2009

IBE-Lite: A Lightweight Identity-Based Cryptography for Body Sensor Networks

Chiu C. Tan

Qun Li
William & Mary

Haodong Wang

Sheng Zhong

Recommended Citation

Tan, C. C., Wang, H., Zhong, S., & Li, Q. (2009). IBE-lite: a lightweight identity-based cryptography for body sensor networks. IEEE Transactions on Information Technology in Biomedicine, 13(6), 926-932.

IBE-Lite: A Lightweight Identity-Based Cryptography for Body Sensor Networks

Chiu C. Tan, *Member, IEEE*, Haodong Wang, *Member, IEEE*, Sheng Zhong, *Member, IEEE*, and Qun Li, *Member, IEEE*

***Abstract*—A body sensor network (BSN) is a network of sensors deployed on a person's body for health care monitoring. Since the sensors collect personal medical data, security and privacy are important components in a BSN. In this paper, we developed IBE-Lite, a lightweight identity-based encryption suitable for sensors in a BSN. We present protocols based on IBE-Lite that balance security and privacy with accessibility and perform evaluation using experiments conducted on commercially available sensors.**

***Index Terms*—Body sensor network, identity-based encryption, privacy, security.**

I. Introduction

THE USE of wireless sensors for health care monitoring creates new ways of providing quality health care. A diverse array of specialized sensors can be deployed to monitor for instance an at-risk patient with a history of heart attacks. By continuously collecting a patient's physiological data in an unobtrusive manner, these sensors can provide doctors with additional information for better medical diagnosis. A body sensor network (BSN), is an important component in this monitoring scheme. A BSN consists of sensors placed directly on a patient's body or woven into the patient's clothes, and "travels" with the patient collecting data.

The fact that a BSN is "always on," continuously collecting data, creates additional security and privacy demands. A patient will rightly want to limit the access and scope of the collected data to different people. For the purposes of this paper, we assume the patient wishes to control data access according to the date, time, and the identity of the person who will access the data. For example, a patient may want to limit a physical therapist's access to BSN data collected on January 1st between 9 and 10 a.m., and no other times. In practice, more stringent access conditions can be adopted such as restricting access to specific locations where the data was collected.

In this paper, we focus on a BSN deployed for medical monitoring. The data collected by the BSN can be stored in the sensors themselves, on a home computer, or forwarded to a publicly accessible website. We use the term *storage site* to refer to where the data is stored. There is a certificate authority (CA) that helps a patient store and regulate access to decryption keys. Examples of possible CAs include a local police department or veteran's affairs (VA) clinic. A patient will register with a CA ahead of time, authorizing the CA to release permissions under different conditions to the appropriate personnel. A doctor wanting to obtain the data will first contact the CA for the appropriate keys, and then obtain the needed information from the storage site. We consider an adversary that seeks unauthorized access to the patient's data. These adversaries include criminal elements like identity thieves, as well as "snoopy" elements like employers. The adversary in the latter case may have *partial* access to some of the data, but may try to learn more. Details of these attacks will be elaborated later in the paper.

A. Encryption Techniques

We can provide the necessary security protections by designing the BSN to encrypt data with different keys. For instance, the data collected between 9 and 10 a.m. will be encrypted with a different key from the data collected between noon and 1 p.m. the same day. This way, the patient can assign the appropriate decryption key to different people to limit access to the information.

Symmetric Key Encryption: In symmetric key encryption, the same key is used to both encrypt and decrypt the data. So for a patient wearing a BSN that monitors a patient 24 h a day for an entire month and only wants his primary doctor to access the information, will need to store $24 \times 30 \times 1 = 720$ symmetric keys in the BSN, assuming that a different key is used every hour. If the patient wishes to control access to different people (other doctors, in-house caregiver, and so on), then more keys will have to be assigned to the BSN.

A problem occurs when the BSN or a single sensor from the BSN is stolen. When this happens, the adversary will be able to decrypt the data, since the same key is used for both encryption and decryption. One solution is to increase the number of encryption keys by letting each sensor use a different key to encrypt the data collected at the same time. For a BSN with 100 sensors, we will have $10 \times 24 \times 30 \times 1 = 72\,000$ keys in the example given above. This makes key management complicated since the decrypting party may not know in advance

Manuscript received September 29, 2008; revised January 17, 2009. First published September 29, 2009; current version published November 4, 2009. The work of C. C. Tan and Q. Li was supported in part by the US National Science Foundation under Grant CNS-0721443, Grant CNS-0831904, and CAREER Award CNS-0747108. The work of S. Zhong was supported in part by National Science Foundation under Grant CNS-0524030, Grant CNS-0845149, and Grant CCF-0915374.

