Implementation and Performance Analysis for Key Divergent and Evolution Protocols in Wireless Sensor Network

Han Chiang Tan, Jun Wen Wong, Jianying Zhou
Institute for Infocomm Research
21 Heng Mui Keng Terrace, Singapore 119613
{hctan, jwwong, jyzhou} @i2r.-star.edu.sg

ABSTRACT

A wireless sensor network (WSN) is composed of many sensor motes that are resource-constrained tiny devices, usually driven by battery. It is critical to save the energy consumption, which is mainly due to the communication overheads, in order to extend the life-span of WSN. Key management protocol is an important component to support secure communications in WSN. Two very efficient protocols appeared in the literature: the "*Key Divergent Protocol*" (KDP) designed by Ren et al. and the "*Key Evolution Protocol*" (KEP) designed by Klonowski et al. The KEP is an improvement on the KDP to provide forward security by using a cryptographic hash function rather than flipping a random bit in the process of key update, at the expense of a bit higher computational cost. These two protocols are designed specifically for the WSN environment, without incurring additional communication overheads for establishing a secret key between any pair of sensor nodes. In this paper, we present the implementation and performance analysis results that we have conducted on the KDP and KEP. We demonstrate their feasibility in the real WSN testbed and provide the APIs that are ready for integration into WSN applications.

Keywords – Wireless Sensor Network Security, Key Management, MicaZ, Energy Analysis.

1. INTRODUCTION

Wireless sensor networks (WSN) are useful in a variety of domains, and can perform both military and civilian tasks. However, such networks are vulnerable against various attacks, mainly due to resource constraint of low cost sensor motes. As a result, it is essential to incorporate appropriate security mechanisms into WSN. Key management protocol is an important component to support secure communications in WSN [2-8].

It is estimated that the cost of data communication in WSN is more costly as compared to the computational cost in term of the energy consumption [1]: 1 bit transmission = 2090 clock cycles for energy consumption. As the sensor motes usually get their power supply from the battery, the life-span of WSN mainly depends on its overall communication overheads. It is critical to minimize the communication overheads thus reducing the energy consumption in designing efficient key management protocols.

A popular key management scheme used in support of secure communications in WSN is the random key pre-distribution [2, 5, 6]. In this scheme, the sensor nodes are pre-loaded with a set of keys that are randomly selected from a large key pool. Any two communicating nodes must have at least one common key that is found in their pre-loaded key sets before they can communicate securely with each other. The problem for this scheme is that two nodes will not be able to communicate if a common key is not found within their key sets. Another weakness is that if one of the nodes is compromised, the pre-stored keys found in the compromised node will allow an attacker to exploit the known valid keys to launch possible attacks on other nodes.

Recently, two efficient key management protocols have been proposed for the WSN environment: the "*Key Divergent Protocol*" (KDP) [3] and the "*Key Evolution Protocol*" (KEP) [4]. The objective of these two protocols is to minimize the energy consumption in communication when establishing a secret key between any pair of sensor nodes. The main idea is to allow any pair of nodes sharing a common secret key at the deployment stage to update the shared key continuously during their data communication. As the process of key update is integrated in the data communication, it does not incur additional communication overheads. The security of these two protocols is based on the assumption that no adversary is monitoring the communication between a pair of sensor nodes all the time. In other words, if the adversary missed some consecutive messages exchanged between the nodes, it will not be able to decrypt the subsequent messages, even if it holds a previously compromised secret key. The major difference between these two protocols is that the KEP uses a one-way hash function to generate the new key whereas for the KDP, the new key is derived by randomly flipping a bit in the previous key. Thus the KEP is able to provide forward security, but at the expense of a bit higher computational cost due to the hash function being used.

There are two contributions in this paper.

- We present the implementation details of the KDP and KEP, and provide performance analysis results.
- We demonstrate their feasibility in the real WSN testbed, and provide the APIs that are ready for integration into WSN applications.

MobiQuitous 2008, July 21 - 25, 2008, Dublin, Ireland

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3856
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3856

The rest of this paper is organized as follows. We first give a brief description of the KDP and KEP protocols as well as their implementation details in Section 2 and Section 3, respectively. In Section 4, we compare and analyze the performance and efficiency between the two protocols. Section 5 concludes the paper.

