Secure Key Renewal and Revocation for Wireless Sensor Networks

Ismail Mansour, Gérard Chalhoub, Pascal Lafourcade, and François Delobel
Clermont University, LIMOS laboratory, France
firstname.name@udamail.fr

***Abstract*—Once a secure mechanism for authenticated communication is deployed in a Wireless Sensor Network (WSN), several situations may arise: a node can leave the network, a new node can join the network, an intruder could try to join the network or capture a node. Therefore it is important to revoke and renew certain keys that are learned by a malicious node. We propose several secure WSN protocols for revocations and renewal of cryptographic keys in the network based on symmetric encryption and elliptic curve cryptography (ECC). For all our solutions, we provide a formal analysis of the security of our protocols using Scyther, an automatic verification tool for cryptographic protocols. All the proposed protocols are proven secure but have different security levels by using different types of keys. Finally we implemented all our protocols on real testbeds using TelosB motes and compared their efficiency.**

Keywords*—*Key Renewing, Key Revocation, WSN, Security.

I. Introduction

Wireless Sensor Networks (WSNs) are more and more used to monitor our environment and to interconnect modern devices. Some applications are critical and often require cryptographic mechanisms in order to achieve security [9]. Then it is important to design secure communication mechanisms between nodes of the network, using cryptography. Once this secure communication channel is established, a node can leave the network (running out of battery or even being destroyed), or a new node can join the network. In addition, an intruder could capture a node and learn all his secret data (including secret keys). An intruder node could also try to join the network and be part of the authenticated nodes. In order to detect such intruders several Intrusion Detection Systems (IDS) have been proposed in the literature. In general IDS either monitor the internal behavior of one node, or search for signs of malicious activity in the network. For this, several methods are used like signature-based or anomaly-based detection technique. Based on the results of IDS, the next step is to revoke malicious nodes that have been identified and to renew the network keys. Moreover, it is widely admitted that, in WSNs, using asymmetric encryption primitives based on exponentiation, like for instance RSA or Elgamal, is not realistic, due to the limited resources of sensor nodes. However several lightweight cryptographic primitives that are more adapted for WSN exist *e.g.* [2]. The low level of security of lightweight primitives remains a real obstacle for their deployment. For instance, in [10], using an improved differential fault analysis, authors can break a lightweight block cipher for WSNs called *LBlock* using a typical personal computer within one hour. It is clearly not surprising that lightweight encryption can be attacked in few hours with more computation power than that of a sensor node. In this context, it is therefore crucial to have efficient revocation and renewal mechanisms in WSNs.

Related Work: One of the first key revocation protocols was proposed in [4] and [3]. This proposition has been enhanced by authors of [5] where they proposed a distributed collaborative key revocation mechanism that divides the network into regions. In each region, nodes collaborate to identify a malicious node. Once a malicious node is identified, the base station is informed and it sends a broadcast message containing a list of keys to revoke. Unlike [4], [3], their contribution is able to revoke all the keys with which the revoked node is involved and does not need to know the network topology before the deployment. The key revocation message sent by the base station is based on trivariate polynomial authentication and verified by each node according the region to which it belongs. In [8], authors propose a key distribution based on key chains where each key is obtained by applying a hash function on the next key in the chain. These key chains are either exchanged through a secure channel or installed in nodes before deployment. Authenticating the keys is done by verifying the hash result. A central node is used as a key service entity that manages the key revocation process. This entity shares a secret key with every node in the network. Authors do not explain how this secret key is shared neither how this secret is renewed. Protocols based on key chains cannot easily update the chains, for their key renewal is based solely on the hashing function and the root value of the chain. Moreover chains should be long enough to last for the life duration of the network. In [15], authors proposed a periodic key renewal based on fragmentation of the key generation function. This method supposes that nodes will assemble the fragments sent by the base station. The process of assembling the function is not authenticated and might be easily falsified for it uses only one key which is a shared key between all the nodes of the network, thus any compromised node is able to interfere in the process. In [11], authors propose a key renewal mechanism that is managed by a special entity called the Command Node. The network is organized into clusters with one Cluster Head node that is used for key revocation. This cluster head receives the list of new keys for renewal and sends them to the other cluster gateways which in turn send them to the sensor nodes. Authors use the shared key with the Command Node to update keys and do not propose a mechanism for updating this shared key.

