stream is composed of several channels of contexts - for example, time, location, button press detection, and so on. The number of channels depends on the privacy policy rules determined by the *data-window* field in the privacy rules set by the user. For example, if a *data-window* has a time range constraint, then time must be included in the context stream. By using this context stream, we can inform what policies are relevant to this time instance of a stream.

B. Access and Downstream Enforcement

When an entity accesses a segment of a sensor data stream to perform an operation or analysis, we must evaluate whether or not the access query attributes satisfies the privacy policies. This involves using the context stream to determine the relevant policies and comparing the query against those policies. In order for the access decision to succeed, the current entity must satisfy the *operations* and *intents* fields of each relevant policy of this data stream for access to be granted. For example, if the user queries for GPS data between 9am and 10am for the purpose of identifying travelling behaviors, then we must identify policies for which the query satisfies. We determine the relevance of the policies by attaching them to windows of time, informed by the context stream, shown in Algorithm 2. We then evaluate these relevant policies against a query on the data, shown in Algorithm 3. A query consists of a time instance of the sensor data

Algorithm 2 Policy Relevance Function

$P_{all} \leftarrow$ Set of mPolicy policies for this entity and sensor stream
$C_{stream} \leftarrow$ context stream
for C_t in C_{stream} **do** ▷ For context values at every time instance t of the context stream
    $P_{relevant} \leftarrow$ Relevant policies at this instance of t
    for P in P_{all} **do**
        $P_{data-windows} \leftarrow$ Set of data window propositional functions for policy P
        $Evaluation \leftarrow$ Dictionary of Logical quantifiers, initially all set to True
        for $prop$ in $P_{data-windows}$ **do**
            $Quantifier \leftarrow$ Logical quantifier for this data window constraint
            $Negation \leftarrow$ true or false value, if the proposition function should be negated
            if Quantifier == "ALL" **then**
                $Evaluation["ALL"] = Evaluation["ALL"] \wedge \neg(prop(C_t)Negation)$
            else if Quantifier == "ANY" **then**
                $Evaluation["ANY"] = Evaluation["ANY"] \vee \neg(prop(C_t)Negation)$
            else if Quantifier == "NONE" **then**
                $Evaluation["ALL"] = Evaluation["ALL"] \wedge (prop(C_t)Negation)$
        if $Evaluation["ALL"] \wedge Evaluation["ANY"] \wedge Evaluation["NONE"]$ **then**
            $P_{relevant}$.append(P)

stream to be accessed, the operation to be performed on the retrieved data, and the intent behind the access. If the policies relevant to this query match the operation and intent, then the access decision is approved [3].

Once the entity is finished processing the stream, it may have an output stream with a set of privacy policies. Before the output stream can be disseminated further, the current entity must complete the functions explicitly described in the *data-stream* section of the privacy policy. Semantically, this requires the current entity to perturb or manipulate the data values before transmitting downstream.

VII. Evaluation

To evaluate realistic settings using mPolicy, we explore the overhead of enforcing these policies using the methods described in Section VI. We are mainly interested in three overheads - the overhead of determining which policies are relevant, the overhead of determining whether a query is appropriate by comparing it against the relevant policies, and lastly, the overhead of combining two policies together. These correspond to the major functionalities described in Section VI that are needed to build systems supporting mPolicy.

We performed these evaluations on a PC with an AMD Ryzen 7 3800X 8-Core processor. Each experiment is repeated 20 times, and the results shown are the median latencies across these 20 repeated experiments. For our experiments, we would like to highlight an assumption we have made. In our experiments, we have used the same policy for all experiments (i.e. evaluating 25 of the same policies). Without this assumption, every query would be performed on an inconsistent number of policies, since different policies means not all of them will be relevant for evaluating the query. Although this assumption is unrealistic, it offers us a way of measuring the worst case time in each of our experiments.

[3]For this case, we implicitly assume that the querying entity and the entity referred to in the policy both match.