2. KDP – KEY DIVERGENT PROTOCOL

A scheme proposed by Ren et al. in [3] limits the effects of the compromised node to only the keys used for communicating with the other nodes. Several assumptions made in their paper include:

- The attacker can only monitor a small fraction of the communication links during deployment.
- Active attacks cannot be launched during deployment.
- Nodes are able to perform symmetric encryption/decryption efficiently.

In the variant scheme, when a node wants to communicate with its neigbouring node, it will first randomly flip one of the bit of its secret key. The node will then retrieve the hash of the previously transmitted message. Using the "flipped" secret key, the node will encrypt the message together with the counter value and hash of the previous message before transmitting the ciphertext to its neighbour.

In our implementation, AES-128 is used as the symmetric block cipher and for ease of implementation, the Electronic Code Book (ECB) mode is chosen. The following flow charts show the implementation design for both receive and send functions implemented in a MicaZ mote (see Figure 2 and Figure 3).

In our implementation, we assume that each sensor node is pre-deployed with a common key. This common key will be used by the node to establish the initial communication with the other node. For each communicating party, the sensor node will need to store 1 shared key, 2 counter values (to prevent replay attack) and 2 hash values (for verification of the key update).

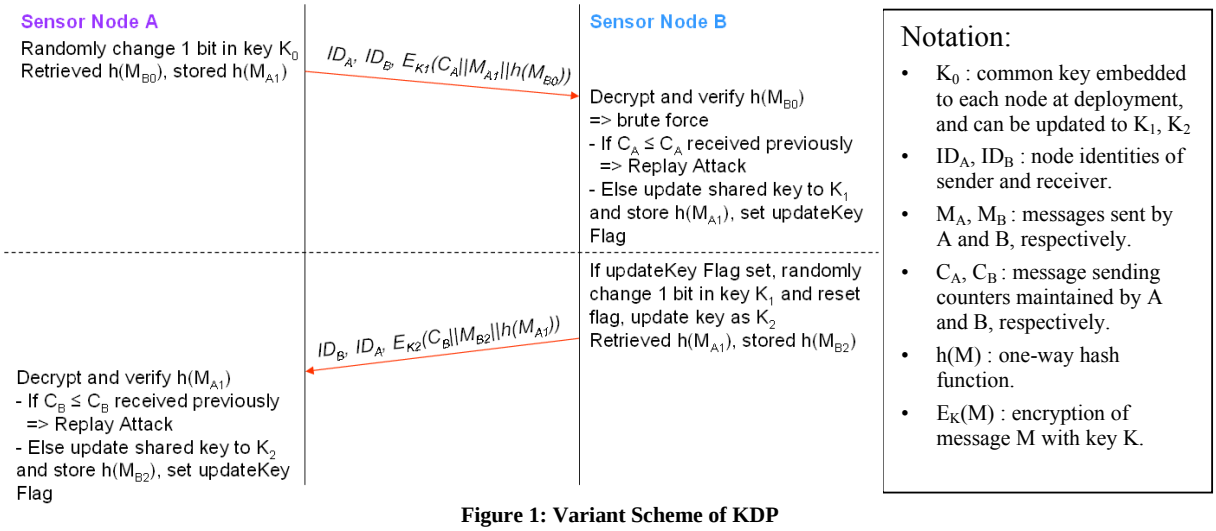

Figure 1: Variant Scheme of KDP

In this scheme, the keys embedded in each node are changed continuously forcing the need for the adversary to monitor all the communication links in order to compromise the protocol. This protocol can also be used together with the random key pre-distribution protocols [2, 5, 6] in the case the two nodes did not find a common key within their pre-loaded key sets.

Besides a basic scheme, there is a variant scheme discussed in [3]. The difference is that in the variant scheme, the hash value of the previous message will be piggybacked on the current message to replace the acknowledgement message. This is to further reduce the communication overhead in the protocol. The hash value is used to acknowledge that the previous key has been correctly retrieved, and the new key is generated from the previous established link key. We have implemented the variant scheme (shown in Figure 1) as it is more efficient and less costly than the basis scheme.

During our implementation, we found that the protocol designed by Ren et al. does not take into consideration the issues of data loss during the transmission of the message. Therefore we modified the protocol slightly, and used the hash values of the previously sent messages to allow the shared key to be updated only if the receiver has successfully received the sent message.

When communicating, each sensor node will encrypt the message using a key that differs by one bit from the shared key and store the new key in the memory. However the sensor node will not be able to update the shared key if the previous key update is initialized by itself. Thus the shared key can only be updated by the other party.

