Mobile Netw Appl (2014) 19:331–344
DOI 10.1007/s11036-013-0447-x

Hide-n-Sense: Preserving Privacy Efficiently in Wireless mHealth

Shrirang Mare · Jacob Sorber · Minho Shin · Cory Cornelius · David Kotz

Published online: 7 June 2013

Abstract As healthcare in many countries faces an aging population and rising costs, mobile sensing technologies promise a new opportunity. Using mobile health (mHealth) sensing, which uses medical sensors to collect data about the patients, and mobile phones to act as a gateway between sensors and electronic health record systems, caregivers can continuously monitor the patients and deliver better care. Furthermore, individuals can become better engaged in monitoring and managing their own health. Although some work on mHealth sensing has addressed security, achieving strong privacy for low-power sensors remains a challenge. We make three contributions. First, we propose an mHealth sensing protocol that provides strong security and privacy properties at the link layer, with low energy overhead, suitable for low-power sensors. The protocol uses three novel techniques: adaptive security, to dynamically modify transmission overhead; MAC striping, to make forgery difficult even for small-sized Message Authentication Codes; and asymmetric resource requirements, in recognition of the limited resources in tiny mHealth sensors. Second, we demonstrate its feasibility by implementing a prototype on a Chronos wrist device, and evaluating it experimentally. Third, we provide a security, privacy, and energy analysis of our system.

Keywords Network protocols · Energy efficient protocols · Privacy · mHealth

1 Introduction

As healthcare systems struggle worldwide to cope with increasing demand (due to an aging population) and economic pressures (due to rising costs), many countries are seeking new models of healthcare. Recent improvements in mobile computing and developments in miniature medical sensors have enabled mobile health (mHealth) sensing, which promises to simultaneously reduce cost and improve the quality of healthcare [19]. In an mHealth sensing system, patients wear or carry one or more sensing devices, and their mobile phone acts as a gateway between the sensors and a repository that makes the data accessible to patients or their caregivers.

An mHealth sensing system can monitor a patient's health continually and unobtrusively without interfering with daily activities, providing timely reports about medical conditions, in situ monitoring by healthcare providers, and fewer re-hospitalizations [7]. These systems promise to improve long-term care for the chronically ill [1] and senior citizens [2], to change unhealthy behaviors [5], and even to improve athletic performance [9]. (Although the monitored individual in these applications may be a resident of an assisted-living facility, a family member, an athlete, or simply yourself, for simplicity in this paper we always refer to the subject of sensing as the "patient"). The security and privacy challenges are largely the same, at the point of sensing;

S. Mare · C. Cornelius · D. Kotz
Computer Science Department, Dartmouth College, Hanover, USA

J. Sorber
School of Computing, Clemson University, Clemson, SC, USA

M. Shin (✉)
Department of Computer Engineering, Myongji University, Yongin, Gyonggido, South Korea
e-mail: shinminho@gmail.com

in this paper we focus on the *mobile* edge of the mHealth ecosystem and thus the solutions proposed in this paper apply equally well to all above settings. Given the inherent sensitivity of health-related data, an mHealth sensing system should have mechanisms to preserve patient privacy and maintain data integrity.

To achieve the necessary security and privacy for mHealth requires one to solve four problems: securing the data in the sensor nodes, securing the data during communication between the sensors and the mobile phone, securing the data (and processing) inside the mobile phone, and securing the data inside the back-end servers. We address the third problem in another paper [22], we assume tamper-resistant sensor nodes, and expect the servers to be secured by other conventional solutions.

In this paper we address the second problem at the link layer, and seek a body-area mHealth network (BAHN) protocol that provides privacy and security efficiently. To provide privacy we mask node identifiers and encrypt all headers. To improve efficiency we reduce packet overhead, which introduces integrity risks; our paper shows how to do so with reasonable integrity guarantees so that ultimately we get an efficient secure and privacy-preserving protocol. Others have proposed protocols for Wi-Fi networks that provide secure and private communication [3, 10, and their references], and we take inspiration from some of the earlier work, using some of their techniques. Among these, SlyFi [10] is the state-of-the-art privacy preserving protocol for Wi-Fi, but it is poorly suited to the needs of low-power mHealth networks because of its large packet transmission overhead. In Wi-Fi networks with large packet sizes, these packet transmission overheads are insignificant; however, when applied to mHealth networks with much smaller packets, we show that the extra transmissions can consume enough energy to reduce a sensor's battery life by more than 50 %, even at low data rates. SlyFi also requires all nodes to have real-time clocks and to stay reasonably synchronized, but not all sensors have real-time clocks, and participating in a time-synchronization protocol requires spending energy on extra wireless transmissions.

In this paper we make three contributions. First, we propose Hide-n-Sense (HnS), a wireless protocol that provides strong security and privacy properties at link layer for mHealth sensing with low energy overhead for the sensors. We use three novel techniques to improve energy efficiency: (1) adaptive protocol overhead to balance security and efficiency, (2) MAC striping to make the protocol strongly resistant against selective forgery for small-sized message authentication codes (MACs), and (3) asymmetric energy requirements among the communicating devices. We demonstrate these techniques using an adaptation of SlyFi; however, these techniques can be applied to other protocols to make them energy-efficient while maintaining their security and privacy properties. Second, we implemented a prototype of the HnS protocol on TI's eZ430 Chronos wireless device, and we demonstrate experimentally that the protocol is feasible on low-powered devices. Third, we provide a security, privacy, and energy analysis of our protocol.

2 Security model

In this section we give an overview of the system architecture, define our threat model and adversary model, identify our security goals, and list trust assumptions.

2.1 System model

Whether for remote-patient monitoring or for personal health management, in a typical mHealth-sensing system the patient carries some sort of *mobile node* (MN), most likely a mobile phone, that acts as a gateway between a body-area network of *sensor nodes* (SN) and the Internet. Although an MN may include embedded sensors, in the context of this paper we are concerned with external sensors used/worn by the patient. The SNs collect data at the instruction of the MN, and send the data to the MN, which may process or aggregate the information before presenting it to the patient (locally) or forwarding it via the Internet to a remote back-end server for use by a remote consumer (health provider, caregiver, researcher) or for their own later use.

As context for our design, and the basis of our security analysis in Section 4, we first detail our underlying assumptions.

System assumptions. We make the following assumptions about hardware and software capabilities of the MN and SN.

S1. *Crypto.* Each SN has the cryptographic capabilities needed for message confidentiality and authenticity; that is, each SN has enough resources to support cryptographic primitives (e.g., AES, SHA-1) either in software or hardware.

S2. *Platform.* The MN is a general-purpose mobile platform, such as a smart phone, with a short-range wireless interface for body-area communication with the SNs (e.g., Bluetooth or Zigbee) and an optional long-range wireless interface for Internet communication (e.g., Wi-Fi or 3G).

S3. *Out-of-band (OOB) channel.* There exists a secure channel between the MN and each SN, which can be used for exchanging secret keys during the pairing process. The OOB channel can leverage one of many different available pairing methods [12].

2.2 Adversary and threat model

We assume a powerful adversary that has the following capabilities:

A1. *Full access to the wireless channel.* The adversary can *observe, inject, modify* or *disrupt* any message transmitted over the wireless channel between MN and SN.

A2. *No access to out-of-band channel.* The adversary has no access to the out-of-band channel (**S3**) used by the MN and the SN for the pairing operations.

A3. *Computationally bounded.* The adversary is not able to break cryptographic primitives, like AES and SHA-1.

A4. *No compromise of MN or SN.* The adversary does not have the ability to compromise either the software or hardware of the MN or the SN (at least, without it being immediately evident to the patient). We realize that this assumption may be difficult to achieve. We address MN security in another work [22]; in this paper, we assume the SN packaging is sealed or otherwise tamper-resistant. Thus, for the purpose of this paper the adversary cannot discover the secret keys by compromising the MN or SN.