C. C. Tan and Q. Li are with the Department Computer Science, College of William and Mary, Williamsburg, VA 23187-8795 USA (e-mail: cct@cs.wm.edu; liqun@cs.wm.edu).

H. Wang is with the Department of Math and Computer Science, Virginia State University, Williamsburg, VA 23806-0001 USA (e-mail: hwang@vsu.edu).

S. Zhong is with the Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY 14260-7022 USA (e-mail: szhong@cse.buffalo.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITB.2009.2033055

which key is used. For this reason, many conventional protocols such as SSL on the Internet use symmetric keys to encrypt data, but use public keys to encrypt the symmetric key before transmission.

Conventional Public Key Encryption: In conventional public key encryption like Rivest-Shamir-Adleman algorithm (RSA), two keys are used, an encryption key and a decryption key. Here, the encryption key is stored on the BSN, and the decryption key is stored safely elsewhere at a trusted location like the CA. When the BSN is compromised, the adversary will only learn the encryption key and cannot decrypt the data.

However, once the secret key is revealed, all encrypted data is vulnerable. This poses a problem when temporary access to the BSN data is needed. For instance, consider an on duty doctor wanting to access the BSN data. If only one public key is used to encrypt all the data, the doctor after learning the secret key will be able to decrypt all data even when he is off duty. A possible defense is to store many public keys in the BSN. However, this will also lead to similar key management problems as with symmetric key encryption.

Identity Based Encryption (IBE): IBE is a form of asymmetric cryptography like RSA. However, unlike RSA which requires both public and private key to be generated together, IBE allows a public key to be generated from an arbitrary string [1]. The corresponding private key can be generated separately later. For example, the patient may instruct the CA to release the keys to any ER doctor. Each day, the patient's BSN will create a new public key using the string str $= \{\text{date} \,|\, \text{time} \,|\, \text{ER}\}$. The CA does not have to create the corresponding private key. When an ER doctor wants to obtain data for January 1st between 9 and 10 a.m., he will first authenticate himself to the CA. The CA will then create the decryption key using that same string str $= \{\text{date} \,|\, \text{time} \,|\, \text{ER}\}$. This key can only decrypt data collected on that date and time.

Storing the syntax in the BSN is secure even if the sensors are compromised due to the asymmetric property of IBE. An adversary with access to a BSN sensor and knowledge of the syntax can only create a public key that cannot decrypt any information. Only the CA (or the patient himself) can create the private key to decrypt the data. Key management is also simplified, since the CA can generate a particular secret key based on the syntax, for instance date and time, on demand.

B. Our Contributions

We design protocols based on IBE that provide security and privacy protections while allowing flexible access to stored data. While IBE has been actively studied and widely applied in cryptography research, conventional IBE primitives are computationally demanding and cannot be efficiently implemented on BSN sensors. We developed IBE-Lite, a lightweight IBE suitable for a BSN. Through a proof-of-concept implementation of IBE-Lite on commercially available sensors, our experimental results show that IBE-Lite gives reasonable performance when executed by resource-constrained sensors.

The rest of the paper is as follows. Section II presents our IBE-based scheme, and Section III presents the security analysis and performance of our protocols. Related work is found in Section IV and Section V concludes.

TABLE I
SIZE OF BASIC ECC PRIMITIVES. y AND P ARE BOTH 320 BITS LONG. p, q, AND $h(.)$ ARE 160 BITS EACH

Secret key	x	160 bits
Public parameters	$(y, P, p, q, h(.))$	1120 bits

II. IBE-LITE SOLUTION

The simple examples presented in the previous section rely on conventional IBE that cannot be efficiently executed by a sensor in a BSN. Instead, we introduce IBE-Lite, a lightweight IBE that retains the properties of conventional IBE, and yet can be executed on a BSN sensor is needed. The two useful properties are the ability to use an arbitrary string to generate a public key, and the ability to generate a public key separately from the corresponding secret key. IBE-Lite is built upon elliptic curve cryptography (ECC), a public key primitive suitable for BSN [2].

To setup an ECC, we need to derive a secret key x, and public parameters $(y, P, p, q, h(.))$. Table I shows the size of these parameters in bits. For the rest of the paper, we denote encrypting a message m using public key y as ECCEncrypt(m, y), and decryption of ciphertext c generated by the ECCEncrypt using the secret key x is given as ECCDecrypt(c, x). Details for generating the parameters as well as ECCEncrypt and ECCDecrypt are found in [3].