Policies Evaluated	1000	10,000	100,000
Latency (ms) per 250 samples, RP	150	1734	41,029
Latency (ms) per 250 samples, QD	2	222	21,723

TABLE I: Latency for determining relevant policies (RP) and query decisions (QD)

Number of Participants	200	2,000	20,000
Latency (ms) per 250 samples	1,314	12,676	133,026

TABLE II: Latency for determining combining policies

For the purposes of this evaluation, we will consider the following scenario that a study entity might expect: a study consists of 200 to 20,000 study participants each expressing 5 policies corresponding to a particular type of data stream (i.e. GPS, IMU). If we assume the maximum sensor sampling rate on a user's smartphone or wearable to be a ∼250Hz, the context stream will also produce values at around the same sampling rate.

If the entity aggregates all 200 participant policies together, it means evaluating 1000 policies at 250Hz to determine which policies are relevant at every time stamp. If this is a larger study with 2000 participants, then the entitiy evaluates 10,000 policies at 250Hz. Similarly, for 20,000 participants they evaluate 100,000 policies at 250Hz. For this first set of experiments, we

Algorithm 3 Policy Query

$t \leftarrow$ Time instance of query access
$P_{relevant} \leftarrow$ Relevant policies at t
$Q_i \leftarrow$ Intent of query access
$Q_o \leftarrow$ Operation for access data
decision = APPROVE
if $P_{relevant}$ == none **then** ▷ If there are no relevant policies, DENY access
    decision = DENY
else
    for P in $P_{relevant}$ **do**
        $P_i \leftarrow$ List of allowed intents for policy P
        $P_o \leftarrow$ List of allowed operations for policy P
        if NOT Q_i in P_i **then** ▷ If querying intent does not match allowed intents, DENY access
            decision = DENY
        if NOT Q_o in P_o **then** ▷ If querying operation does not match allowed operations, DENY access
            decision = DENY
return decision

measure the time it takes to attach relevant policies for 4 seconds of a context stream (1000 samples). As mentioned earlier, the actual policies themselves are exact copies. These results are shown in Table I.

Our results for time taken to evaluate a query against the same number of policies is also shown in Table I. We have used a predefined query so that all policies allow the access, thus requiring the algorithm to evaluate all policies in order to measure the worst case query time.

Lastly, we evaluate the overhead of combining two policies. For the sake of space, we have omitted the policy example, but feel free to consult our Github[4] for such examples. Following the scenario from before, if we assume that the entity is generating byproduct data by combining two data streams at a rate of 250Hz, then the entity is combining 2 policies per participant for a 250Hz context stream. We show these corresponding latencies in Table II.

It appears that the overhead grows rapidly for all operations. Especially for studies involving large numbers of users, these operations will never be able to keep up with the actual sensor sampling rates on participant mobile devices. This may not be a problem if the entity is only collecting and analyzing data offline, this is not acceptable in settings where entities require continuous monitoring. Particularly in the second phase of studies, as described in Section II, deployed models (i.e. for real time medical intervention) will suffer from long delays when attempting to query user data. Although these latencies may be ameliorated by amortizing over more capable servers, the scalability of this library remains questionable - this motivates us to seek optimizations to our library in the future. Although we have not mentioned it in Section VI, it is not necessary to evaluate the context stream at every point. In our implementation, we determine if policies are relevant when the context stream changes. For example, if we have policies that only depend on a GPS location, we only need to evaluate the context stream when the location changes.

[4]https://github.com/nesl/mPolicy-and-privLib

VIII. Related Work

There are a number of works on enforcing fine-grained control of user policies specific to systems such as Android. Examples include IpShield [20], FlaskDroid [21], and Mr. Hide [22]. [20] provides enforcement of context-aware policies with respect to time and location over Android OS. [21] provides a policy language based on Security Enhanced Android [23] to produce both context-aware policies as well as attribute enforcement of various Android objects (i.e. Activity, Intent). [22] develops a policy language that expresses parameterized permissions (i.e. restricting data access to a particular column of a database), a more fine-grained variant of the Android permission model.

