Body Sensor Network Security: An Identity-Based Cryptography Approach

Chiu C. Tan
College of William and Mary
cct@cs.wm.edu

Haodong Wang
College of William and Mary
wanghd@cs.wm.edu

Sheng Zhong
SUNY at Buffalo
szhong@cse.buffalo.edu

Qun Li
College of William and Mary
liqun@cs.wm.edu

ABSTRACT

A body sensor network (BSN), is a network of sensors deployed on a person's body, usually for health care monitoring. Since the sensors collect personal medical data, security and privacy are important components in a body sensor network. At the same time, the collected data has to readily available in the event of an emergency. In this paper, we present IBE-Lite, a lightweight identity-based encryption suitable for sensors, and developed protocols based on IBE-Lite for a BSN.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Security

Keywords

Security, Body Sensor Networks, Identity-based cryptography

1. INTRODUCTION

Applying wireless sensors toward health care monitoring allows for new ways to provide quality health care to patients. A diverse array of specialized sensors can be deployed to monitor, for instance, at-risk patients with history of heart attacks, or senior citizens living independently at home. These sensors provide continuous, long term monitoring in an unobtrusive manner, allowing doctors to diagnose problems more effectively.

A body sensor network, or BSN, is a network of sensors deployed on a person's body to collect physiological information. In this paper, we focus on a BSN deployed for medical monitoring. We term the person wearing the BSN as the *patient*, and the person access the data as the *doctor*. The term "doctor" is used loosely, and refers to any person wanting to access the data. The data collected by the BSN is either stored on the sensors, or forwarded and archived on publicly accessible site known as the *storage site*. Figure 1 illustrates a BSN.

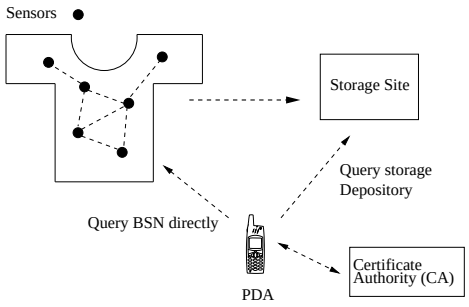

Figure 1: A Body Sensor Network

1.1 Motivating example

Privacy and security for a BSN is important since the data collected is directly associated with a particular patient. At the same time, the data must be easily accessible to relevant personal in the event of an emergency. The following scenarios serve to better illustrates such concerns.

1. Alice wears a BSN that monitors her EKG data when she is working out. One day, Alice suddenly falls unconscious and is sent to the emergency room. The data collected by the BSN should be stored in a public place and made easily accessible in an emergency.
2. After the incident, Alice instructs her BSN to collect some additional data. Alice would like to restrict this information to only physicians in an ER. However, Alice cannot predict which doctor or hospital will treat her. Alice may not be physically competent to authenticate anybody when she is admitted to a hospital. A BSN security scheme should be able to tolerate this form of ambiguity.
3. Furthermore, due to privacy reasons, a doctor requiring two days worth of data prior to Alice's illness should only be able to obtain data collected within those two days. However, since Alice does not know when she might have a relapse, a BSN should be able to limit access to the collected data, even when the data is stored in a public space.

WiSec'08, March 31–April 2, 2008, Alexandria, Virginia, USA.

4. Alice's family and friends are worried about her condition. To allay their concerns, Alice would like some of her family members to be able to have partial access to her BSN data. Since the data collected by the BSN belongs to Alice, a security scheme should be flexible enough to allow Alice to easily add additional access permissions to people she chooses.

1.2 BSN security requirements

While a complete BSN system will have many different components, a crucial factor is that any security design must be lightweight enough to be executed by a sensor in a BSN since a sensor can be lost or stolen, leaving the data stored within the sensor exposed.

From the scenario presented, we derive the following security and privacy requirements for a BSN.

1. Protect patient privacy from the storage site. Since the data is stored on a third party storage site, we cannot trust the storage site with the data. We assume an honest-but-curious storage site that will not maliciously delete the data, but may attempt to learn the contents of a patient's data.
2. Tolerate compromised BSN sensors. The BSN sensors may be misplaced or stolen, and a compromised BSN sensor should not allow an adversary from obtaining the patient's data.
3. Prevent unauthorized access to information. This includes a doctor with permissions to access some data and not others. We assume that a doctor may attempt to obtain additional data about a patient beyond what was authorized. Since storage site is not trusted, access control can only be performed by the CA.
4. Flexibility in granting permissions. The patient may decide to allow different people to access the BSN data, and the BSN should be able to generate keys on-the-fly without additional interaction with the CA.