A. IBE-Lite

From the basic ECC primitives, we derive the following IBE-Lite primitives, `setup`, `keygen`, `encrypt`, and `decrypt`.

The intuition behind using IBE-Lite is to let a sensor independently generate a public key on-the-fly using an arbitrary string. For example, a sensor collecting EKG readings on Monday 1 p.m. will first create a string str $= (\text{Monday} \,|\, 1 \text{ p.m.} \,|\, \text{EKG})$. Using this string, the sensor can derive a public key, y_{str} to encrypt the data and send it to the storage site. There is no corresponding secret key created. In fact, the sensor *cannot* create the secret key needed to decrypt the message.

When the CA wishes to release this information to a doctor, the CA will derive the corresponding secret key x_{str} by using the same string str $= (\text{Monday} \,|\, 1 \text{ p.m.} \,|\, \text{EKG})$. This secret key only allows the doctor to decrypt messages encrypted by a sensor using the same string. This simplifies the key management, since the CA can generate the secret key on-demand without keeping track of which keys were used to encrypt which data. The only requirement is that the string used to describe the event is the same. Our primitives are as follows.

`Setup`: The patient selects an elliptic curve E over $GF(p)$, where p is a big prime number. We also denote P as the base point of E and q as the order of P, where q is also a big prime. The patient then generates n secret keys $x_1, \ldots, x_n \in GF(q)$ to generate the master secret key

$$X = (x_1, \ldots, x_n). \tag{1}$$

The n public keys are then generated to make up the master public key

$$Y = (y_1, \ldots, y_n) \tag{2}$$

where $y_i = x_i P$, $1 \leq i < n$. Finally, the patient selects a collision resistant one-way hash function h: $\{0,1\}^* \rightarrow \{0,1\}^n$. The parameters

$$\langle Y, P, p, q, h(.)\rangle \tag{3}$$

are released as the system public parameters.

`Keygen`: To derive a secret key x_{str} corresponding to a public key generated by a string str, the patient executes $\texttt{Keygen}(\text{str}) = x_{\text{str}}$

$$x_{\text{str}} = \sum_{i=1}^{n} h_i(\text{str}) x_i \tag{4}$$

where $h_i(\text{str})$ is the ith bit of $h(\text{str})$.

`Encrypt`: To encrypt a message m using a public key derived from string str, the sensor does $\texttt{Encrypt}(m, \text{str})$ to determine the ciphertext c. Algorithm 1 shows the process. Note that Algorithm 1 lines 1 and 2 need only be run once to derive the public key y_{str}.

Algorithm 1 $\texttt{Encrypt}(m, str)$

1: Determine string str using agreed upon syntax
2: Generate public key y_{str} where $y_{str} = \sum_{i=1}^{n} h_i(str) \cdot y_i$
3: Execute $EccEncrypt(m, y_{str})$ to obtain c

`Decrypt`: The doctor executes $\texttt{Decrypt}(c, x_{\text{str}})$ to obtain the original message m which was encrypted using a secret key derived from str. The process is shown in Algorithm 2.

Algorithm 2 $\texttt{Decrypt}(c, str))$

1: Requests permission from CA to obtain data described by str
2: CA runs $\texttt{Keygen}(str)$ to derive x_{str}
3: Doctor executes $EccDecrypt(c, x_{str})$ to obtain m

B. BSN Security Protocols

Here we describe the protocols built upon IBE-Lite. First is the initialization phase where the patient first uses the BSN. Next is the data collection phase, which outlines how a sensor encrypts the collected data. This is followed by the data transfer phase that describes how a BSN transfers data to a storage site. Finally, the query phase which occurs when a doctor needs to obtain data from the storage site.

We assume that an agreed upon syntax is used to describe the public key, and this description is termed as str. For example, the patient deciding to collect data on a hourly basis will set the sensors in the BSN to affix a timestamp rounded to the nearest hour when creating str. In other words, two EKG readings collected on Monday at 1:05 p.m. and 1:20 p.m. will both be described using the same string $\text{str} = \{\text{Monday} \,|\, 1 \text{ p.m.} \,|\, \text{EKG}\}$.

As mentioned earlier, we assume an honest-but-curious storage site, which will try to learn the contents of the stored data, but will otherwise not delete the stored data. We also assume a separate security mechanism is in place so that only the patient can store BSN data onto the storage site.