A5. *Local adversary.* The adversary does not compromise any of the back-end services. If the adversary has access to the back-end services, he can get access to the patient's sensitive data, defeating the purpose of a secure wireless body-area mHealth network (BAN) protocol.

These assumptions begin to define the scope of our solution: we propose an energy-efficient protocol that provides security and privacy at the link layer; we leave physical-layer attacks to other physical-layer solutions.

Given the capabilities of the adversary, we focus on the following threats. For a more thorough treatment of privacy-related threats to mHealth, see our survey [4] and paper [11].

T1. *Threat to privacy:* The adversary wants to learn sensitive information about the patient, such as medical conditions (e.g., disease or treatment type), sensing situation (e.g., types or number of sensors), or other personal information deemed private (e.g., location or activity). For this threat, the adversary tries to eavesdrop on the SN-MN communication in order to discover sensitive information from the messages (if available in cleartext), or by linking communication to/from a node and applying traffic-analysis techniques [23]. To address this threat a protocol should provide unlinkability and anonymity, which is one of our security and privacy goals (**SP1**).

T2. *Threat to data integrity and authenticity:* The adversary wants to cause the MN to accept incorrect, invalid, or duplicate data, by either *forging* a message that looks legitimate to the MN, *tampering* with a legitimate message from the SN, or *replaying* a previously sent message.

T3. *Threat of resource exhaustion attack:* The adversary wants to exhaust the MN's or the SN's battery to prevent them from collecting the victim's medical data. For this threat, the adversary may send the MN or the SN invalid messages, forcing them to consume power to receive, process, and discard the messages.

2.3 Security and privacy goals

We must be precise in defining the set of properties that an mHealth-sensing system should achieve; the following set of goals directly address the set of threats defined above. We first list the security properties that are essential to protect security and privacy in MN-SN communications; these properties define our design goals. Unless otherwise specified, the term 'node' refers to both the MN and the SN.

SP1. *Node anonymity.* The protocol should not reveal information about the nodes to an (active or passive) observer (addressing **T1**). To preserve node anonymity, transmissions to/from a node must be *unlinkable*; that is, given two transmissions, the adversary should not be able to tell whether they are to/from the same node. The HnS protocol provides weak unlinkability; that is, given two messages the adversary cannot tell whether they are from/to the same node using *only the content of the messages*. To achieve strong unlinkability, a protocol must also prevent an adversary from linking messages to nodes via traffic analysis. HnS does not provide strong unlinkability; in Section 6 we discuss this further.

SP2. *Data confidentiality.* The protocol should not reveal any information about the message content to an observer – active or passive (addressing **T1**).

SP3. *Data integrity and authenticity.* A node should be able to verify that a received message was generated by the node that claims to have generated it, and that the message was not modified in transit (addressing **T2**).

SP4. *Data freshness.* A node should be able to ignore any duplicate or out-of-order messages. If a node receives such a message, it should discard it (addressing **T2**).

SP5. *Efficient message filtering.* The MN and the SN should be able to ignore messages that are not for them, quickly and with minimum energy expenditure (addressing **T3**).

2.4 Trust model

Any trustworthy system is built on certain assumptions about who trusts whom to do what. We outline our assumptions about the three types of principals – manufacturer, health provider, and patient – in our system. A manufacturer is an entity that produces the sensors or mobile nodes, and distributes them to the health provider and the patient. A patient is a person that uses sensors (obtained either directly from a manufacturer or from a health provider) to get information about his or her own health. The patient can choose to forward this information to a healthcare provider for consultation or use it for self-monitoring. A healthcare provider is an entity that provides health services to the patient, including providing and configuring sensors, monitoring the resulting data, and providing health advice or treatment.

TR1. The patient and the healthcare provider trust the SN manufacturer to produce calibrated sensors that operate correctly, so that the patient and the health provider can trust the sensor to provide the right reading.

TR2. The patient and the healthcare provider trust the MN and HnS manufacturers to write correct software; thus, they protect confidentiality and integrity according to the above goals.

TR3. The patient trusts the healthcare provider not to disclose the sensor data to unauthorized parties.

The manufacturer has no stake in the system, so the manufacturer assumes nothing about other principals.

3 Hide-n-Sense

We begin with a brief use case of a patient using the Hide-n-Sense (HnS) protocol to secure communication of her BASN. The patient obtains a new SN and wishes to use it with her MN. She *pairs* the SN and the MN, using one of the many available secure pairing methods [12], allowing them to authenticate each other and share three secret keys used for discovery $\left(k_d^h, k_d^p, k_d^m\right)$, a threshold for consecutive failed discovery attempts (w_d), and a nonce (N). Later, when she wears the SN, the SN automatically discovers the MN (which she routinely carries) using the HnS discovery protocol, and the two nodes establish a secure session by sharing three more secret keys. Then the SN securely uploads the sensed data to the MN, using the HnS session protocol, until the patient removes her SN.

3.1 HnS discovery

Once two nodes are paired, they do not necessarily remain in radio contact. To communicate, the nodes must first use a discovery protocol to detect that they are in radio range. Most discovery protocols, however, reveal the identity of the seeker (or the nodes being sought), violating **SP1**.

The HnS discovery protocol is an adaptation of *Tryst* from the SlyFi protocol, with two important differences: (1) the SN does not have to update its address table (see discussion below), and (2) the SN and MN do not need to maintain synchronized clocks for discovery. These changes shift computational burden from the resource-constrained SN to the more powerful MN, and remove an unnecessary constraint (time synchronization), which increases the variety of SNs that can be used and makes HnS more robust.

By design, we allow only the SN to initiate discovery. There are two reasons for this choice. First, it can turn off its radio when not in use, and save energy. Second, to filter incoming messages, we show below that a node must maintain a hash table of expected headers; if the SN does not anticipate receiving a message (without it having initiated the communication), the SN does not have to maintain any hash table, and thus, it can save memory and computation.

Figure 1 shows the discovery format of a discovery message. The header, payload, and message authentication code (MAC) for a discovery message are generated using

$$\begin{aligned} \mathcal{H}_d(i_d) &= \text{AES}_{k_d^h}(N + i_d) \\ Payload &= \text{AES-CTR}_{i_d, k_d^p}(data) \\ \mathcal{M}_d(i_d) &= \text{AES-CMAC}_{k_d^m}(\mathcal{H}_d(i_d)||Payload) \end{aligned} \tag{1}$$

where $\mathcal{H}_d$ and $\mathcal{M}_d$ represent the header and MAC of a discovery message (the subscript d, denotes a discovery variable); i_d is the discovery message number; k_d^h, k_d^p, k_d^m are the keys shared during pairing (note the superscripts h, p, m; k_d^h is used to encrypt the header, k_d^p is used to encrypt the payload, and k_d^m to generate the MAC); and N is the nonce, also shared during pairing. (Table 1 shows a list of important notations used in the paper.) The discovery message number i_d is used as a nonce in the CTR mode to generate the key stream that is used to encrypt the payload; that is, the first payload block is encrypted with the key generated using $\text{AES}_{k_d^p}(i_d||1)$, the second block is encrypted with the key generated using $\text{AES}_{k_d^p}(i_d||2)$, and so on. We chose CTR mode because it produces cipher text of same size as the plaintext, a desired property in low-power mHealth networks, since transmission consumes a lot of energy.

