Timestamp Integrity in Wearable Healthcare Devices

Muhammad Siddiqi[†], Vijay Sivaraman[†], Sanjay Jha[*]
[†]School of Electrical Engineering and Telecommunications
[*]School of Computer Science and Engineering
[†*]University of New South Wales, Sydney, Australia
{m.siddiqi@student.,vijay@,sanjay@cse.}unsw.edu.au

***Abstract*—Wearable sensors for heart-rate, ECG, blood pressure, and blood glucose are gaining increasing prominence in home-based healthcare. Though the medical sensory data is now routinely encrypted and signed, the timestamp associated with the data, which is needed for accurate correlation and reconstruction of medical events, remains poorly secured. In this paper[1] we first motivate the problem by demonstrating with two medically approved devices on the market today that timestamps can be easily tampered to backfill medical data. We then propose a novel solution that works within the resource constraints of wearable devices to secure timestamps against malicious or ill-configured gateways. Lastly, we evaluate our solution via simulation and experimentation across multiple network access technologies.**

I. Introduction

Healthcare costs are ballooning in much of the western world. Increased life expectancy, sedentary lifestyles, and poor diet are contributing to a larger proportion of our population living for longer periods with chronic conditions like obesity, heart disease, and diabetes. This is increasing the burden on primary care delivery, and exacerbating acute events requiring hospital care. A significant opportunity exists to curtail these growing demands on our healthcare system by engaging patients in at-home medical management by using emerging wireless sensor technology.

Wearable sensors range from being fitness-based ones for cardio and calorie monitoring (e.g. Fitbit Flex, Jawbone UP, etc.) to medically-approved monitors for ECG (AliveCor Heart Monitor), respiration rate (Toumaz Sensium Plaster), and blood pressure (Fora Diamond Cuff BP). Major technology companies are teaming up with pharmaceutical and healthcare companies to develop medical monitoring platforms and provide remote healthcare to patients – examples include Google's partnership with Novartis and prototype of the "smart contact lens" for wireless glucose monitoring [1], Apple's partnership with Mayo Clinic [2] and development of its HealthKit platform for the Apple Watch [3], and Samsung's healthcare platform called SAMI for its Galaxy series of smartphones [4].

While wearable devices can collect vast amounts of data on the evolution of chronic conditions and the effectiveness of treatments, significant research is needed to increase confidence and trust in the data thus collected. Our previous work has developed mechanisms for low-complexity encryption and digital signatures of the medical data [5], [6] by the wearable device – while these mechanisms help secure the data itself, they do not secure the associated *meta-data* or *context*, such as the time at which the data was collected, which is the focus of this work.

[1]This work is funded by Australian Research Council's Discovery Grant DP150100564.

Correct timestamping of medical data is vital for medical practitioners and insurance providers. For a patient who has had a heart attack, the doctor reading the ECG needs to know the heart rate and blood pressure immediately before and after the event, at sub-second time-scales [7], in order to identify potential causes and consequences. Similarly, a medical insurance company evaluating a medical claim needs to know the timing of events, including whether the patient is in compliance with terms (e.g. routines regarding exercise, smoking, and diet) that can affect insurance pricing [8]. Timestamps of medical data from wearable devices are not reliable today – they often operate on low-accuracy or misconfigured clocks, synchronize over variable-latency Internet paths, and are susceptible to being tampered by users who have incentive to do so.

At first thought timestamping may seem trivial to achieve – either the wearable device could timestamp each data item as it generates it, or the medical server could timestamp it as it receives it. The problem with the first approach is that the wearable device, being extremely resource constrained, neither has an accurate and reliable on-board clock, nor can it run a sophisticated protocol (such as NTP) to bootstrap and synchronize its on-board clock. The second approach, whereby the receiving server timestamps the data, is not only susceptible to unpredictable Internet latencies, but also fails to work when the wearable device or the gateway (smartphone) uploads data not in real-time but in bursts (indeed many wearable devices like Fitbit work in this way). There is a third alternative, whereby the timestamping is done by the smartphone, but this approach makes the timestamps susceptible to tampering by the user, as we will demonstrate shortly.

