Systematic Information Flow Control in mHealth Systems

Chandrika Bhardwaj
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Email: chandrika@cse.iitd.ac.in

***Abstract*—This paper argues that the security and integrity requirements of mHealth systems are best addressed by end-to-end information flow control (IFC). The paper extends proposals of decentralized IFC to a distributed smartphone-based mHealth system, identifying the basic threat model and the necessary trusted computing base. We show how the framework proposed can be integrated into an existing communication stack between a phalanx of sensors and an Android smartphone. The central idea of the framework involves systematically and automatically labelling data and metadata collected during medical encounters with security and integrity *tags*. These mechanisms provided can then be used for enforcing a wide variety of complex information flow control policies in diverse applications. The chief novelty over existing DIFC approaches is that users are relieved of having to create tags for each class of data and metadata that is collected in the system, thus making it user-friendly and scalable.**

I. Introduction

Traditionally, information *at rest* is secured by access control mechanisms and information *in motion*, i.e., in communication channels between two devices, is secured by encryption/decryption mechanisms. However these mechanisms do not suffice as the information is also processed by applications and is transmitted outside the domain of the access control mechanisms [1]. For example, if user/application *A* is allowed to read a patient Puja's data, Puja cannot control how *A* distributes the information that it has read. Essentially, the traditional mechanisms do not address patients' requirements, which include controlling information propagation instead of merely preventing information release and relying on secure transmission.

For addressing all such requirements, *decentralized* information flow control (DIFC) mechanisms need to be adopted and integrated into mHealth systems. DIFC systems track and restrict the release of the data as they flow through the system [2]. Earlier work on information flow control [3] has been extended in DIFC to protect data for different users, each with their individual policy [1].

In this paper, we present SIFT, a *Systematic Information Flow Tagging* framework to address the confidentiality and integrity requirements specific to the mHealth domain. SIFT is designed to enable control on flows of all data and metadata that are captured during a medical encounter, including images and videos. The middleware component (which resides on the smartphone) tags the data and contextual data captured from the sensor hardware and the smartphone apps. The tags keep track of the confidentiality level or the integrity level of the data items. SIFT supports two types of restrictions: *Confidentiality restrictions* which ensure that only the authorized principals get access to the data and *Integrity restrictions* which ensure that the trustworthiness of the data is not compromised while propagating the data onto servers. SIFT enables *principals* such as patients, doctors, hospital administrators etc. to specify their confidentiality and integrity constraints based on the sensitivity and purpose of the data, at a fine grained level. These policies can then be enforced easily on the tagged database created by the SIFT framework.

This paper makes following contributions:

- *We propose the SIFT framework for equipping data-collecting mHealth systems with inbuilt provisions for maintaining confidentiality and integrity of the data captured from the sensors*, in accordance with the fine grained policies defined by individuals. The SIFT framework allows for any policy to be implementable.
- *The proposed framework provides the required infrastructure for implementing DIFC mechanisms* along the lines proposed by recent efforts [2], [4], [5]. The crucial concept in DIFC mechanisms is placing tags with data and metadata, which specify the sensitivity of the data. In the cited work, the tags used in these mechanisms have been assumed to be present in the database courtesy the user. However,
  - users may find it extremely hard to think of appropriate tags on their own and to tag the individual data records themselves, more so because users are not even aware of all data and metadata that are typically captured in a medical encounter;
  - there is high probability of introduction of errors when users attempt to manually tag each field or record. These may even lead to data discrepancies.

  The proposed framework automatically annotates the data such that the users can directly define policies, which can then be implemented using those tags.
- In the proposed framework, we have incorporated this *automated tagging* into an existing communication stack for sensors communicating with mobile nodes [6]. As both data and metadata are being tagged, with tags being provided by appropriate layers of the modified stack, the

system efficiently adapts existing DIFC techniques with little overhead or burden to the principals.

Rest of the paper describes the motivating example, security model considered, information flow concepts and SIFT's system design.