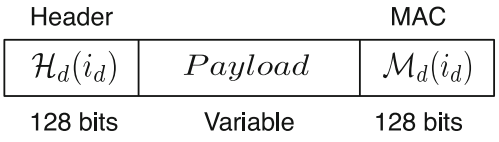

Fig. 1 Format of the i_d^{th} discovery message

Table 1 Important notations used in the paper

Notation	Description
k^h, k^p	keys to encrypt header and payload, respectively
k^m	key used to generate MAC
k_d^m, k_s^m	k^m for discovery and session packets, respectively
k_s^h, k_s^p	keys to encrypt session packet header and payload
k_d^h, k_d^p	keys to encrypt discovery packet header and payload
k_s^x	key to generate random sequence (in MAC striping)
w_d	max. number of allowed failed discovery attempts
w_s	max. number of allowed session packet loss
i_d, i_s	discovery and session message numbers, respectively
$\mathcal{H}_d, \mathcal{M}_d$	discovery packet header and MAC
$\mathcal{H}_s, \mathcal{M}_s$	session packet header and MAC
h, m	length of header and MAC in a packet
T	duration of a time period in SN's lifespan
ρ	max. allowed successful forgery probability during T
β	successful forgery prob. threshold during SN's lifespan

The MN maintains a hash table with w_d future discovery headers; that is, it maintains headers from $\mathcal{H}_d(i_d+1)$ to $\mathcal{H}_d(i_d+w_d)$. The MN filters incoming messages (i.e., determines whether the received message was sent by the SN) by doing a hash-table lookup on the header of the received message.

The HnS discovery protocol works as follows:

1. When the SN needs to discover the MN, it generates (using Eq. (1)) and sends a discovery message, say the i_d^{th} discovery message. The data in the payload does not matter here; it can be application-related data (e.g., sensor status) or it may be empty.
2. MN replies with the $(i_d+1)^{th}$ discovery message. The payload contains session keys $(k_s^h, k_s^p, k_s^m, k_s^x)$, session message-loss threshold (w_s), and the length of header and MAC (h_0, m_0) for session messages. The MN uses a different session key for each SN; w_s and (h_0, m_0) are same for all SNs in a BASN, and are set by the MN. The MN updates its hash table: it removes old headers (i.e., headers up through i_d+1) from the hash table, and adds future discovery message headers, such that the number of headers in the address table remains w_d; the MN also adds w_s future session headers and removes all session headers from an old session, if any. At this point, a secure session is established from the MN's perspective.
3. After sending a discovery message, the SN waits for incoming messages. If the received message is from the MN (i.e., the message header matches $\mathcal{H}_d(i_d+1)$) and if the message is authentic (i.e., MAC verification is successful), the SN decrypts the payload, and saves the data (i.e., keys and other parameters). At this point, a secure session is established from the SN's perspective. It records internally the number i_d+2 for use in a future discovery message. If the SN does not receive a reply from the MN, for a pre-determined period, it gives up and turns off its radio. In our implementation, the SN will retry twice before giving up.

3.2 HnS session protocol

The HnS session protocol is used to secure the sensor data that the SN sends to the MN. A session begins after a successful conclusion of the discovery protocol, and it ends if the SN loses w_s consecutive session messages (a message is considered lost if the SN does not receive an ACK for it from the MN, within the ACK timeout period, as defined by the underlying medium-access control protocol).

In the session protocol (as in the discovery protocol) we allow only the SN to initiate the communication (again, to save energy and memory at the SN). This design works well for a BAHN, where most of the communication is from the SN to the MN (i.e., sensor data), with the MN occasionally sending control commands to the SN. The message filtering mechanism at the MN is similar to the discovery protocol.

Figure 2 shows the session message format. The header, payload, and MAC for the i_s^{th} message are generated by

$$\mathcal{H}(i_s) = \text{AES}_{k_s^h}(i_s)$$
$$Payload = \text{AES-CTR}_{i_s,k_s^p}(data)$$
$$\mathcal{M}(i_s) = \text{AES-CMAC}_{k_s^m}(\mathcal{H}(i_s)||Payload)$$
$$\mathcal{H}_s(i_s) = \text{MSB}_h(\mathcal{H}(i_s)) \; ; \quad \mathcal{M}_s(i_s) = \text{MSB}_m(\mathcal{M}(i_s))$$

where $\mathcal{H}_s$ and $\mathcal{M}_s$ represent the header and MAC of a session message; k_s^h, k_s^p, k_s^m are the session keys shared during discovery (the subscript s denotes a session variable); i_s is the session message number, and (h, m) represent the length of header and MAC (at the beginning of a session, $h = h_0, m = m_0$, where (h_0, m_0) were shared during discovery). Again, similar to discovery protocol, HnS uses i_s as a nonce for generating the key stream in the CTR mode to encrypt the payload. Note that the actual bits sent as header and MAC are the h and m most significant bits (MSB) of $\mathcal{H}$ and $\mathcal{M}$, respectively (as shown in Fig. 2a). The actual session message that is transmitted is shown in Fig. 2b, which is

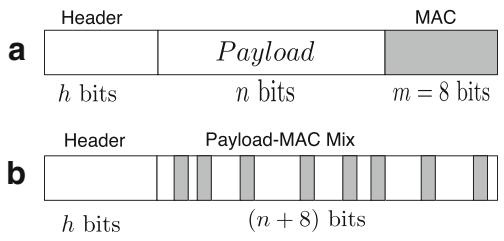

Fig. 2 Session message format (for illustration we show 8-bit MAC): **a** Before MAC striping, **b** After MAC striping (one shaded rectangle represents one MAC bit)

generated after MAC striping the message shown in Fig. 2a; we describe the MAC striping technique next.

Unlike discovery messages, session messages are transmitted frequently, so we wish to keep the session message overhead (size of header and MAC) as small as possible. However, reducing the size of header and MAC also decreases the protocol's resistance to forgery. So, to reduce the message overhead while maintaining reasonable security, we propose two techniques: *MAC striping* (to make the session protocol resistant against selective forgery, even for small MAC sizes), and *adaptive security* (to dynamically change message overhead to maintain security against existential forgery).

3.2.1 MAC striping

MAC striping provides strong resistance against selective forgery. In selective forgery the adversary tries to forge a message with a chosen payload. For the adversary to forge a message of format shown in Fig. 2a, with a chosen ciphertext payload, requires work proportional to 2^{h+m} (he needs to guess the correct h-bit header and m-bit MAC). However, an adversary may intercept[1] a message (**A1**) to discover a valid header, and thereby reduce the work to 2^m. MAC striping changes the message format slightly (as shown in Fig. 2b), which makes the protocol strongly resistant against selective forgery, even with a small MAC. The MAC bits are interspersed in the payload at different offsets, represented by shaded rectangles in Fig. 2b. These bit locations are different for each message, based on a pseudorandom sequence generator function f:

$$\langle x_0, x_1, ..., x_m \rangle = f\left(k_s^x, i_s, n, m\right), \tag{2}$$

where $x_i (< n)$ is the randomly-chosen offset for the i^{th} most significant bit of $\mathcal{M}_s(i_s)$, k_s^x is the key that was shared during discovery, i_s is the session message number, n is size of the payload, and m is size of the MAC. Note that these offsets are chosen at random by the MN and the SN independently. We used the AES encryption algorithm as a secure pseudorandom number generator; thus, the adversary cannot guess the offsets of MAC bits better than random guess.

When a node (SN or MN) receives a session message (it will be of format Fig. 2b) with a valid header, the node computes the pseudorandom sequence $\langle x_0, x_1, \ldots, x_m \rangle$, and separates the MAC bits from the payload (to recover a message of the format in Fig. 2a).

[1]To intercept a message, the adversary captures the message header when it is being transmitted, and then disrupts some bits in the payload or the MAC so that the receiver discards the message because it will fail the MAC verification process.