Initialization: The patient first executes `Setup` to obtain the master secret key $X = (x_1, \ldots, x_n)$, and public parameters $\langle Y, P, p, q, h(.)\rangle$. The patient loads the parameters $\langle Y, P, p, q, h(.)\rangle$ into every sensor in the BSN. The master secret key is registered with the CA.

Data Collection: Let the sensor collect data d at event str. The sensor executes Algorithm 3 to encrypt its data.

Algorithm 3 Sensor encrypting data

1: Derive the string str, and generate a random number n
2: Calculate $m_1 = (flag|n)$ where $flag$ is a known bitstring
3: Calculate $m_2 = (d|n)$
4: Calculate $c_1 = \texttt{Encrypt}(str, m_1)$
5: Calculate $c_2 = \texttt{Encrypt}(str, m_2)$

The tuple (c_1, c_2) is then stored in sensor memory. The flag is a commonly known bitstring several bits long.

Data Transfer: Periodically, each sensor in the BSN will transfer its data to the storage site. This is done by first aggregating all the data into a cellphone like device [4]. The cellphone then forwards the aggregated data to the storage site. Assuming that there are k tuples generated by the BSN, the cellphone will forward the set $\{(c_1^1, c_2^1), \ldots, (c_1^k, c_2^k)\}$. Alternatively, a sensor with enough storage capacity can opt to store the data within the sensor itself. In this case, there is no data transfer process.

Querying: A doctor wishing to obtain data collected under some str will first contact the CA for permission. After the CA agrees, the CA will run `Keygen(str)` to derive the corresponding secret key x_{str} needed to decrypt data.

The doctor then contacts the storage site and retrieves the data as shown in Algorithm 4. When the data is stored within the sensor, the role of the storage site will be executed by the sensor.

Algorithm 4 Doctor querying for data

1: **for** every (c_1^i, c_2^i) $i \in k$ for patient **do**
2: Storage site sends c_1^i to doctor
3: Doctor runs $\texttt{Decrypt}(c_1^i, str)$
4: **if** the initial bits of the result match $flag$ **then**
5: Doctor requests corresponding c_2^i from storage site
6: Doctor executes $\texttt{Decrypt}(c_2^i, str)$ and checks whether the n matches the value from c_1^i
7: Doctor accepts d if both are correct
8: **end if**
9: **end for**

Since all the data is encrypted, the storage site cannot return a specific encrypted data to the doctor. Instead, the storage site simply lets the doctor try to decrypt each tuple (c_1, c_2) belonging to the patient. The reason the storage site first returns c_1 for the doctor to verify instead of returning c_2 directly is to improve

efficiency. Since the length of c_1 is much shorter than that of c_2, letting the doctor first attempt to decrypt c_1 before sending the much longer c_2 reduces transmission time.

Notice that c_2 embeds the same random number n in both c_1 and c_2, and the doctor will only accept the data in c_2 to be legitimate only if both random numbers match. This random n is known only to the sensor encrypting the data. Consider for example two sensors belonging to the same BSN encrypting some data using the same string str. Since both sensors are legitimate, the use of the random n prevents an adversary from swapping the c_2s from different sensors to confuse the doctor.

C. Query Improvements

A potential bottleneck is the amount of time needed for a doctor to query a storage site. Consider a storage site with k tuples $(c_1^1, c_2^1), \ldots, (c_1^k, c_2^k)$, and a doctor receives l secret keys from the patient. The storage site will have to transmit $c_1^i, i \in k$ to the doctor, and the doctor will have to try every key $x_j, j \in l$ on each c_1^i to determine whether there is any desired data in the storage site. This takes $O(kl)$ amount of time.

This poor performance is because the storage site is unable to index any of the tuples since the storage site cannot determine the actual content of the tuples. This feature protects the privacy of the patient at the cost of slower searching time. For instance, consider a storage site have many tuples belonging to the same patient, and one of the tuples is encrypted using the string str $= \{\text{date} \mid \text{ER}\}$. An ER doctor with the corresponding secret key will still have to go through every tuple in the storage site to determine whether that single tuple. This is inefficient when a storage site contains many different tuples.

We can improve the search performance by letting the sensor encrypt additional hints about the tuples. This hint is a variable that can summarize several tuples together. For example, the sensor may have created two tuples (c_1^1, c_2^1) and (c_1^2, c_2^2) using two different descriptions $\{\text{date} \mid \text{ER}\}$ and $\{\text{date} \mid \text{gym}\}$. Since both descriptions contain the same condition *date*, we can create a hint $\eta = \texttt{Encrypt}(m, \text{str})$ where

$$m = (\text{flag}|n|i_1^1|i_1^2) \tag{5}$$

and str $= \{\text{date}\}$. Here i_1^1 and i_1^2 refer to the indices pointing to c_1^1 and c_1^2.

