Step Towards Better Security: Attacks and Defenses for Low Power Fitness Trackers

Mahmudur Rahman
Florida International University
Miami, Florida
Email: mrahm004@cs.fiu.edu

Bogdan Carbunar
Florida International University
Miami, Florida
Email: carbunar@cs.fiu.edu

Umut Topkara
IBM Research
Yorktown Heights, NY
Email: umut@us.ibm.com

***Abstract*—Wearable personal fitness trackers automatically collect sensor data about the user throughout the day and integrate this data into social network accounts. The increasing popular interest in personal telemetry, also called the Quantified Self or ‘‘lifelogging’’, has induced this flourishing new product category. The makers of these trackers have to strike a balance between many constraints including time to release, cost of tracker hardware, battery life, features, mobility, usability, and utility to end user. Unfortunately, such a constrained design process usually puts security to the back seat, making the collection, storage and transmission of personal fitness data in the trackers that we purchase, vulnerable to security attacks. We studied a popular product called Fitbit, which uses i) a lightweight tracker that stores sensor data, ii) a base station for communication, and doubles as charging dock, and iii) a backend for long-term storage of historical data. We first describe how we reverse engineered and identified security vulnerabilities in Fitbit, and introduce FitBite, a tool that enables an attacker to launch eavesdropping, injection and denial of service attacks against Fitbit. Second, we devise FitLock, a general secure data storage and communication solution, for use by makers of affordable and lightweight personal trackers. FitLock exploits a time-space trade-off to reduce the hardware and computation requirements on the tracker into simple and cheap bitwise XOR and EQUAL operations, and yet FitLock ends up with no additional storage cost on the tracker.**

I. Introduction

Wearable personal trackers which collect sensor data about the wearer throughout the day, have long been used for patient monitoring in healthcare. Holter Monitors [**?**], with large and heavy enclosures, that use tapes for recording, have recently evolved into affordable personal fitness trackers (e.g., [**?**]). Fitness trackers have successfully penetrated the consumer electronics market, thanks to advances in mobile and wearable computing, as well as in sensor and wireless technologies.

Meanwhile, health centric *social sensor networks* have emerged, with large numbers of users. Products like Fitbit [1], Garmin Forerunner[2], Wahoo [3], Jawbone UP [4], RunKeeper [5] and RunMeter [**?**], require users to carry wireless trackers that continuously record a wide range of fitness and health parameters (e.g., steps count, heart rate, sleep conditions). Trackers report recorded data to a providing service, either directly (e.g., through cellular connection) or through a specialized wireless base, that connects to the user's personal computer. The services that support these trackers enable users to analyze their fitness trends with maps and charts, as well as share with friends in their social networks.

All happening too quickly both for vendors and users alike, this data-centric lifestyle, popularly referred to as the Quantified Self or ‘‘lifelogging’’, is now producing massive amounts of intimate personal data. For instance, BodyMedia has created one of the worlds largest libraries of raw and real-world human sensor data, with 500 trillion data points [6]. This data is becoming the source of significant privacy and security concerns: information about locations and times of user fitness activities can be used to infer surprising information, including the times when the user is not at home [7], and even company organizational profiles [8].

In this paper we demonstrate vulnerabilities in the collection, storage and transmission of personal fitness data in a popular tracker, Fitbit [1]. For instance, we show that the privacy of users can be breached by capturing and modifying the data stored on any tracker situated within a radius of 15 ft. We further prove the ability to inject arbitrary data on social networking accounts belonging to nearby tracker owners. This attack can enable malicious users to fraudulently accumulate a variety of rewards and even health insurance discounts. Health insurance companies have expressed interest in providing discounts to customers that use trackers to prove a healthy lifestyle [9].

We believe that, the vulnerabilities and the gaps that we identified in the security of Fitbit are a symptom of the nature of quickly introducing new wearable consumer devices to the market. In an effort to help add security to new product designs, we introduce FitLock, a solution that guarantees secure sensor and fitness data storage, and transmission for lightweight personal trackers. FitLock exploits a time-space tradeoff to reduce the hardware and computation requirements imposed on trackers, to fast bitwise XOR and EQUAL operations. FitLock however imposes no additional storage cost on the tracker.

FitLock achieves security by making it impossible to read the plaintext data outside of the backend server. This effect could be trivially achieved by using public key encryption on the tracker. However public key algorithms are known to be computationally expensive, hence are not practical for use in simple trackers. Furthermore, many fitness trackers display instant summaries for users. Such data either has to be computed with counters as the sensor is read, and stored

in the clear, or should be retrieved from the backend server if the privacy requirements bar even summaries from being stored in the clear.

Since the difficulty of adding security on a new product will be shared by other vendors with good product ideas, we design FitLock with generality in mind. By imposing small computation overheads on trackers, minimizing user interaction and requiring only generic bases, FitLock's approach may benefit other products similar to Fitbit. Thus, the contributions of this paper are the following:

- Reverse engineer the semantics of the Fitbit tracker memory banks, command types and the tracker-to-social network communication protocol [Section III-B].
- Build FitBite, a tool that exploits the design of Fitbit design to prove the feasibility of a wide range of attacks [Section IV].
- Devise FitLock, an efficient solution that imposes only fast, bitwise XOR and EQUAL operations on tracker devices. Show that FitLock is resilient even to powerful probing attackers, capable of reading and altering the memory of captured trackers [Section V].
- Evaluate FitLock through an implementation on a testbed of Android devices. FitLock significantly reduces the overhead imposed by a solution based on public key cryptography. FitLock imposes only negligible (1.08%) overhead on Fitbit [Section VII].

As a courtesy, we have contacted the Fitbit team and reported our results. We are also making the source code of FitBite and FitLock publicly available on the project website [10].

II. Related Work

Tsubouchi et al. [8] have shown that Fitbit data can be used to infer surprising information, in the form of working relations between tracker carrying co-workers. This information could be used to surreptitiously learn the organizational profile of a company. This work assumes access to the fitness data of other users, a task that (part of) our paper undertakes.

