SecRose: a data transportation layer security mechanism for wireless sensor networks

Ekonomou, E and Booth, KM

SecRose: a data transportation layer security mechanism for Wireless Sensor Networks

Elias Ekonomou Kate Booth
E.Ekonomou@pgr.salford.ac.uk K.M.Booth@salford.ac.uk
Centre for Networking and Telecommunications Research
School of Computing, Science and Engineering
The University of Salford
Salford, Greater Manchester M5 4WT, UK

***Abstract*- We introduce the final version of SecRose, a mechanism for secure transferring of data in a Wireless Sensor Network (WSN). SecRose is based on improvements in existing proposals plus a set of distinctive novel characteristics. Major Features include encrypted and authenticated communications, integrated key management and categorisation of communication types. SecRose is secure, scalable and extremely lightweight.**

I. INTRODUCTION

A. Wireless Sensor Networks

A typical wireless sensor network (WSN) consists of a number of wirelessly interconnected sensor nodes that are used to gather information from the environment. They might operate in a hostile environment and they should be resistant to physical or electronic attacks. They can acquire information using any standard electronic component, e.g. microphones, photometers, thermometers etc and so they can be used for a number of tasks. They can operate in a wide variety of applications including military, commercial, industrial and even domestic or personal applications.

The network consists of sensor devices which use a single integrated circuit which embeds all the electronic components required. This chip interfaces with a sensor device which could be mechanical or electronic or both. The whole sensor is powered by a small battery which means the network's life is highly dependent on the energy consumption of the sensor.

In addition to the sensors the network uses a base station which is the network's interface point to the rest of the world. The base station is usually a more powerful and less energy dependent device than the sensor devices. Some networks might include more than one base station.

Existing WSNs are mainly experimental and might consist of up to a few hundreds of sensors. Future WSNs are expected to consist of tens of thousands of nodes, although the actual size of the future typical sensor network is disputed[1].

Each sensor node is a mini computer, complete with processing unit, volatile and non volatile memory and interfacing capabilities via radio or LEDs. It also includes a minimalistic operating system, with TinyOS [2] being the current standard.

B. Research objective

The area of WSNs attracts considerable research interest mainly because their greatly exciting potential. In order to achieve that potential the research community has to overcome the security obstacle which poses great challenges [3], [4].

Privacy and security is an essential requirement of many applications in the modern world. By enabling security in the WSNs, we create the potential of using them for demanding requirements. A well designed security protocol is essential for the further development and the success of wireless sensor networks.

Our objective was to provide an efficient and easy to use security mechanism for WSNs. The process of achieving our objectives resulted in a paper [5] which described our approach and the initial design. In this paper we describe the final design of SecRose. The mechanism, at its present form, meets our security requirements without significant energy cost.

C. SecRose design goals

Since every WSN application is subject to its own threat model the security requirements are not homogenous. Therefore we had to set a number of security properties and require them to be provided by SecRose in order to characterise it as secure. These properties were originally defined in [3] and are as follows:.

SecRose must provide data confidentiality. Data have to be encrypted in a way difficult enough to avoid any realistic possibility of reading them while they are still useful.

It must also provide data authentication and integrity. We have to guarantee that the identity of the source is indeed what is claimed to be and that the data have not been altered during transit.

Finally, we have to provide data freshness so that an attacker cannot record and replay messages that would appear valid to the network. A solution to this problem usually means that the network is also protected against cryptanalytic attacks.

We had to provide the above security properties utilising the minimum possible energy. It was also desired to provide the security properties transparently. We would enhance the TinyOS's equivalent of the data transportation layer, not the way that applications interface with that layer. SecRose would therefore be extremely portable and migration from plain TinyOS to SecRose-enabled applications would be effortless.

II. OTHER WORK

Research in the past was largely conducted under military funding and thus affected by military goals set by the funding institutions. Their goal was to limit the effect of node capturing, also known as node compromise by an attacker. Such event would give access to the memory of the captured node which would include the keys in use. If these were revealed then the whole security of the network would be compromised. Proposed countermeasures to node capturing included key management schemes [6], [7], [8] and location-aware security [9]. Most of that research was done for extremely large networks, which are yet to be deployed anywhere.