Now the doctor requesting permissions will get an extra key from the CA for the date to decrypt the hint. The use of hints improve the performance by reducing the number of transmissions between the storage site and the doctor, since the doctor will only request c_1s from hints he can decrypt. This scheme is still secure since the doctor still needs the correct key x_{str} to decrypt a particular c_1. The privacy of the patient is still protected from the storage site since the storage site learns nothing from the hints.

III. Analysis and Evaluation

A. Security

Here, we analyze the security of our proposed protocols. Encryption and decryption are performed using the keys x_{str} and y_{str} derived from string str. Both x_{str} and y_{str} do not violate the discrete logarithm property where, given $y = xP$, it is infeasible to determine x given y and P, since both are derived from addition of points on the same curve.

Eavesdropping Attack: In this attack, the adversary eavesdrop on the message transmitted from the BSN to the storage site and learns the tuple (c_1, c_2). The adversary succeeds in his attack if he is able to determine the data d after observing as many tuples as he wishes. Since our protocol encrypts all data before broadcast, the adversary learns nothing from the ciphertext.

Tracking Attack: Here, the adversary attacks the patient's privacy by observing multiple transmission between a BSN and a storage site. The adversary is considered able to track the patient if given two tuples, the adversary is able to determine whether they come from the same BSN. In our protocol, each ciphertext (c_1, c_2) includes a new random number n. In fact, even if *identical* data encrypted using a public key derived from the same string str in two different broadcasts cannot be linked together since a different random n will be used. This is important when the BSN monitors data such as body temperature that may remain relatively static for long periods of time.

Compromised Sensor: We assume that the adversary compromises one or more sensors in the BSN, and is able to extract all data that is stored on the sensor. The adversary succeeds in this attack if he is able to use the information to determine previously encrypted data. Since each sensor only stores the public parameters $(Y, P, p, q, h(.))$, the adversary learns no secret knowledge that can enable him to decrypt any tuples (c_1, c_2).

Matching Attack: The adversary launches a matching attack by first creating many public keys using different strings str. The adversary then encrypts all possible values using the different public keys to determine whether there is a match for the tuple (c_1, c_2). This is possible since the number of potential EKG readings for example are bounded. However, both c_1 and c_2 contains a random number n generated by the sensor. Since the adversary cannot predict the value of n, the matching attack fails.

Honest-But-Curious Storage Site: This type of storage site will not delete the user's data but may attempt to determine the contents of the data. This assumption is common for many web-based applications. For instance, an e-mail service provider can generally be assumed to *not* delete the user's e-mails, but may try to use some of the content to place advertisements. This requirement also covers instances where the storage site is compromised and data exposed to an adversary. In our protocol, all data stored on the storage site is encrypted, and no secret keys are stored in the storage site. Therefore, an adversary with access to all the ciphertexts cannot decrypt the data.

Note that a malicious storage site can still cause disruption by deleting the patient's data. While our protocols do not prevent this, a practical defense is to store the same encrypted data at different storage sites so that the data is still recoverable.

Complexity Analysis: Given n public keys $Y = (y_1, \ldots, y_n)$, the time complexity for using IBE-Lite to generate a public key y_{str} using string str is $O(n)$, and the time needed to perform the encryption with y_{str} is $O(1)$. Similarly, the decryption requires

a time complexity of $O(n)$ to generate the secret key $x_{\rm str}$ and $O(1)$ to decrypt the data.

Since our protocols rely on asymmetric key encryption, only public keys are stored in the sensors. Thus, our schemes are resistant against an adversary that can compromise all sensors in the BSN. However, our protocols are vulnerable to attack if there are $O(n)$ colluding users each with a single secret key $x_{\rm str}$. The colluding users can use their individual secret keys to derive the master secret key X given in (1). This vulnerability can be defended by the rekeying process given in the following.

B. Limitations

A limitation of our scheme is that we can only release n secret keys $x^1_{\rm str}, \ldots, x^n_{\rm str}$. Once more than n secret keys are released, the master secret key X (in 1) is vulnerable to compromise. The idea is that since each secret key $x^i_{\rm str}$ is computed from a linear combination of the master secret key X [shown in (4)], an adversary that can obtain n secret keys $x^1_{\rm str}, \ldots, x^n_{\rm str}$ can solve for the master secret key X.