Given the above challenges, our objective in this paper is to develop and evaluate a mechanism that assures reliable timestamping of medical data. Our first contribution is to show, using medically approved devices on the market today, that timestamps are insecure and can be tampered easily. Our second contribution is to develop a scheme that ensures reliability of timestamps of the medical data; our scheme operates within the tight resource constraints of wearable devices, and accommodates variable latencies over Internet paths. Our third contribution is to evaluate our scheme via simulation and implementation to quantify the reliability of timestamps, in the face of various threats from the insecure components in the system.

The rest of the paper is organised as follows: §II reviews related prior work in this area, while in §III we demonstrate that today's medical devices are insecure against tampering of timestamps. In §IV we develop our protocol for assuring correctness of timestamps of the sensor data, while in §V we evaluate the performance of our protocol via analysis and simulation, and the paper is concluded in §VI.

II. Related Work

Broadly, existing schemes for timestamping either send the data to a trusted third-party to timestamp, or try to synchronize clocks across all relevant entities that record the timestamps. Seminal works in the former category include [9], [10]. These schemes require a centralized trusted timestamp authority that provides the absolute timestamp on request. Adams *et al.* [11] introduced the idea of additional entities called Temporal Data Authorities (TDAs) that add unpredictable timing information (such as stock exchange or match score) to the absolute time to distribute the trust rather than relying on a single entity. A decentralised timestamping scheme with distributed storage, called "broadcast and save", was proposed by [12] in which user sends the hash of the data to all other users that save it with current time appended. In these schemes, trusted third party time servers are required to timestamp the data. It requires Internet connectivity, which sensors do not usually have and gateways (smartphones) might not have at all times. Moreover, frequent timestamping is not possible as it involves power consuming public key decryption operations. Unpredictable network delays also make these schemes inaccurate and imprecise. For these reasons, conventional digital timestamping protocols cannot be adopted for sensor networks.

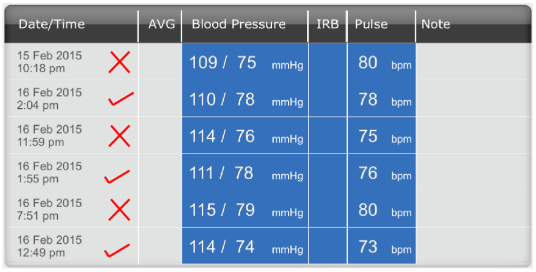
Synchronising wireless sensor clocks has been studied deeply in the past two decades. Elson *et al.* proposed *Reference Broadcast Scheme (RBS)* [13], which synchronizes receiver with receiver instead of sender to receiver thus improving precision by removing uncertainty related to sender. *Gradient Time synchronization Protocol (GTSP)* [14] is a distributed time synchronization protocol where nodes periodically send synchronization beacons to peers and try to agree on a common logical time. These two protocols, RBS and GTSP, are not suitable for the problem in hand where we need to secure absolute timestamps rather than relative timestamps of the sensor data. *Timing-Sync Protocol for Sensor Networks (TPSN)* [15] is based on sender-receiver synchronization and uses tree-based architecture in which the synchronization process is triggered by the root node and the first hop nodes synchronize with the root node by two-way messaging between them. Subsequent nodes get synchronized to first hop nodes in the same fashion and so on until all the nodes get synchronized. *Flooded Time Sync Protocol (FTSP)* [16] is similar to TPSN in that all the nodes are synchronized to the root node. Root node is elected periodically and broadcasts the timing information in a single message to the nodes within broadcast domain. TPSN or FTSP, which allow absolute time to be taken from root node, can be used to get the wearable device synchronized to the smartphone time if smartphone time is to be used.

III. Attacking Timestamps: A Demonstration

Users of wearable medical devices may have incentives to tamper with their records. Indeed, studies show that about half of the patients lie to their doctors about their medical history, habits, and symptoms [17]. By tampering their records, users can obtain physical benefits like sick leave, as also monetary benefits like lower insurance premium. For example, a patient who is enjoying lower insurance premium in exchange for daily exercise may have an incentive to backfill data from his activity tracker for a period of missed exercise. We demonstrate next that these threats are real, not just hypothetical.