Halperin et al. [11] demonstrated attacks on pacemakers and implantable cardiac defibrillators (ICDs). They proposed patient warnings, authentication and key exchange defenses. The devices considered communicate over radio frequency (RF) technologies, enabling the harvesting of induced RF energy to power the implementation of the proposed defenses. Rasmussen et al. [12] proposed a proximity based access control solution for securing pacemakers and ICDs, by verifying the distance from the communicating peer before initiating wireless communication. Li et. al. [13] demonstrated successful security attacks on a commercially deployed glucose monitoring and insulin delivery system and provided defenses against the proposed attacks. Our constraints are different, as we consider an application domain with insufficient resources (and interest so far) to provide API support even for efficient cryptographic operations such as modular exponentiation, pseudo-random permutations (e.g., AES) or one-way functions.

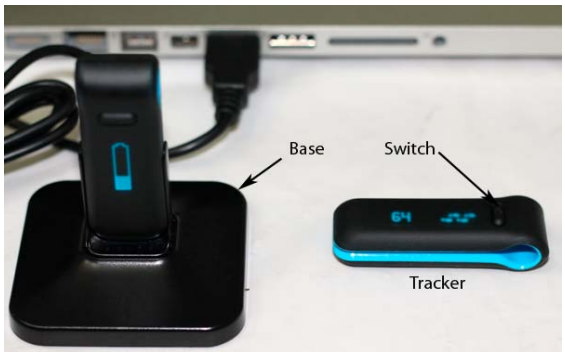


Fig. 1. Fitbit system components: trackers (one cradled on the base), the base (arrow indicated), and a user laptop. The arrow pointing to the tracker shows the switch button, allowing the user to display various fitness data.

Lim et al. [14] analyzed the security of a remote cardiac monitoring system where the data originating from the sensors is sent through a Body Area Network (BAN) gateway and a wireless router to a final monitoring server. Muraleedharan et al. [15] proposed DoS attacks including Sybil [16] and wormhole [17] attacks, for a health monitoring system using wireless sensor networks. They further proposed an energy-efficient cognitive routing algorithm to address such attacks.

Barnickel et al. [18] targeted security and privacy issues for HealthNet, a health monitoring and recording system. They proposed a security and privacy aware architecture, relying on data avoidance, data minimization, decentralized storage, and the use of cryptography. Marti et al. [19] described the requirements and implementation of the security mechanisms for MobiHealth, a wireless mobile health care system. MobiHealth relies on Bluetooth and ZigBee link layer security for communication to the sensors and uses HTTPS mutual authentication and encryption for connections to the backend.

III. Background and Model

We center our model on Fitbit [1], a popular health centric social sensor network (see Figure 1). The Fitbit system consists of user tracker devices, user USB base stations and an online social network. Here we detail each component.

Fitbit tracker. The *tracker* is a wearable device that measures the daily steps taken, distance traveled, floors climbed, calories burned, and the duration and intensity of the user exercise. It consists of four IC chips, (i) a MMA7341L 3-axis MEMS accelerometer, (ii) a MEMS altimeter to count the number of floors climbed, (iii) a MSP430F2618 low power TI MCU consisting of 92 KB of flash and 96 KB of RAM and (iv) a nRF24API 2.4 GHz RF chip supporting the ANT protocol (1 Mbits/sec @15 ft transmission range). The user can switch between displaying different real-time fitness information on the tracker, using a dedicated hardware *switch* button (see the arrow pointing to the switch in Figure 1). Each tracker has a unique id, called the *tracker public id* (TPI).

The base. The *base* connects to the user's main compute center (e.g., PC, laptop) and is equipped with a wireless communication chip that enables it to communicate with any

tracker within a range of 15 ft. It acts as a bridge between trackers and the online social network. It sets up wireless connections (over the ANT protocol) with trackers within its transmission range, then relays commands issued by the webserver. Figure 1 shows a snapshot of two trackers and a base, connected to a laptop through a USB port.

The webserver. The online social network webserver, allows users to create accounts from which they befriend and maintain contact with other users. Upon purchase of a Fitbit tracker and base, a user binds the tracker to her social network account. Each social network account has a unique id, called the *user public id* (UPI). When the base detects and sets up a connection with the tracker, it automatically collects and reports tracker stored information (step count, distance, calories, sleep patterns) to the corresponding user's social network account. In the following, we use the term *webserver* to denote the computing resources of the online social network.

Tracker-to-base communication: the ANT protocol. Trackers communicate to bases over ANT, a 2.4 GHz bidirectional wireless Personal Area Network (PAN) ultra-low power consumption communication technology, optimized for transferring low-data rate, low-latency data.

Data conversion. The tracker relies on the user's walk and run stride length values to convert the step count into the distance covered. The tracker also extrapolates the user's Basal Metabolic Rate (BMR) [20] values and uses them to convert the user's daily activities into burned calories values.

A. Attacker Model

We consider external attackers that attempt to learn and modify the fitness information reported by the trackers of other users, as well as disrupt the Fitbit protocol. We assume attackers are able to capture wireless communications in their vicinity. Thus, the attackers can impersonate system participants, sniff, inject and jam communications. Attackers also control any number of bases, except the ones belonging to honest users. We assume that the Fitbit service (e.g. the social network servers) will not collude with attackers to facilitate false data reports.

While the attacks we propose in this work do not require physical access to victim trackers, we consider also attackers that have access to trackers. Furthermore, we also consider attackers that are able to perform JTAG and boundary scan based attacks (e.g., [21]) to read/write information directly from/to the memory board. We call such adversaries, *JTAG attackers*.

B. Reverse Engineering Fitbit

We have reverse engineered the Fitbit communication protocol. This includes the message communication format among the participating devices and the organization of the tracker memory. We have partially relied on libfitbit [22], providing information on open source fitness hardware. Furthermore, we have exploited Fitbit's lack of communication encryption to implement a USB based filter driver that logs the data flowing through the base.