1.3 Identity-based encryption

Our solutions are based on a type of cryptographic primitive known as identity-based encryption (IBE). After an initial setup phase, IBE allows a public key to be generated from an arbitrary string. The corresponding secret key can be derived separately by a trusted party. For example, Alice may want to encrypt a message for the doctor in charge on Monday. Alice can independently generated a public key using the strings "Monday" and "doctor" to encrypt her data without further contact with the trusted party. To decrypt the message, a doctor will have to convince the trusted party that he is a doctor in charge on Monday. The same doctor working on Tuesday cannot decrypt messages encrypted using the string "Tuesday" and "doctor" even if he knows the secret key from Monday.

The simple example cannot be easily accomplished without using IBE. Alice can try to generate many public/secret key pairs, one for each occasion. However, Alice will have to store the secret key created with the trusted party *each* time a new public/secret key pair is generated. Otherwise, the trusted party cannot derive the secret key on its own. This is inefficient.

Another possible alternative is for Alice to include some instructions with each of her message, and encrypt everything with the trusted parties public key. When a doctor receives an encrypted message, the doctor will forward it to the trusted party. The trusted party can decrypt can obtain Alice's instructions. The trusted party will release the message to the doctor only if he meets Alice's instructions. This solution is also inefficient since a BSN may generate many pieces of data, each of which has to be forwarded to the trusted party for decryption. Using IBE, the doctor only needs to be given a single secret key once.

1.4 Our contributions

In this paper, we design protocols based on identity-based encryption (IBE) that provide security and privacy protections to a body sensor network, while allowing flexible access to stored data. The use of IBE in a medical setting has been proposed by several researchers [22, 21, 19]. However, conventional IBE cannot be efficiently implemented on sensors used in a BSN. In this paper, we propose IBE-Lite, a lightweight IBE scheme that is suitable for sensors. We implement a proof-of-concept of our schemes based on IBE-Lite on commercially available sensors similar to the ones used in a BSN. While IBE schemes have been suggested by previous researchers to protect medical data, we are the first to present a lightweight IBE suitable for body sensor networks.

The rest of the paper is as follows. The next section presents our protocols and Section 3 contains the security analysis. Our schemes are evaluated in Section 4, with related work presented in Section 5. Section 6 concludes.

2. OUR SOLUTION

Our protocols are based on a lightweight IBE scheme IBE-Lite. IBE-LIE shares two properties with conventional IBE, namely 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. We begin by first reviewing Elliptic Curve Cryptography (ECC), a public key primitive suitable for BSN [18], followed by the modifications made to derive IBE-Lite. Finally, we present our protocols based on IBE-Lite.

2.1 Basic ECC Primitives

To setup ECC, we first select a particular 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. We then pick a secret key x, and the corresponding public key y, where $y = x \cdot P$, and a cryptographic hash function $h()$. Finally, we have the secret key x and public parameters $(y, P, p, q, h(.))$.

We denote encrypting a message m using public key y as $EccEncrypt(m, y)$. The resulting ciphertext is denoted by c. The decryption of ciphertext c using the secret key x is given as $EccDecrypt(c, x)$. The algorithms for EccEncrypt and EccDecrypt are found in Alg. 1 and Alg. 2 respectively.

Algorithm 1 EccEncrypt(m, y)

1: Generate a random number $r \in GF(p)$. Encrypt m with r, $E_r(m)$
2: Calculate $A_r = h(r) \cdot y$
3: Calculate $B_r = h(r) \cdot P$
4: Calculate $\alpha_r = r \oplus \chi(A_r)$, where $\chi(A_r)$ is the x coordinate of A_r
5: Return ciphertext $c = \langle \alpha_r, B_r, E_r(m) \rangle$

Algorithm 2 EccDecrypt(c, x)

1: Calculate $x \cdot B_r = x \cdot h(r) \cdot P = h(r) \cdot y = A_r$
2: Determine the x coordinate, $\chi(A_r)$
3: Derive symmetric key r with $\alpha_r \oplus \chi(A_r) = r \oplus \chi(A_r) \oplus \chi(A_r) = r$
4: Apply r to $E_r(m)$ to return m

2.2 IBE-Lite

From the basic ECC primitives, we derive the following IBE-Lite primitives, `setup`, `keygen`, `encrypt` and `decrypt`. For ease of explanation, we assume in this subsection that all primitives are executed by the patient. The actual protocols involving the patient, CA and doctor are explained in the next subsection.