A. Tampering with the Clock

We procured a smart blood pressure monitor, called "Fora Diamond Cuff BP", manufactured by ForaCare. It is worn on the arm, and transmits the blood-pressure measurements wirelessly via Bluetooth to the smartphone. The App timestamps the reading and uploads to their server, where the user and medical practitioner can view it online.



Date/Time		AVG	Blood Pressure	IRB	Pulse	Note
15 Feb 2015 10:18 pm	✗		109 / 75 mmHg		80 bpm	
16 Feb 2015 2:04 pm	✓		110 / 78 mmHg		78 bpm	
16 Feb 2015 11:59 pm	✗		114 / 76 mmHg		75 bpm	
16 Feb 2015 1:55 pm	✓		111 / 78 mmHg		76 bpm	
16 Feb 2015 7:51 pm	✗		115 / 79 mmHg		80 bpm	
16 Feb 2015 12:49 pm	✓		114 / 74 mmHg		73 bpm	

Fig. 1. Blood pressure readings on iFora BP App taken between 12:49pm to 2:04pm on 16 Feb

We were able to tamper the timestamps of the measurements simply by changing the time on the smartphone clock before measuring the blood pressure. To illustrate this, we took several blood-pressure measurements within a time window (12:49pm to 2:04pm) on a specific day (16 Feb), but changed the smartphone clock settings between readings. As shown in Fig. 1, the App and the online server report the readings with the doctored time: rows 1, 3, and 5 show the readings as if they were taken outside of the measurement window. This lack of protection of timestamp, even in a medically approved device, shows the ease with which a user can backfill their missing readings.

The above behavior is not unique to the device considered above (which does not have an internal clock and relies on the smartphone to timestamp its data), but also to devices which have their own clock. For example, we experimented with the Withings Pulse O_2 activity tracker, which has an internal clock, but still needs to sync its clock with the smartphone to get the absolute time (much like Fitbit). We verified that changing the smartphone clock between the synchronizations allowed us at will to change the timestamp of the medical data.

B. *Tampering with the App*

One could argue that the problem above can be overcome by having the App maintain its own time rather than relying on the phone's clock setting. We show that a dedicated user can still circumvent this by tampering with the App itself. There are a host of tools (Apktool, Androguard, Cydia, Clutch, Hopper, iRET) that can decompile legitimate Apps, both for Android and iOS platforms (indeed, Santoku linux provides many of such tools built in for mobile forensics), and these tampered Apps can then be recompiled and run on the smartphone.

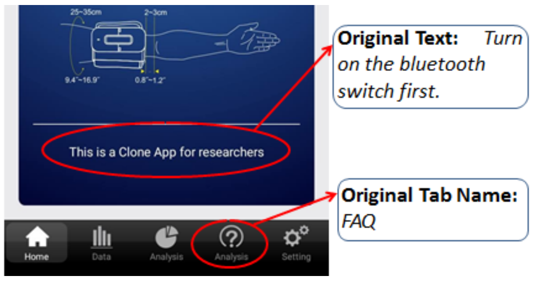

25~35cm
2~3cm
9.4"~16.9"
0.8"~1.2"
This is a Clone App for researchers
Original Text: *Turn on the bluetooth switch first.*
Original Tab Name: *FAQ*
Home
Data
Analysis
Analysis
Setting

Fig. 2. Home screen (cropped) of clone Android App of iFora BP

To illustrate the feasibility of this approach, we used Apktool to decompile the Apps supporting the Fora and Withings devices above (inspired from [18]); we were then able to make modifications to the Apps, and recompile them for the Android smartphone with appropriate digital signatures (uploaded online [19][20]). In Fig. 2 we show how the front-end of our clone App differs from the original, as proof that our App installs and runs on the smartphone. Further, our doctored App is transparent to both the medical device and to the server. In other words, the device still uploads data to the App and the server still accepts data from the App as before. This demonstrates that a motivated user can doctor the App on their smartphone to change the clock value used by the App, which in turn falsifies the timestamp associated with the medical data flowing from the wearable device to the server.

IV. Our Protocol to Secure Timestamps

A. *Challenges*

It is important that timestamping is performed in such a way that it is nearly impossible for any potential malicious party to fudge the time when data is produced. Such guarantee is vital for the success of remote healthcare technology, however, there are challenges to achieve this goal. There are three places in the network where data can be timestamped: 1) Server 2) Sensor and 3) Smartphone; however, none of these by itself is ideal as explained below.