While conventional IBE systems [1] do not have this limitation, such systems cannot be implemented on BSN hardware. IBE-Lite allows us similar properties as conventional IBE so long as a limited number of secret keys are released. We believe that this is a reasonable tradeoff due to the following reasons.

First, as a practical matter, since a secret key $x^i_{\rm str}$ is only released to a trusted party, such as a doctor verified by the CA, it is less likely that the secret key will be abused. Even if the trusted party is later found to be an adversary and attempts to colludes with others to derive the master secret key, the adversary will still need to determine the identity of these "others." Since the CA is responsible for issuing secret keys, only the CA knows the identities of the other holders of secret keys. Given the CA is always trustworthy, the adversary cannot easily determine which other doctors have the remaining keys.

Second, we can select a large enough n such that we will never release n secret keys. As we will show in later, we can easily let n be a few thousand keys without incurring heavy storage penalty. For instance, for $n = 500$, the BSN can still encrypt more than 500 pieces of data so long as less than 500 secret keys are released. This is unlike a symmetric key solution where each piece of encrypted data requires a new key. This distinction is important because a BSN that is continuously worn by a patient will always collect information, but only a subset may be ever be used. Since we cannot determine in advance what data will be requested, we have to ensure we have enough keys to encrypt everything.

Finally, we can rekey the BSN by creating a new set of n secret keys as the master secret key, and store the new public information in the sensors. This rekeying does *not* have to be done by the BSN itself. A powerful laptop can be used to generate these keys, and the information then stored into the BSN and CA. This is akin to changing a password for a bank account online. To reduce the rekeying frequency, the CA can be configured to inform the patient when to rekey his BSN after it has released a certain number of secret keys, thus avoiding unnecessary rekeying.

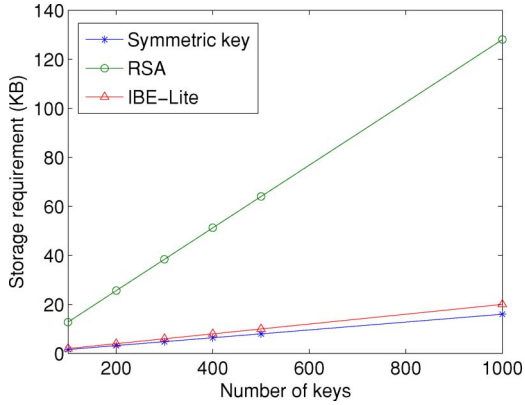


Fig. 1. Amount of storage needed to store n keys for different encryption methods.

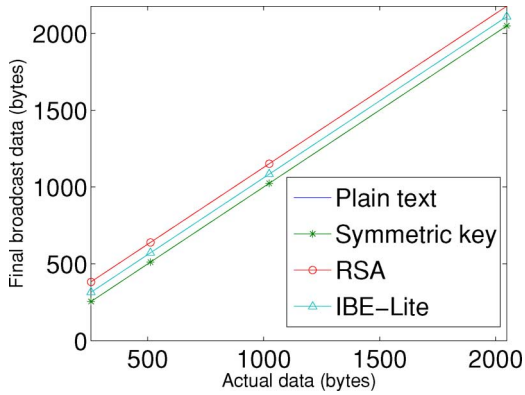


Fig. 2. Data transmission overhead for different encryption schemes. All values in bytes.

C. Performance

We evaluate our protocols using experiments conducted on commercially available Tmote Sky sensors. The sensor has a 8 MHz TI MSP430 CPU, 10 KB on-chip RAM, 48 KB programming ROM, and a 802.15.4/ZigBee radio.

Fig. 1 shows the amount of storage needed for different encryption schemes. We represent a conventional asymmetric key encryption scheme using RSA. We see that a symmetric key encryption requires the least amount of storage, while RSA encryption uses the most amount of storage. Our IBE-Lite gives us the advantages of asymmetric key encryption while using a little more storage space than a symmetric key scheme.

Fig. 2 shows the data transmission overhead of the various encryption schemes. For both the RSA and IBE-Lite, the data itself is not encrypted. Instead, a symmetric key is first used to encrypt the data, and then the asymmetric key is used to encrypt the data. This is the conventional method when designing protocols using asymmetric key encryption.