3.2.2 Adaptive security

In wireless protocols the receiver node uses the header to filter incoming messages (that is, to determine whether the incoming message is addressed to this node), and it uses the MAC $\mathcal{M}$ to verify whether the received message is authentic. The overhead (header and MAC) in these protocols is usually fixed, and for strong security, the protocols choose long header and long MAC. However, a node is not always in a hostile environment, so using large overhead all the time is unnecessary. In many mobile devices, the transmission is expensive (energy-wise), so a low-power sensor should minimize transmission overhead.

Adaptive security provides strong security when it is required (e.g., when a network is under attack), but saves energy, at both the sender and receiver, when it is not. In HnS, the nodes dynamically increase the size of the header and/or MAC sent in the message if MN detects the presence of an adversary who is trying to forge a message; otherwise, the nodes use a short header and MAC. In our illustration, the nodes dynamically change the MAC size, keeping header size constant; in Section 6, we discuss when it would be beneficial to change header size too.

Adaptive security: How to adapt. Consider a simple BAHN with a set of sensor nodes (SNs), and a mobile node (MN, e.g., a smartphone). The MN chooses the header and MAC sizes to be used by all the SNs in the BAHN, and shares them with each SN during the discovery process. When the MN decides that it needs to change the MAC size, it notifies the SNs by sending a *reissue* message. During this adaptive process, the MN ensures that communication is not disrupted due to inconsistency in message overhead sizes.

The MN maintains a set σ of MAC sizes: the current MAC size (which the SN should use), and the old MAC size (which the SN might be using). When the MN receives a message, it parses the message to get the header, and verifies the MAC using the current MAC size from σ. If the message header is valid, but the message MAC is wrong, the MN will verify the MAC using the old MAC size from σ. This ensures that the MN can receive messages sent by the SN, in case the SN is using old MAC size. If the SN is using the old MAC size, the MN will send a *reissue* message with the new MAC size. Once the MN knows that the SN has successfully adapted to the current MAC size (i.e., when it receives a message from SN with the current MAC size), it removes the old MAC size from σ. Figure 3 shows how the MN and the SN adapt the MAC size: initially they are using m_0. At time t_1, MN decides to use m_1, but SN is still using m_0, and so MN issues a *reissue* command message telling SN to use m_1, and when MN receives a message from SN with m_1 MAC size, it removes m_0 from σ. This approach generalizes to multiple SNs (σ may contain an old

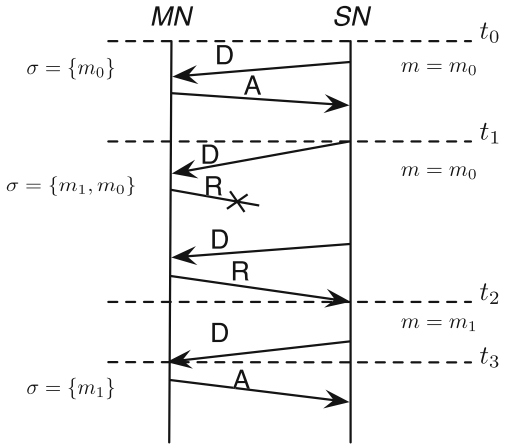

Fig. 3 Adaptive security at work (D = data, A = ack, R = reissue)

m for each SN) and to header-size reductions (σ contains (h, m) pairs).

Adaptive security: When to adapt. The MN decides when the BAHN must adapt to new header and MAC sizes; the decision on *when* to adapt can be based on many factors, such as application requirements, upper bound on security against forgery, and network bandwidth optimization. Below we give an example of how the MN adapts based on a *success probability threshold*, an upper bound on security against forgery. We discuss the parameters involved in this adaptation in Section 6.

The MN uses the header to filter incoming messages. When the header of the incoming message is valid (i.e., the header matches with one of the headers in the MN's address table), the MN does MAC verification to determine whether the message is authentic. If the MAC verification fails, it implies that either the incoming message was from the SN but got corrupted in transit, or that the message is of a neighboring BAHN that happened to use a header considered valid by the MN, or that the message is a forgery attempt. It is hard for the MN to distinguish between these three cases. We take a cautious approach, and consider a failed MAC verification to be a forgery attempt.

In adaptive security MN dynamically changes the header and MAC sizes to save energy and to maintain a certain level of security. We can define this certain level of security as: the probability that an adversary will successfully forge a message should always be below β, where β is chosen by the application.

The SN's lifespan (perhaps years) can be divided into n time periods; thus, lifespan of the sensor $= \sum_{i=1}^{n} T_i$, where T_i is the duration of period i in SN's life span. Let ρ be our chosen successful forgery probability threshold for each time period. To maintain the security guarantee that an adversary's success probability is never greater than β, we choose ρ such that:

$$\beta \geq 1 - (1 - \rho)^n \tag{3}$$

Now, for a given ρ, the HnS adaptive security technique should adapt such that the successful forgery probability for every time period is less than ρ, regardless of the number of forgery attempts by an adversary.

The MN tracks the number of failed MAC verifications. The probability that one failed MAC verification would succeed (that is, probability of a successful forgery given that the adversary guessed the header correctly) is 2^{-m}, where m is the MAC size. Thus, the probability of successful forgery in x failed MAC verifications, $P = 1 - \left(1 - 2^{-m}\right)^x$. To achieve $P \leq \rho$, x should be

$$x \leq \frac{\log(1 - \rho)}{\log(1 - 2^{-m})} = \alpha, \text{ the threshold} \tag{4}$$

Thus, if the number of failed MAC verifications exceeds the threshold (α), the MN increases the MAC size. Although the MN will not accept messages with the old MAC size during the current time period, if the message is valid the MN sends a reissue command to indicate a change in MAC size.

The MN falls back to a smaller MAC size after the time period T expires. The MN can choose to keep T and ρ constant, but it can improve performance by initially choosing ρ close to β and then varying ρ and T for future time periods, to maintain the condition in Eq. (3) throughout the lifespan of the sensor.

4 Security and privacy analysis

In this section, we explain how HnS achieves the security properties mentioned in Section 2.3.

4.1 Node anonymity (**SP1**) and Data Confidentiality (**SP2**)

If two messages to/from a node are indistinguishable from random bits, then an adversary cannot link them, and hence they provide **SP1** and **SP2**. HnS uses the same cryptography primitives as SlyFi for computing the header and MAC in the messages. Pang provides a formal analysis for indistinguishability of headers and MACs in SlyFi [14], and we need only show that our optimizations do not compromise that indistinguishability.

HnS discovery messages improve on SlyFi in that HnS discovery messages are generated using different nonce values, while SlyFi discovery messages use the same nonce for certain period, making linking obvious during that period. Therefore, it is easy to see that the discovery message of HnS provides strictly better indistinguishability than SlyFi.

HnS session messages differ from SlyFi in two ways: (1) the header and MAC are truncated, and (2) the MAC is striped. We argue that these changes do not compromise the indistinguishability of SlyFi.

Claim 1 *A truncated* h*-bit header and* m*-bit MAC is as indistinguishable from random bits as the original* n*-bit header and* n*-bit MAC* ($h < n$ *and* $m < n$).[2]

Proof Suppose not. Then, an adversary can distinguish more easily the n-bit headers or MACs from random bits, by just looking at the first h-bits (for header) or m-bits (for MAC). Contradiction. □

Claim 2 *The plaintext+MAC of HnS is as indistinguishable from random bits as the original without MAC-striping.*

Proof The MAC scheme of SlyFi is a special case of HnS; choose a pseudorandom sequence generator that does not change the locations of the original MAC bits. Therefore, an adversary who can break the indistinguishability of HnS should be able to break that of SlyFi. □

By Claim 1 and 2, the whole packet of HnS is as indistinguishable from random bits. Therefore, HnS provides as high confidentiality and unlinkability as SlyFi.