1) Timestamping data at server: Servers are far removed from medical devices. When data reaches healthcare servers over the Internet, it is subject to unpredictable Internet delays and loses the accuracy of time. Many wearable devices and/or their smartphone Apps upload data not in real-time but in bursts (Fitbit, Withings Pulse O_2). Moreover, data can be held back by the smartphone with malicious intent and released after a certain period of time, which makes timestamping at the server erroneous.

2) Timestamping data at sensor: Sensors are usually equipped with hardware oscillators to provide logical time but do not have any reference to the absolute time. They usually do not have direct access to the Internet and rather communicate with smartphone through low power consuming wireless technologies like Bluetooth. The resource constrained nature of sensors prohibits the implementation of traditional cryptographic absolute timestamping techniques. Sensors can get their clocks synchronized to the smartphone, however, time provided by the smartphone could be wrong making it untrusted. Moreover, the hardware clocks drift away from the actual value over time due to inherent and environmental factors. As a result, sensor devices must keep getting synchronized to an external reliable source of time after certain period of time to prevent excessive drift.

3) Timestamping data at smartphone: According to a research report by Arxan Technologies, 97% of top 100 paid Android Apps and 87% of top paid iOS Apps have been hacked [21]. Therefore, even if the smartphone App manages a software clock and uses an Internet protocol like NTP to synchronize it to the absolute time, the App is vulnerable to hacking [22], as we have shown in §III as well. The hacker can tamper the clocking mechanism of the smartphone that makes smartphone unreliable for timestamping.

B. *Design decisions*

Having established that none of the single entity above can solely be responsible for timestamping medical data, we mandate the following roles of the three entities.

The **sensor** node performs the actual timestamping, for two reasons: (a) we want high accuracy, in that timestamping should be performed at or very close to where the data originates; indeed prior studies have required sub-second accuracy in diagnosis of medical conditions such as cardiac events and neurological conditions [23]; (b) we want the timestamp to be digitally signed by the sensor so it is non-repudiable and cannot be fudged by any other entity, including the smartphone.

The **smartphone** bootstraps the absolute timestamp on the sensor, since it can run Internet protocols like NTP to synchronize time accurately, and also because it is proximitous to the sensor. However, this time given by the smartphone to the sensor may be untrusted and will be verified by the server, as described next.

The **server**, henceforth referred to as the Time Inspector, verifies the validity of the timestamp by challenging the sensor periodically; it does not assign timestamps, rather just checks the timestamp. This allows the verification role to be optional; the system design is modular so that the timestamp validation can be excluded if the application does not require it.

The detailed operation of the protocol is described next.

C. *Protocol to validate time of the sensor*

Our proposed protocol is shown in Fig. 3 and operates as follows. When a patient registers his wearable sensor device with the authentication server from the healthcare provider,