The intuition behind 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 pm will first create a string $str = (monday|1\ pm|EKG)$. 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 patient wishes to release this information to a doctor, the patient can derive the corresponding secret key, x_{str}, by using the same string $str = (monday|1pm|EKG)$. 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 patient 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.

`Setup:` We select 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. A set of n secret keys $x_1, \cdots, x_n \in GF(q)$ is chosen to generate the master secret key,

$$X = (x_1, \cdots, x_n).$$

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

$$Y = (y_1, \cdots, y_n)$$

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

$$\langle Y, P, p, q, h(.) \rangle$$

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 `Keygen`$(str) = x_{str}$,

$$x_{str} = \sum_{i=1}^{n} h_i(str) \cdot x_i,$$

where $h_i(str)$ is the i-th bit of $h(str)$.

`Encrypt:` To encrypt a message m using a public key derived from string str, the sensor does `Encrypt`(m, str) to determine the ciphertext c. Alg. 3 shows the process. Note that Alg. 3 lines 1 and 2 need only be run once to derive the public key y_{str}.

Algorithm 3 `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 $Decrypt(c, x_{str})$ to obtain the original message m which was encrypted using a secret key derived from str. The process is shown in Alg. 4.

Algorithm 4 `Decrypt`(c, str))

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

2.3 BSN Security Protocols

We first describe the initialization phase where the patient configures the BSN, followed by the data collection phase which outlines how a sensor encrypts the collected data. The data transfer phase describes how a BSN transfers data to a storage site, and 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 string needed to derive a 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 pm and 1:20 pm will both be described using the same string $str = \{monday|1\ pm|EKG\}$.

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 $X = (x_1, \cdots, x_n)$, 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 patient then registers the master secret key together with additional instructions with the CA.

Data collection: Let the sensor collect data d at event str. The sensor executes Alg. 5 to encrypt its data. The

Algorithm 5 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 $c_1 =$`Encrypt`(str, m_1)
5: Calculate $c_2 =$`Encrypt`(str, m_2)

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 [28]. 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 $\{(c_1^1, c_2^1), \cdots, (c_1^k, c_2^k)\}$. 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 Alg. 6. When the data is stored within the sensor, the role of the storage site will be executed by the sensor.

Algorithm 6 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 `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 `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 tuple 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 for returning c_1 to the doctor first instead of returning c_2 directly is to improve efficiency. Since the length of c_1 is much shorter than c_2, letting the doctor first attempt to decrypt c_1 before sending the much longer c_2 reduces transmission time.

The doctor can check if the data obtained from the storage site belongs to his patient by checking whether the same random number n is used in both c_1 and c_2. Since this random n is known only to the sensor encrypting the data, only that sensor can embed the same n in both c_1 and c_2.

3. SECURITY ANALYSIS

We begin by examining the basic primitives, followed by an analysis of the protocols themselves.

3.1 Analysis of Basic Primitives

Our `Setup` is similar to that of the basic ECC setup scheme, except that instead of picking a single secret x, our `Setup` picks n secrets and n corresponding public keys. Knowing only one x_{str} and $h(str)$, the doctor cannot determine the patient's master secret X since there are n unknown x_i. The doctor is only able to determine X when he has in this possession n different secret keys $x_{str}^1, \cdots, x_{str}^n$.

The use of x_{str} and y_{str} as the private key and public key derived from string *str* does not violate the discrete logarithm property of ECC where, given a $y = x \cdot P$, it is infeasible to determine x given y and P, since both are simply the result of addition of points. Also, both `Encrypt` and `Decrypt` are secure since both rely on well established ECC encryption and decryption methods.