Access control policy languages such as those of eXtensible Access Control Markup Language (XACML) [9] and Platform for Privacy Preferences (P3P) [10] are expressed in the XML markup language, designed to compare user defined policies against access requests (as in the case of XACML) or website policies (in the case of P3P). Primelife Policy Language (PPL) [24] builds off of XACML to provide better *data handling* capabilities via an obligation language, where accesses to user data may trigger notifications. It also adds fields for data access purpose, as well as policies for downstream handling. Accountability PPL (A-PPL) [25] builds off of PPL to enable auditing of an entity via logging of data accesses. Purpose To Use (P2U) [26] is another

policy language that improves upon the limitations of P3P to prevent improper secondhand sharing of data via explicitly defined purposes, length of data retention, and other factors.

Our work differs from these two types of languages. Although system specific user policy languages can provide context aware policies, they do not target the notion of downstream data. The policies are designed to remain on the device, and can not be enforced downstream. On the other hand, general privacy policy languages, such as the access control policy languages, are not designed to address the need for context-aware policies that are found in the pervasive sensing setting. Our work seeks to provide a language that addresses both of these missing gaps, which are needed in mHealth studies.

The concept of creating a new set of privacy policies for output streams of data isn't new either. [18] seeks to attach a *residual policy* of a program's output. This residual policy is computed by combining the attributes from the input policies and removing the attributes that are less restrictive. [9] provides a number of *rule combining algorithms* that determining the output policy given a combination of input policies. In a similar vein, works such as [27], [28] seeks to provide automated decisions of privacy configurations in exchange for a reward. [27] provides an automated negotiation agent to determine what data permissions to grant based on a virtual currency reward. [28] introduces the *economics of privacy*, where privacy of user attributes (such as demographics) has an associated utility (web search relevance), and seeks to optimize the attributes shared to meet an associated utility threshold. Although our byproduct policy is inspired by the similar abilities demonstrated in these prior works, we look to other factors for guiding the creation of output policies that these works have not addressed.

IX. Limitations and Future Work

One limitation for our byproduct policy creation comes from the fact that it is defined by a single static threshold. In reality, a single threshold is not sufficient to encapsulate the possible combinations of optimal privacy-adaptive byproduct policies. We believe that this is a necessary improvement for future work.

A number of limitations also arise from our prototype library privLib. Many of the propositional functions expressed in real context aware policies have yet to be implemented (such as "While participant is connected to the WiFi, use policy 1"). In our implementation, we have only considered GPS location, time ranges, and button presses (a user defined context). There are such many propositional functions to create, and we hope to add these to privLib in the future. There is also limited logical evaluation functionality for the creation of byproduct policies in privLib. For example, byproduct policies assume that there is no negation for data-windows (i.e. we can not combine "Not between 9am and 12pm" with "between 8am and 2pm"). Addressing these flaws are a necessary step to further improve privLib in the future.

A fair part of this work involves adding additional privacy rules to existing policies informed by the *trust* in different entities and interaction objects of an mHealth study. These new privacy policies include perturbations and other privacy mechanisms, albeit in this work they are rather coarse grained. The concept of differential privacy [29] can play a role in our approach, as the area offers more fine-grained and measurable privacy guarantees via privacy notions such as k-anonymity, l-diversity, or t-closeness [30]–[33]. We leave the integration of these mechanisms into future work.

One additional challenge that we have not addressed is the evolution of data products over time. As deployed models change (i.e. become more accurate), the privacy policies should also change to address new risks. For example, improved precision of the data byproducts may result in higher risk of identifying participants. We hope to address the challenge of evolving data byproducts in future work.

Lastly, it is important to note the barrier to usability for policy languages such as mPolicy. Participants might find the syntax abtruse or are unsure of what they *want to express* in the first place. We believe this challenge can be addressed by providing an automated software agent acting on behalf of the participants to express privacy policies. Though such work remains outside the scope of this paper, it is an excellent next step in establishing a real system that enables privacy policies in mHealth.