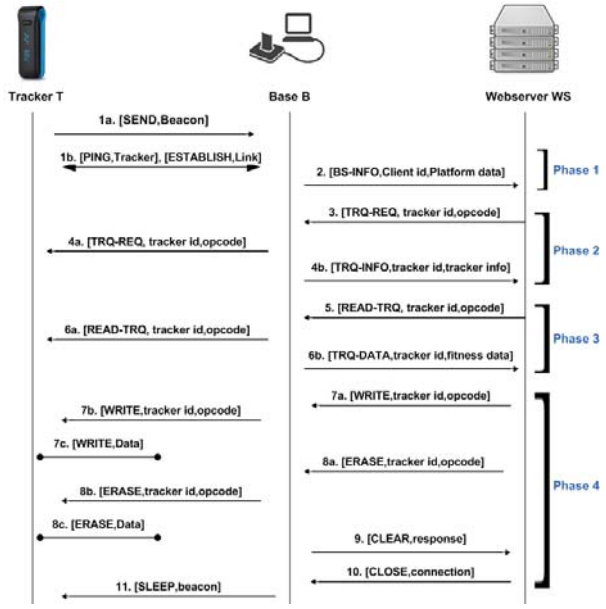


Fig. 2. Fitbit Upload protocol. Enables the tracker to upload its collected sensor data to the user's social networking account on the Fitbit webserver. The initial connection is established by the tracker, but subsequent commands are issued by the webserver.

Tracker memory organization. A tracker has both *read banks*, containing data to be read by the base and *write banks*, containing data that can be written by the base. The read banks store the daily user fitness records. The write banks store user information specified in the "Device Settings" and "Profile Settings" fields of the user's Fitbit account. The tracker stores current sensor readings (e.g., step and floor count values) in a L1D cache. The tracker periodically commits these values to the read bank. This enables the collection of a fine grained user fitness history. The tracker can store 7 days worth of minute-by-minute sensor readings [23].

The Fitbit communication protocol. The communication between the webserver and the tracker through the base, is embedded in XML blocks, that contain base64 encoded opcodes – commands for the tracker. All opcodes are 7 bytes long and vary according to the instruction type (e.g., TRQ-REQ, READ-TRQ, WRITE, ERASE, CLEAR). In the following, for brevity, we use the notation "HOME" to denote the URL http://client.fitbit.com. The data flow between the tracker, base and the webserver during the data upload operation, illustrated in Figure 2, is divided into 4 phases, beginning at steps 2, 3, 5 and 7:

1) Upon receiving a beacon from the tracker, the base establishes a connection with the tracker.
2) **Phase 1:** The base contacts the webserver at the URL HOME/device/tracker/uploadData and sends basic client and platform information.
3) **Phase 2:** The webserver sends the tracker id and the opcode for retrieving tracker information (TRQ-REQ).
4) The base contacts the specified tracker, retrieves its information TRQ-INFO (serial number, firmware version, etc.)

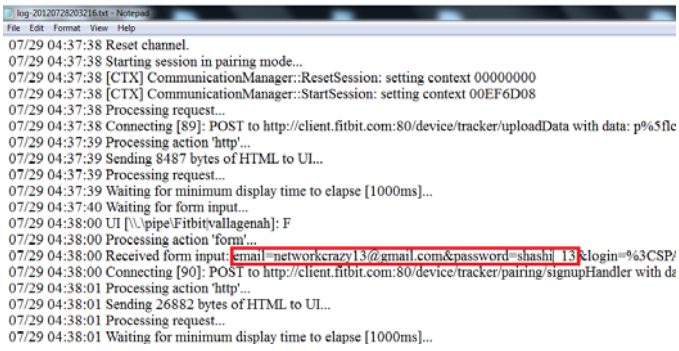


Fig. 3. Fitbit service logs: Proof of login credentials sent in cleartext in a HTTP POST request sent from the base to the webserver.

and sends it to the webserver at HOME/device/tracker/dumpData/lookupTracker.

5) **Phase 3:** Given the tracker's serial number, the webserver retrieves the associated tracker public id (TPI) and user public id (UPI) values. The webserver sends to the base the TPI/UPI values along with the opcodes for retrieving fitness data from the tracker (READ-TRQ).
6) The base forwards the TPI and UPI values and the opcodes to the tracker, retrieves the fitness data from the tracker (TRQ-DATA) and sends it to the webserver at HOME/device/tracker/dumpData/dumpData.
7) **Phase 4:** The webserver sends to the base, opcodes to WRITE updates provided by the user in her Fitbit social network account (device and profile settings, e.g., body and personal information, time zone, etc). The base forwards the WRITE opcode and the updates to the tracker, who overwrites the previous values on its write memory banks.
8) The webserver sends opcodes to ERASE the fitness data from the tracker. The base forwards the ERASE request to the tracker, who then erases the contents of the corresponding read memory banks.
9) The base forwards the response codes for the executed opcodes from the tracker to the webserver at the address HOME/device/tracker/dumpData/clearDataConfigTracker.
10) The webserver replies to the base with the opcode to CLOSE the tracker.
11) The base requests the tracker to SLEEP for 15 minutes, before sending its next beacon.

IV. FitBite: Attacking Fitbit

During the reverse engineering process, we discovered two fundamental vulnerabilities, which we describe here. We then detail the attacks we have deployed to exploit them.

A. Vulnerabilities

Cleartext login information. During the initial user login via the Fitbit client software, user passwords are passed to the webserver in cleartext (as part of POST data) and then stored in log files on the base. Figure 3 shows a snippet of captured data, with the cleartext authentication credentials emphasized.

Fig. 4. Outcome of Tracker Injection (TI) attack on Fitbit tracker: The daily step count is unreasonably high (167,116 steps).

Cleartext HTTP data processing. No encryption or authentication is used when the tracker uploads its data to the webserver. All the requests issued by the webserver and the answers provided by the tracker are sent in clear.