However, recent advantages in integrated circuit design have resulted in more secure microprocessors. This statement is supported by the work in [10], [11], [12] and [13]. Therefore we argue that the whole idea of node capturing is certainly outdated. Not only work in other disciplines challenged node capturing, but even the most recent work in WSNs does not address the issue.

The attempt for traditional secret-key solution was introduced in [4] but it was never fully described or implemented [14]. Subsequently the research seems to have shifted in providing lightweight cryptographic mechanisms, for example, TinySec [14] which is the mechanism that is currently shipped with TinyOS and included in commercial products like the ones from Crossbow [15]. We are also aware of the work of Li *et al.* in SenSec [16] and Luk *et al.* in MiniSec [17]. Unfortunately we were unable to find an implementation of SenSec for reference and evaluation and thus we had to rely on their published work only in order to identify its performance potential.

III. THE SECROSE DESIGN

A. Design Assumptions

We assume a network consisting of a few hundreds of nodes. The nodes are mica2 models from Crossbow [15], one of the less capable commercially

available node. A mica2 node has an 8MHz processor, 128KB runtime memory, 512KB storage memory and 4KB non-volatile memory. They can communicate at the rate of 38.4Kbps and they run TinyOS.
Traffic flow follows a pattern determined by the application. We believe that most traffic is directed from the nodes to the base station. The reason for that assumption is derived from the ultimate goal of a WSN; to get information from the network. There are also two more possible flow patterns. Fist, the base station will occasionally broadcast a command or other data to the whole network. Secondly the network nodes, or the base station, can directly communicate with each other. We assume that the frequency of the last case is significantly less that the other two cases.
We also assume that all the sensors use integrated circuits that are tamper-resistant. Thus, in case a node is captured the attacker is unable to extract data from the sensor.

B. Packet types

As we distinguish three possible communication types we have defined three types of packets to cover all the possible communication needs inside a WSN while allowing for maximum optimisation. A fourth type is used for state preservation but it allows for many more future uses.
A packet of type *normal packet* is used when a node wants to communicate with the base station. We expect this type of packet to be the most used type during the lifetime of a typical WSN. *Normal packets* lack a destination address, since that is always the base station. Fig. 1a shows the format of a *normal packet*.
A packet of type *broadcast packet* should be used by the base station if there is a need to command or query the whole network. Broadcast packets lack source and destination address since they are can only originate from the base station and they are destined to the network in general. Fig. 1b shows the format of a *broadcast packet.*
Finally, a packet of type *long packet* is to be used in point-to-point communications between any given pair of nodes in the network, including communication from the base station to a specific node. A *long packet* does not lack source and/or destination address as the other packet types do. Fig. 1c shows the format of a *long packet.* The fourth type, a *control packet,* is a specially constructed normal packet. That type is explained further in section IV.

C. Key Management

The key management system represents the network as a set of communication pairs. Each communication pair has its own *pair key* which is derived from an *initial key*. The *initial key* is stored in the pairs during deployment and can be uniform for all the pairs. Even a broadcast communication is regarded a pair, with the base station on one side and the whole network on the other side of the pair.
The *pair keys* are derived using the *initial key* and a *counter* value. The *initial key* is 96 bits and the *counter* is 32 bits. The pairs are required to produce a new *pair key* after each successful communication. The whole process is further explained in section IV.
The method guarantees data freshness, as keys are usually not reused. It also prevents cryptanalysis, since the same plaintext would produce a different ciphertext. This scheme shares some of the properties of key management schemes and thus it also limits the impact of a node compromise attack.

a. Normal packet:

Len	*Source*	*Type*	*Group*	*Data*	MAC
1	2	1	1	0…29	4

b. Broadcast packet:

Len	*Type*	*Group*	*Data*	MAC
1	1	1	0…29	4

c. Long packet:

Len	*Source*	*Dest*	*Type*	*Group*	*Data*	MAC
1	2	2	1	1	0…29	4

Fig. 1. SecRose packet types. All numbers express sizes in bytes. The fields Type and Group are irrelevant to our mechanism but required parts of all packets by the TinyOS. Text in *italics* is protected by the MAC. Text in *underline italics* is encrypted and protected by the MAC.