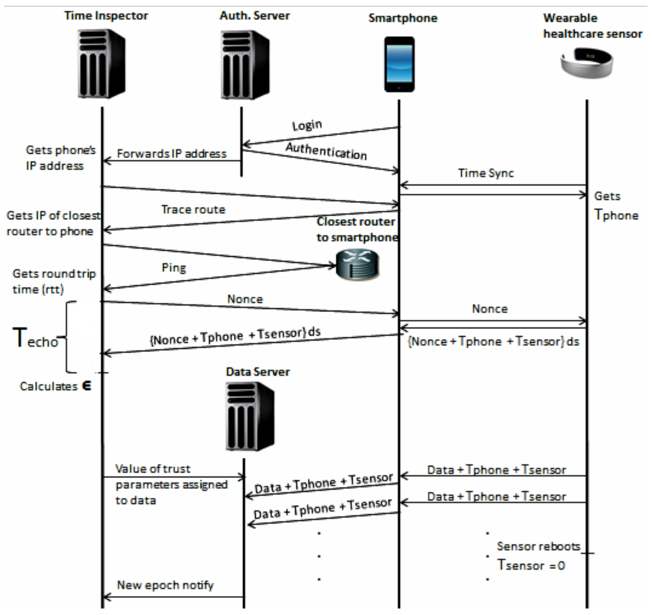


Fig. 3. Flow diagram of the protocol to validate time of the sensor

the smartphone's IP address is notified to the Time Inspector, so that the timestamp verification process may commence. When the patient pairs his medical device with the phone, the wearable device gets the absolute time from the smartphone. This time T_{phone} acts as the baseline reference for an absolute time from which the wearable device can count time using its internal relative clock T_{sensor}. Periodically, the Time Inspector validates the clock value $\{T_{phone} + T_{sensor}\}$ at the sensor node. In order to do so, it determines the network latency by conducting multiple *pings* to the penultimate hop router to the smartphone, determined by using a *traceroute* {note that the Time Inspector does not trust the smartphone and hence does not ping it directly}. Denote by μ_{rtt} the mean round-trip-time to the penultimate hop router. The latency between the smartphone and the penultimate hop router can be neglected for being too small. The Time Inspector now verifies the time of the sensor by sending a $nonce$ to the sensor, which the sensor responds to by concatenating the nonce with its local absolute time $\{nonce+T_{phone}+T_{sensor}\}$, digitally signed. The $nonce$ ensures the smartphone does not replay any previously signed messages from the sensor. Upon receiving the $nonce$ reply, the server calculates the time error ϵ as:

$$\epsilon = T_{server} - (T_{phone} + T_{sensor} + \mu_{rtt}/2) \tag{1}$$

where $\mu_{rtt}/2$ is an approximation of the network delay between the server and the smartphone.

ϵ captures the residual offset between the server and sensor's clocks after accounting for network latency, and will be used to calculate timestamp trust parameters as described next.

D. Trust parameters of the timestamps

The Time Inspector periodically sends $nonce$ challenges, and computes the mean μ_ϵ and standard deviation σ_ϵ over successive time errors ϵ. These parameters (μ_ϵ,σ_ϵ) are called trust parameters and help us determine the extent to which timestamps are skewed. Defining an "epoch" as the period between two time synchronizations, the Time Inspector calculates the trust parameters for each epoch. Medical data generated during an epoch is associated with the timestamp trust parameters for that epoch. Each time the sensor syncs its time with the smartphone, that time needs to be verified as the malicious smartphone might give a wrong time to the sensor. A μ_ϵ (offset) that is close to zero in an epoch indicates that the sensor has a right global time for that epoch. A large value of μ_ϵ indicates one of two things: 1) The smartphone has a constant offset from the server time; or 2) The smartphone has given a wrong time to the sensor with either malicious intent or out of negligence. A constant offset can be verified by geographical location of the smartphone or by looking at the offsets in multiple epochs to see if they are consistent or not. A large value of σ_ϵ (distrust) reduces the trust on validity of timestamps.

E. Effect of sensor reboot

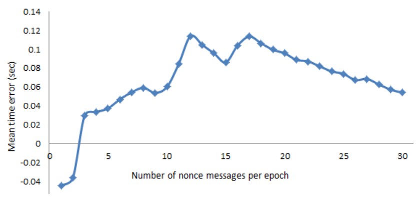
Each time the sensor synchronizes its time to the smartphone, it updates the base value of its global time T_{phone}. When the sensor reboots, T_{sensor} resets to zero and a new value of T_{phone} is fetched from the smartphone. The datalog server, which receives timestamped data, can detect a reboot whenever T_{sensor} resets without expected addition in the base value T_{phone}. In which case, the data server sends a message to the Time Inspector to check the time of the sensor as it is the start of a new epoch.

F. Periodicity of challenges

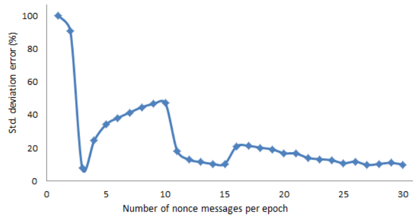
In this section, we discuss how the periodicity of the $nonce$ probing should be adjusted depending on the frequency with which the sensor synchronizes its clock with the smartphone. Hardware clocks within sensors drift with time and are known and specified by the manufacturers. A device manufacturer may choose any frequency mode to sync sensor's time with the smartphone depending upon the drift rate and the requirement of the application. The $nonce$ messages must be distributed evenly in time within an epoch, rather than all at the start, in order to detect any drift in a sensor's clock.