B. The FitBite Tool

We have built FitBite, a tool that exploits the above vulnerabilities to attack Fitbit. FitBite consists of two modules. The Base Module (BM) is used to retrieve data from the tracker, inject and upload fabricated values into the account of the corresponding user on the webserver. The Tracker Module (TM) is used to read and write the tracker data. FitBite implements the following attacks.

Tracker Private Data Capture (TPDC). Use the TM module to discover any tracker device within a radius of 15 ft and capture the fitness information stored on the tracker. This attack can be launched in public spaces, particularly those frequented by Fitbit users (e.g., parks, sports venues, etc). This attack is particularly damaging as Fitbit trackers store sensor readings with a one per minute frequency. This enables the attacker to collect up to 7 days of fitness data history.

Tracker Injection (TI) Attack. Use knowledge about the format of communication packets, opcode instructions and memory banks, to modify the fitness data stored on neighboring trackers. FitBite allows the attacker to choose the data fields to be modified. The TM module can simultaneously modify multiple fitness records (memory banks). Thus, FitBite allows the attacker to modify even the fitness history of the victim. Figure 4 shows an example of a victim tracker, displaying an unreasonable value for the (daily) number of steps taken by its user.

User Account Injection (UAI) Attack. Hijack the data reported by trackers in the vicinity of the attacker, through the attacker's corrupt USB base. Use the BM module to launch a data injection attack: fabricate a data reply embedding the desired fitness data (encoded in the base64 format). The BM module sends the reply as an XML block in an HTTP request to the webserver. The webserver does not authenticate the request message and does not check for data consistency – thus it accepts the data. Using this attack, we have successfully

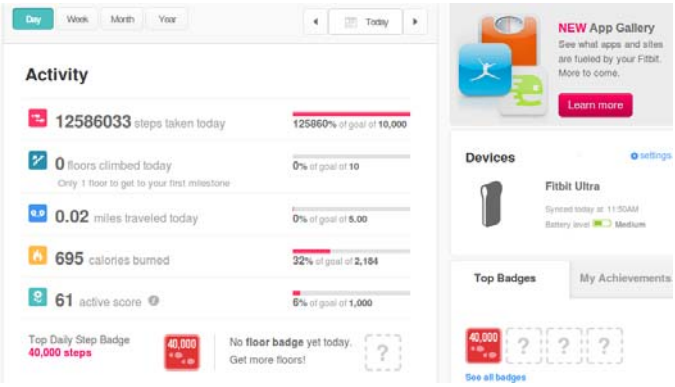


Fig. 5. Snapshot of Fitbit user account data injection attack. In addition to earning undeserved badges (e.g., the "Top Daily Step"), it enables insiders to accumulate points and receive financial rewards through sites like Earndit [24].

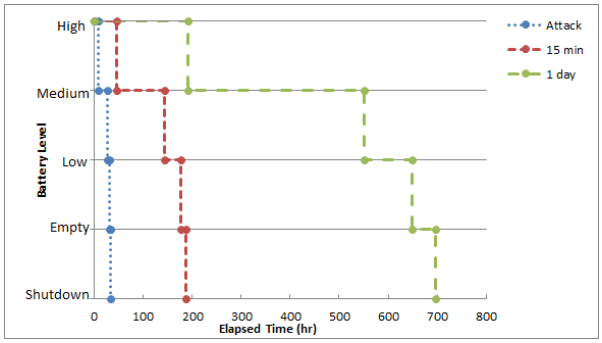


Fig. 6. Battery drain for three operation modes. The *attack* mode drains the battery around 21 times faster than the 1 day upload mode and 5.63 times faster than the 15 mins upload mode.

injected unreasonable daily step counts, e.g., 12.58 million, see Figure 5. The attack also proves that Fitbit does not check for data consistency: the 12.58 million steps correspond to a distance traveled of 0.02 miles.

Free Badges and Financial Rewards. By successful injection of large values in their social networking accounts, FitBite enables insiders to achieve special milestones and acquire merit badges, without doing the required work. Figure 5 shows how the injected value of 12.58 million steps, being greater than 40,000, enables the account owner to acquire a "Top Daily Step" badge.

Furthermore, Fitbit users are encouraged to link their social networking accounts to systems that reward users for exercising. For instance, Earndit [24] provides gift cards and financial prizes through accumulated points: 0.75 points for each "very active" Fitbit minute and 0.10 points for a "fairly active" minute. By keeping the BM module running and continuously updating the tracker data (once each 15 minutes), we have accumulated a variety of undeserved rewards, including 200 Earndit points, redeemable for a $20 gift card.

Battery Drain Attack. FitBite allows the attacker to continuously query trackers in her vicinity, thus drain their batteries at a faster rate. To understand the efficiency of this attack, we have experimented with 3 operation modes. First, the *daily upload* mode, where the tracker syncs with the USB base and the Fitbit account once per day. Second, the *15 mins upload* mode, where the tracker is kept within 15 ft. from the base, thus allowing it to be queried once every 15 minutes. Finally, the *attack* mode, where FitBite's TM module continuously (an average of 4 times a minute) queries the victim tracker. To avoid detection, the BM module uploads tracker data into the webserver only once every 15 minutes. Figure 6 shows our battery experiment results for the three modes: FitBite drains the tracker battery around 21 times faster than the 1 day upload mode and 5.63 times faster than the 15 mins upload mode.

C. Perspective

We have used a three pronged approach to hack Fitbit. First, we have built a USB based filter driver, enabling us to collect tracker information from the USB base. Second, we have exploited Fitbit's raw usage of ANT and its lack of encryption (including passwords). Finally, we have used reverse engineering efforts coupled with a trial and error approach, to bypass the procedure binding the tracker to the user's social networking account.

V. Efficient Security

A. Solution Requirements

Our goal is to develop a health centric social sensor network solution that satisfies the following requirements:

- **Secure.** Provide security assurances against the attackers described in Section III-A.
- **Lightweight tracker.** Minimize the computation overhead imposed on the low power trackers.
- **Flexible sync.** Allow trackers to sync with the social network account of their users, from multiple bases.
- **Generic base.** Minimize the use of proprietary solutions on bases.
- **User friendly.** Minimize user interaction.