4.2 Data integrity and authenticity (**SP3**)

HnS prevents unauthorized changes to the header or payload by use of a message authentication code (MAC). Nonetheless, the adversary may attempt to alter the message content or construct a new message without being detected; such an attack is called *forgery attack*. We consider two types of forgery attacks: *selective forgery* (when the adversary tries to get the receiver to accept a message with a payload of his choice), and *existential forgery* (when the adversary tries to get the receiver to accept a message with any payload).

Selective forgery In selective forgery the adversary tries to find the corresponding MAC for a chosen payload (i.e., ciphertext).[3] Since the adversary does not know the MAC key (shared at discovery), the adversary must pick a random MAC out of $\{0,1\}^m$, where m is the length of the MAC.

It may be possible to do better than random to guess a MAC, by exploiting a weakness in a crypto algorithm or its implementation (i.e., using cryptanalysis), but we use a standard algorithm (AES) and an existing hardware implementation, and assume the adversary cannot break them. If these algorithms later are broken, any suitable crypto algorithm may be substituted for AES. It is, however, important to assess whether our techniques provide any additional information to an adversary that can help him do better than random. In MAC striping, no additional bits are transmitted in a packet; the MAC bits are intermixed in the payload based on a pseudorandom sequence, which is computed independently by the sender and receiver. In adaptive security an adversary may attempt to learn the adaptation parameters, but these parameters are independent of the crypto algorithm or its encryption keys, so knowing these parameters does not enable the adversary to improve its ability to guess a MAC.

The HnS discovery messages use a 128-bit MAC, so it is secure against selective forgery. The HnS session protocol, however, uses a small-sized MAC. Without MAC striping (Fig. 2a), the probability that the attacker succeeds in selectively forging a session message with one random guess is 2^{-m}. With MAC striping (Fig. 2b), the MAC bits are interspersed with the payload bits at locations determined by Eq. (2). Without knowing the key k_x^s and the message number j, it is computationally hard for the adversary to know which bits among the $(\ell + m)$ bits are the MAC bits, where ℓ is the length of the payload. Therefore, to forge a message of his choice, the adversary has to guess the matching MAC bits out of 2^m possibilities, and MAC-bit locations out of $\binom{\ell+m}{m}$ possible MAC-bit locations, making the probability of success $\frac{1}{2^m\binom{\ell+m}{m}}$. To illustrate: when the payload is 10 bytes long and the MAC is 2 bytes long (i.e., $\ell = 80, m = 16$), the success probability of selective forgery without MAC striping is 2^{-16}, and with MAC striping it becomes approximately 2^{-75} (since $\binom{96}{16} \approx 2^{59}$). Therefore, the MAC striping technique drastically decreases the success probability of selective forgery (from 2^{-16} to 2^{-75} in the example) without increasing the MAC size.

Existential forgery In existential forgery the adversary tries to find any matching payload-MAC pair that the MN would accept; the adversary has no control over the payload content. Among the $2^{\ell+m}$ possible payload-MAC pairs, 2^{ℓ} are matching payload-MAC pairs. Without knowing the MAC key, the adversary can only guess, and the success probability of such a random guess is 2^{-m}.

[2] A string is said indistinguishable from random bits if any computationally bounded adversary cannot guess correctly whether the string is truly random or not with a non-negligibly higher probability than the probability that she guesses incorrectly. The formal treatment of this security property can be found in [14].

[3] For the adversary to inject sensor data chosen by itself, the adversary needs to compute the corresponding ciphertext, which is difficult because it requires knowledge of the encryption key and nonce. As a forgery attack, however, it suffices to make the MN accept the ciphertext chosen by the adversary, whatever the decrypted data might be.

The HnS discovery protocol uses a 128-bit MAC, so it is secure against existential forgery. For the HnS session protocol, the work required for existential forgery is the same (i.e., 2^m) with and without MAC striping if the guess is made uniformly at random. The HnS session protocol uses adaptive security to ensure that the probability of a successful existential forgery is never greater than β (as described in Section 3.2.2).

Remark about adversary and our analysis In our analysis we assume a very powerful adversary who discovers the header of messages by observing the message header and at the same time corrupting the message. This attack requires specialized hardware and skills, and even so, to target a user, the adversary needs to know which message to observe, which is hard since HnS messages are unlinkable. In practice, adversaries will not know the headers, and hence, the work required to selectively or existentially forge a message increases by a factor of 2^h, where h is the header length.

4.3 Data freshness (**SP4**)

Each message in the HnS protocol has an encrypted header that contains the message sequence number. Recall that the MN maintains a hash table of headers of expected messages, and when it receives a message from the SN it removes all the headers with the sequence number less than the received message from the hash table. Thus, the MN will only accept messages in increasing order of their sequence number, achieving **SP4**.

4.4 Efficient message filtering (**SP5**)

If the MN requires lots of processing to discard invalid messages, an adversary can launch a resource-exhaustion attack, where the adversary will try to drain the MN's battery by sending a stream of invalid messages. HnS uses the message filtering method proposed in SlyFi. The MN uses hash-table lookups to filter incoming messages, and it discards invalid messages (i.e., message with headers that are not available in the hash table). For any message that has a matching header in the hash table, the MN does MAC verification, and discards the message if the MAC verification fails. Both the operations (hash-table lookup and MAC verification) require negligible energy and processing time on the MN; on TI's Chronos device, on average, AES encryption (or decryption) and MAC verification on a 16 byte message takes 0.57 μJ and 2.94 μJ, respectively. Even if the adversary sends 100 attempts per second (which saturates the network in our implementation), it succeeds only in drawing 2.5 % of the idle power, and on a smartphone MN the percentage would be far less. Thus, the message filtering mechanism is efficient enough to thwart resource-exhaustion attacks.

In terms of resource-exhaustion attack, the adversary's knowledge of the threshold x in Eq. (4) helps little. With the knowledge of x, the adversary can keep the MAC size at the maximum with the minimum cost; no message needs to be sent until the MAC size drops below the maximum. However, at the same time, the HnS will also save its energy for verifying the MAC field of attack messages. Therefore, for the sake of resource-exhaustion attack, it is the best strategy for the adversary to keep sending attack messages at the cost of its own energy for the attack.

5 Evaluation

In this section, we describe our prototype implementation of HnS, and its evaluation. Our experiments measure the energy use and network performance of HnS as it adapts to forgery attempts from a synthetic adversary. We also compare the energy cost of HnS with SlyFi and SPINS [16] (a security protocol for low-power wireless sensor networks).

To evaluate the efficacy of HnS, we implemented HnS on the eZ430 Chronos wireless wrist device [8] from Texas Instruments, which integrates a 16-bit ultra low-power MCU (MSP430), a CC1101 wireless transceiver with hardware support for AES-128 encryption, 32 KB flash, 4 KB RAM, three integrated sensors (3-axis accelerometer, pressure, and temperature), and 255 mAh battery, into a low-power wearable platform. We use the watch's sensors as a proxy for mHealth sensors in our experiments.

Using this hardware setup, we measure 1) HnS's ability to respond to an attack, and 2) HnS's impact on system energy usage.

5.1 Adaptive security

Our first experiment uses three Chronos watches, acting as an SN, an MN and an adversary. Using a watch as MN, instead of a phone, allows us to implement HnS at the link layer. The MN and SN use the HnS protocol, and the adversary tries to forge a message. Throughout the experiment we observe how the MN adapts to forgery attempts, and how the SN and MN coordinate the change in the number of security bits used.

Experiment setup The SN imitates a temperature sensor that sends a reading every 15 s; the sensor payload is 6 bytes. We assume an unusually strong attacker, who knows the headers in MN's table. (As mentioned earlier, such an attack requires intercepting packet headers, which is extremely difficult.) The adversary attempts to forge messages for 10 min

at a rate of 100 forgery attempts per second (this rate saturates the wireless channel), after which communication returns to normal.