3.2 Analysis of Protocols

Our protocols protects the privacy of the patient's data by encrypting all the information before forwarding the data to the storage site. After a sensor collects the data, the sensor encrypts the data using `Encrypt`, resulting in the tuple (c_1, c_2). The storage site receives an aggregated set of tuples $\{(c_1^1, c_2^1), \cdots, (c_1^k, c_2^k)\}$. Since all tuples are in ciphertext, the storage site learns nothing about the patient's data.

The protocols also prevent unauthorized access to the patient's data. Each piece of data collected by a sensor is encrypted with a y_{str}, the public key derived from the string *str*. When the doctor receives permissions to access data encrypted under *str*, the doctor receives the secret key x_{str}, which cannot be used to decrypt any other ciphertext not encrypted using y_{str}.

A compromised sensor does not allow the adversary to obtain any useful data about a patient from the storage site since the sensor only stores the publicly known parameters. At most the adversary obtains the ciphertext pair (c_1, c_2). The adversary can try to launch 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.

Finally, our protocols provide flexibility. The string *str* can be used to specify access to the data, without using additional certificates. For instance, consider the string $str = \{Date \mid ER \mid Doctor\}$ used to encrypt data. A doctor wanting to obtain the corresponding secret key will have to convince the CA that he is indeed an ER doctor on the given date. The process of specifying what *str* to construct can be programmed by the patient without additional permissions from the CA.

3.3 Additional Discussion

As mentioned earlier, a doctor knowing one x_{str} and $h(str)$ cannot determine the master secret X. However, when the doctor receives n secret keys $x_{str}^1, \cdots, x_{str}^n$, the doctor is able to solve for X. We can prevent such an attack by selecting a large enough n and periodically rekeying the BSN.

The value of n is limited by the storage capacity of an individual sensor in the BSN. As we will show in the evaluation section, a typical sensor used in a BSN can accommodate an n of several hundred public keys with reasonable performance. Since the BSN is worn on the patient's body, the patient can rekey the BSN by periodically creating a new master secret X and derive the new public parameters for the BSN. The frequency of the rekeying is related to the number of secret keys given out by the patient.

4. EVALUATION

We evaluate our protocols using experiments conducted on commercially available sensor hardware. Since every sensor has to perform certain cryptographic operations independently from each other, we are primarily interested in the performance of a sensor within a BSN, rather than the performance of an entire BSN.

We use the Tmote Sky motes as the sensors that make up our BSN. The Tmote Sky sensor has a 8MHz TI MSP430 CPU, 10KB on-chip RAM, 48KB programming ROM, and 1MB permanent flash storage. A portion of the flash memory to store the master public keys Y, thus enabling us to

evaluate IBE-Lite with a larger set of keys, i.e. with larger values of n. We show in our subsequent evaluation that the processing delay due to loading keys from flash memory to RAM is negligible. Radio communication is performed with a 802.15.4/ZigBee radio, and an integrated antenna provides up to 125 meter radio transmission range.

The IBE-Lite primitives are based on Elliptic Curve Cryptography (ECC). Optimizations to improve ECC performance can be found in [10, 15, 26, 27]. Due to space constraints, we omit further discussion on the ECC primitives. Table 1 briefly summerizes our performance. For details, please refer to [27].

PubKey	Sign	Verify	Code	Data
0.74 s	0.77 s	1.12 s	25 KB	1.6 KB

Table 1: PubKey refers to the time taken to generate a public key, Sign, time taken to generate a signature. Verify refers to the time taken to verify a signature verification time. Code and Data is the size of the binary code and data respectively.

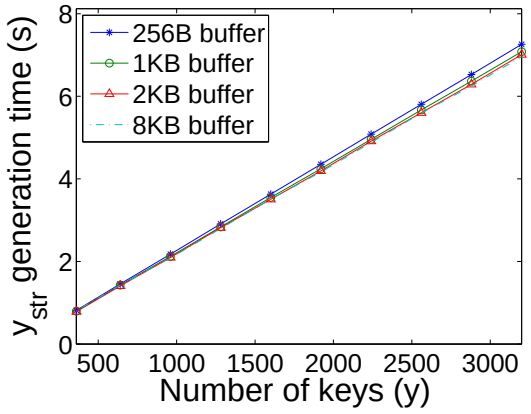

Figure 2: Time needed to derive one y_{str} using different n number of public keys, $y_1, \cdots, y_n$.