Upon successful decryption of the message, the sensor node will check if the counter value of message is greater than the counter value stored in the memory. If not, the node will assume a reply attack and reject the message. Otherwise, the node will update its own counter and verify if the hash value corresponds to its previous sent message hash value. If the hash values are the same,

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3856
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3856

the node will update the shared key during the next communication initialized by itself.

A disadvantage of the KDP is that it continuously changes the shared key between 2 communicating nodes and uses the brute force attack to retrieve the shared key from the received message. Although the cost of the brute force attack is insignificant as compared to the cost of transmitting the shared key through a secured channel, it may not be suitable for WSN with high data traffic where messages are transmitted frequently. This is because, the sensor node might be sending out messages at a very short interval and hence the overall cost of continuously updating the keys and performing the brute force attack to retrieve key might be high.

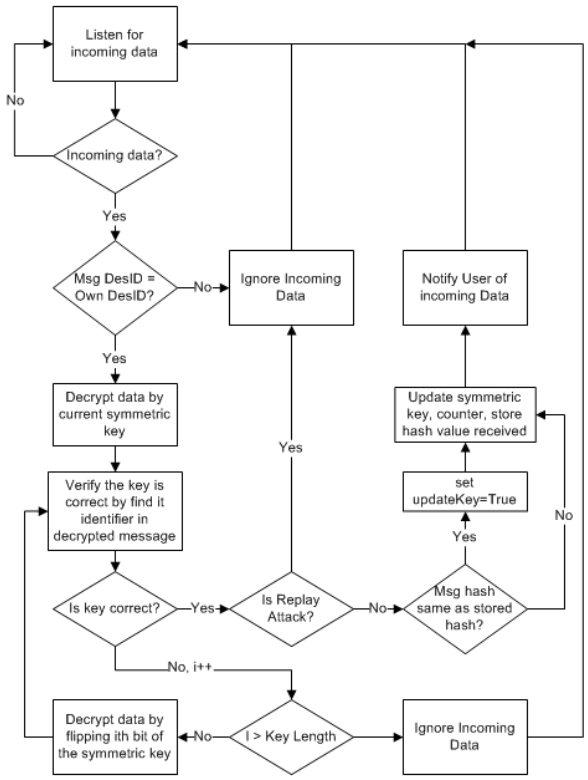

Figure 2: KDP Receive Function

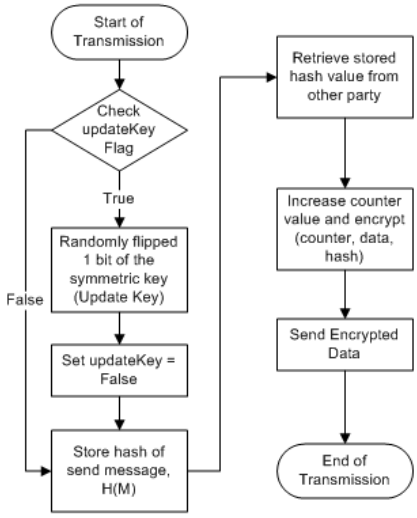

Figure 3: KDP Send Function

IMPLEMENTATION OF KDP

a. A initialized the communication to B

Before node A encrypts the message, A will check the *updateKey* flag. If the flag is set, A will randomly flip 1 bit in the pair-wise shared key with B (thus the key is updated from K_0 to K_1) and set the *updateKey* flag to false. A will also increase its own counter, C_A, and encrypt $C_A||M_{A1}||h(M_{B0})$ using K_1. A will then hash the message M_{A1} and store it. This hash value will be used in the decision to set the *updateKey* flag in the future. Finally A will transmit the encrypted message to B.

b. B received message from A

Upon receipt of an incoming message designated for B, B will use the current pair-wise shared key with A to decrypt the message. If it is not successful, B will do 'brute force attack' on the shared key to retrieve the new key. After the message is retrieved successfully, B will compare the counter value to verify if the received message is fresh. B will then compare the hash value, $h(M_{B0})$, with its stored hash. If they are the same, B will set the *updateKey* flag.

c. B sent message to A

B will check the *updateKey* flag. If the flag is set, B will randomly flip 1 bit in shared key (thus the key is updated from K_1 to K_2). B will then set the *updateKey* flag to false. Next, B will hash its message M_{B1} and store the hash value, which will be used in the decision to set the *updateKey* flag in the future. B will then increase its own counter, C_B, and encrypt $C_B||M_{B1}||h(M_{A1})$. Finally B will transmit the message to A.

d. Key update in KDP

Since we use AES-128, our key is 16 bytes. The sensor node will update its key by flipping the bit from a random location l, where $0 \leq l < 128$. The new shared key will only be generated when the *updateKey* flag is set.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3856
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3856

e. Key recovery in KDP

During key recovery phase, the sensor node will perform a brute force attack on the shared key. This is done by trying out the temporary keys obtained by flipping the bit from position 0 to 127, to decrypt the received message and verify the identifier obtained.