X. Conclusion

In this work we present a privacy policy language for meeting the privacy demands of users in the context of mHealth studies. Our language is both context-aware and capable of data handling to handle a variety of expressions in the context of mHealth. We also address the need for creating policies on byproduct data produced by different entities in the mHealth study, and enable *privacy-adaptive* policy creation on this byproduct data. We have discussed some preliminary results from our implementation for enforcing and combining mPolicy policies. These results encourage us to further optimize and improve the library, moving us one step closer to establishing a practical and extensive framework for creating privacy policies in mHealth studies.

Acknowledgements

This research was supported in part by the National Science Foundation (NSF) under award OAC-1640813 and OAC-1636916, and by the NIH Center of Excellence

for Mobile Sensor Data-to-Knowledge (MD2K) under award 1-U54EB020404-01. Any findings in this material are those of the author(s) and do not reflect the views of any of the above funding agencies.

REFERENCES

[1] "ResearchKit and CareKit." [Online]. Available: http://www.apple.com/researchkit/

[2] "ResearchStack." [Online]. Available: http://researchstack.org/

[3] "AWARE – Open-source Context Instrumentation Framework For Everyone." [Online]. Available: https://awareframework.com/

[4] E. D. Perakslis, "Using digital health to enable ethical health research in conflict and other humanitarian settings," *Conflict and Health*, vol. 12, no. 1, p. 23, May 2018. [Online]. Available: https://doi.org/10.1186/s13031-018-0163-z

[5] E. R. Dorsey, Y.-F. "Yvonne" Chan, M. V. McConnell, S. Y. Shaw, A. D. Trister, and S. H. Friend, "The Use of Smartphones for Health Research," *Academic Medicine*, vol. 92, no. 2, p. 157, Feb. 2017.

[6] M. Gjoreski, H. Gjoreski, M. Lutrek, and M. Gams, "Automatic Detection of Perceived Stress in Campus Students Using Smartphones," in *2015 International Conference on Intelligent Environments*, Jul. 2015, pp. 132–135.

[7] S. Tucker, "Welcome to the world of mHealth!" *mHealth*, vol. 1, Mar. 2015. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5344173/

[8] "Personal Data for the Public Good," Mar. 2014. [Online]. Available: https://www.rwjf.org/en/library/research/2014/03/personal-data-for-the-public-good.html

[9] "eXtensible Access Control Markup Language (XACML) Version 3.0." [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[10] "P3p: The Platform for Privacy Preferences." [Online]. Available: https://www.w3.org/P3P/

[11] K. Strimbu and J. A. Tavel, "What are Biomarkers?" *Current opinion in HIV and AIDS*, vol. 5, no. 6, pp. 463–466, Nov. 2010. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3078627/

[12] P. Dagum, "Digital biomarkers of cognitive function," *npj Digital Medicine*, vol. 1, no. 1, pp. 1–3, Mar. 2018. [Online]. Available: https://www.nature.com/articles/s41746-018-0018-4

[13] S. Kumar, G. Abowd, W. T. Abraham, M. al'Absi, D. H. Chau, E. Ertin, D. Estrin, D. Ganesan, T. Hnat, S. M. Hossain, Z. Ives, J. Kerr, B. M. Marlin, S. Murphy, J. M. Rehg, I. Nahum-Shani, V. Shetty, I. Sim, B. Spring, M. Srivastava, and D. Wetter, "Center of Excellence for Mobile Sensor Data-to-Knowledge (MD2k)," *IEEE Pervasive Computing*, vol. 16, no. 2, pp. 18–22, Apr. 2017.

[14] ——, "Center of Excellence for Mobile Sensor Data-to-Knowledge (MD2k)," *IEEE Pervasive Computing*, vol. 16, no. 2, pp. 18–22, Apr. 2017.

[15] "Home - Open Humans." [Online]. Available: https://www.openhumans.org/

[16] S. Moore, A.-M. Tassé, A. Thorogood, I. Winship, M. Zawati, and M. Doerr, "Consent Processes for Mobile App Mediated Research: Systematic Review," *JMIR mHealth and uHealth*, vol. 5, no. 8, Aug. 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5597795/

[17] N. Li, Q. Wang, P. Rao, D. Lin, E. Bertino, and J. Lobo, "CERIAS Tech Report 2008-9 A Formal Language for Specifying Policy Combining Algorithms in Access Control," 2008.

