Providing Transparent Security Services to Sensor Networks

Hamed Soroush, Mastooreh Salajegheh and Tassos Dimitriou
Athens Information Technology
Email: {hsor,msal,tdim}@ait.edu.gr

Abstract

In this paper we introduce a link layer security platform for wireless sensor networks. At the heart of this platform, lies our key management module facilitating an efficient scalable post-distribution key establishment that allows the platform to provide different security services. We have developed this framework under TinyOs and have tested it with MICA2 motes. To the best of our knowledge this is the first implemented security platform for sensor networks that provides acceptable resistance against node capture attacks and replay attacks. The provision of security services is completely transparent to the user of the framework. Furthermore, being highly scalable and lightweight, this platform is appropriate to be used in a wireless sensor network of hundreds of nodes.

I. Introduction

Wireless sensor networks (WSN) have recently attracted many researchers due to a variety of new challenges they introduce. The unique set of resource constraints in WSN nodes (such as finite on-board battery power and limited network communication bandwidth) results in very different design trade-offs than those in general-purpose systems.

As sensor networks are usually deployed in hostile environments, many of their applications require that data must be exchanged in a secure and authenticated manner. However, the conventional security schemes which are used in other types of networks such as mobile or ad-hoc networks cannot be applied to WSN due to their mentioned constraints. End-to-end security mechanisms such as SSL are considered highly inappropriate since they don't allow in-network processing and data aggregation which play an important role in energy-efficient data retrieval. On the other hand, since the usual traffic pattern in WSN is many-to-one, pre-loading one-to-one keys between two sensors and refreshing the keys are practically impossible tasks. Public key cryptography is also considered to be computationally expensive for WSN and therefore, light-weight, yet reasonably secure key management schemes are crucial in order to bring about acceptable security services in WSN. In addition to this, any WSN security protocol has to be flexible and scalable enough to easily allow nodes to join or leave the network.

In this paper, we introduce a new approach for design and implementation of a security platform for sensor networks. At the heart of this approach is a flexible and scalable post-distribution key management module which provides basic cryptographic services. This component can be easily merged with other components and used to secure different operations at different layers. We propose a new security platform based on this module and describe its implementation details under TinyOs [12], a popular component-based event-driven operating system for WSN.

The organization of the paper is as follows: Section II gives a description of the design goals of our approach and the assumptions we make to achieve them. In Section III, we review proposed security platforms as well as some of the important available cryptographic key management schemes for WSN. Our key management module is described in detail in Section IV and performance measurements are presented. In Section V our proposed security platform is discussed. Finally, Section VI concludes the paper.

II. Problem Formulation

There are various WSN applications (military, health, etc.) that require a strong level of security. Consequently, a security platform that copes with constrained resources of nodes while being flexible and lightweight eases the application development process and contributes to widespread deployment of sensor networks. In order to provide such a platform we have made a few reasonable assumptions. We assume that the sensor nodes in the network are not mobile. We also suppose that the base station is safe and adversaries cannot compromise it. Our approach does not place *any* trust assumption on the communication apart from the obvious fact that there is a non-zero probability of delivering messages to related destinations.

We introduce the following requirements for a practical WSN security platform:

- *Flexibility*: Various security services have to be supported but not imposed to the application level communications.
- *Scalability*: Adding or deleting nodes has to have minimum overhead in terms of energy consumption and memory usage as well as having no effect on the functionality of the security scheme.
- *Transparency*: The provision of security services has to be performed transparently to other components or services.
- *Lightweightness*: The constrained resources of sensor nodes especially limited memory and computational power has to be taken into account at design time.
- *Node Capture Resistance*: The effects of node capture attacks should be minimal.
- *Simplicity*: The integration of this scheme with other services or components should have a minimal overhead.

In the following sections we will show how each of these requirements are addressed in our approach.

III. Related Work

There has been several research works trying to address security issues in sensor networks. In this section we give a brief overview of the available security solutions.