APIs for KDP

The APIs for our KDP implementation is shown in Table 1. They can be easily integrated into WSN applications.

API	Description	Input Param.	Output Param.
start	Initialization of the protocol	NIL	NIL
send	Transmission of data	destID, srcID, msg, msgLen	NIL
receive	Reception of data	destID, srcID, msg, msgLen	NIL
stop	Termination of protocol	NIL	NIL

Table 1: APIs for KDP

3. KEP – Key Evolution Protocol

The KEP is an improvement to the KDP as suggested by Klonowski et al. [4]. The KEP seeks to address one of the weaknesses found in the KDP. As discussed in the paper, the KDP does not provide forward security. This is because an attacker is able to retrieve all the shared keys that have been used between the two nodes if the sensor node is compromised and all the previously transmitted data are recorded.

Hence the KEP aims to prevent the mentioned attack by providing forward security through the use of a cryptographic one-way hash function. This is because the attacker will not be able to retrieve back the previous keys used in securing the data communication due to the cryptographic one-way hash properties.

Similar to the KDP, the KEP assumes that the two communicating nodes have already established a common shared key.

The advantages of the KEP are also similar to the KDP. They include:

- No overheads incurred in the updating of the shared key.
- Key update is controlled by the communication traffic.
- Forward security (only for the KEP).
- Constant monitoring is necessary to decipher the future transmitted messages.

As shown later in the performance results, the main limitation of the KEP is that the protocol efficiency is mainly dependant on the choice of cryptographic hash function used. Hence in order for the KEP to be efficient, an efficient hash function is necessary.

Implementation of KEP

In our implementation, the general architecture used in the KEP is similar to the KDP. Overall, the KEP follows the same implementation architecture as shown in Figure 2 and Figure 3 used by the KDP.

The only difference is that instead of the random 1-bit flip in the key generation phase found in the KDP, the KEP will use SHA-1 as the cryptographic hash function to generate the new shared key. This is the same in the key recovery phase which SHA-1 will also be used to retrieve the shared key.

Since AES-128 is used to provide data confidentiality in our implementation, we only use the first 16 bytes of the hash function output. This is because the secret key used by AES-128 is 16 bytes long whereas the output for SHA-1 is 20 bytes long.

a. KEP Key Generation

Instead of randomly flipping one of the bits in the shared key, the KEP will generate the key through the use of a hash function. In their paper, Klonowski et al. suggested 2 schemes. The basic scheme is to generate the key by hashing the current key together with a random number, $K' = h(K||R)$, *where* $R \leq L$. L is the parameter used to control the convergence rate.

A variant scheme is that the new key is generated by hashing the current shared key together with the current index of the key and a random number, $K' = h(K||I||R)$. The index keeps track of the number of times the key is updated as it increases with each transformation of the key.

b. KEP Key Recovery

Similar to the KDP, the KEP also performs a brute force attack on the received message to recover the shared key. Instead of running through the entire shared key length to retrieve the flipped bit position, the KEP obtains the shared key by trying out with a different random number that is input into the hash function. This is similar for both the basic and the variant scheme. In the variant scheme, the sensor node will need to store the index of the key. This will prevent the replay attack as the index will increase with each key update.

APIs for KEP

The APIs for our KEP implementation is the same as for our KDP implementation as shown in Table 1 because the key generation and key recovery phases are transparent to the user.

4. Performance Analysis

In this section we provide some analysis that has been done on the two protocols, KDP and KEP. The experimental setup (see Figure 4) shows the measurement circuit that we used in measuring and estimating the energy consumption for the two protocols. The current drawn from the battery is calculated by dividing the measured value at the multi-meter, V1 and the resistor value, R = 9.7 Ω. The operating voltage of the MicaZ is measured by the oscilloscope, V2. Thus the operating power value is calculated by

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3856
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3856

multiplying the operating current to the operating voltage, P = I*V. Energy consumption is then calculated based on the formula, E = P*t, where t is the time consumed by the operation (either the KDP or the KEP).