[18] L. Wang, J. P. Near, N. Somani, P. Gao, A. Low, D. Dao, and D. Song, "Data Capsule: A New Paradigm for Automatic Compliance with Data Privacy Regulations," *arXiv:1909.00077 [cs]*, Aug. 2019, arXiv: 1909.00077. [Online]. Available: http://arxiv.org/abs/1909.00077

[19] A. Barth, A. Datta, J. Mitchell, and H. Nissenbaum, "Privacy and contextual integrity: framework and applications," in *2006 IEEE Symposium on Security and Privacy (S P'06)*, May 2006, pp. 15 pp.–198, iSSN: 1081-6011, 2375-1207.

[20] S. Chakraborty, C. Shen, K. R. Raghavan, Y. Shoukry, M. Millar, and M. Srivastava, "ipShield: A Framework For Enforcing Context-Aware Privacy," 2014, pp. 143–156. [Online]. Available: https://www.usenix.org/node/179736

[21] S. Bugiel, S. Heuser, and A.-R. Sadeghi, "Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies," 2013, pp. 131–146. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/bugiel

[22] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein, "Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications," in *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, ser. SPSM '12. New York, NY, USA: ACM, 2012, pp. 3–14, event-place: Raleigh, North Carolina, USA. [Online]. Available: http://doi.acm.org/10.1145/2381934.2381938

[23] "SEforAndroid - SELinux Wiki." [Online]. Available: https://selinuxproject.org/page/SEforAndroid

[24] C. Ardagna, L. Bussard, S. De, C. Vimercati, G. Neven, S. Paraboschi, E. Pedrini, F.-S. Preiss, D. Raggett, P. Samarati, S. Trabelsi, and M. Verdicchio, "Primelife policy language," Jan. 2009.

[25] M. Azraoui, K. Elkhiyaoui, M. Önen, K. Bernsmed, A. S. De Oliveira, and J. Sendor, "A-PPL: An Accountability Policy Language," in *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, ser. Lecture Notes in Computer Science, J. Garcia-Alfaro, J. Herrera-Joancomartí, E. Lupu, J. Posegga, A. Aldini, F. Martinelli, and N. Suri, Eds. Cham: Springer International Publishing, 2015, pp. 319–326.

[26] J. Iyilade and J. Vassileva, "P2u: A Privacy Policy Specification Language for Secondary Data Sharing and Usage," in *2014 IEEE Security and Privacy Workshops*, May 2014, pp. 18–22.

[27] T. Baarslag, A. T. Alan, R. Gomer, M. Alam, C. Perera, E. H. Gerding, and m. schraefel, "An Automated Negotiation Agent for Permission Management," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '17. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 380–390, event-place: São Paulo, Brazil. [Online]. Available: http://dl.acm.org/citation.cfm?id=3091125.3091184

[28] "[1401.3859] A Utility-Theoretic Approach to Privacy in Online Services." [Online]. Available: https://arxiv.org/abs/1401.3859

[29] C. Dwork, "Differential Privacy," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, Eds. Berlin, Heidelberg: Springer, 2006, pp. 1–12.

[30] R. Agrawal and R. Srikant, "Privacy-preserving Data Mining," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '00. New York, NY, USA: ACM, 2000, pp. 439–450, event-place: Dallas, Texas, USA. [Online]. Available: http://doi.acm.org/10.1145/342009.335438

[31] L. Sweeney, "k-ANONYMITY: A MODEL FOR PROTECTING PRIVACY," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, Oct. 2002. [Online]. Available: https://www.worldscientific.com/doi/10.1142/S0218488502001648

[32] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, "L-diversity: privacy beyond k-anonymity," in *22nd International Conference on Data Engineering (ICDE'06)*, Apr. 2006, pp. 24–24, iSSN: 1063-6382, 2375-026X.

[33] N. Li, T. Li, and S. Venkatasubramanian, "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity," in *2007 IEEE 23rd International Conference on Data Engineering*, Apr. 2007, pp. 106–115, iSSN: 1063-6382, 2375-026X.