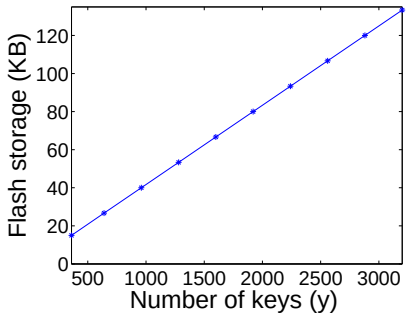

Figure 3: Flash memory storage needed to store different values of n public keys, $y_1, \cdots, y_n$.

The main overhead of our protocols is the amount of time needed to generate a single y_{str} from a string str using n number of public keys $y_1, \cdots, y_n$. For clarity, we refer to the public key generated using str as simply y_{str}. The n number of keys $y_1, \cdots, y_n$ are referred to as public keys. Note that the value of n is *not* related to the number of different y_{str}s that can be generated. The BSN can continue to generate as many y_{str} on-the-fly as needed, regardless of the value of n. Once y_{str} is generated, the remaining encryption is the same as that for a regular ECC encryption.

Fig. 2 shows the amount of time needed to generate a single y_{str} with varying values of n. All n public keys are initially stored in the flash memory. Fig. 3 shows the amount flash storage need to store different n public keys. We see from Fig. 2 that, for $n = 360$, we need only 0.9 seconds to generate y_{str}. Usually, 200∼300 public key bases are sufficient for the BSN applications we are concerned about. For example, a BSN requiring a new public key for data encryption in every hour requires 168 keys for an entire week. At the end of week, the public key bases can be refreshed.

Note that a key y_{str} once generated can be used for a considerable long period of time. Therefore, while the key generation time is longer than the data encryption time, this does not pose an excessive overhead for common BSN applications.

Since all n keys are stored in the flash memory, we ran the code using different size buffers in RAM and evaluate the performance. Large buffer allows us to fetch multiple pages to the buffer in one operation. We observe that the performance gain from choosing a 8KB buffer over a 256B buffer is very small. This is because reading multiple pages in one operation takes approximately the same amount to read one page at a time. Furthermore, the speedup from a larger buffer is only apparent for large values of n. This is a beneficial finding, since this means IBE-Lite can be executed with a smaller buffer cache in RAM with negligible performance penalty. Fig. 4 shows the time needed to perform the

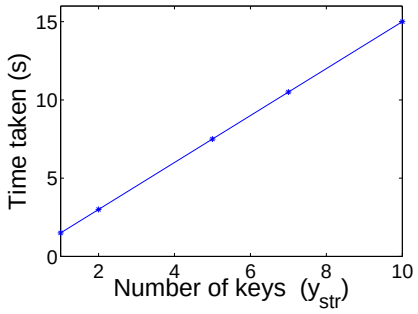

Figure 4: Time needed for different number of public keys y_{str} to encrypt data.

encryption once the public key y_{str} has been derived. For a given data, encrypting with just one y_{str} requires about 1.5 seconds. Again this is the encryption time for the symmetric key (r), which will be used then to encrypt the raw data. The symmetric key can be used for a period, say 10 minutes. The cost of the 1.5s can be amortized over the 10 minute period. The amount of time needed for multiple y_{str}s to encrypt the same data is proportional to the number of y_{str}s. While in Fig. 4 the amount of time needed for 10 different y_{str} is close to 15 seconds, in practice we are unlikely to use many different public keys to encrypt the same event. For example, a string $str = \{ER \mid doctor\}$ can be

used to cover all ER doctors. Thus even if there are many potential doctors that might access the data, the same *str* can be used.

5. RELATED WORK

The motivation behind a BSN is to place low cost sensors directly on the patient for health care monitoring. With this in mind, several research prototypes have been developed [18, 28, 8, 19]. The use of identity-based cryptography (IBE) [25, 2, 5] for medical applications was also suggested by [22, 21], but our work presents practical implementaion on actual sensors rather than a general architecture. Other applications of IBE include [13, 1, 11].