II. Motivating Example

Consider Puja, who is a patient under the supervision of Dr Divya; Heena is a health worker who routinely visits Puja's home to capture significant data and metadata. Dr Rita is a researcher at WHO and Bimla is the billing executive in the hospital to which Puja goes when consulting Dr. Divya. Puja may specify the following confidentiality policy (illustrated in Figure 1) for data owned by her:

Policy 1: "*Share Puja's medical data and contextual data with her doctor, her health worker, the billing department of her hospital and researchers; But do* not *share her DateOfBirth and her Name with researchers.*"

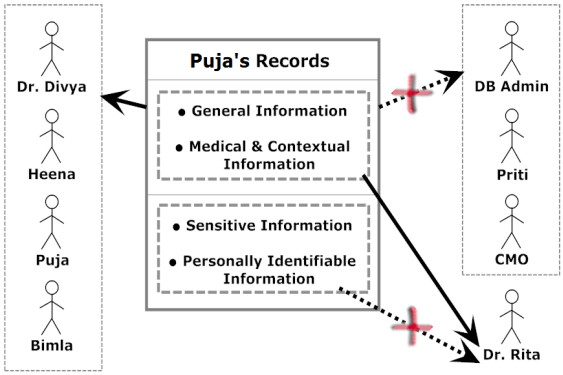

Fig. 1. Puja's Confidentiality Requirements. Bold black lines represent the allowed information flow.

Note: Malicious application software can complicate these basic policies. For example, a health worker may download games or mHealth applications which take permission at installation time to access the SMSs, calendar events, SD card contents and other confidential information. These applications may upload all this sensitive information to a public web server. So, under typical security plans, Puja can only convince themselves that the mHealth applications do not reveal their data to anyone other than the people/applications to whom they have granted the permission to do so. But with DIFC, they can reason about an mHealth application's behaviour by observing behaviour at the interprocess communication level instead of merely validating by code level inspection. [4].

III. Security Model

We overview the system architecture, threat model, security and integrity goals, and the trusted computing base.

A. System Assumptions

We do not focus on the specific security policy compliance, though our work is motivated by the data security compliance regulations. We make the following assumptions about the mHealth system:

- **S1** We assume that there are some existing policies/provisions for deciding the individual ownership of data, and that the hospitals and other organizations involved provide the required infrastructure for defining decentralized ownerships.
- **S2** We assume appropriate DIFC security and integrity policies are defined by relevant principals and that these are implemented correctly using some existing DIFC mechanism [5].
- **S3** We assume both microcontroller boards and mobile nodes are capable of supporting sufficiently good cryptographic primitives (such as AES, SHA-1, etc.) to ensure the confidentiality and authenticity of data packets being exchanged between them.

B. Threat Model

The following threats arise in relation to the security critical assets of mHealth systems:

- **T1** *Threat to Confidentiality of 'data being processed'*: The adversary wants to access information (both data and metadata) about patients or other principals involved in the mHealth system, while the information is being processed or propagated by the applications in the mobile devices.
- **T2** *Threat to Integrity of 'data being processed'*: The adversary wants the mHealth system to accept and upload corrupted information about patients or other principals.
- **T3** *Threat to Confidentiality and Integrity of 'data at rest' and 'data in motion'*: Confidentiality and integrity of data packets (being transmitted via Bluetooth) may be compromised if sufficiently good encryption and authentication mechanisms are not used by the microcontroller software. This threat is addressed by system assumption **S3**.
- **T4** *Threat to Usability of DIFC-enabled mHealth system framework*: The adversary wants users to not understand the dynamics of the DIFC-enabled system framework properly and to accept/define DIFC policies that ultimately leak user data.
- **T5** We do not address here threats such as the data captured using sensors being inaccurate due to the sensors being faulty or compromised. The adversary may adapt the sensors to give to the microcontroller board valid readings that are not accurate or precise.

Enforcement of confidentiality and integrity policies ultimately depends on the correct implementation of some DIFC mechanism in the smartphone. This threat is addressed by system assumption **S2**.