D. Flags

All packet types defined before are identified by a flag. There are four packet types and thus 2 bits of information were required in order to represent them. This 2-bit value can be safely overloaded with the value of the packet's *length*. We have also moved the *length* in the very beginning of the packet's data stream so that it allows for appropriate and rapid handling of any packet.
The length of the packet previously contained the amount of data payload in this packet. That is a value in the range of 0 to 28 which can be represented using 5 bits. So the first 3 most significant bits of the 8-bit data length value are always unused. We overload the 2-bit flag in the first 2-bits of the length in order to eliminate the flag's overhead in the radio. We end up with a robust mechanism with significant saving potential and minimum overhead.
The use of flags allows us to save significant amounts of energy by introducing different packet types. Energy is particularly saved when using *normal packets* and *broadcast packets.* Since both types have a predetermined destination, the nodes can determine the packet type and its destination by just reading the flag. They can therefore save energy by not sending the destination address at all.
The flag mechanism, in association with a routing table, can help a node determine if a packet is intended to be forwarded by itself or not. Neighbour nodes that happen to overhear a communication can stop receiving the rest of the packet at a very early stage, saving the otherwise wasted energy.

E. Message authentication

Message authentication provides protection against a possible attempt to alter a message during transit. It also guarantees that the source of the data is the expected source and not an adversary. We authenticate our data by replacing the aged CRC value of the packets with a Message Authentication Code. Such code is generally a small amount of data appended in the end of the packet. SecRose produces a MAC by feeding the *pair key* and the data stream to our encryption function as recommended by Dworkin in CMAC[18].
We selectively authenticate the important parts of each packet depending on the packet type. As illustrated in Fig. 1, the *AM, Group* and *Data* contents of the packet are always authenticated by the MAC. In addition we authenticate the *source* in *normal* packets and the *destination* in *long* and *control* packets.
We state that a 32-bit MAC provides adequate protection against MAC collisions, intentional or not. The length of the maximum possible protected data stream is sufficiently small to be protected by a 32-bit MAC. Nevertheless, the CMAC produces a code that is equal in size with the block size of the block cipher used. In our case that size is a minimum of 64 bits. We use the rest of the 64 bits for other functionality.

F. Counter calculation

As stated above, the *pair keys* are produced by the *initial key* and a changing *counter* value. The counter is not simply incremented by one. It is incremented by the value of the 5th byte in the output of the MAC calculation. Thus after each packet's MAC is calculated the counter is also determined and is derived by the actual data of the packet plus the previous value of the counter.
Each time a MAC is calculated the increment of the *counter* is also determined. When the MAC is later verified by the receiving node, it is also able to calculate the new counter value as well. This method allows us to maintain a counter state without transmitting its value over the radio and thus saving the required energy and the potential security implications.

G. Acknowledgements and maintenance of state

Since we use counters to produce the pair keys we also need to be assured that the state of these counters is properly maintained for each node in the pair. Loss of counter state leads to a broken communication capability. We use an acknowledgement method to ensure that state is maintained.
Each receiver is required to send an *ACK packet* back to the sender each time a valid packet with a valid MAC is received. An *ACK packet* is essentially a *control packet*, complete with its MAC, which contains no actual data payload. However, it is prepared using the new counter value, as derived from the received packet.
Should the receiver be unable to verify the new MAC of the *ACK packet* or if there is no *ACK packet* at all then the communication in this pair is either problematic or under attack. It is much more possible for a radio link to be problematic than to be under attack. Therefore SecRose can be programmed to allow for a number of communication attempts before a pair is designated as broken. The exact mechanism of how this happens is described in section IV.

The novelty of our acknowledgement method is that we do not send a traditional ACK value between hops each time a packet is sent/forwarded over a hop. That method would be a security disadvantage[19] and would waste energy for a functionality that is already covered by the *ACK packet*.

H. Encryption function

We recommend, and use, the Tiny Encryption Algorithm [20] (TEA) as the encryption function for SecRose. The particular algorithm was selected for its security properties and for its small block size that suits small plaintext lengths. The most important property of TEA is that it combines both a small block size and a large 128-bit key. It also features small memory footprint and limited computational requirements. Its latest revision was published by Russell under the name of Corrected Block TEA in 2004.

Although we believe that TEA is the most suitable for use in WSNs, there is no security risk in using any other encryption function with the SecRose mechanism. The features of our algorithm do not rely on the encryption function itself but in how the function is used.