One approach to create secure platforms in WSN is providing link layer cryptographic primitives or libraries. TinySec [1], SecureSense [3] and SenSec [5] are examples of this approach. TinySec and SecureSense assume having *a global common secret key* among the nodes which is assigned before the deployment of the network and is used to provide security services such as encryption and authentication in link layer. The problem with this approach is that it is not resistant against node capture attacks in which an adversary can pollute an entire sensor network by compromising only one single node. In SenSec [5], there are three types of keys: *Global Key*, *Cluster Key* and *Sensor key*. The global key is generated by the base station, pre-deployed on each sensor node and shared by all nodes. This key is used to broadcast messages in the network. However, this protocol again falls prey to node capture attacks in which dedicated attackers can find this global key and broadcast commands or data to the network. Table I gives a brief comparison of these mentioned security solutions in WSN.

The provision of maximum level of security for all types of communication in each sensor node, as the one which appears in SenSec, is not appropriate to be used in a general security platform for WSN since it can lead to unnecessary waste of system resources and noticeably reduce the network lifetime. Although there has been an attempt made in SecureSense to address this issue, its solution cannot be well integrated with higher level services appropriately (we will discuss this in more details in section V).

TABLE I
Comparison of available security platforms in WSN

Security Scheme	TinySec	SecureSense	SenSec	Ours
No. of Key Types	1	1	3	3
Capture Resistance	no	no	partially	yes
Scalability	yes	yes	yes	yes
Flexibility	no	yes	no	yes
Ease of use	yes	yes	yes	yes

In other solutions, like secure information routing protocols such as SPINS [2] and LEAP [4] or security-aware middleware services such as secure localization [7] or secure time synchronization [8] cryptographic key management plays an important role. Generally there are three major approaches for key management in WSN namely, i) deterministic pre-assignment, ii) random pre-distribution and iii) deterministic post-deployment derivation. Examples of the first approach are SPINS [2] and LEAP [4] in which unique symmetric keys shared by the nodes with the base station are assigned before the network is deployed. Using this approach, cryptographically strong keys can be generated, however, this involves a significant pre-deployment overhead and is not scalable. Random-key distribution schemes like those in [9], [10] and [11] and PIKE [6] refer to probabilistically establishing pair-wise keys between neighboring nodes in the network. Usually in this approach a random subset of keys from a key pool is pre-assigned to every node; two nodes establish a pair-wise key based on the subset of the shared keys between them. This framework is quite flexible; the choice of protocol parameters determines the tradeoff between scalability and resiliency to node capture. However, most of the key pre-distribution schemes rely on sensor nodes to broadcast a large number of pre-loaded key IDs to find pair-wise keys between neighboring nodes, thus leading to a huge communication overhead. In addition, to guarantee network connectivity, each node has to store several hundreds keys or key spaces, which may greatly decrease the memory availability. In the third general approach, deterministic post-deployment key generation, nodes use some globally shared secret and pseudo-random number generators to derive the keys at runtime. LEAP [4] use this approach in order to establish pair-wise and group keys. A node erases the global secret after the completion of the initial key establishment phase to provide resilience against possible node compromises. However, most of the techniques based on this approach make it infeasible for even uncompromised nodes to generate new keys at a future time making these protocols inefficient for dynamic sensor network topologies. A comparison of the mentioned key management schemes is presented in Table II. PW stands for pair-wise keys, G stands for a global key common among all nodes, NB is the node-base key common between each one of the nodes and the base station, and BC is the broadcast key of each node common between the node and its neighbors.

TABLE II

COMPARISON OF KEY ESTABLISHMENT PROTOCOLS IN WSN

Protocol	SPINS	LEAP	PIKE	Ours
Preloading Overhead	yes	yes	yes	no
Capture Resistance	no	partially	partially	yes
Scalability	no	no	yes	yes
Preloaded keys	NB	NB,G	PW	G
Key Types	PW,NB	PW,NB,G	PW	PW,NB,BC

IV. KEY MANAGEMENT MODULE

In this section we describe our key management module. Our approach is a post-deployment key management scheme which addresses flexibility and scalability issues and is resistant to node capture attacks. Although this module forms the core of our proposed security platform as described in Section V, it can be used as a stand alone component as well. It can also be easily integrated with other components providing the related services to them.