If a sensor synchronizes its time **periodically**, it forms epochs of equal lengths. Each epoch can be distinguished by a unique base value of time T_{phone}. In this case, on-going periodic challenges are sent with a time period *N* times the period of clock synchronization, where *N* is the number of challenges per epoch. The values of trust parameters (μ_ϵ,σ_ϵ) are calculated and sent to the data server at the end of the epoch to be associated with the data.

If a sensor gets its time synchronized only **once** when it boots up, and never syncs again unless it reboots, only one epoch is formed. In this case, the sensor's clock will keep drifting away from the actual value of absolute time, which results in the change of mean time error μ_ϵ with half the rate of drift (challenges evenly distributed). A few challenges evenly distributed over a significant interval of time are sufficient to validate time and determine the drift rate. σ_ϵ is not affected by the drift due to the absence of randomness.



(a) Mean time error μ_ϵ



(b) σ_ϵ error (%)

Fig. 4. Error in trust parameters vs number of nonce messages per epoch

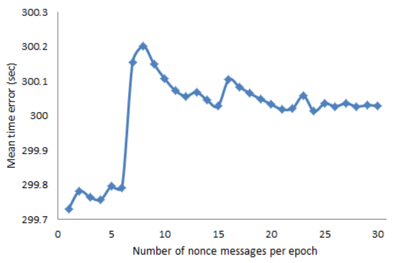


Fig. 5. Catching the time offset of sensor at server

If a sensor gets its time synchronized **non-periodically** e.g. whenever the user syncs its device with the smartphone (as in Withings Pulse O_2), it forms epochs of unequal lengths. Here we assume that when the device is synchronized with the smartphone, the smartphone uploads the data to the data server right away, which is true in case of Withings Pulse O_2. In this mode, as soon as the data server gets the data from a sensor, it sends a message to the Time Inspector to notify the start of a new epoch. On-going periodic challenges with an appropriate time period (depending upon the power constraints and requirement of the application) are recommended in this mode.

V. Evaluation of the Protocol

As argued in §III, the attacker could be the patient himself who wears the sensor device and owns the smartphone. Moreover, smartphone is the one that provides global time to the sensor device in our protocol as well as in most of the medical devices in the market today, which is evident from our attacks on Fora and Withings devices. The attacker cannot change the contents of the challenge response messages from the sensor as they are digitally signed by the sensor, however, he can perform the following attacks through the smartphone:

1) Smartphone may give the wrong time to the sensor. (**Attack-1**)
2) Smartphone may choose not to forward the challenge messages from server to the sensor or vice versa thus defying the inspection protocol. (**Attack-2**)
3) Smartphone may give a future time to the sensor and hold the challenge or challenge response message until that future time to pretend he is further away from the server. (**Attack-3**)

In this section we demonstrate how the server detects the above mentioned attacks and discuss how the number of $nonce$ messages in an epoch affects the performance of our protocol. Intuitively, increasing the number of $nonce$ messages in an epoch should increase accuracy of the trust parameters (μ_ϵ,σ_ϵ). On the other hand, more $nonce$ messages in an epoch will consume more sensor power as it involves digital signatures. For example, a telosB mote consumes 10mJ energy to perform ECDSA algorithm [24]. Energy consumption increases linearly with the increase of number of $nonce$ messages, e.g. 100mJ for ten $nonce$ messages.

To evaluate our protocol, we implemented the Time Inspector (server) and the client modules (smartphone) simulating attacks in Python (uploaded online [25]). Evaluation of our protocol under no attack (reference) and the attacks mentioned above is presented next.

A. No Attack

To establish the baseline for our protocol, we developed a smartphone module named **TrustedClient**, which obtained its time $\{T_{phone} + T_{sensor}\}$ using NTP. Similarly the server obtained its time T_{server} using NTP. We noticed that the discrepancy between the two times was up to 100ms. The server Time Inspector and smartphone modules were running in different cities. We measured the network latency between server and client modules to be $\mu_{rtt} = 350ms$, $max(rtt) = 810ms$, and a standard deviation of $\sigma_{rtt} = 250ms$, which was calculated by running a *ping* command 1000 times from the client to the server.