IV. Description of Operation

A. Preparation

The first step in sending a packet is to determine and prepare its flag. The sender can determine the packet type at transportation level by looking on the address that the application wishes to send to. If the address be the base station address then the flag is 0. If it is the broadcast address the flag becomes 1. At that stage anything else has a flag of 2. After determination the flag has to be overloaded in the length field of the packet. This is achieved by appropriately flipping the 2 most significant bits of the length in order to represent the value of the flag.

The next and final preparation task is to encrypt the *data* of the packet using the current *pair key*. The *pair key* is produced as explained in paragraph H of this section.

B. Packet type specifics

After the initial task the sender has to calculate the MAC of the packet. The process involves determining what needs to be protected, as implied by the design and explained in section III. Then the encryption function is employed to produce a MAC code as defined in CMAC. This process also produces the exact value to increment the *counter* with. However that value is not used at this stage.

While sending the packet the sender must keep track of what information to send, based on the design. Specifically, *broadcast packets* need neither the *source* nor the *destination* to be sent, while *normal packets* replace the content in the space allocated for the *destination* address with the *source* address.

C. Post-send counter handling

After sending a packet the sender updates the *counter* with the increment value produced while calculating the MAC. Then the counter is stored in a temporary memory location, identifiable by the node ID of the receiver. The receiver's node ID is also the ID of the pair, from the sender's perspective. Should the destination be the broadcast address then a special value is used as the pair ID.

At this stage the sender can continue with other tasks while it waits for the *ACK packet* to be received.

D. Reception

On the other end of the communication the receiver is notified about the incoming packet event and the first action to execute is to receive and examine the *length* of the packet. The *flag* can be extracted from the *length*, since it was overloaded there by the receiver. After determining the *flag* the receiver can properly receive the packet's fields using the equivalent to the sending process.

Upon complete reception the node has to calculate the MAC of the packet and compare it with the MAC received over the radio. Again the packet type specifics have to be taken into account for appropriate MAC calculation and the counter value is produced as well. Packets with a verified MAC are then decrypted and higher layers are notified about the successful reception event.

If the MAC does not verify with the current key the receiver is allowed to use the *backup key*. That is the key used in the most recent previous communication, as defined below.

Should the MAC fail to verify again the receiver is powerless to solve the problem and thus it has to ignore this packet.

E. Sending of acknowledgement

If the MAC is verified the receiver should proceed with sending the *ACK packet* to the sender. The initial stage for this process is to update the current counter with the new counter increment value, which was calculated before. Then the *ACK packet* is prepared. That packet is a *long packet* with *length* equal to zero and 3 as the packet flag.

The packet is immediately sent over the radio as for any other packet. The receiver saves the new current counter while keeping a backup of the previous counter. Then the receiver can proceed with other tasks.

F. Receiving the acknowledgement

When the sender receives the *ACK* packet it should immediately identify it using its flag and it should also be able to properly receive it using the normal reception process.

After reception the MAC has to be verified and upon successful verification the sender has to re-assign its current and backup keys. The temporary key that was stored after sending now becomes the current key. The current key is kept as a backup key.

If the MAC cannot be verified the receiver is allowed to use the backup key and try verification again. If the verification fails again the receiver does nothing. Although that fact means that the next time will reuse an already used key we have to allow it to happen in order to accommodate possible radio problems. The amount of times that a key is allowed to be reused can be programmed in SecRose.

G. Intermediate nodes

All intermediate nodes can determine the destination of a packet. This is true since the *destination* field is not encrypted and the *flag* designates *normal* and *broadcast* packets. They can therefore consult their routing table and decide if they ought to receive and forward this packet or not. Nodes that belong to the path of a specific pair should receive and immediately forward the packet byte-to-byte without attempting to decrypt or validate it.

H. Counters and key production

Before each cryptographic or MAC calculation the node has to mix the active *counter* with the *initial key* in order to produce the *final key*. The process is executed by breaking the 96-bit *initial key* into four 24-bit blocks and the 32-bit *counter* into four 8-bit blocks. Then the first block of the *initial key* becomes the first block of the *final key*, followed by the first block of the *counter*. The process is repeated four times until the *final key* is finalised. Fig. 2 illustrates this process.

V. Evaluation

A. Security properties

As mentioned before, SecRose was designed with a set of goals that defined when a communication is secure. These goals state the need to provide confidentiality, authentication, integrity and freshness. Other work in the area also meets these goals, namely TinySec [14], SenSec [16] and MiniSec [17].