One challenge thus is to provide a solution that is secure against powerful adversaries, with minimal tracker and user involvement.

B. FitLock Overview

We now propose FitLock, a secure and lightweight Fitbit extension. We identified 3 basic procedures, for which we provide a solution: RegisterBase, RecordData and Upload.

FitLock prevents trackers from synchronizing with the social network accounts of their users through unknown bases. In fact, we consider such an operation to be insecure. Instead, the RegisterBase procedure enables users to register trusted bases with the webserver. Only registered bases can then later be used for uploading tracker data.

The tracker periodically runs the RecordData procedure to commit to memory intermediate sensor readings. Figure 7 illustrates the FitLock tracker storage organization: A one time pad, OTP is written on the tracker (during RegisterBase and Upload procedures). When RecordData is run, the tracker computes a bitwise XOR between current sensor readings

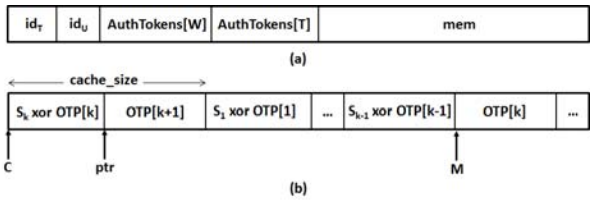


Fig. 7. FitLock tracker "read memory" organization. (a) General organization, showing the TPI, UPI and authentication tokens. mem denotes the area available for storing sensor readings. (b) Zoom-in into mem. The cache (shown part of mem) can store 2 sensor records, of which only the first (the k-th overall) has been written. The subsequent mem space stores the written records up to the $k-1$-th instance. The remaining mem contains unused OTP sequences.

and part of the OTP string. The Upload procedure allows the tracker to sync its (OTP encoded) data to W, through a registered base. After the sync completes, the Upload procedure overwrites the tracker's memory with a freshly generated one time pad. A pair of single use authentication tokens, $AuthTokens[W]$ and $AuthTokens[T]$ is used by the webserver and the tracker to mutually authenticate during RegisterBase and Upload runs. Thus, instead of expensive cryptographic constructs, FitLock's security relies on safe synchronization through registered bases and on one time pads. OTPs provide information theoretic security assurances.

C. FitLock Details

In the following, U denotes a user, T denotes her tracker, B is a base and W is the Fitbit webserver. Let id_U, id_B, and id_T denote the public unique identities of U, B, and T. U has an account with W, with the UPI id_U and a password P_U.

The protocol F for each procedure (Upload, RecordData, RegisterBase) running between participants $P_i \in \{U, T, B, W\}$, each with its own input arguments, is denoted by $F(P_1(args_1), .., P_n(args_n))$.

We assume that any base can setup a secure connection with the webserver, using standard protocols like OAuth and SSL, and keying material (e.g., public key certificates) embedded in the base. Being standard cryptography, for brevity, we omit their presentation.

Without loss of generality, we describe the procedures as if each user has one tracker. The webserver stores and maintains a database Map that has an entry for each user: $[id_U, id_T, H(P_U), Bases, AuthTokens, OTP]$. P_U is U's password and $H()$ is a cryptographic hash function. $Bases$ is a list of registered bases. OTP is a one time pad used to encode tracker readings. $AuthTokens$ is an array which holds a copy of mutual authentication token pairs for the base to be used in two of the procedures: Upload, and RegisterBase. Note that OTP and $AuthTokens$ are generated by W, and stored by both T and W at the end of the Upload procedure, which is also called by the RegisterBase procedure.

We begin with the RecordData procedure, that enables tracker T to commit recorded sensor data to its local memory. The pseudocode of RecordData is shown in Algorithm 1. The organization of the tracker memory is shown in Figure 7.

Algorithm 1: RecordData pseudocode. The one time pad encoding requires only simple bit wise xors and pointer handling operations. The OTP is stored in mem at position M. The cache C (initialized with OTP from mem) is used to store sensor readings. Once C is full, it is written over mem.

```
1. Object implementation Tracker;
2.  mem : bit[];              #tracker memory
3.  C : int;                  #index of cache
4.  cache_size : int;         #size of cache
5.  ptr : int;        #moving index inside cache
7.  M : int;          #index to currentmemory
8.  Operation int RecordData(A : sensordata)
9.      xor mem[ptr], A;
10.     ptr = ptr + A.size;
11.     if (ptr > C + cache_size) then;
12.         mov mem[M], mem[C]; #copy cache to main mem
13.         M = M + cache_size;
14.         mov C, M, cache_size; #copy OTP to cache
15.         ptr = C #move ptr to start of cache
16. fi end
```

RecordData($T(sensor\_data)$). T has available memory mem and cache C. We model C as being part of mem for simplicity, however, C is a L1C cache in Fitbit. M is the current writing index in mem, denoting the beginning of the yet unused bits of the one time pad OTP. Given $sensor\_data$ (A) as input, T xors it into the next available position in the cache C (line 9). If all the space in C has been exhausted (line 11), copy C's content over mem starting with index M (line 12). Then, overwrite the cache C with a yet unused chunk from mem's OTP (line 14). Then, prepare to write the next sensor reading at the beginning of the cache (line 15).

RecordData does not impose a storage overhead on the existing Fitbit solution: the result of the XOR is written into the same amount of memory taken by the standard Fitbit sensor readings. We note that the RecordData procedure above is designed to apply to any sensor readings. In the case of Fitbit, the sensor data consists of step and floor counts. Thus, A stores the daily step (or floor) count values.

We now present the Upload protocol, that enables T to automatically upload its $data$ (sensor readings) to its user U's social network account on W, through one of the bases added with RegisterBase (see next).