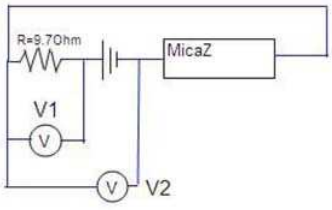

Figure 4: Measurement Circuit

In our implementation, we have chosen MicaZ, developed by Crossbow Technology, as our sensor mote with TinyOs, developed by Berkley, as the operating system. AES-128 has been developed by Technische Universitat Graz (TUG), Austria and we have implemented our own CBC mode in NesC. For SHA-1, we have adopted the SHA-1 implementation developed by North Carolina State University (NCSU), USA. For the KEP implementation, we have set L = 128 so that the number of brute force attacks in the worst case scenario are the same for both the KDP and KEP.

In our analysis, we found that that Chipcon CC2420 does not switch to the idle mode when the mote is in computation state. This affects the measurements done on MicaZ as the transceiver is still active and consumes a significant amount of energy. Therefore we have to manually disable the transceiver during the mote is doing computation to get a better estimation of the energy consumed by the protocol itself.

a. Memory Analysis

Table 2 presents the memory usage for MicaZ to install the KDP, the basic KEP and variant KEP. In terms of the Read-Only Memory (ROM) usage, the KDP requires an additional 8446 bytes as compared to the basic encryption scheme without any key update protocol. The ROM usage for both KEP is the same which is 20894 bytes. (All measurements shown in terms of bytes)

	Basic Encryption	**KDP**	**Basic KEP**	**Variant KEP**
RAM	608	608	608	608
ROM	11950	20396	20894	20894

Table 2: Memory Analysis Results

As seen from Table 2, the KEP uses slightly more ROM than the KDP. This is due to the SHA-1 function used in the protocol. The table also shows that both the KDP and KEP need significantly higher ROM usage than the basic encryption model. Although the increase in the ROM usage is close to about 10 kB, this is quite insignificant even in the constrained devices such as MicaZ, which has an available ROM of size 128 kB.

b. Timing Analysis

Table 3 presents the average time taken by MicaZ for the key recovery phase for both the KDP and KEP protocols. In our experiment, we recorded the timing taken for two protocols for 500 times and obtained the average time taken for the key recovery phase.

	KDP	**Basic KEP**	**Variant KEP**
Average time taken for key recovery (s)	0.1734	0.8321	0.8549
Worst case (s)	0.4	1.4	1.594

Table 3: Timing Analysis Results

It is clearly shown that the KDP is more efficient than the KEP. By using the KDP, we can save 0.6815s as compared to the KEP's variant scheme and 0.6587s as compared to the basic scheme. The main reason why the KEP is significantly slower than the KDP is because the KEP uses SHA-1. This slows down the recovery of the shared key.

The timing results showed that although the KEP is able to provide forward secrecy, there is a trade-off in terms of its efficiency. A difference of 0.6s (or approximately 400% slower) per key update can be significant in the wireless sensor network environment where data are transmitted almost continuously in some system applications. This overhead might be the bottleneck for such applications.

c. Energy Analysis

Table 4 shows the average energy consumed by MicaZ for the key recovery phase for both the KDP and KEP. We calculated the power required by MicaZ based on the multi-meter and oscilloscope reading as shown in Figure 4 (as explained earlier on). We then multiplied the calculated power and the time taken for the key recovery to obtain the energy consumption.

	KDP	**Basic KEP**	**Variant KEP**
Average energy consumption for key recovery (mJ)	2.682	12.866	13.218

Table 4: Energy Analysis Results

As shown from the results, the KDP is more efficient than the KEP. By using the KDP, we can save 10.536mJ as compared to the KEP variant scheme and 10.184mJ as compared to the KEP basic scheme. As discussed earlier, the additional energy consumed by the KEP is due to the SHA-1 function in the key recovery phase.

The wireless sensor motes are often deployed in the open areas and the only power source available is the battery pack attached to it. Since the battery pack has limited energy capacity, the energy needs to be utilized carefully to avoid malfunction of the senor mote due to the lack of power in the batteries. As seen from the results, the KDP consumes approximately 10mJ less energy with each key recovery phase. This is a significant difference as the lifetime of the mote can be a difference of about 6 times longer for the KDP as compared to using the KEP.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3856
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3856

d. Security Analysis

In our implementation, AES-128 is used as the symmetric block cipher for both KDP and KEP. For the KDP, a legitimate neighbouring node will need an average of 64 tries to guess the position of the bit that is changed from the last known secret key. This is significantly a lot lesser than an attacker, who does not has the secret key. On the average, an attacker will need 2^{127} tries to retrieve the secret key.