We executed the protocol in which server calculated time error ϵ (Eq. 1) and the running average μ_ϵ each time it received a $nonce$ reply from the client. We plotted the running average μ_ϵ as a function of number of $nonce$ messages in an epoch (Fig. 4(a)). We also plotted the standard deviation σ_ϵ error (%) from the pre-calculated value (125ms = $\sigma_{rtt}/2$, standard deviation for one way trip) (Fig. 4(b)). From the plots in Fig. 4, it can be noted that increasing the number of $nonce$ messages brings the mean time error μ_ϵ closer to the actual time difference between server's time and sensor's time (approx. zero in this case). Also, it brings the standard deviation of the time error σ_ϵ closer to the actual value (125ms in this case) as well. This is an expected behaviour due to **unpredictable network latency** and we conclude that more number of challenges in an epoch result in more accurate values of trust parameters (μ_ϵ,σ_ϵ), however, energy constraints would not allow us to increase that number beyond limits (depending on the device). Plots in Fig. 4 show that, in this particular instance, mean time error μ_ϵ remained quite stable (within 120ms accuracy) even with a few number of $nonce$ messages per epoch while σ_ϵ error stabilized (under 20%) after 20 $nonce$ messages.

B. Attack-1: Giving Wrong Time

This is the most significant and likely attack. To evaluate the performance of the protocol under this attack, we set smartphone module's time 300 seconds (5 minutes) behind the global time and named this module **OffsetClient**. We executed the protocol and the server calculated the mean time error μ_ϵ against the number of $nonce$ messages. From the plot in Fig. 5, it is evident that μ_ϵ is converging to 300 seconds. The behaviour of μ_ϵ remained same as in Fig. 4(b), as expected. This shows that the server can catch the **offset** of the client whether due to negligence or **malicious** intent.

C. Attack-2: Defying Inspection

To study this attack, we developed a smartphone module named **DropClient** that dropped the challenge messages from the server. As in our protocol, we have set the challenge response to be timed out in T_{echo} seconds (calculated based on the value of $max(rtt)$), the server found all the challenges in an epoch timed out. Failure of $nonce$ response to come back in time T_{echo} could be either due to network or malicious user. The Time Inspector can query the data server to check whether the medical data is being received or not. In the later case, the Time Inspector flags the client as malicious with **drop** tag. Although, the value of μ_ϵ (offset) in this case is found zero, the server sets the value of μ_ϵ (distrust) very high (infinity).

D. Attack-3: Disguising Locality

To study this attack, we developed a smartphone module named **DelayClient** that simulated attack-3 in which it gave a future value to the sensor and held back the response from sensor to that future time to pretend it was further away from the server. We executed the protocol and the server found the challenge response messages being received after the timeout value T_{echo}. Although the time error values ϵ (Eq. 1) calculated by server represented no significant offset, the measurements could not be trusted due to late arrival of the challenge response messages. The server flags such client as malicious with **delay** tag. The value of μ_ϵ (offset) in this case is close to zero, however, the server sets the value of μ_ϵ (distrust) relatively high.

E. Sensor's clock drift

To study the role of sensor's **clock drift**, we set clock drift in the client module **DriftClient** to be "-1 sec/hour" (a decrease of 1 second per hour). A number of $nonce$ messages were sent in an epoch at different times. We plotted the time error ϵ against the epoch time expressed in hours (Fig. 6). The drift δ

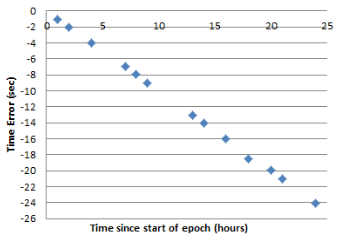


Fig. 6. Catching the clock drift of sensor at server

can be approximated by averaging out the drifts between two $nonce$ messages using the following formula.

$$\delta = \frac{1}{N-1}\sum_{i=2}^{N}\frac{\epsilon_i - \epsilon_{i-1}}{T_i - T_{i-1}} \quad (2)$$

where N is total number of successful $nonce$ messages and T_i is the time of ith $nonce$ since the epoch. We found $\delta \approx -1sec/hour$ as expected. In the case of one-off synchronization, the sensor's clock drift is the dominating factor in causing the mean time error μ_ϵ to increase. The server binds the drift rate with the timestamps of data.