A. Protocol

All of the direct communications in wireless sensor networks can be divided into the two types of one-to-one and one-to-many. To secure these communications our key establishment module establishes the following kinds of keys:

1) *Pair-wise (PW) key* that is established between two neighbors to protect their one-to-one communications.
2) *Broadcast (BC) key* that is established in order to secure the broadcast messages sent by a node to its neighbors.
3) *Node-Base (NB) key* that is established in order to secure the communication between a node and the base station (note that this communication is not necessarily direct). A message encrypted by this key, can only be decrypted by the base station.

Since the *pair-wise* and *broadcast keys* are essentially established among neighboring nodes, the first phase of key establishment is *neighbor discovery*. This is achieved in two steps by a pair of handshake messages. In the first step, node i broadcasts a specific type of message containing its ID so that every other node in i's communication range (like j for example) can receive it. We refer to this message as a *ping message*. Every node receiving the ping message answers back to the sender (i) with a *pong message* containing its ID (steps 1 and 2 in Table III). Node i can then add j to its own neighbor list. After a sufficient amount of time (see Table IV and more explanations in Section IV-B), i will discover all of its neighbors and this phase will be finished.

When the neighbor discovery phase is over, node i computes its own node-base key and its pair-wise keys with its neighbors as well as their broadcast keys as follows:

$$NB_i = F(i||baseStationAddress||K)$$

$$PW_{i,j} = F(min(i,j)||max(i,j)||K)$$

$$BC_i = F(i||K)$$

where "||" is the concatenation operator and F is a secure pseudo-random function usually implemented by a hash function such as SHA-1 or MD5. K is a global master key that is distributed to all nodes before deployment of the network. As we will explain later, K will eventually be deleted from the memory of the nodes in order to make the scheme more secure against node capture attacks.

TABLE III
STEPS OF KEY ESTABLISHMENT PROTOCOL

Step	Message
1	$i \rightarrow j : \{i\}$
2	$j \rightarrow i : \{j\}$
3	$i \rightarrow j : \{i, PW_{ij}, N_A\}_{NB_j}$
4	$i \rightarrow j : \{i, BC_i, N_B\}_{NB_j}$
5	$j \rightarrow i : \{j, N_A, N_B\}_{PW_{ij}}$
6	i Deletes master key K and node-base key of j

When these calculations are over, node i has a complete table of related keys. However, node j's key table is not quite complete as it does not have any entry corresponding to node i. Thus, node i has to send a message M containing these keys to node j. Obviously, M should not be sent in plain. Therefore, node i should calculate an appropriate key to encrypt M with it and then send the encrypted version of M to node j. A proper key, as we will see, is the node-base key of node j which can be derived by i as follows:

$$NB_j = F(j||baseStationAddress||K)$$

Having this key, node i can encrypt and send to j the key it shares with it as well as its own broadcast key. The related messages are the following (Steps 3 and 4 in Table III):

$$i \rightarrow j : \{i, PW_{i,j}, N_A\}_{NBj}$$

$$i \rightarrow j : \{i, BC_i, N_B\}_{NB_j}$$

where N_A and N_B are two nonces to guarantee the freshness of these messages[1].

After sending these two messages, node i will delete the node-base key of node j from its memory. Therefore the only non-base station node that can decrypt these messages is node j (note that we assume the base station is secure). Node i will also delete the master key K from its memory.

Upon receiving the keys, node j will answer back to node i by sending a message containing the nonces N_A and N_B. This message is encrypted with the pair-wise key of i and j (Sstep 5 in Table III). At this point, key establishment is complete.

Notice how this message exchange enforces the *scalability* aspect of our protocol: related keys can be established when a new node is added to a previously deployed network. Any new node that joins the network (such as i) can initiate the key establishment phase by broadcasting a *ping message*. Following that, related keys are calculated by the new node. Then the broadcast key of this added node, as well as its pair-wise keys with each of its neighbors are sent to related neighbors, encrypted with their node-base keys. Note that using the node-base keys for this purpose is quite an appropriate choice in order to make the protocol scalable and secure. This is because the already available network nodes have already deleted the master key K from their memory and consequently cannot use it to either calculate the keys or decrypt any message encrypted with it. It is not a good idea to use the broadcast key of previously joined neighbor nodes (like j) since other neighbors of j have that key available and can decrypt messages encrypted with it; a fact that results in providing a looser security scheme.