As mentioned earlier, one of the limitations found in the KDP is that it does not provide forward secrecy. Once the attacker is able to retrieve a valid secret key, the attacker is able to retrieve any previously encrypted message easily. This is because with the valid secret key, the attacker only needs an average of 64 tries to retrieve either the secret key that is used before or after the retrieved key.

The KEP is able to provide forward secrecy through the used of a cryptographic hash in its key generation. This prevents the attacker from retrieving any past messages from a known key due to the properties of the cryptographic hash function. Although the attacker is not able to retrieve any past messages, the attacker is still able to retrieve the future keys easily with the knowledge of the current secret key. The difficulty of the retrieval of keys is dependant on the random number used in the key generation phase. If |R|=128, the attacker would takes a slightly longer time then the KDP to retrieve the next instance of the secret key, although the attacker will also requires an average of 64 tries. This is because the cryptographic hash function consumes slightly longer time as compared to flipping 1 bit of the secret key.

5. CONCLUSION

One of the major concerns in the implementation of a symmetric key crypto-system is the need to establish and refresh the shared key in a secure manner. As mentioned previously that the sensor nodes in the WSN environment have limited resources, especially the energy supply, the basic key management protocols used in a wireless environment [9] is unsuitable as the amount of overheads incurred in the transmission of messages will significantly deplete the energy level of the sensor nodes.

The two protocols investigated in this paper (KDP and KEP) seek to minimize the energy consumption by replacing the communication overheads, found in the basic key management protocols, to computation overheads. This is because transmission of data incurs significantly more energy than computation. Our implementation of these two protocols demonstrated their feasibility in the real WSN testbed. APIs are also provided for integration into WSN applications.

As shown in the performance analysis, we can see that the KDP uses fewer resources and is more efficient than the KEP. Although the KEP is able to provide forward security, it may not be suitable to be used in WSN with high data traffic as it will shorten the life-span of the sensor nodes significantly.

Instead, the KDP may be a better choice to provide the key refreshment in a resource constrained WSN over the KEP, although the KDP does not support forward secrecy. Actually, it is quite infeasible for the attacker to be able to decipher the previously encrypted messages from a compromised node as the attacker cannot afford to miss collecting the transmitted data for more than 2 key refreshments with the node, after which the attacker will not be able to retrieve the key back.

6. ACKNOWLEDGEMENT

This work is partially funded by the European Union project SMEPP-033563.

7. REFERENCES

[1] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "*Energy analysis of public-key cryptography for wireless sensor networks*", 2005 IEEE International Conference on Pervasive Computing and Communications (PerCom'05), pp. 324-328.

[2] L. Eschenauer and V. D. Gligor, "*A key-management scheme for distributed sensor networks*", 2002 ACM Conference on Computer and Communications Security (CCS'02), pp. 41-47.

[3] M. Ren, K. D. Tanmoy, and J. Zhou, "*Diverging keys in wireless sensor network*", 2006 Information Security Conference (ISC'06), LNCS 4176, pp. 257-269.

[4] M. Klonowski, M. Kutylowski, M. Ren, and K. Rybarczyk, "*Forward-secure key evolution protocol in wireless sensor networks*", 2006 International Conference on Cryptology and Network Security (CANS'07), LNCS 4856, pp. 102-120.

[5] H. Chan, A. Perrig, and D. Song, "*Random key predistribution schemes for sensor network*", 2003 IEEE Symposium on Security and Privacy, pp. 197–213.

[6] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "*A pairwise key predistribution scheme for wireless sensor networks*", 2003 ACM Conference on Computer and Communications Security (CCS'03), pp. 42–51.

[7] Y. H. Kim, H. Lee, J. H. Park, L. Yang, and D. H. Lee, "*Key establishment scheme for sensor networks with low communication cost*", ATC 2007: 441-448.

[8] B. C. Lai, D. Hwang, S. Kim, and I. Verbauwhede, "*Reducing radio energy consumption of key management protocols for wireless sensor network*", 2004 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'04), pp. 351-356.

[9] C. Boyd and A. Mathuria, "*Key establishment protocols for secure mobile communications: A selective survey*", Lecture Notes in Computer Science, vol. 1438, pp. 344–355, 1998.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3856
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3856