VI. Conclusion

In this paper we have discussed the importance of accurate timestamping of data from wearable healthcare devices. We have shown that the mechanisms these devices employ to timestamp their data are susceptible to malicious attacks. An overlay solution has been developed, which verifies that the time used to timestamp data from these devices is precise and has not been altered maliciously. This can be used to increase confidence in timestamps associated with medical data. Our evaluation has quantified the trade-off between the reliability and energy cost of the protocol and has shown how well timestamps can be validated. We believe this is a step towards securing the time-context of data from wearable healthcare devices.

References

[1] "Novartis to license Google "smart lens" technology," http://goo.gl/ygXAmX, Jul. 2014, [Online; accessed 12-Feb-2015].
[2] "Apple partner Mayo Clinic to reportedly demo HealthKit integration at media event," http://goo.gl/e2JKm6, Dec. 2014, [Online; 12-02-15].
[3] "Apple Watch," https://www.apple.com/watch/, 2015.
[4] R. Lawler, "Samsung Opens Simband Applications And SAMI Data Exchange Platform To Developers," http://goo.gl/ZAE2M9, Nov. 2014, [Online; accessed 13-Feb-2015].
[5] S. T. Ali *et al.*, "Eliminating reconciliation cost in secret key generation for body-worn health monitoring devices," *IEEE Trans. Mob. Comput.*, 2014.
[6] ——, "Authentication of lossy data in body-sensor networks for cloud-based healthcare monitoring," *FGCS*, vol. 35, pp. 80–90, 2014.
[7] "Overview of online symptom checkers: the basic things you need to know," http://goo.gl/HD0dYv, [Online; accessed 21-08-2015].
[8] Steve Lohr, "Carrots, Sticks and Lower Premiums," http://goo.gl/7o42SO, [accessed: 02-03-15].
[9] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," *Journal of Cryptology*, vol. 3, 1991.
[10] F. Pinto and V. Freitas, "Digital time-stamping to support non repudiation in electronic communications," in *the SECURICOM, Paris*, 1996.
[11] Adams *et al.*, "Internet x.509 public key infrastructure time stamp protocols," *IETF*, 1999.
[12] J. Benaloh and M. de Mare, "Efficient broadcast time-stamping," Tech. Rep., 1991.
[13] J. Elson *et al.*, "Fine-grained network time synchronization using reference broadcasts," in *Proceedings of the 5th Symposium on OSDI*, 2002.
[14] P. Sommer and R. Wattenhofer, "Gradient clock synchronization in wireless sensor networks," in *International Conference on IPSN*, 2009.
[15] S. Ganeriwal *et al.*, "Timing-sync protocol for sensor networks," in *ENSS*, 2003.
[16] M. Maróti *et al.*, "The flooding time synchronization protocol," in *ENSS*, 2004.
[17] Kevin McCarthy, "Study: 50 percent of patients withhold information from their doctor," http://goo.gl/6E0w2U, [Online; accessed 21-10-15].
[18] Pau Oliva Fora, "Beginners guide to reverse engineering android Apps," http://goo.gl/xeWbrM, Feb. 2014, [Online; accessed 17-May-2015].
[19] "A modified clone app of iFora BP for research purposes," https://github.com/sidz81/iFora-BP.
[20] "A clone app of Withings for research purposes," https://github.com/sidz81/Withings, Jun. 2015.
[21] Arxan Technologies, "State of Mobile App Security. Apps Under Attack," http://goo.gl/eB5Es5, Nov. 2014, [Online; accessed 27-Feb-2015].
[22] "Gmail smartphone app hacked by researchers," http://www.bbc.com/news/technology-28895304, 2014, [Online; accessed 17-Feb-2015].
[23] A. Zani and A. M. Proverbio, in *The Cognitive Electrophysiology of Mind and Brain*. San Diego: Academic Press, 2003.
[24] Arpit and A. Kumar, "Analysis of optimized elliptic cryptographic protocol on resource poor tiny node," in *ICSCA, Singapore*, 2011.
[25] "Time Inspector and Client modules in Python," https://github.com/sidz81/TimeInspector-Server_Clients.