C. Security and Integrity Goals

We define the set of properties that an mHealth system framework should satisfy to ensure the confidentiality and integrity of the medical data and metadata being captured

(using sensors) in a mobile node. These properties define our security and integrity goals.

- **G1** *End-to-end Confidentiality of 'data being processed'*: The system framework should provide appropriate mechanisms to track the flow of sensitive information inside the mobile node, to ensure that unauthorized disclosure/leak of information is prevented altogether (addressing **T1** and **T3**).
- **G2** *End-to-end Integrity of 'data being processed'*: The system framework should provide appropriate mechanisms to track the end-to-end integrity of sensitive information inside the mobile node, to ensure that untrusted data does not corrupt trusted data (addressing **T2** and **T3**).
- **G3** *Usable DIFC-enabled mHealth system framework*: The system framework should be usable by naive users without requiring them to know a lot of details about how the system works (addressing **T4**). This is the central focus of our paper. We address the challenge of *automatically and systematically* annotating the data and metadata, securely with appropriate tags for security and integrity policy compliance by DIFC mechanisms.

The work in this paper does not seek to address malicious data disclosure, which could be due to collusion between applications or side channel attacks or malicious daemons operating outside our confinement, or users who may transcribe the data from the screen. We specifically focus on the data residing and processed on the smartphones, and do not address data exposure from cloud services (when data leave the smartphone).

D. SIFT *Architecture*

SIFT equips a distributed mHealth system setting with decentralized information flow control capability. As only trusted mobile nodes are allowed to enter the system, each mobile node is expected to run the SIFT framework. SIFT tracks information on a fine-grained per-field level within the database because sensitivity of each field varies. For example, the date of birth of a patient is likely to be more sensitive than the date of a medical encounter or even the age group. Allowing someone to access the complete information of a record related to a medical encounter will be unwise. Consequently, we choose to provide field-level control over the flow of information in the mHealth system framework, i.e., a patient can control whether the doctor gets access to her date of birth or just her age.

E. Trusted Computing Base

In order to build a trusted mHealth system, various components need to be in the *trusted computing base* (TCB). The proposed SIFT framework and the DIFC mechanism running atop it are both part of the TCB, as both work together to enforce the information control policies of principals involved. Apart from the application runtime, the TCB of our framework also consists of the lower layers on which it runs, i.e. the OS (Android in our case) and the hardware. The application components which authenticate the users, grant them privileges and define the information policies also need to be trusted. Lastly, the drivers for the Arduino board, the board itself and the Arduino software which captures the data readings from the sensors and transmit these to the smartphone are also included in the trusted computing base.

IV. Information Flow Concepts

This section describes the main concepts in the SIFT framework and decentralized information flow control in an mHealth system.

A. Tags, Labels and Principals

The SIFT framework uses tags and labels to track the flow of information in the system and principals' authority [5] to restrict the flow according to the security and integrity policies specified in the mHealth system.

1) Principals: Principals are entities, such as users and roles in the mHealth system framework, who have security interests or are relevant to other principals' security interests. The processes which access data from records also run with the authority of these principals.

2) Tags: Tags are identifiers which denote the sensitivity of the data to which they are attached. For example, *Puja_PII* can be the 'security' tag on all data objects which belong to the category of personally identifiable information of patient Puja. Similarly, *SensorStack_Validated* can be the 'integrity' tag on all the medical data which have been collected from sensors and have also been validated based on the hardware specification by the SensorStack to be of correct format, uncorrupted and belong to the valid range of values possible for a particular sensor.

3) Labels: Labels are just sets of tags signifying the sensitivity of composite data-records depending on the data items from which it has been derived/composed. Labels are of two types: 'Security Labels', and 'Integrity Labels'. Say data-record A is labelled with label L_A and data record B is labelled with label L_B. Then information is allowed to flow from A to B iff :

- For Confidentiality: $L_A \subseteq L_B$
- For Integrity: $L_B \subseteq L_A$ [3].

Although the labels on the data items are updated while being processed in the Secure Sensor Stack (ref. V-B2), the labels of the data records become immutable after they are written to the database and cannot be changed thereafter.