The main overhead of IBE-Lite over other encryption schemes is the time needed to generate an encryption key $y_{\rm str}$ from a string str using n number of public keys $y_1, \ldots, y_n$. In both symmetric key and RSA, the public keys are precomputed and stored in the sensor. Fig. 3 shows the amount of time needed to generate a single $y_{\rm str}$ with varying values of n. All

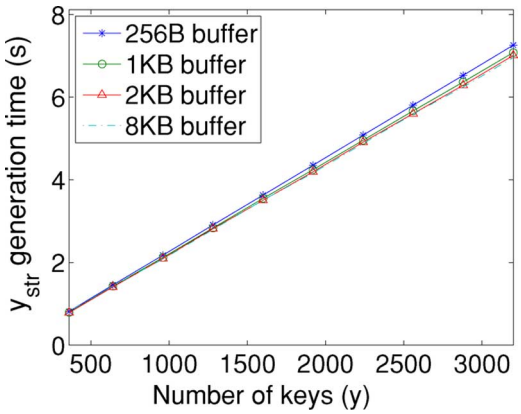


Fig. 3. Time needed to derive one y_{str} using different n number of public keys, $y_1, \ldots, y_n$.

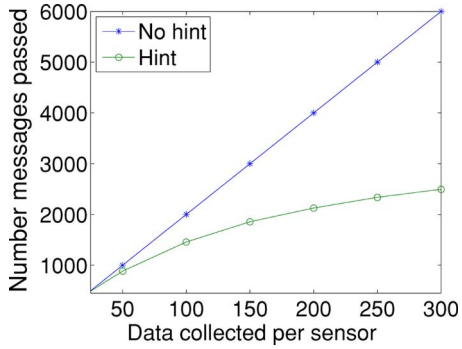


Fig. 4. Number of passed messages.

n public keys are initially stored in the flash memory. We see for instance that for $n = 360$, we need only 0.9 s to generate y_{str}. While IBE-Lite does require an additional key generation time, we note that we can achieve the properties of asymmetric key encryption like RSA using a much small amount of storage space. In addition, the amount of time a single public key can be used is typically much longer than the time needed to generated a new key.

We use simulation to evaluate the search improvements. Since the searching is performed by a doctor with access to a more powerful machine, the simulations focused on the number of messages passed between a doctor and a storage site. We assume there are 1000 different possible time periods where a sensor may collect data. A sensor randomly selects a time period to collect data, and encrypts the data using a public key derived from that time period. The doctor is assumed to randomly select data from five time periods. The results are the average over 100 trials. Fig. 4 shows the improvement when hints are used.

IV. Related Papers

The motivation behind BSNs is to place low-cost sensors directly on the patient for health care monitoring, and several research prototypes have been developed [4], [5]. Our paper differs from these in that we focus on the security issues in BSNs.

IBE is a relatively new type of asymmetric key encryption [1], [6]. Zhang *et al.* [7] developed protocols using IBE on sensors used in large scale sensor networks. The sensors used in a BSN have to worn on the patient, and are likely to be smaller and weaker. Thus unlike [7], our paper uses IBE that does not rely on bilinear pairings such as Weil or Tate pairings in our primitives. Practical public key encryption for sensors have been proposed by [8]–[10], but these research only focus on conventional public keys, and do not support the IBE properties mentioned in this paper.

Mont *et al.* [11] uses IBE in a medical setting to secure the communications in a hospital, and Malasri and Wang [12] considered a security architecture for BSN focusing on the problem of key exchange between a sensor and a base station. The main difference between these work and our research is that we focus on deploying IBE on resource constrained devices to achieve practical performance. A key distinction is that our work presents evaluation results based on actual hardware.

Bao *et al.* [13] proposed a secure system for BSNs using symmetric keys. While symmetric key schemes use less storage space per key and generate a smaller ciphertext, they do not have the asymmetric property of public keys schemes like RSA or IBE-Lite. Our paper takes advantage of recent advances that allows us to perform faster computation than earlier sensor hardware platforms. Another paper by Bao *et al.* [14] uses the variability of a patient's heart rate as a means of person authentication. This paper complements our IBE-Lite encryption since the patient's heart rate can be used as an input string to generate encryption keys. Tan *et al.* [15] also consider using IBE on BSN, but suffers from slow query performance when searching over a lot of ciphertext.

V. Conclusion

In this paper, we presented IBE-Lite, a lightweight IBE method suitable for a BSN. We provided protocols based on IBE-Lite and evaluated the protocols using a combination of security analysis, simulations, and practical implementation on actual sensors.

Acknowledgment

The authors would like to thank the anonymous reviewers whose valuable comments significantly improved this manuscript.