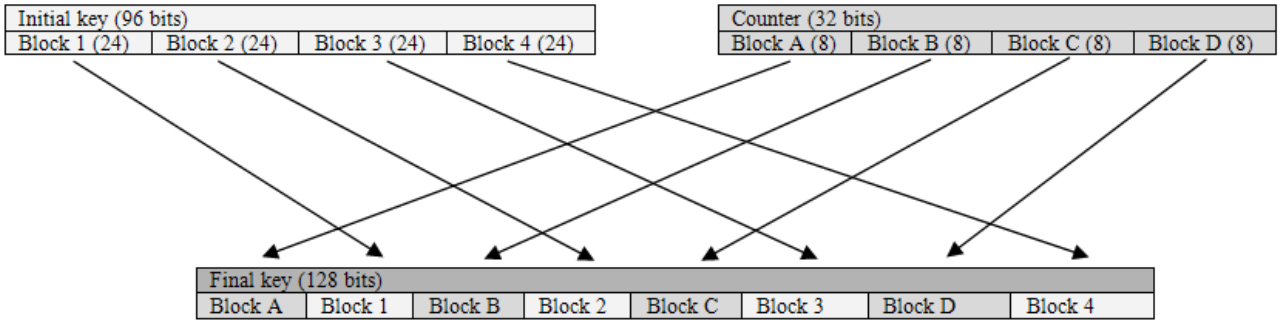

Fig. 2. Final keys are created from a 96-bit key and a 32-bit counter. The numbers in parenthesis represent the length of the block in bits.

This paragraph demonstrates why SecRose proposes a better solution to achieve most of these goals than the other mechanisms.
In order to provide confidentiality, all mechanisms follow the logical path of encrypting the exchanged data. The difference of SecRose is our integrated key management mechanism and the use of TEA for encryption. While the other proposals make no mention of how the keys are deployed and managed, we have integrated a process to do exactly that. The key management is an essential part of SecRose and poses an extremely small energy overhead while greatly enhancing the overall confidentiality of the exchanged data. Also, our mechanism protects the acknowledgement packets as well, a fact that is not seen in any of the other designs.
Additional differences regarding confidentiality lie in the selected block cipher of the other proposals. SecRose and MiniSec use the Skipjack block cipher [21] for their cryptographic needs. Although Skipjack is suitable because of its small block size it only uses 80-bit keys. To tackle this problem, SenSec uses a variant of Skipjack which accepts 128-bit keys but the security of the variant is yet to be analysed by the cryptographic community. On the other hand, TEA operates with 128-bit key lengths and it was designed like that from the very beginning.
All the other proposals use initialisation vectors (IVs) in order to provide data freshness and to minimise the vulnerability of the design to a cryptanalysis attack. Such vectors are essentially counters which are altered with every communication and are then fed to the encryption function as a part of the data payload which needs to be encrypted. This process guarantees that repeated encryptions of the same plaintext always result in different ciphertext. The IVs are sent over the radio as a part of the packet. SecRose also uses a pseudo-randomly incremented counter for the same task but in a fundamentally different approach; our counter is not transmitted with the packet, it is kept secret.
The major advantage of this approach is that the counter does not have to be small. Currently we use a 32-bit counter while the IVs of TinySec and SenSec are only 16-bits. Only the IV of MiniSec is of equal size to SecRose. But the IVs are constructed including the destination address, AM type and Group values. This makes the IVs inferior to a pure counter since they are not guaranteed to be variable at all times. Therefore SecRose provides greater freshness that the other proposals. This allows for greater complexity and higher traffic volumes while it leaks less cryptanalytic information.
Finally, all mechanisms use their encryption function in CBC mode to produce a MAC. The MAC guarantees message authentication and integrity. SecRose follows the same approach and as far as we can say there is no observable difference in any way.