Upload($T(id_T, AuthTokens, data), B(), W(Map)$). The following process takes place, building upon the 11 step Fitbit protocol (see Figure 2):

1) T sets up a connection to the base B in its vicinity (following steps 1a and 1b in Figure 2). B sets up an authentic, secure connection with W (extending step 2). According to steps 3 and 4, T sends (id_U, id_T) to B, which relays it to W as (id_U, id_T, id_B).
2) W retrieves $Map[id_U]$, the Map entry corresponding to id_U: $[id_U, id_T, H(P_U), Bases, AuthTokens, OTP]$. If $id_B \notin Bases$ (see RegisterBase), W aborts the protocol. Otherwise, it includes in the TRQ-REQ message (step 5) the server part of the mutual authentication token pair $AuthTokens[W]$. The message is sent over the secure

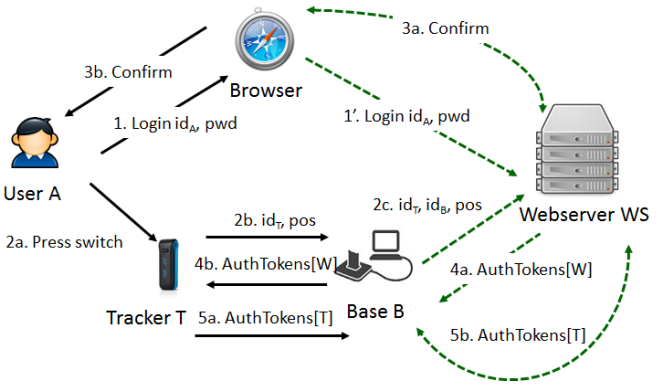


Fig. 8. FitLock RegisterBase procedure. Green dotted lines represent secure (e.g., SSL) connections. The user confirmation (step 3) prevents a DoS attack: attacker impersonates T (in steps 2) and forces W to consume $AuthTokens[W]$. A and W mutually authenticate in steps 4 and 5.

connection with B.

3) B forwards the $AuthTokens[W]$ to T. T verifies the equality of the bits received from B with its local copy of the mutual authentication token. If the verification fails, T aborts the protocol. Otherwise, it sends $AuthTokens[T]$, the tracker part of the mutual authentication token, as well as $data$ to B (step 6b), that forwards it over the secure session to W.
4) W verifies that $AuthTokens[T]$ matches the local copy. If not, it aborts the protocol. Otherwise, W computes $data \oplus OTP$ in order to decode the contents of the $data$ field. It then stores the result in U's social networking account. Steps 7 proceed just like in Fitbit.
5) W creates new $AuthTokens$, and one time pad values and writes them (through B) on T. The WRITE messages rely on and replace the ERASE messages (steps 8). Steps 9-11 proceed as in Fitbit.

The RegisterBase procedure described next and illustrated in Figure 8, enables U to bind a trusted base B to her tracker T and her UPI on W. T can then use B for Upload (see above).

RegisterBase($U(Id_U, P_U), T(id_T), B(id_B), W(Map)$). U, T, B and W execute the following sequence of steps:

1) U logs into her W account, using a secure connection from her browser, using her username and password, (id_U, P_U). W retrieves the entry for $Map[id_U]$: $[id_U, id_T, H(P_U), Bases, AuthTokens, OTP]$. If no such entry exists (the first RegisterBase run by U), W creates one. $Bases$ is an empty set, $AuthTokens$, and OTP are reset to contain 0 bit values.
2) U brings T into RegisterBase mode, e.g. by long pressing the switch. If T discovers no nearby base, it aborts the protocol. T sends id_T to B. B sets up a secure connection to W, and sends id_T and id_B.
3) W notifies the user U through the browser, and asks to confirm the registration of a new base. If U does not confirm, W aborts the protocol.
4) W sends the server part of the mutual authentication token $AuthTokens[W]$ to B, which relays it to T.
5) T verifies that $AuthTokens[W]$ matches the locally stored copy. If the verification fails, T aborts the protocol. Otherwise, T sends to W the tracker part of the mutual authentication tokens, $AuthTokens[T]$.
6) W verifies that the $AuthTokens[T]$ received from T matches its own copy. If the verification fails, W aborts the protocol. Otherwise, W adds id_B to $Bases$. W and T (through B) engage in the Upload procedure, that creates new $AuthTokens$ values.

The first time U runs RegisterBase, there is no $AuthTokens$ mutual authentication token set shared by T and W. Then, both T and W send 0 as the $AuthTokens[]$ bits.

D. Data Consistency Verifications

As mentioned in Section III, there exists a strong relationship between the different activity parameters tracked by Fitbit. However, as demonstrated by the UAI attack in Section IV, Fitbit does not verify the consistency of the data reported by trackers. We propose to address this vulnerability through the following additional verifications. The verifications are performed by W, based on the data reported by tracker T.

- Verify that the number of steps between consecutive records on T does not exceed a maximum value achievable in that interval. The maximum step count can be either generic, or personalized, computed over user data during an initial interval when the tracker is unlikely to be under attack.
- Use the initial, attack-free interval to train time series forecasting tools (e.g., ARIMA, Artificial Neural Networks), predict future activity levels and raise red flags when the recorded values significantly exceed predictions. Ask the user to personally confirm flagged readings.
- Rely on the user's walk/run stride length and BMR values (see Section III) to verify the relations between the number of steps (walking and running) and the distance traversed and the calories burned by the user.

E. Analysis

During the RegisterBase procedure, standard authentication and secure communications protocols are used between B and W. A is authenticated to W through her password P_U. A's confirmation (step 3) is required to prevent a DoS attack: the attacker forces W to reveal $AuthTokens[W]$ (step 4) by impersonating T in steps 2. B is implicitly authenticated by the user, who decides to run RegisterBase in its vicinity. W and T authenticate each other through the one time use $AuthTokens$ pair. The security of the procedure relies thus on the the user running this operation in attacker-free environments: at home, work, or at a friend's house (and base).