According to our knowledge, none of the existing revocation and key renewal protocols were verified using an automatic formal verification tool, in addition, most of the existing results are obtained through simulations or complexity

estimation when evaluating the cost of the cryptographic scheme. Our mechanism achieves authenticated key renewal in the presence of malicious nodes to update all the keys in a multihop network, we provide automatic formal verification and real testbed implementation.

II. Authenticated Join Protocols

We use public key Elliptic Curve Cryptography (ECC), using parameters secp160r1 given by the Standards for Efficient Cryptography Group [14]. Our implementation of ECC on TelosB is based on optimized TinyECC library [12]. More precisely we use Elliptic Curve Integrated Encryption Scheme (ECIES 160 bits), the public key encryption system proposed by Victor Shoup in 2001. For all symmetric encryptions we use an optimized implementation of AES with a key of 128 bits proposed by [13]. We use the following notations to describe exchanged messages in our protocols: I: a new node that initiates the protocol; R: a neighbor of node I; S: the sink of the network (also called *base station*); n_A: a nonce generated by node A; $\{x\}_k$: the encryption of message x with the symmetric or asymmetric key k; $pk(A)$: the public key of node A; $sk(A)$: the secret (private) key of node A; $K(I,S)$ or $K(S,I)$: the symmetric session key between I and S; NK: the symmetric network key between all nodes of the network; $K_{DH}(N,S)$ or $K_{DH}(S,N)$: the shared symmetric key between N and S using the Diffie-Hellman key exchange without interaction described below.

Before deployment, each node N knows the public key $pk(S)$ of the sink and also its own pair of public and private keys, denoted $pk(N)$ and $sk(N)$ respectively. Based on ECC, we have that $pk(N) = sk(N) \times G$, where G is a generator point of the elliptic curve. Using this material, each node N can compute a shared key with the sink S using a variation of the Diffie-Hellman key exchange without interaction, denoted $K_{DH}(N,S) = K_{DH}(S,N)$.

III. Renewal and Revocation Protocols

a) Key Revocation Protocol (KR): The sink collects the IDS results and determine the nodes that have to be revoked. Then he sends a revocation request to node I using the protocol (KR) described in Figure 1. In this protocol, the sink sends to node I the list $M_1, \ldots, M_k$ of all revoked nodes in the neighborhood of I and a nonce n_S encrypted with $K_{DH}(S,I)$. Then node I deletes all shared session keys with all nodes included in the list and do not accept any further communications with these nodes. In order to confirm the reception of the list, node I sends back the nonce n_S encrypted with $K_{DH}(S,I)$. Nonce n_S acknowledges the reception of the list by node I, it also ensures the authentication.

b) Renewing Symmetric Keys (RSK): Figure 2 presents protocol RSK that allows an initiator node I to renew a session key with its neighbor R. The protocol consists in sending the new session key $K'(I,R)$ encrypted with the previous session key $K(I,R)$ in order to confirm to R that I has the previous session key. Then the message is encrypted again with the public key of R. Notice that an intruder should obtain $K(I,R)$ and $sk(R)$ in order to learn the new session key $K'(I,R)$. This protocol clearly increases the security but will take a lot of execution time due to the extra public key encryption (see Table I). Finally, the new key $K'(I,R)$ is used as a nonce by R to confirm the reception by sending back to I the message $\{K'(I,R)\}_{K'(I,R)}$.

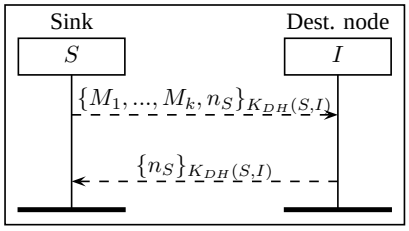


Figure 1: Protocol KR: Revocation of k malicious neighbor nodes of node I.