4) Lattice Structure: Labels form a lattice under the partial order based on the subset relation [3] [7].

5) Authority and Capabilities: Each tag $h(enid)\hat{}st_p$, where $h(enid)$ is the keyed hash of unique medical encounter id, is owned by some principal(s) who define(s) the information flow control policy for the data tagged with this tag. The *authority* of the principals is defined with the help of labels S_{Pid} and I_{Pid}, where Pid is the principal's unique id. S_{Pid} and I_{Pid} contain the security and integrity tags respectively, whose information the principal with identity Pid is allowed to access/affect.

Principals also have *capabilities*, defined as a set C_{Pid}.

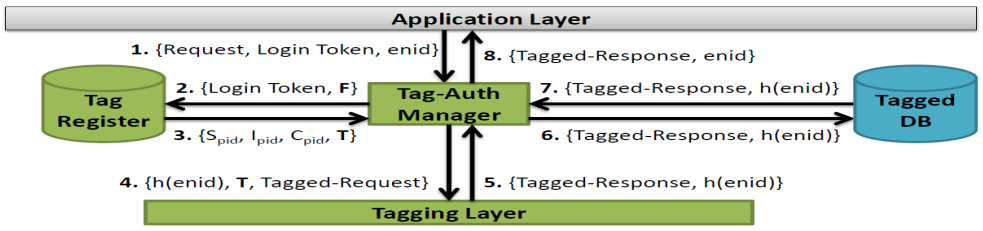

Fig. 2. Main tagging-specific components of Secure Sensor Stack. Here, **F** and **T** represent the set of fields/categories of data objects requested by the application layer, and the set of partial tags corresponding to those fields, respectively.

There are two capabilities defined per tag $enid\hat{}st_p$, namely, $enid\hat{}st_p^+$ and $enid\hat{}st_p^-$. If capability $enid\hat{}st_1^+ \in C_{Pid}$ then a process running with the authority of principal Pid can add the tag to its label and read the information tagged with tag $enid\hat{}st_p$. If capability $enid\hat{}st_p^- \in C_{Pid}$ then a process running with the authority of principal Pid can remove the tag from its label, allowing it to declassify the information tagged with the tag $enid\hat{}st_p$ [4]. Our SIFT framework allows an arbitrary number of principals to be involved in the mHealth applications and each principal has a globally unique identifying number in this framework.

B. Decentralized Privilege

In the SIFT framework, the owners of the tags will be decided by a mutual understanding between the principals in the mHealth system. But once the owners of the tags have been decided, all the owners can specify their individual policies for the flow control of the information owned by them. This scheme of operation decentralizes the control over flow of information and gives the principals 'real control on the data owned by them'.

C. Domain Specific Compound Tags

In order to aid the understanding of the principals for the task of defining policies, the framework also supports the collating of many tags [5].

V. SIFT System Design

The system framework has inbuilt provisions for information flow control and a layered architecture which resides on a smartphone acting as middleware between a smartphone app and the sensor hardware. It consists of four major parts:

- The first part comprises creation of appropriate tags and attaching the tags to the data and metadata being captured using the sensors and the mHealth application running on the smartphone.
- The second part involves securely writing the tagged data objects in files which reside on the smartphone and are also uploaded to the server.
- The third part involves various principals of the mHealth system defining the security and integrity policies for the data owned by them.
- In the fourth part, the middleware enforces the complex security and integrity policies which have been defined by various principals in the mHealth system.

TABLE I
Mapping between *partial tags* and data fields.

Partial Tag	Field
st_d	date_of_birth
st_{dt}	datetime_medical_encounter
st_l	location
st_n	name
st_g	gender
st_p	PII

In this paper, we describe only the first two parts because the focus of this paper is to make the existing DIFC mechanisms user-friendly.

A. Creation of tags

A complete tag of a data object comprises the *keyed hash* of the unique encounter id appended to a partial tag corresponding to the category of the data object. The partial tags are stored in a hashtable in a **Tag Register** (see table I), where they are mapped to the category of the data object.