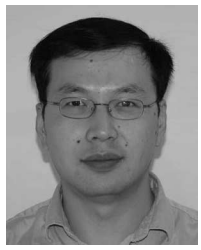
References

[1] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *Proc. CRYPTO*, 2001, pp. 213–229.
[2] B. Lo and G. Z. Yang, "Key technical challenges and current implementations of body sensor networks," in *Proc. Body Sensor Networks (BSN)*, 2005, pp. 1–5.
[3] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, pp. 203–209, 1987.
[4] L. Zhong, M. Sinclair, and R. Bittner, "A phone-centered body sensor network platform: Cost, energy efficiency and user interface," in *Proc. BSN*, 2006, pp. 179–182.
[5] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton, "Codeblue: An ad hoc sensor network infrastructure for emergency medical care," presented at the BSN, London, U.K., 2004.

[6] C. Cocks, "An identity based encryption scheme based on quadratic residues," in *Proc. LNCS*, 2001, vol. 2260, pp. 360–363.
[7] Y. Zhang, W. Liu, W. Lou, and Y. Fang, "Location-based compromise-tolerant security mechanisms for wireless sensor networks," in *Proc. IEEE J. Sel. Areas Commun.*, 2006, pp. 247–260.
[8] H. Wang and Q. Li, "Efficient implementation of public key cryptosystems on mote sensors (short paper)," presented at the Int. Conf. Inf. Commun. Security (ICICS), Raleigh, NC, 2006.
[9] H. Wang, B. Sheng, and Q. Li, "Elliptic curve cryptography based access control in sensor networks," *Int. J. Security Netw.*, vol. 1, no. 3/4, pp. 127–137, 2006.
[10] H. Wang, B. Sheng, C. C. Tan, and Q. Li, "Comparing symmetric-key and public-key schemes in sensor networks," in *Proc. IEEE ICDCS*, 2008, pp. 11–18.
[11] M. Mont, P. Bramhall, and K. Harrison, "A flexible role-based secure messaging service: Exploiting IBE technology for privacy in health care," in *Proc. Int. Workshop Database Expert Syst. Appl.*, 2003, pp. 432–437.
[12] K. Malasri and L. Wang, "Addressing security in medical sensor networks," in *Proc. HealthNet*, 2007, pp. 7–12.
[13] S.-D. Bao, Y.-T. Zhang, and L.-F. Shen, "A new symmetric cryptosystem of body area sensor networks for telemedicine," in *Proc. Conf. Jpn. Soc. Med. Electron. Biol. Eng.*, 2005, p. 654.
[14] S.-D. Bao, Y.-T. Zhang, and L.-F. Shen, "Physiological signal based entity authentication for body area sensor networks and mobile healthcare systems," in *Proc. IEEE Eng. Med. Biol.*, 2005, pp. 2455–2458.
[15] C. C. Tan, H. Wang, S. Zhong, and Q. Li, "Body sensor network security: An identity-based cryptography approach," in *ACM Conf. Wireless Security (WiSec)*, 2008, pp. 148–153.

Chiu C. Tan (M'07) received the B.S. degree in computer science, and the B.A. degree in economics (honors) from the University of Texas, Austin, in 2004.

He is currently a Graduate Research Assistant in the Department of Computer Science, College of William and Mary, Williamsburg, VA. His research interests include ubiquitous computing, embedded systems, large scale radio frequency identification systems, vehicular networks, and wireless security.



Haodong Wang (M'08) received the Ph.D. degree in computer science from the College of William and Mary, Williamsburg, VA, in 2009.

He is currently an Assistant Professor in the Department of Mathematics and Computer Science, Virginia State University, Williamsburg. His research interests include information assurance in cyber-physical systems, privacy preserving and user access control in sensor networks, efficient information storage, search and retrieval in pervasive computing, and MAC design in IEEE802.11 wireless local area network.

Sheng Zhong (M'07) received the B.S. and M.S. degrees from Nanjing University, Nanjing, China, in 1996 and 1999, respectively, and the Ph.D. degree from Yale University, New Haven, CT, in 2004, all in computer science.

He is currently an Assistant Professor in the Department of Computer Science and Engineering, State University of New York, Buffalo. His current research interests include cooperation and security problems in distributed computing.

Qun Li (M'05) received the Ph.D. degree in computer science from Dartmouth College, Hanover, NH, in 2004.

He is currently an Associate Professor in the Department of Computer Science, College of William and Mary, Williamsburg, VA. His research interests include wireless networks, sensor networks, radio frequency identification, and pervasive computing systems.

Dr. Li received the National Science Foundation Career Award in 2008.