[1] The reason that message *M* is broken into two consecutive messages is only a practical nuance. The overall size of M – combination of the two mentioned messages – which would be 32 bytes (node ID is 2 bytes and the pair-wise key, the broadcast key and the nonce are 10 bytes each) is larger than the maximum allowed message size in TinyOs which is 29 bytes. Hence, we were forced to break M into two different messages. However, if keys are 8 bytes long then these two messages can be merged to one.

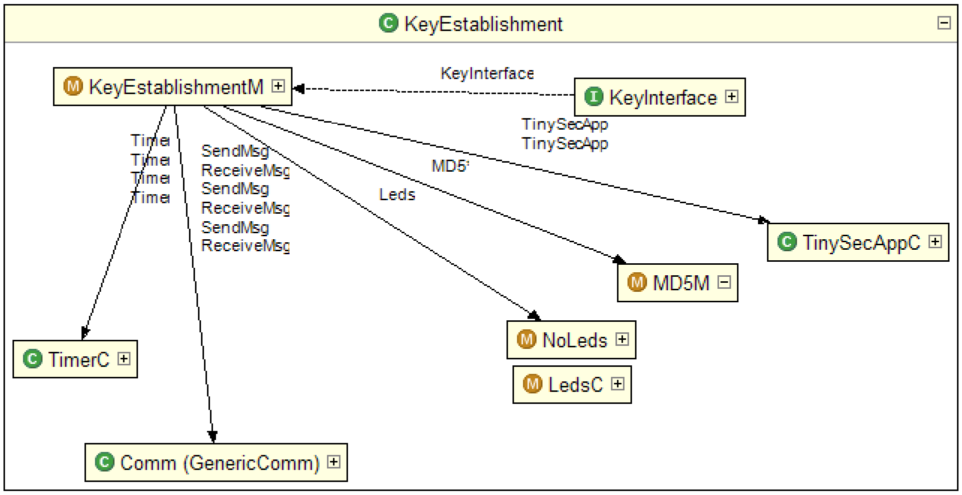

Fig. 1. KEY ESTABLISHMENT MODULE ARCHITECTURE

The deletion of master key K and the temporarily calculated node-base key of j by i as mentioned above, makes the protocol resilient to node capture attacks by reducing the effects of capturing a node to its neighborhood and *not the entire network*. Since the needed time for key establishment is negligible, we can assume that the adversary does not have enough time to find the master key K before it is deleted from the memory of the nodes (see also LEAP [4] for a similar assumption). On the other hand, newly joined nodes must come with the master key K in order to calculate the cryptographic keys. Therefore, the adversary cannot gain any useful information by introducing new nodes to the network as a result of not having access to K. In addition to that, it is important to note that if one of the above mentioned messages in key establishment protocol is not delivered, the receiving node will not get stuck. If node i does not receive the last message of the protocol (Step 5 in Table III), it will not add any entry for node j in its key table.

B. Implementation and Performance

Our key management module is implemented in TinyOs [12] which is an event-driven operating system commonly used on WSN nodes (motes). A program written in nesC, the programming language used for TinyOS, consists of one or more reusable components assembled or *wired*, to form an executable application. Since this wiring mechanism is independent of the implementation of components, each application can customize the set of components it uses. Therefore, unused components or services can be excluded from the application. In Figure 1, the components of our key establishment module are depicted.

As a stand alone library, the key establishment module implements an interface *KeyInterface* which contains three commands and three events. Command $init(k)$ initializes the key establishment module. When all of the keys are established among current nodes, the user will be notified by an *initDone*() event. Other commands provide security services such as encryption, decryption and MAC computation. The higher level application can use these services to design and implement its own security scheme. The complete platform discussed in Section V provides even more transparent security services embedded in implemented interfaces, *Send*() and *Receive*(). The MD5 module is used as pseudo-random function that is needed to establish the different type of keys.

The memory overhead of our key management module for each node can be calculated as follows:

$$Overhead\ M = [(|BC| + |PW|) * d] + |NB|,$$

where $|BC|$, $|PW|$ and $|NB|$ are the size of broadcast key, pair-wise key and node-base key respectively and d stands for the maximum number of neighbors each node may have[1]. The default size of all types of keys in

[1] In our current implementation of neighbor discovery phase, a node willing to discover its neighbors, broadcasts a ping message and waits for t milliseconds to receive pong messages from the potential neighbors. Yet it discards pong messages if they arrive after t milliseconds or if the number of discovered neighbors is already d. Values of d and t are decided during deployment time and play important roles in network connectivity.