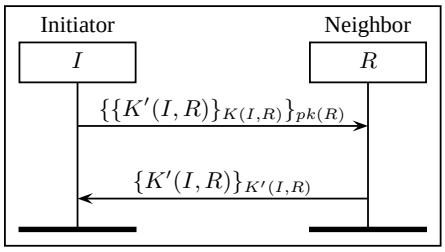


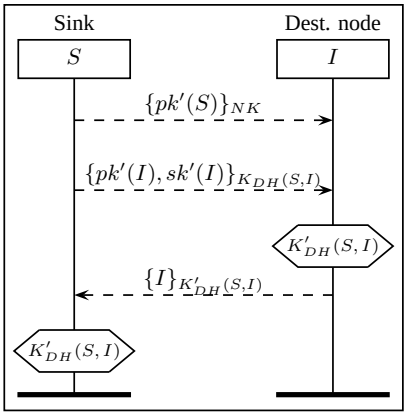
Figure 2: Protocol RSK: Renewing a symmetric or session key.

c) Renewing Asymmetric Keys (RAK): We give four protocols to renew all asymmetric keys of the network. These protocols use the existing key infrastructure to securely replace all the asymmetric keys between the sink and all nodes of the network. For this, the sink creates his own new public/private keys $(pk'(S), sk'(S))$ and a new pair of public/private keys $(pk'(I), sk'(I))$ for each node I in the network. Our four protocols (RAKnk$_a$, RAKnk$_b$, RAKdh$_a$ and RAKdh$_b$) are based on the same idea: First S securely sends $pk'(S)$ and the new pair of keys for node I; then node I replies by sending back his identity with the new shared key $K'_{DH}(S,I)$. We use different cryptographic primitives to distribute these new keys.

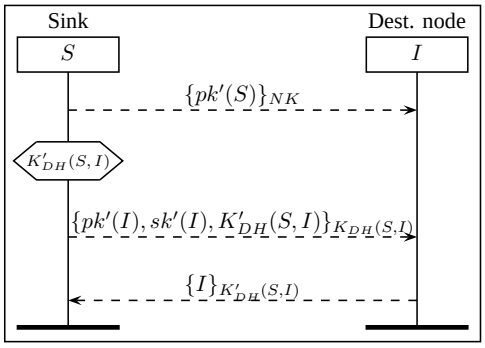
In Figure 3, we present two protocols RAKnk$_a$ and RAKnk$_b$, where the new public key of the sink $pk'(S)$ is broadcast to all nodes using NK. In the first protocol RAKnk$_a$, depicted in Figure 3a, the sink only sends to each node I the new pair of keys using $K_{DH}(S,I)$. Then I computes the new shared key $K'_{DH}(S,I) = sk'(I) \times pk'(S)$. In order to save computation time for node I, we propose a second version RAKnk$_b$ in Figure 3b, where S pre-computes $K'_{DH}(S,I) = sk'(S) \times pk'(I)$ without using the secret key of I.

An alternative is to use the pre-shared key $K_{DH}(S,I)$ instead of NK in the distribution of $pk'(S)$, as depicted in Figure 4. In Figure 4a, we explain the protocol RAKdh$_a$ with computation of the new key performed by node I. In Figure 4b, we present the protocol RAKdh$_b$ where the sink pre-computes $K'_{DH}(S,I)$. These two protocols use symmetric shared keys on each hop preventing an intruder to learn the new key of the sink by learning the network key as it is the case in protocols of Figure 3. Nevertheless this solution requires more load on the network since the transmission of the public key of the

sink is not a broadcast using the network key but an unicast using a symmetric shared key between two nodes.



(a) Protocol RAKnk$_a$: S and I compute $K'_{DH}(S,I)$.



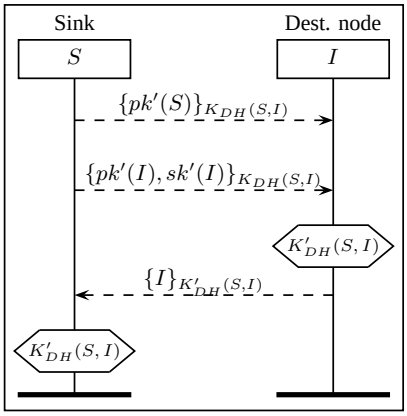
(b) Protocol RAKnk$_b$: S computes $K'_{DH}(S,I)$ and sends it to I.

Figure 3: Renewing asymmetric keys of a node I using the network key NK to broadcast $pk'(S)$.