Sensor network security is a widely researched area [24, 12], with solutions focusing on key deployment [7, 14, 17, 16, 4], public key cryptography [15, 23, 9] and management [6, 20, 3]. Unlike prior work, our security protocols incorportate identity-based cryptography primitives.

6. CONCLUSION

In this paper, we presented IBE-Lite, a lightweight identity based encryption method suitable for a body sensor network. We provided protocols based on IBE-Lite to provide security and privacy support for a BSN. We evaluated our protocols using a combination of security analysis, simulations, and practical implementation on actual sensors.

Acknowledgments

The authors would like to thank all the reviewers for their helpful comments. This project was supported in part by US National Science Foundation award CCF-0514985 and CNS-0721443. Sheng Zhong was partially supported by NSF CNS-0524030.

7. REFERENCES

[1] N. Asokan, K. Kostiainen, P. Ginzboorg, J. Ott, and C. Luo. Applicability of identity-based cryptography for disruption-tolerant networking. In *MobiOpp 2007*.

[2] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO 2001*.

[3] S. Capkun, L. Buttyán, and J.-P. Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE TMC 2003*.

[4] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE SP 2003*.

[5] C. Cocks. An identity based encryption scheme based on. quadratic residues. In *LNCS 2260 (2001)*.

[6] W. Du, R. Wang, and P. Ning. An efficient scheme for authenticating public keys in sensor networks. In *MobiHoc 2005*.

[7] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *CCS 2002*.

[8] R. Ganti, P. Jayachandran, and T. Abdelzaher. Satire: A software architecture for smart attire. In *Mobisys 2006*.

[9] J. Girao, D. Westhoff, E. Mykletun, and T. Araki. Tinypeds: Tiny persistent encrypted data storage in asynchronous wireless sensor networks. *Ad Hoc Networks 2007*.

[10] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S. C. Shantz. Sizzle: A standards-based end-to-end security architecture for the embedded internet. In *PerCom 2005*.

[11] U. Hengartner and P. Steenkiste. Exploiting hierarchical identity-based encryption for access control to pervasive computing information. In *SecureComm 2005*.

[12] C. Karlof, N. Sastry, and D. Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *SenSys 2004*.

[13] A. Kate, G. Zaverucha, and U. Hengartner. Anonymity and security in delay tolerant networks. In *SecureComm 2007*.

[14] L. Lazos and R. Poovendran. Serloc: Secure range-independent localization for wireless sensor networks. *ACM TOSN 2005*.

[15] A. Liu, P. Kampanakis, and P. Ning. Tinyecc: Elliptic curve cryptography for sensor networks (version 0.3). 2007.

[16] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *CCS 2003*.

[17] D. Liu, P. Ning, S. Zhu, and S. Jajodia. Practical broadcast authentication in sensor networks. In *MobiQuitous 2005*.

[18] B. Lo and G. Z. Yang. Key technical challenges and current implementations of body sensor networks. In *BSN 2005*.

[19] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton. Codeblue: An ad hoc sensor network infrastructure for emergency medical care. In *BSN 2004*.

[20] D. J. Malan, M. Welsh, and M. D. Smith. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography. In *SECON 2004*.

[21] K. Malasri and L. Wang. Addressing security in medical sensor networks. In *HealthNet 2007*.

[22] M. Mont, P. Bramhall, and K. Harrison. A flexible role-based secure messaging service: exploiting IBE technology for privacy in health care. In *International Workshop on Database and Expert Systems Applications 2003*.

[23] E. Mykletun, J. Girao, and D. Westhoff. Public key based cryptoschemes for data concealment in wireless sensor networks. In *ICC2006*.

[24] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *Mobicom 2001*.

[25] A. Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO 1984*.

[26] H. Wang and Q. Li. Efficient Implementation of Public Key Cryptosystems on Mote Sensors (Short Paper). In *International Conference on Information and Communication Security (ICICS 2006), LNCS 4307*.

[27] H. Wang, B. Sheng, C. C. Tan, and Q. Li. WM-ECC: an Elliptic Curve Cryptography Suite on Sensor Motes. In *Technical Report WM-CS-2007-11*, 2007.

[28] L. Zhong, M. Sinclair, and R. Bittner. A phone-centered body sensor network platform: cost, energy efficiency and user interface. In *BSN 2006*.