TABLE IV

REQUIRED TIME AND ENERGY BEFORE THE GLOBAL KEY DELETION

Phase	Neighbor discovery	Key computation	Key Sending
Time	1000ms	10ms	10ms
Energy	1592640nJ	157nJ	38049000nJ

our key establishment module is 10 bytes, which provides strong security (2^{80} bit key space) for sensor network applications. As a result, in a very dense network where $d = 50$ will have $M \approx 1KB$. Although this value of d is far more than enough to keep the network connected, this memory overhead is well within the memory capabilities of motes (MICA2 motes have 4KB of RAM).

During the key establishment phase, prior to deletion of the master key, an adversary has a chance to find it and use it to derive all the other keys. However, this time is so small that probability of having an adversary capture a mote during it is minimal. Table IV shows the related duration that it takes to delete the master key from memory of a newly added mote during its initialization phase[2]. These results are of simulations using Tossim (an internal simulator coming with TinyOs).

The estimated amount of energy consumption for each phase of key establishment for the same network ($d = 50$) is presented in Table IV as well. This estimation was performed by multiplying the total amount of communications by an average communications cost of 18 $\mu J/bit$ (see PIKE [6] for a similar assumption). As a result, the estimated energy consumption of our key management scheme is approximately $0.4J$ (note that as presented in Table IV the energy needed for key computation is quite smaller than the needed energy for communication) comparing to PIKE-2D [6] that is more than $8J$ or PIKE-3D [6] which is around $6J$. This high energy efficiency of our platform comes with a comparable cost in terms of memory overhead; it uses about 1000 bytes of memory to establish and manage the keys while PIKE-2D and PIKE-3D need around 600 bytes and 500 bytes respectively.

In our scheme the effects of having a node captured is reduced to its neighborhood, i.e. the captured node's pairwise keys with its neighbors, its broadcast key and its node-base key are only keys that can be discovered by the adversary. This is a small fraction of established keys and secure communication still remains possible in other parts of the network.

Our code is built under TinyOs stable release 1.1.0 – also compiles and works under latest beta release 1.1.15 – and is integrated on MICA2 [12] motes.

V. PROPOSED SECURITY PLATFORM

Here we describe our new security platform for WSN.

A. Platform Features

Based on our key management module described in section IV and the basic framework provided by TinySec [1], the new platform has several advantages over known security platforms in WSN. This platform is designed to be transparent and easy to use. The process of key establishment as well as related computations regarding the provision of security services such as confidentiality and authentication is completely hidden from the applications using the platform. Moreover, the flexibility in the platform allows the developers to customize it for their application security needs, an important requirement, especially in resource constrained systems such as sensor networks. As different messages being exchanged in the network require different security services, a security platform has to be flexible enough to address all the security needs of different types of communications while not imposing extra overhead due to redundancies.

An approach similar to the one used in SenSec [5] is not appropriate to be used in a general security platform for WSN since it provides the highest possible degree of security for all of the exchanged messages. This flexibility is provided in our platform by using the most significant *three* bits of the data length field in the packet format as an indicator of the security service(s) to be used. These three bits are never used as the maximum data length

[2] The higher the value of t, the higher the time prior to the deletion of master key. The current value of $t = 1000ms$ as appears in Table IV is quite appropriate for a very dense network where $d = 50$ and all of the nodes are supposed to be able to discover all of their potential neighbors.

in TinyOs is appropriately chosen to be 29 bytes (see also TinySec [1] for a similar approach). Consequently, the provision of this feature comes with no overhead.