Figure 4 shows the result of this experiment. The number of security bits used is shown over time for the MN and SN under HnS, and for SPINS [16]. SPINS is a security protocol for wireless sensors, and we use SPINS as a benchmark for comparison. SPINS uses a 64-bit MAC as protection against message forgery. When not under attack, HnS achieved much lower overhead than SPINS. During the attack, HnS increased its overhead; however, the overhead never needed to be raised to SPINS's level.

During the experiment, both the MN and SN start using a 2-byte header and 3-byte MAC (total overhead =40 bits). The MN chooses $\beta = 2^{-24}$, $\rho = 2^{-16}$ and $T = 30$ min. At time $t = 60$ sec the attacker begins sending messages with random payload-MAC bits but using a header it knows is in the MN's hash table. Such a strong attacker only has to transmit 256 messages to force the MN to increase its MAC size, m (see Eq. (4)). After the MN increases m to 32 bits, the attacker must send 2^{16} messages before m increases again, but the attack ends before that. After the attack has ended, the MN and SN both slowly return to their previous security level after time T.

In the experiment, the attack lasted 10 min. Figure 5 shows how HnS would adapt if a persistent attack were to continue indefinitely. Over time, the MN will continue to use more security bits in response to continued forgery attempts (represented as a log-scale on x-axis; the x-axis also represents the duration of attack, since attack rate is constant). The y-axis represents the security bits used by the protocol's header and MAC (i.e., $h + m$). In the plot, the initial overhead is set to 48 bits, and the probability that a forgery succeeds is set to 2^{-16} (i.e., $\rho = 2^{-16}$). The plot also compares HnS's overhead to that of the SlyFi (256-bit)

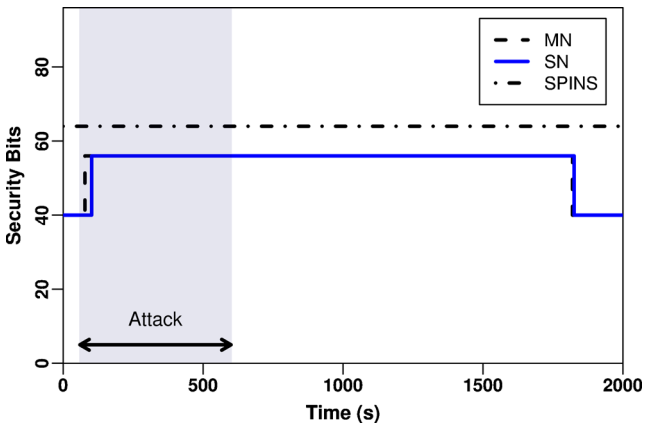

Fig. 4 HnS's adaptive security in response to an attack; as the attack begins, MN and SN increase their MAC size

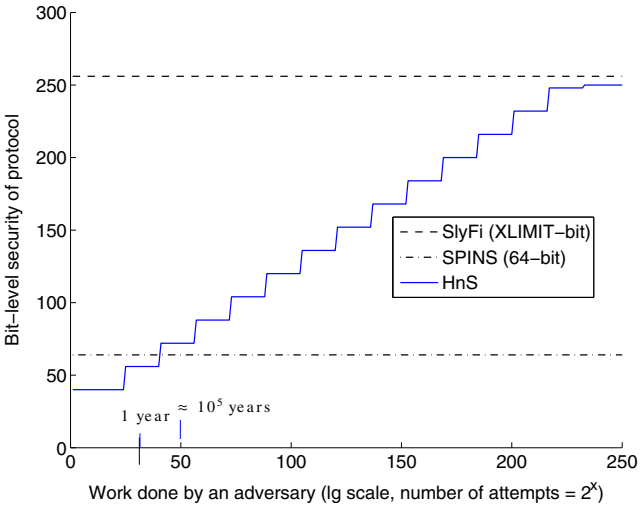

Fig. 5 Projection of how HnS would adapt if a forgery attack continues till infinity

and SPINS (64-bit) protocols. In *theory*, HnS will eventually incur the same overhead as SlyFi; however, we don't expect any attack, any device, or even the Earth, to last that long.

5.2 Energy analysis

Our second experiment measures the energy required to run a SlyFi-like privacy-preserving wireless protocol, SPINS-like protocol, and HnS. To measure energy for SPINS and SlyFi, we implemented a simple protocol that uses the SPINS and SlyFi message format, but involves no encryption or security features. This gives us a slightly lower energy consumption values for these protocols than they actually would consume.

We simulated the traffic produced by a six-electrode ECG sensor, which sends 10-byte messages (a timestamp and 1 byte per electrode) at a rate of 10 Hz, for 60 sec. We measured the energy consumed by the SN using a Monsoon power monitor [13]. Table 2 shows the average energy consumed by each protocol.

Table 2 Comparing energy cost of three protocols

	SlyFi	SPINS	HnS
Transmissions overhead	256-bit	64-bit	40-bit[a]
Security and Privacy[b]	Yes	No	Yes
Energy (J)[c]	0.78 (0.013)	0.39 (0.013)	0.38 (0.01)
Battery life	52.33 (1.27)	103.6 (3.74)	107 (2.75)

[a]HnS overhead can vary between 16 and 256 bits

[b]Whether the protocol achieves the goals described in Section 2.3

[c]Mean (and standard-deviation) of 10 trials

Since communication cost dominated the energy consumption, among the three protocols, the HnS protocol had the least overhead and hence it consumed less power and had greater battery life. Compared to SlyFi, HnS provides similar security and privacy properties at about half the energy cost, and compared to SPINS, HnS provides more security and privacy properties with no compromise in sensor life. Such substantial improvements are important in the context of continuously-worn health sensors.

5.3 Network performance

Our third experiment measures packet loss in the presence of an adversary. The setup is similar to the second experiment, except that we introduce an adversary who is trying to forge messages by sending random messages, and we measure packet loss rate instead of energy. The SN, the MN, and the adversary were placed about 1.2 m from each other. At the start of the experiment, the adversary began transmitting packets at a fixed rate. The SN then began sending Data packets to MN for 1 min, at the rate of 5 messages per second, and expected an ACK from the MN for every Data packet. The total number of transmitted packets is the sum of the number of Data packets transmitted by the SN and the number of ACKs sent by the MN. The total number of lost packets is the sum of the number of lost Data packets (i.e., Data packets sent by the SN but not received by the MN) and the number of lost ACKs. Thus, the packet-loss ratio is the ratio of the total number of lost packets to the total number of transmitted packets. We repeat this experiment 5 times for each protocol (HnS, SPINS, and SlyFi) and transmit rate.

Figure 6 shows the average packet-loss percentage for the three protocols for different adversary rates. The large overhead of SlyFi explains its higher packet loss: larger packets are more susceptible to collisions with the adversary's packets. Thus, the large overhead not only consumes more transmission energy, it also causes more retransmissions due to packet losses in a congested network, causing sensors to expend more energy. SPINS and HnS are designed for sensor networks, have smaller overhead, and incur less packet loss.

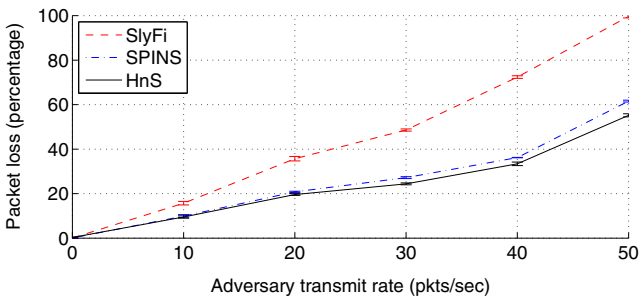

Fig. 6 Packet loss for different protocols in presence of an adversary. Each point is a mean of 5 trials, with error bars showing 1 standard deviation

5.4 Microbenchmarks

We measured the time and energy required for encrypting 16 bytes, computing a AES CMAC for 16 byte message, and updating the hash table in HnS. We ran each of the above operations 1000 times and measured the time and energy. Table 3 shows the time and energy required for one operation of each type. As shown in the table, these operations use negligible energy, and have almost no impact on the sensors' battery life. For instance, encrypting and computing MAC for each message in the experiment described in Section 5.2 would reduce the expected battery life of the sensor by less than 1 min.