An attacker with physical access to a tracker but not to the tracker's memory cannot read the sensor data stored on the device. This is because the attacker does not know W's authentication value $AuthTokens[W]$. The chance of guessing the l bits of $AuthTokens[W]$ is 2^{-l}, e.g., 1 in a million for $l = 20$. By pressing the tracker's switch, the attacker can read the latest sensor readings. The readings are

however anonymized and contain also input from the attacker (that has carried the tracker from the reception to the attack spots). This attack can be prevented by requiring the user to authenticate to her tracker. We have omitted the presentation of a seamless authentication procedure based on a paired mobile device, due to its standard form and space constraints.

A sniffing attacker has no advantage in the Upload procedure (except detectable DoS attacks). The base B is authenticated by T through W: W authenticates B through standard SSL and verifies $id_B \in Bases$ and sends its one time use $AuthTokens[W]$. T authenticates B by its knowledge of W's $AuthTokens[W]$ value. Thus, vicinity to a safe, registered base provides a first level of security. The second level is provided by the fact that the data reported by T has been previously xored with a one time pad OTP. Thus, the data is secure with information theoretic assurance.

Resilience to JTAG attacks. We consider now an attacker powerful enough to steal the tracker, open the case, and use JTAG attacks to read its memory. This includes $AuthTokens[W]$, $AuthTokens[T]$ and either OTP (for memory not yet written) or $data \oplus OTP$ (for previously sampled sensor data). Thus, the attacker cannot read the sensor history of the user: past values are stored xored with random one time pad bits, no longer available on the tracker.

The attacker can attempt to inject fraudulent sensor data – but only after the point in time when it cracked the tracker. If corrupted, old $data \oplus OTP$ values will produce inconsistent values, once decoded with the OTP bits only known by W. Inconsistencies will be detected by the techniques described in Section V-D. Even then, for this attack to be meaningful, the attacker either (i) needs to subsequently return the tracker to its owner, or (ii) bring the tracker in the vicinity of one of the user's trusted bases. Thus, FitLock significantly raises the stakes even for a powerful JTAG attacker with physical access to the victim tracker.

Other Requirements. FitLock is user friendly. The user is explicitly involved only during infrequent RegisterBase procedure runs. FitLock requires only a lightweight tracker, capable of performing xor operations and equality tests. FitLock bases need only be able to setup standard secure SSL connections to W, and forward traffic between T and W. FitLock is sync flexible, allowing users to add bases through which their trackers can automatically sync with W. FitLock also does not impose storage overhead on trackers: sensor data is XORed into place with the memory's one time pad.

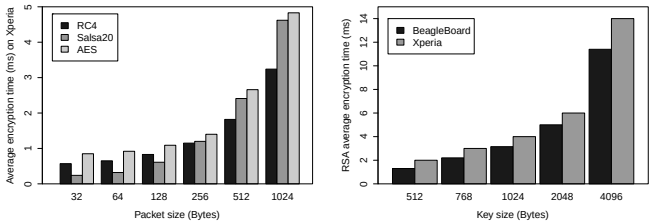
VI. Evaluation

We have contacted Fitbit and reported our results. While interest for the security of users was expressed, Fitbit declined our offer to collaborate in implementing FitLock on the Fitbit platform. We have thus implemented FitLock in Android.

A. Experiment Setup

In the experiments, we have used both a Sony Ericsson Xperia X11 mini smartphone with ANT+ support and a Revision C4 BeagleBoard [25] as trackers, a Dell laptop equipped with a 2.3GHz Intel Core i5 and 4GB of RAM to connect the base, and a 2.4GHz Intel Core i5 Dell laptop with 4GB of RAM for the webserver (built on the Apache web server 2.4).

Fig. 9. Snapshot of testbed for FitLock, consisting of BeagleBoard and Xperia devices used as Fitbit trackers.



(a) (b)

Fig. 10. Encryption overhead: (a) AES, RC4, Salsa20 on Xperia. (b) RSA encryption on Xperia and BeagleBoard.

We implemented a client-server Bluetooth [26] socket communication protocol between the tracker (Xperia smartphone) and the base using PyBluez [27] python library. We used Wi-Fi for the connectivity between the base and the webserver. Figure 9 illustrates our testbed. We report all values as averages taken over at least 10 independent protocol runs.

B. FitCrypt

We have compared the performance of FitLock against Fitbit and also FitCrypt, an alternative secure solution for Fitbit. We have considered two versions of FitCrypt, one using only symmetric encryption solutions to authenticate participants and secure communications, and one relying only on public key cryptography. While more efficient, the fundamental drawback of symmetric key crypto for FitCrypt is that it needs to store the encryption key on the tracker. Thus, this solution is not resilient to JTAG attacks: the attacker can recover the secret key and decrypt both all previously captured and all the future transmissions between the victim tracker and W. We note that a hybrid solution can be devised, using PKC for tracker to webserver communications and symmetric key crypto for webserver to tracker communications.

C. Results

FitCrypt tinker: encryption overheads. We have experimented with the use of symmetric and public key encryption

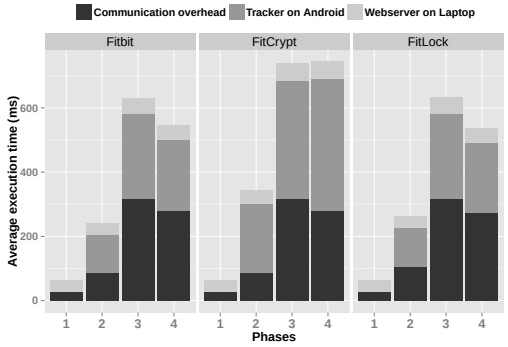


Fig. 11. Comparison of end-to-end delay between the current Fitbit solution and our proposed encrypted solution

algorithms on Xperia and Beagleboard devices. We have first evaluated AES [28], RC4 [29] and the Salsa20 [30] stream cipher, with a key size set to 128 bits and packet sizes ranging from 32 to 1024 bytes (the Fitbit packet size is at most 80 bytes). Figure 10(a) shows the execution time of the three protocols on the Xperia smartphone. For small packet sizes, Salsa20 performs the best. As the packet size increases, RC4 performs slight better than Salsa20. Both RC4 and Salsa20 outperform AES for any packet size. Figure 10(b) compares the encryption overhead of RSA when running on the BeagleBoard and on the Xperia devices, for key sizes ranging from 512 to 4096 bytes while the packet size was set to 1024 bytes. The BeagleBoard performs better due to its more powerful CPU.