As another approach to bring run-time composition of security services, SecureSense [3] has introduced a new byte field in its packet format called SCID as an indicator of the services required by the message. SCID has replaced the TinyOs active message type (AM) field in order not to increase the packet length and consequently, the packet transmission time. However, the removal of active message type field introduces several major problems for upper layer services as it directly affects the *Active Message Model* of TinyOs. According to this model, each packet on the network specifies a handler ID that will be invoked on recipient nodes. When a message is received, the receive event associated with this ID is signaled. This mechanism allows different network protocols to operate concurrently without conflict. Removing the AM ID, therefore, significantly affects implementation of such protocols under TinyOs and introduces new complexities. Being motivated by TinySec, we provide run-time composition of security services *without* removing the AM ID or adding extra fields to support integration of services. Each one of the higher three bits of data field of the packet stands for a security service (from higher order bit to the lower: Replay Attack Protection, Access Control and Integrity, Confidentiality) and having *any* bit set means that the related service is provided for that packet. Thus the desired services for different packets can be composed at runtime.

B. Security Attacks and Services

This platform provides security against several types of attacks as follows:

- **Replay Attacks:** A common defense to protect a network from message replay attacks is to either timestamp the messages using some network time synchronization protocol or include a monotonically increasing counter in related messages in order to be able to detect replayed old messages and reject them. Providing time synchronization in WSN is usually quite complicated. Moreover, doing it in a secure manner is even more demanding and has to rely on a security platform like the one we provide here. As a result, we have decided to use the increasing counter value to guarantee the freshness of the messages. However, the memory cost of having every recipient maintain a table of the last received counter value of all of the other nodes is quite high. This is the reason why TinySec, as pointed out by the authors of its paper, does not provide acceptable security against replay attacks. As a matter of fact, a complete provision of such a service in link layer is not practically feasible. This is because information about the network's topology and communication patterns seem to be mandatory to provide protection against replay attacks, yet this information is not available in the link layer. However, having neighbor information available at the end of the neighbor discovery phase, our platform lets each sensor maintain only the last counter value for its *neighbors* and not for all of the nodes. Thus while a message is being routed to an specific destination, the counter value is updated on each hop. The amount of memory needed to keep the neighbors' counter value is quite small (for example, if each counter requires 4 bytes, the total amount of memory needed to keep the counter values in a *very dense* network where each node has 20 neighbors will be 80 bytes). This is a major advantage of our platform as to the best of our knowledge none of the available security platforms provide any acceptable solution to address replay attacks.
- **Node Capture Attacks:** Our platform is also the first one providing acceptable resistance against node capture attacks. This feature minimizes the effects of having the adversary capture a node to the neighborhood of that node –and not the entire network– while at the same time making no assumptions about nodes being tamper resistant. While tamper resistance might be a thorough solution for node capture attacks it is considered noticeably expensive for the sensor nodes that are intended to be very inexpensive and often no solution at all [13].
- **Denial of Service Attacks:** An effective security solution against denial of service attacks is to detect unauthorized packets before delivering them to application layer for further processing and stop them from spreading into the network. These types of packets are detected when they are received in link layer, providing not complete but sufficient protection against DoS attacks.
- **Message Modification and Impersonation Attacks:** Proper Message Authentication Codes (MAC) can be used to let the receiver nodes detect any modifications of received messages from the original one. The MAC generation is performed by applying a pseudo-random function usually implemented by a hash function to the concatenation of the message and the related established key. The default hash function that is used in our

TABLE V

PACKET FORMAT FOR FULL SECURITY SERVICE. NUMBERS ARE IN BYTES.

Destination	Length	AM	Source	Counter	Data	MAC
2	1	1	2	2	29	4

TABLE VI

OPERATIONAL MODES AND RELATED SETTINGS.

Mode	SetBits	Omitted Fields	Omitted Operations
"RAC"	111	-	-
"RA"	110	-	Encryption
"RC"	101	MAC	MAC
"R"	100	MAC	MAC & Encryption
"AC"	011	-	Counter Saving
"A"	010	Counter	Counter Saving& Encryption
"C"	001	MAC	Counter Saving & MAC
"-"	000	All Security Fields	All Security Operations

platform is MD5 however related settings can be easily changed to use any other function of choice such as SHA-1. Since the already established keys (like pair-wise keys established among the neighbors) are used to generate the message authentication codes, network nodes are able to verify the authenticity of the received messages. Using this service unauthorized nodes will not be able to send legitimate messages into the network. Thus, an access control service is also provided using generated MACs.

- **Attacks on Confidentiality:** In order to protect the messages being exchanged among the nodes from eavesdropping by unauthorized parties, appropriate encryption mechanisms are provided. The default cipher that is used by our platform for this purpose is *SkipJack*, however, the platform is not bound to use an specific cipher and related settings can be changed easily by the platform user (the other currently available cipher is RC5).