6 Discussion

HnS for other domains Although we proposed HnS as a privacy measure for mHealth systems, one may apply HnS to other kinds of systems, even to non-sensing systems. For example, without modification, HnS can provide its privacy-preserving mechanism to any system where multiple nodes report data to a paired sink node within one-hop distance. Extending HnS to different topologies such as multi-hop networks and its scalability in such settings, are beyond the scope of this paper.

Forgery attacks on SN In the sections above, we describe forgery attacks only on the MN because there is only a very small attack window for the SN; the MN never initiates communication and thus the SN rarely listens for messages. The SN's radio is on only when it is waiting for an ACK, seeking for only one particular header in the incoming message (unlike the MN who has to match the header of incoming message with all the headers from the hash table). Thus, an adversary will have one chance to forge an ACK, and the success probability for that is $2^{-(h+m)}$. Moreover,

Table 3 Time and energy required for encrypting 16 bytes, computing CMAC of a 16 byte long message, and updating the MN's hash table in HnS

	Time in msec mean (stdev)	Energy in μJ mean (stdev)
AES encryption	0.07 (0.0014)	0.57 (0.011)
AES CMAC	0.37 (0.0021)	2.94 (0.011)
Hash table update	0.32 (0.0041)	2.41 (0.030)

Mean (and standard-deviation) of 10 trials

since the entire packet (including header) is encrypted it is hard for an adversary to attack a particular SN using information at the link layer. However, in a worst-case scenario, if the adversary does succeed in forging an ACK, the SN will think that its last message was successfully received by the MN, and it will not send that sensor data to the MN, and the MN may get one less data point. If an adversary wants to drain SN's battery he would have to have to corrupt all ACKs forcing the SN to do multiple retries–a denial of service type attack, which is out of scope of this paper.

Energy cost at MN Although energy consumption by the MN is a concern, in the preceding sections we focus on energy consumption in the SNs because MN has a large battery capacity and can be recharged daily. It is difficult to recharge SNs daily; users expect SNs to last weeks, if not for months.

Selective vs. existential forgery In the context of mHealth sensing it should be easy for the MN to detect existential forgery attempts, because any message accepted and decrypted will produce arbitrary sensor data values, since the adversary has no control over the ciphertext payload. These values will likely be discarded due to semantic checks on the format or reasonable value range for such data. In case of selective forgery, the adversary can choose the ciphertext payload. Although the adversary cannot choose the underlying sensor data because he does not know the encryption key, he can use the previously observed ciphertexts to generate (or choose) a ciphertext that will give a valid sensor data when the MN decrypts the ciphertext. Thus, it is important to have strong resistance against selective forgery in mHealth sensing, as provided by HnS.

Forgery probability and ρ, T In Section 3.2.2, T refers to the duration of time periods in a SN's lifespan (recall that we divide the SN's life span into a series of time periods). For simplicity (and without loss of generality), we assume that all time periods have the same duration (i.e., T), but in practice they can be different. A period is a duration in an SN's life in which HnS can guarantee that the success probability of forgery will be less than ρ (for some chosen value of ρ for that period). We introduce this concept of time period in SN's life span to make brute-force attacks on HnS for small-sized MACs difficult.

In adaptive security, HnS increases the MAC size when an adversary attempts a forgery, and eventually HnS reduces the MAC size. An adversary can wait for the event when HnS reduces its MAC size, thereby attacking only when HnS is weakest. As a counter measure, once the MAC size is increased in a time period, HnS never reduces MAC size in that same time period. This delays the brute-force attack by a factor of $\overline{T}$, where $\overline{T}$ is the average duration of all time periods. Thus, to delay such attacks, long time periods are desirable. However, longer time periods also mean that once MAC size is increased in a period, nodes have to use larger overheads for a longer time, which costs more energy.

ρ is the success probability of forgery in a given time period. Like T, ρ can be different for different time periods. A small ρ implies that HnS has a small threshold (4) for failed MAC verifications; thus, MN will increase its MAC size quickly in response to forgery attempts. Thus, it is desirable to have a small ρ, however, ρ is roughly inversely proportional to T, so, one needs to balance ρ and T. A careful study of this tradeoff is left for future work.

Traffic-analysis Traffic-analysis attacks rely on linking messages to a particular source, and then using various analysis techniques to identify patterns in those messages that reveal some information about the source; thus, a protocol that provides true unlinkability defeats traffic analysis.

In a BASN, there are two levels of linkability: node-level linkability, linking messages to a particular node in a BASN, and user-level linkability, linking the user's BASN traffic to the user, making users vulnerable to tracking. To provide node anonymity (**SP1**) and prevent user tracking, a protocol must provide both node-level and user-level unlinkability.

HnS does not provide user-level unlinkability, nor do SPINS or SlyFi. HnS unlinkability is "weak": although messages cannot be linked based on message content, an adversary may be able to use message transmission timings and message sizes to link messages. HnS can provide strong node-level unlinkability by incorporating some countermeasures against traffic-analysis; for instance, Buttyan et al. [6] propose one such countermeasure, traffic shaping, a technique that makes the message transmission pattern uniform for all nodes. However, one has to be careful when applying countermeasures that require message padding or dummy messages, which can be costly (energy-wise) for SNs.

Limitations HnS is a link-layer protocol. It does not provide any defense at the PHY layer. An adversary may use PHY-layer fingerprinting to link together a set of transmissions from a node, breaking the unlinkability property. However, such attacks require special hardware, raising the bar for the adversary [15].

7 Related work

Several research projects have proposed, and even demonstrated, prototypes for mobile healthcare sensing, but providing a comprehensive security and privacy solution that is energy-efficient for low-power sensors is still an open

question. The HnS protocol that we present in this paper is part of the HnS architecture, which is our effort towards building such a complete solution.

Privacy-preserving wireless protocols The class of privacy-preserving wireless protocols (PPWP) represent all protocols that try to provide user privacy in a wireless network, by obfuscating any information in the transmission that an adversary can use to get some information about the user (or her device). There are several PPWP proposed in the literature [3, 10, 21, and their references]. We use techniques from SlyFi [10] (a relatively strong PPWP), as a basis for comparison while evaluating the HnS protocol. Unlike SlyFi, our focus is to provide strong privacy and security with low energy overhead suitable for low-power sensors in a BASN. HnS differs in the following ways: 1) in HnS we made the resource requirement asymmetric between the two nodes involved in the protocol (that is, one node requires less resources than the other), so that even a lower-capability SN can participate without undue energy consumption; 2) HnS uses smaller header and MAC sizes, which improves energy efficiency and network bandwidth, but to maintain strong security we use an adaptive security model and MAC striping; 3) we provide an energy evaluation for the protocol; 4) in HnS the nodes do not rely on synchronized clocks for discovery, reducing complexity, and allowing SNs to participate without a real-time clock.

Perrig et al. [16] proposed the SPINS security protocol for low-power wireless sensors. They did not consider node privacy, however. To achieve data integrity and authentication, they propose using a constant-length 8-byte MAC. HnS uses less overhead (as low as 4 bytes), and adapts to network traffic changes to maintain a desired level of security.