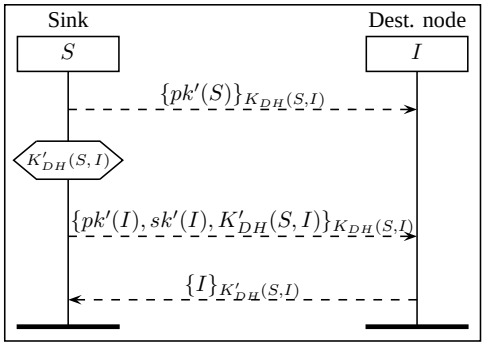
d) Renewing the Network Key (RNK): The sink decides when to renew the network key NK. We propose a secure way for the sink to distribute this new key to all authenticated nodes. This protocol, denoted RNK, is described in Figure 5. It allows the sink to be sure that all nodes receive the new key before to start using it. The sink generates a new network key NK' and a nonce $n_{(S,I)}$ and encrypts them using the shared symmetric key $K_{DH}(S,I)$ and sends them to node I. Then, it waits until it receives all nonces before to start using NK'.

e) Formal Security Evaluation: Evaluating the security of cryptographic protocols is not an easy task and is easy to design flawed protocols. Moreover during the last decade several tools have been developed to automatically verify cryptographic protocols [1], [7]. In order to prove the security of all our protocols we use the cryptographic protocol verification tool Scyther[7]. We choose this tool since it is one of the fastest tools as it has been shown in [6] and one of the most user-friendly. Scyther automatically proves security properties or give an attack on a cryptographic protocol for bounded and unbounded numbers of sessions and provides an easy way to model security properties like secrecy and authentication.

We verified automatically all our protocols using Scyther in few seconds on a regular PC. Scyther concludes that all our protocols are secure and proves the secrecy of all sensitive data exchanged (keys and nonces) and also the authenticity of the communication[1]. Moreover for each protocol, we minimize the amount of exchanged data. For example, we use in the protocol of Figure 2 the new symmetric key as a nonce.



(a) Protocol RAKdh$_a$: S and I compute $K'_{DH}(S,I)$.



(b) Protocol RAKdh$_b$: S computes $K'_{DH}(S,I)$ and sends it to I.

Figure 4: Renewing the asymmetric keys of a node I using the symmetric shared key $K_{DH}(S,I)$.

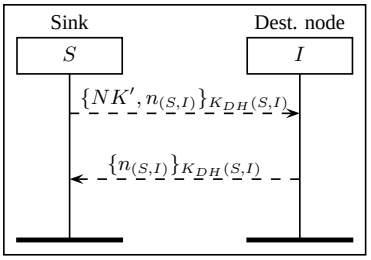


Figure 5: Protocol RNK: Renewing the Network Key.

IV. Experiments

Using our own implementations on TelosB motes, we compare the time execution of our different protocols.

[1]http://sancy.univ-bpclermont.fr/~lafourcade/scyther-lcn-code.tar

Protocol	Name	Figure	Time with S (ms)	Time without S (ms)	Gain	Standard deviation (ms)
Revocation	KR	1	155.37	87.58	44%	3.82
Renewing SymKey	RSK	2	10042.32	10042.32	0%	76.49
Renewing AsymKey	RAKnk$_a$	3a	6797.75	3436.24	49%	4.26
	RAKnk$_b$	3b	3646.05	254.62	93%	3.95
	RAKdh$_a$	4a	6797.75	3436.24	49%	4.26
	RAKdh$_b$	4b	3646.05	254.62	93%	3.95
Renewing Network Key	RNK	5	221.09	121.4	45%	3.73

Table I: Time execution of all protocols.

Settings: To evaluate the efficiency of our solutions, we used TelosB motes. These motes have a 8 MHz microcontroller with 10 Kb of RAM, 48 Kb of ROM and a CC2420 radio using the IEEE 802.15.4 standard. During the experiments, we considered topologies without intermediate nodes, since these nodes would only forward the packets without doing any modification on the packet. The cost of these communications is therefore negligible compared to the encryption and decryption costs. Moreover, this cost is the same for all protocols, only the load of the network can change between unicast and broadcast protocol. Hence for each situation we only consider a minimal topology containing only the nodes involved in the cryptographic operations.