The Tagging Layer (see Figure 2) of the Secure Sensor Stack is responsible for creation and appending of tags to data objects. We choose to append the *keyed hash* of the unique encounter id to the partial tags for following reasons:

- It gives *fine grained control* over the data captured during encounters. For example, Puja can now restrict the researcher's access to information of only selected encounters, specific to particular conditions, instead of all her records.
- Granting *access permissions for varying time durations* becomes feasible, because encounter ids are linked to a specific date and time. For example, if required, Puja can reveal her medical history of only the past one year to her new doctor Dipti, to get a second opinion.
- Using a *keyed hash* of the encounter id prevents an adversary from correlating the tags with the actual encounter ids.

B. Tagging of data

Figure 2 shows that the application layer forwards user's request to the *TagAuth Manager* along with the encounter id (enid) and the Login Token using which *TagAuth Manager* fetches the authority and capability sets of the principal making the request. After checking whether the request is allowed, the *TagAuth Manager* forwards the request to the Tagging

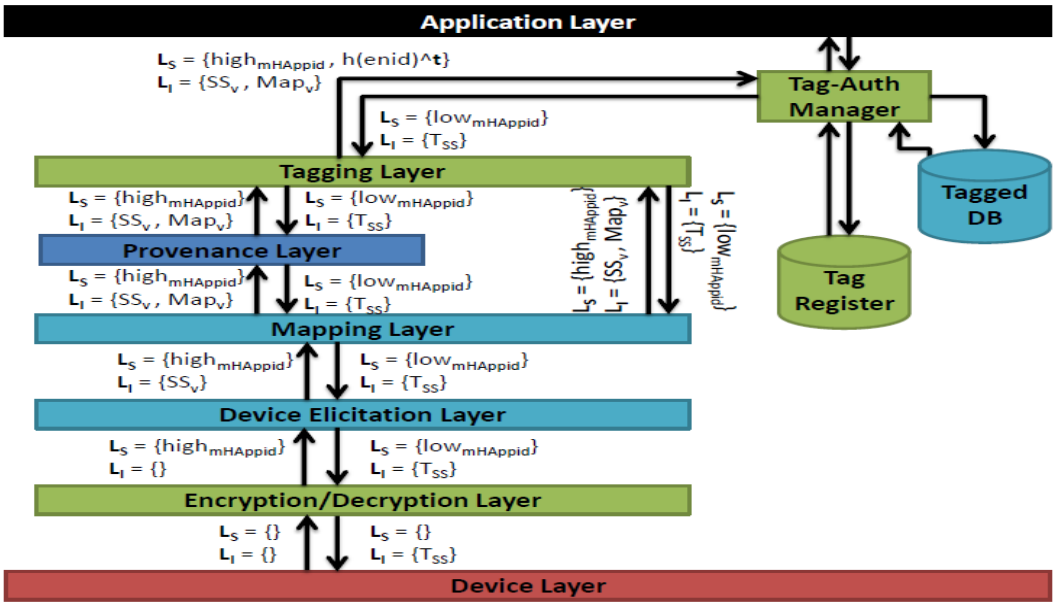

Fig. 3. Secure Sensor Stack, demonstrating the tagging process.

Layer along with the *partial tags* of the data fields requested and the *keyed-hash* of the encounter id. The response-data fetched from the sensors is tagged by the tagging layer and sent to the tagged-database via *TagAuth Manager*.

1) Tag Register: This module is responsible for storing the meaning of the tags. The following information is stored in the Tag Register database:

- The mapping between fields (category of data) and partial tags (see Table I),
- The mapping between principals and their authority (S_{Pid} and I_{Pid}) and capabilities, and
- The authority and capabilities common to all principals, i.e., S_{all}, I_{all} and C_{all}.

The mapping between partial tags and fields (category of data) is globally defined and is stored in a hashtable. Two such hashtables are stored, one keyed on the tags and the other keyed on the category of the data object.