Adaptive protocols Prasad et al. [18] suggested using three modes of security: low-level, medium-level, and high-level security; the different levels of security use different cipher algorithms. Depending on the user location (e.g., home vs. public place) or which devices the user is contacting (e.g., trusted vs. untrusted), the model will choose an appropriate security level. The different levels of security achieve different sets of properties, and they use different cipher algorithms. However, choosing the right level of security is not easy; for example, a familiar location (e.g., home) does not necessarily imply the absence of an adversary. Portilla et al. [17] propose changing ECC parameters to provide different levels of security depending on the energy budget of the node. In practical settings, the SSL cipher suite provides one form of adaptive security; communicating parties negotiate the cryptographic algorithms to be used at the start of a session. All these three approaches described adapt cryptographic primitives (encryption or MAC algorithms) rather than reducing network overhead, which (as in HnS) would provide more energy savings than the proposed computational adaptation.

The hybrid security model proposed by Shon et al. [20] is the closest work to our adaptive security model. In their adaptive scheme they propose using MAC of different sizes to provide different levels of security, which they choose according to the network characteristics (public, commercial, or private) and the data characteristics (application data or control data). Classifying data sensitivity and determining which level of security would be reasonable for a given data type, while reducing energy used, can be tricky. For mHealth sensing, where the medical data is considered sensitive, their approach would always choose the highest-level security, which would be energy inefficient. Our adaptive model, however, adapts the security level dynamically in response to an attack by an adversary; in the absence of an adversary HnS adapts to reduce the overhead and thus would naturally have lower overhead than Schon's approach.

8 Conclusion

We describe a framework for our Hide-n-Sense (HnS) system, which aims to provide a secure and private mHealth sensing environment. We propose the HnS protocol to provide a secure, private, and energy-efficient communication channel between devices in a body-area network. To achieve the desired security and privacy goals and support low-energy sensor devices, we use three techniques: MAC striping, adaptive security, and an asymmetric resource requirement. We demonstrated these techniques using SlyFi, but these techniques can also be applied to other protocols to make them energy-efficient while maintaining their security and privacy properties. Through experiments, we demonstrated that it is feasible to implement and use HnS on low-power devices. In fact, as shown in our experiments, HnS is more energy-efficient than the existing security protocols for low-power sensors, and much more energy-efficient than existing privacy-preserving wireless protocols.

Acknowledgments This research results from a research program at the Institute for Security, Technology, and Society at Dartmouth College, supported by the National Science Foundation under award number 0910842, and by the Department of Health and Human Services (SHARP program) under award number 90TR0003-01. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors.

References

1. Agarwal S, Lau CT (2010) Remote health monitoring using mobile phones and web services. Telemed e-Health 16(5):603–607. doi:10.1089/tmj.2009.0165
2. Arcelus A, Goubran R, Sveistrup H, Bilodeau M, Knoefel F (2010) Context-aware smart home monitoring through pressure measurement sequences. In: Proceedings of the IEEE international workshop on medical measurement and applications (MeMeA), pp 32–37. doi:10.1109/MEMEA.2010.5480223
3. Armknecht F, Girao J, Matos A, Aguiar RL (2007) Who said that? Privacy at link layer. In: IEEE international conference on computer communications (INFOCOM), pp 2521–2525. doi:10.1109/INFCOM.2007.313
4. Avancha S, Baxi A, Kotz D (2013) Privacy in mobile technology for personal healthcare. ACM Comput Surv 45(3) Online at http://www.cs.dartmouth.edu/dfk/papers/avancha-survey.pdf
5. Buttussi F, Chittaro L (2010) Smarter phones for healthier lifestyles: an adaptive fitness game. IEEE Pervasive Comput 9(4):51–57. doi:10.1109/MPRV.2010.52
6. Buttyan L, Holczer T (2012) Traffic analysis attacks and countermeasures in wireless body area sensor networks. In: IEEE international symposium on a world of wireless, mobile and multimedia networks (WoWMoM), pp 1–6. doi:10.1109/WoWMoM.2012.6263774
7. Chang H-L, Shaw MJ, Lai F, Ko W-J, Ho Y-L, Chen H-S, Shu C-C (2010) U-health: an example of a high-quality individualized healthcare service. Personalized Med 7(6):677–687. doi:10.2217/pme.10.64
8. TI eZ430 Chronos. http://processors.wiki.ti.com/index.php/EZ430-Chronos
9. Coyle S, Benito-Lopez F, Byrne R, Diamond D (2010) On-body chemical sensors for monitoring sweat. In: Wearable and autonomous biomedical devices and systems for smart environment, volume 75 of lecture notes in electrical engineering, pp 177–193. Springer. doi:10.1007/978-3-642-15687-8_9
10. Greenstein B, McCoy D, Pang J, Kohno T, Seshan S, Wetherall D (2008) Improving wireless privacy with an identifier-free link layer protocol. In: Proceedings of the international conference on mobile systems, applications, and services (MobiSys), pp 40–53. ACM. doi:10.1145/1378600.1378607
11. Kotz D (2011) A threat taxonomy for mHealth privacy. In: Proceedings of the workshop on networked healthcare technology (NetHealth). IEEE Press. doi:10.1109/COMSNETS.2011.5716518
12. Kumar A, Saxena N, Tsudik G, Uzun E (2009) A comparative study of secure device pairing methods. Pervasive Mob Comput 5(6):734–749. doi:10.1016/j.pmcj.2009.07.008
13. Monsoon power monitor. http://www.msoon.com/LabEquipment/PowerMonitor/
14. Pang J (2009) Quantifying and mitigating privacy threats in wireless protocols and services. PhD thesis, School of Computer Science, Carnegie Mellon University
15. Patwari N, Kasera SK (2007) Robust location distinction using temporal link signatures. In: Proceedings of the ACM international conference on mobile computing and networking (MobiCom), pp 111–122. ACM. doi:10.1145/1287853.1287867
16. Perrig A, Szewczyk R, Tygar JD, Wen V, Culler DE (2002) SPINS: security protocols for sensor networks. Wirel Netw 8(5):521–534. doi:10.1023/A:1016598314198
17. Portilla J, Otero A, de la Torre E, Riesgo T, Stecklina O, Peter S, Langendörfer P (2010) Adaptable security in wireless sensor networks by using reconfigurable ECC hardware coprocessors. Intern J Distrib Sens Netw 2011(2011). doi:10.1155/2010/740823
18. Prasad NR, Alam M (2006) Security framework for wireless sensor networks. Wirel Pers Commun 37:455–469. doi:10.1007/s11277-006-9044-7
19. Saxon LA, Hayes DL, Gilliam FR, Heidenreich PA, Day J, Seth M, Meyer TE, Jones PW, Boehmer JP (2010) Long-term outcome after ICD and CRT implantation and influence of remote device follow-up: the ALTITUDE survival study. Circulation 122 (23):2359–2367. doi:10.1161/CIRCULATIONAHA. 110.960633
20. Shon T, Koo B, Choi H, Park Y (2009) Security architecture for IEEE 802.15.4-based wireless sensor network. In: Proceedings of the International Symposium on Wireless Pervasive Computing (ISWPC), pp 1–5. doi:10.1109/ISWPC.2009.4800607
21. Singelée D, Preneel B (2006) Location privacy in wireless personal area networks. In: Proceedings of the ACM Workshop on Wireless Security (WiSe), pp 11–18. ACM. doi:10.1145/1161289.1161292
22. Sorber JM, Shin M, Peterson R, Kotz D (2012) Plug-n-Trust: practical trusted sensing for mHealth. In: Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys), pp 309–322. doi:10.1145/2307636.2307665
23. Wright CV, Ballard L, Coull SE, Monrose F, Masson GM (2010) Uncovering spoken phrases in encrypted voice over IP conversations. ACM Trans Inf Syst Secur (TISSEC) 13(4):35:1–35:30. doi:10.1145/1880022.1880029