Results and Discussion: In Table I, we provide the execution time for all our protocols. We also present the results without the execution time of the sink, since in many applications the base station is a special node with extra resources. All results are the averages of 100 experiments of each protocol. We also provide the standard deviations for execution time including time of S. The protocol KR (key revocation) is the fastest. We also note that the sink performs almost half of the cryptographic operations, thus by making it doing more operations we avoid sensor nodes from doing the heavy cryptographic computations. If the size of the list of revoked nodes increases then the protocol KR will take more time. For renewing the asymmetric key we proposed four protocols, two of them use the symmetric network key NK and the other two use symmetric keys K_{DH}. We see that since they are using the same symmetric encryption mechanism they take the same execution time. Hence, the execution time for protocols RAKnk$_a$ and RAKdh$_a$ is the same and similarly for protocols RAKnk$_b$ and RAKdh$_b$. However, the second version of these protocols RAKnk$_b$ and RAKdh$_b$ are faster than protocols RAKnk$_a$ and RAKdh$_a$ (more so if we do not count the sink execution time). It clearly shows that the computation of the new key by a node is expensive. Therefore it is important that a designer takes it into account during the conception of the protocols in order to have efficient protocols and also to preserve resources of the nodes.

V. Conclusion

We have proposed several protocols to revoke a set of nodes, and renew symmetric and asymmetric keys. All our protocols have been automatically verified using Scyther. This ensures the security of our solutions. We also have implemented our protocols on testbeds using TelosB motes in order to compare their efficiency. Then, according to the context (size of the network, size of the battery, type of mote, energy consumption for communication, computation resources of the motes) one solution might be better than another one. All these parameters should be taken into account before to choose one real solution. In our future work, we are planning to adapt these protocols to support mobility.

Acknowledgemnt: This research was conducted with the support of the Digital trust Chair from the University of Auvergne Foundation.

References

[1] B. Blanchet, "Automatic proof of strong secrecy for security protocols," in *IEEE Symposium on Security and Privacy*, May 2004, pp. 86–100.

[2] M. Cazorla, K. Marquet, and M. Minier, "Survey and benchmark of lightweight block ciphers for wireless sensor networks," in *SECRYPT*, P. Samarati, Ed. SciTePress, 2013, pp. 543–548.

[3] H. Chan, V. Gligor, A. Perrig, and G. Muralidharan, "On the distribution and revocation of cryptographic keys in sensor networks," *IEEE Transaction on Dependable and Secure Computing*, vol. 2, 2005.

[4] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *IEEE S & P*, 2003.

[5] S. Chattopadhyay and A. K. Turuk, "A scheme for key revocation in wireless sensor networks," *International Journal on Advanced Computer Engineering and Communication Technology*, vol. 1, 2012.

[6] C. J. Cremers, P. Lafourcade, and P. Nadeau, "Comparing state spaces in automatic protocol analysis," in *Formal to Practical Security*, vol. 5458/2009. Springer, 2009, pp. 70–94.

[7] C. Cremers, "The Scyther Tool: Verification, falsification, and analysis of security protocols," in *CAV'08*, ser. LNCS, 2008.

[8] G. Dini and I. Savino, "An efficient key revocation protocol for wireless sensor networks," in *International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2006.

[9] M. A. Hussain, P. Khan, and K. K. Sup, "Wsn research activities for military application," in *Proceedings of the 11th international conference on Advanced Communication Technology-Volume 1*, 2009.

[10] K. Jeong, C. Lee, and J. Lim, "Improved differential fault analysis on lightweight block cipher lblock for wireless sensor networks," *EURASIP J. Wireless Comm. and Networking*, vol. 2013, p. 151, 2013.

[11] G. Jolly, M. Kusçu, P. Kokate, and M. Younis, "A low-energy key management protocol for wireless sensor networks," in *ISCC'03*, 2003.

[12] A. Liu and N. Ning, "Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks," in *7th International Conference on Information Processing in Sensor Networks*, 2008.

[13] N. Manica, M. Saloni, and P. Toldo, "WSN - secure comunications with AES algoritms," Univ. of Trento - Faculty of Computer Science, 2008.

[14] C. Research, "Standards for efficient cryptography, sec 1: Elliptic curve cryptography," September 2000.

[15] C. Wang, T. Hong, G. Horng, and W. Wang, "A key renewal scheme under the power consumption for wireless sensor networks," *Journal of Colloid and Interface Science*, 2006.