2) Secure Sensor Stack: The design of the Secure Sensor Stack is divided into different layers which communicate with one another and are responsible for providing specific functionalities. The architecture of Secure Sensor Stack is presented in Figure 3. The figure also shows how tagging is done in the layered architecture of the SIFT framework. The layers of Secure Sensor Stack are:

*a) **Application Layer:**:* This layer is where the mHealth applications will be deployed, and make calls to *TagAuth Manager* of the Secure Sensor Stack.

*b) **TagAuth Manager:**:* There are two kinds of requests that an application can make to the *TagAuth Manager*:

- to fetch some fresh data from the sensors.
- to access data that are already stored in database.

In this paper, our focus is on the first type of requests; the latter are handled by schemes already proposed [2].

When an application requests TagAuth Manager to fetch some fresh data points from sensors, it provides TagAuth Manager with a login token that has been generated by the authenticating component of the application. This login token is then used by TagAuth Manager to fetch the authority and capabilities of the principal (with which the process requesting the information is running) from the TagRegister, as illustrated in Figure 2. TagAuth Manager also fetches the set of partial tags **T**, for the fields of information requested, from Tag Register. TagAuth Manager checks if the application is authorized to access the data requested according to rules given in Section IV-A3. If the access is allowed then TagAuth Manager forwards the request to the underlying Tagging Layer along with the keyed hash of the medical encounter id, i.e., *h(enid)* and **T**.

Before forwarding the request, TagAuth appends the security tag '$low_{mHAppid}$' and integrity tag 'T_{SS}' to the requests (Figure 3). The security tag $low_{mHAppid}$ means that the access of the data contained in the requests is restricted to only the Secure Sensor Stack and the requesting application, i.e., one with identifier $mHAppid$. The integrity tag T_{SS} means that the requests are being processed via a trusted sensor stack and therefore are of a certain integrity.

Once TagAuth Manager receives an appropriately tagged response from the underlying Tagging Layer, it securely saves the data in the database and then forwards the response to the application layer. It is responsible for making sure that the data captured in a medical encounter is saved in files or the database along with the tags.

*c) **Tagging Layer:**:* This layer checks if the request from TagAuth Manager is tagged appropriately to reflect the validity of the request. Only then does it forward the request for processing to the underlying layers.

This layer is also responsible for tagging the data freshly fetched from sensors with appropriate tags, specific to the application invoking the calls for fetching data. For each request from TagAuth Manager, it receives *h(enid)* and **T**. So it appends the keyed hash of encounter id, *h(enid)*, to the partial tags $t \in \mathbf{T}$ of the data objects and annotates the data objects with their complete tags. This completion of tags is important because otherwise giving the authority of the tag corresponding to the location field, i.e., st_1 to a principal, say Dr Divya, will give her authorization for accessing the location of all patients whose data are in the system.

d) ***Provenance Layer:****:* This layer is responsible for adding the contextual information relevant to the medical encounter. This contextual information can be captured from both the sensor hardware, by invoking requests to the mapping layer or the sensors already integrated in the smartphone, by directly invoking the smartphone OS.

e) ***Mapping Layer:****:* This layer ensures that a valid request is made to the underlying layers. This layer is responsible for sending the validated sensor values in the desired range/format to the upper layers [6]. In the example illustrated in Figure 3, the layer appends an integrity tag 'Map_v'with all the data that it passes to its upper layers. This tag represents that the data received from the underlying layers has been appropriately mapped to the specified range by Sensor Stack.

f) ***Device Elicitation Layer:****:* This layer is responsible for guiding the device layer about when and what to fetch. This layer also extracts and validates the individual sensor values from the response received from the sensor hardware via the device layer [6]. In the example illustrated in Figure 3, this layer appends an integrity tag 'SS_v'with all the data that it passes to its upper layers. This tag represents that the data has been validated by the Sensor Stack and the data is not incomplete/inconsistent/corrupted.