C. Packet Format

Table V shows the fields included in the platform packet format in the full security mode. The *Source* field of the packet is used to find the appropriate established pair-wise or broadcast key needed for the security services. Note that TinySec does not use the *Source* field in its *Authentication-Only* mode. This is because it assumes that if the attached MAC of a received message is valid then it comes from an authorized source (note that the MAC is derived using an specific *global* key shared among all valid nodes, a bad security practice as we explained in the introduction). However, this assumption is not necessary in our platform that uses established keys in order to resist against node capture attacks. As a result, we must include the *Source* field in the packet format.

In other related service modes, such as replay attack protection mode, the packet format contains a counter (*Counter*). Together with the *Source* field, the two bytes long counter can be used to avoid *IV* reuse in *CBC* encryption mode. In our platform, similar to TinySec, the *IV* includes the destination address, the active message type (AM) type, the data length, the source address and the counter. The $Source||Counter$ format guarantees that each node can send 2^{16} messages with the same AM type and the same destination but with different IV values. As mentioned, another application of the counter value is its role in providing resistance against replay attacks.

Different major security options that are provided include i) Authentication, Access Control and Integrity (A) ii) Confidentiality (C) iii) Replay Attack Protection (R). In the first mode, the *Counter* field is not required, but obviously the *MAC* field is needed. In the second mode, *Source* and *Counter* fields are used in the packet format, however receiver nodes do not save the related counter values. *Source* and *Counter* fields are also necessary in the third mode, but the counter value of each neighbor is kept. As we mentioned earlier, the combination of these modes is also possible in order to provide a combination of security services. Table VI shows more details of different modes, provided services and the related packet format.

VI. CONCLUSION

In this paper we introduced a new link layer security platform for wireless sensor networks. At the heart of this platform lies our post-distribution key management module allowing for the provision of several security services such as acceptable resistance against node capture attacks and replay attacks. It is lightweight and allows for high scalability while being easy to use and transparent to the users. This platform is flexible enough to allow different types of security services for different types of communications among nodes.

References

[1] C. Karlof, N.Sastry, D. Wagner, “TinySec: Link Layer Encryption for Tiny Devices”, *ACM SenSys*, 2004

[2] A. Perrig, R. Szewczyk, V. Wen, D. culler, D. Tygar, “SPINS: Security Protocols for Sensor Networks”, *ACM CCS*, 2003

[3] Q. Xue, A. Ganz, “Runtime Security Composition for Sensor Networks (SecureSense)”, IEEE Vehicular Technology Conference, 2003

[4] S. J. S. Zhu, S. Setia, “LEAP: Efficient security mechanism for large-scale distributed sensor networks”, *ACM CCS*, 2003

[5] T. Li, H. Wu, F. Bao, “SenSec Design”, Institue for InfoComm Research, Tech. Rep. TR-I2R-v1.1, 2005

[6] H. Chan, A. Perrig, “PIKE: Peer Intermediaries for Key Establishment in Sensor Networks”, Proceedings of IEEE Infocom, 2005

[7] S.Capkun, J.P. Hubaux, “Secure positioning of wireless devices with application to sensor networks”, *IEEE Infocom*, 2005

[8] S. Ganeriwal, S. Capkun, C. C. Han, M. B. Srivastava, “Secure time synchronization service for sensor networks”, *ACM WiSe*, 2005

[9] L. Eschenauer and V. D. Gligor, “A key-management scheme for distributed sensor networks”, *ACM CCS*, 2002

[10] H. Chan, A. Perrig, D. Song, “Random Key Predistribution Schemes for Sensor Networks”, *IEEE Symposium on Security and Privacy*, 2003

[11] D. Liu, P. Ning, “Establishing pairwise keys in distributed sensor networks”, *ACM CCS*, 2003

[12] J. Hill, et al, “System architecture directions for networked sensors”, in *Proceedings of ACM ASPLOS IX*, 2000

[13] Anderson, R., Kuhn, M.: Tamper resistance - a cautionary note. In: Proc. of the Second Usenix Workshop on Electronic Commerce, (1996) 1–11