End-to-end Upload comparison. We have implemented FitLock, FitCrypt (both symmetric key and public key versions) and the Fitbit protocols in Android and compared their performance using the Xperia X11 tracker. Figure 11 shows the end-to-end time imposed by the Upload procedure of the implemented solutions. The times shown for each solution are divided into the 4 phases of the webserver-to-tracker communication protocol described in Figure 2. We only show the performance of the public key crypto FitCrypt, using a standard 2048 bit modulus. The symmetric key version performs better than the public key version, however, as mentioned above it is not resilient to JTAG attacks.

The total time of Fitbit's Update procedure is 1481ms. The end-to-end (sum over all 4 phases) time of FitCrypt is 1894ms, a 27.9% increase over Fitbit. In contrast, the end-to-end time of FitLock is 1497 ms, thus introducing an overhead of 16ms (just 1.08%) over Fitbit.

VII. Conclusions

In this paper, we show that the wealth of constraints faced by the makers of wearable personal fitness trackers, make the data they collect and communicate, vulnerable to security attacks. Case in point, we reverse engineer Fitbit, a popular fitness tracking solution. We introduce FitBite, a tool for launching eavesdropping, injection and denial of service attacks against Fitbit. We devise FitLock, a general, efficient and secure extension for Fitbit. FitLock relies only on fast bitwise XOR and EQUAL operations, and does not impose a storage overhead on trackers and the webserver. FitLock is resilient even to attackers able to probe the memory of captured trackers and imposes only 1.6% end-to-end delay on Fitbit.

VIII. Acknowledgements

We thank Madhusudan Banik for help on the FitLock implementation.

References

[1] Fitbit. http://fitbit.com/.
[2] Garmin Forerunner. http://sites.garmin.com/forerunner610/.
[3] Wahoo fitness. http://www.wahoofitness.com/.
[4] Jawbone. https://jawbone.com/.
[5] Runkeeper. http://runkeeper.com/.
[6] Jawbone takes a big bite out of health tech: acquires BodyMedia, launches Up app platform. http://venturebeat.com/2013/04/30/jawbone-takes-a-big-bite-out-of-health-tech-acquires-bodymedia-launches-up-app-platform.
[7] Please Rob Me. http://www.http://pleaserobme.com/.
[8] Kota Tsubouchi, Ryoma Kawajiri, and Masamichi Shimosaka. Working-relationship detection from fitbit sensor data. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, UbiComp '13 Adjunct, pages 115–118, 2013.
[9] Cotton Delo. Insurance Giant WellPoint Commits to Facebook With Fitness Tracker. AdAge digital, http://adage.com/article/digital/wellpoint-commits-facebook-fitness-tracker/237774/, 2012.
[10] FitBite and FitLock: Attacks and defenses on Fitbit Tracker. http://users.cis.fiu.edu/~mrahm004/fitlock.
[11] D. Halperin, T. Heydt-Benjamin, B. Ransford, S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 129–142, 2008.
[12] K. B. Rasmussen, C. Castelluccia, T. S. Heydt-Benjamin, and S. Capkun. Proximity-based access control for implantable medical devices. In *ACM Conference on Computer and Communications Security*, pages 410–419, 2009.
[13] Chunxiao Li, A. Raghunathan, and N.K. Jha. Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system. In *13th IEEE International Conference on e-Health Networking Applications and Services (Healthcom)*, pages 150–156, 2011.
[14] S. Lim, T.H. Oh, Y. Choi, and T. Lakshman. Security issues on wireless body area network for remote healthcare monitoring. In *2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC)*, pages 327–332, 2010.
[15] Rajani Muraleedharan and Lisa Ann Osadciw. Secure health monitoring network against denial-of-service attacks using cognitive intelligence. In *CNSR*, pages 165–170, 2008.
[16] J. Newsome, E. Shi, D. Song, and A.Perrig. The sybil attack in sensor networks: Analysis and defenses. In *Third International Symposium on Information Processing in Sensor Networks(IPSN)*, 2004.
[17] C. Karlof and D.Wagner, editors. *Secure Routing in Sensor Networks: Attacks and Countermeasures*, 2003.
[18] Johannes Barnickel, Hakan Karahan, and Ulrike Meyer. Security and privacy for mobile electronic health monitoring and recording systems. In *the 2010 IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6, 2010.
[19] Ramon Marti and Jaime Delgado. Security in a wireless mobile health care system, 2007.
[20] A. J. Hulbert and P. L. Else. Basal Metabolic Rate: History, Composition, Regulation, and Usefulness. *Physiological and Biochemical Zoology*, 77(6):869–876, 2004.
[21] Ing Breeuwsma. Forensic imaging of embedded systems using JTAG (boundary-scan). *Digital Investigation*, 3, 2006.
[22] Libfitbit: Library for accessing and transfering data from the fitbit health device. https://github.com/qdot/libfitbit.
[23] Fitbit Specs. http://www.fitbit.com/one/specs, Last retrieved on October 1st, 2013.
[24] Earndit: We reward you for exercising. http://earndit.com/.
[25] G. Coley. *Beagleboard system reference manual*. BeagleBoard.org, December 2009.
[26] Bluetooth SIG. Specification of the bluetooth system, 2001.
[27] Pybluez. http://code.google.com/p/pybluez/.
[28] Federal Information Processing and Announcing The. Announcing the advanced encryption standard (aes), 2001.
[29] Rc4. http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4.html.
[30] Daniel J. Bernstein. The salsa20 family of stream ciphers. In *New Stream Cipher Designs*, pages 84–97. Springer-Verlag Berlin, Heidelberg, 2008.