g) ***Encryption/Decryption Layer:****:* This layer encrypts the payload of the request packet to be sent out of the device, which it receives from the upper layers. After encryption, the security tags are stripped off the data as the encrypted data will now be in the public domain and hence the security label becomes empty. In the example illustrated in Figure 3, the integrity tag 'T_{SS}'is propagated untouched, signifying that the request is coming from a trusted Sensor Stack. In the example illustrated in Figure 3, this layer also decrypts the data received from the device layer and appends the security tag '$high_{mHAppid}$'to the decrypted data, which means that the data is meant only for the Secure Sensor Stack as the mHealth application with identifier $mHAppid$, and is not meant for any other application nor is it to be sent outside the system.

h) ***Device Layer:*** *:* This layer is responsible for establishing connection with the sensor hardware. Once the connection is established, this layer is responsible for fetching raw data from the sensors [8]. As no data validation is performed in this layer and the data is in the public domain, the security and integrity labels of the data fetched by this layer are typically empty, illustrated in the example in Figure 3.

VI. Related Work

The crucial concept in DIFC mechanisms proposed by researchers is placing the tags with data and metadata, which specify the sensitivity of the data [1], [2], [4], [5], [9], [10], [11]. In the cited work, the tags used in these mechanisms have been assumed to be present in the database courtesy the user. However, users may find it extremely hard to manually tag individual data items. In addition, this process is highly prone to errors. The framework proposed in this paper automatically annotates the data such that the users can directly define policies, which can then be implemented using those tags.

VII. Conclusion

The novel idea of the paper is *systematically tagging* the data/metadata elements right from the point of data collection for enabling, in a distributed setting, the fine grained individual privacy preferences in a mHealth system. These tags can be the basis for enforcement mechanisms for a wide variety of security/integrity policies in distributed information flow systems. We have described how this framework may be incorporated into an existing communication stack developed for wireless communication between sensors attached to a microcontroller board and an Android smartphone. The design however is far more general, and is applicable in settings beyond particular devices, protocols or even mHealth applications.

References

[1] A. C. Myers and B. Liskov, "A Decentralized Model for Information Flow Control," in *ACM Symp. on Operating Systems Principles*, 1997, pp. 129–142.

[2] D. A. Schultz and B. Liskov, "IFDB: decentralized information flow control for databases," in *Proc. of the 8th ACM European Conf. on Computer Systems*. ACM, 2013, pp. 43–56.

[3] D. E. Denning, "A Lattice Model of Secure Information Flow," *Commun. ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[4] M. N. Krohn, A. Yip, M. Z. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris, "Information flow control for standard OS abstractions," in *ACM Symp. on Operating Systems Principles*, 2007, pp. 321–334.

[5] W. Cheng, D. R. K. Ports, D. A. Schultz, V. Popic, A. Blankstein, J. A. Cowling, D. Curtis, L. Shrira, and B. Liskov, "Abstractions for Usable Information Flow Control in Aeolus," in *USENIX Annual Tech. Conf.*, 2012, pp. 139–151.

[6] A. Kansal and A. Gupta, "Sensor Stack on Android for mHealth Applications," Master's thesis, Department of Computer Science & Engineering, IIT Delhi, 2014.

[7] D. E. Bell and L. J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation. *MITRE Corporation*," 1976. [Online]. Available: http://csrc.nist.gov/publications/history/bell76.pdf

[8] A. Kansal, A. Gupta, K. Paul, and S. Prasad, "mDROID - An Affordable Android based mHealth System," in *International Conf. on Health Informatics*. SciTePress - Science and Technology Publications, 2014.

[9] P. Efstathopoulos, M. N. Krohn, S. Vandebogart, C. Frey, D. Ziegler, E. Kohler, D. Mazières, M. F. Kaashoek, and R. Morris, "Labels and event processes in the Asbestos operating system," in *ACM Symp. on Operating Systems Principles*, 2005, pp. 17–30.

[10] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières, "Making Information Flow Explicit in HiStar," in *USENIX Conf. on Operating Systems Design and Implementation*, 2006, pp. 263–278.

[11] N. Zeldovich, S. Boyd-Wickizer, and D. Mazières, "Securing Distributed Systems with Information Flow Control," in *USENIX Symp. on Networked Systems Design and Implementation*, 2008, pp. 293–308.