B. Energy efficiency

Energy is the asset that has to be paid to obtain security. It is therefore generally accepted that a security mechanism will be less efficient and slower than a plain one. SecRose offers a number of efficiency improvements and we are confident that the overall impact of SecRose in the network's lifetime is lower the other security mechanisms in most cases.
The main energy-saving contributor is SecRose's distinction of packet types and the way of how this distinction is implemented, the flags. The flags consume no extra radio energy while at the same time allow for small energy gains in every single normal of broadcast packet transmitted. The energy savings of the flags are illustrated in Fig. 3 which shows a comparison of SecRose's packets against the other available solutions. Actual benchmarks show that this difference is indeed demonstrated in real life scenarios.
The flags also allow for further savings in potential future versions of SecRose. For example small *control packets* might be used by the application to achieve a suitable communication requirement. They also allow for early rejection and reduced routing information. These features are already implemented in the demonstration of SecRose.

C. Design novelty and completeness

The first transport layer security protocol for WSNs was TinySec. SenSec and MiniSec are based on TinySec and they attempt to produce a similar result with greater energy efficiency.
SecRose was not based on TinySec and it does not simply attempt to improve the energy performance of TinySec. We use a number of completely novel ideas and as a result SecRose incorporates security and efficiency features not present in other security mechanisms. The existence of these unique features allow for an elevated security of the overall WSN. For example we protect the acknowledgements. This allows for dramatic improvements in the security of existing routing protocols, because many of them suffer from an attack known as ACK spoofing [19].
TinySec is the most complete and ready to use security mechanism for WSNs but SecRose is very close to that as well. Although the MiniSec's team have also released some code, this code is limited only a particular sensor node model. The completeness status of SenSec is unknown.

VI. FUTURE WORK

The SecRose mechanism is almost complete but there is always room for improvements. A potential future version could include:.
More functionality for maintaining the state of the counter is possible. Section IV describes two cases where the nodes are powerless to solve a problem. Although this is currently not a major issue it is undesired and poses potential security implications. These problems can be solved by requesting assistance from neighbouring nodes or from the base station, using control packets.
We have also identified a duplicate effort that results from the use of SecDed encoding in TinyOS and a MAC in SecRose. The purpose of the SecDed encoding is to identify an infinite number of transmission errors and correct a finite number of them. But the MAC can also identify transmission errors. We intend to shift the functionality from SecDed to the MAC and thus save significant amounts of currently wasted energy.

VII. CONCLUSION

We have introduced SecRose. It is the latest security mechanism for use in Wireless Sensor Networks. We have shown how it meets its design requirements, how it operates and how it compares with other existing mechanisms that share common goals. We believe that SecRose benefits from certain unique features and also builds on existing ideas currently present in the literature. The result is a robust, complete and improved mechanism that is implemented and available to use with any existing or future WSN application.

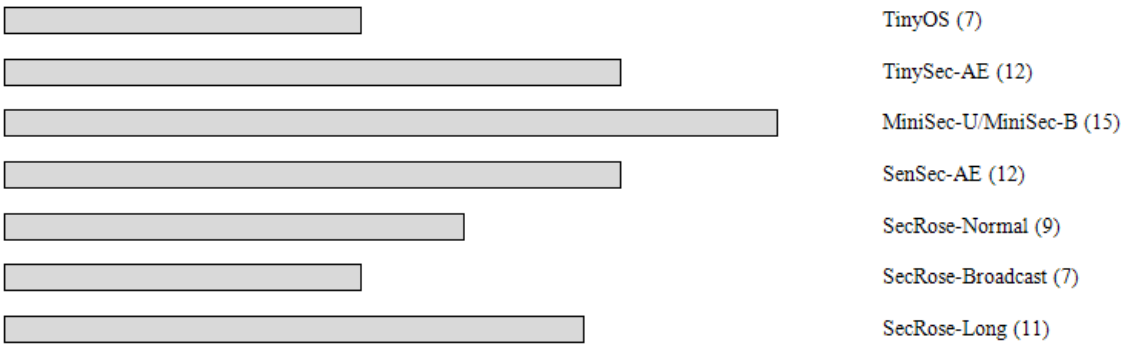

Fig. 3. Comparison of packet overhead of all proposed solutions. Numbers in parenthesis indicate the size in bytes.

REFERENCES

1. Gamage, C., et al. Security for the Mythical Air-Dropped Sensor Network. in Proceedings of the 11th IEEE Symposium on Computers and Communications. 2006.
2. Berkeley, U., *TinyOS*. 2004.
3. Perrig, A., J. Stankovic, and D. Wagner, *Security in wireless sensor networks.* Communications of the ACM, 2004. **47**(6): p. 53 - 57.
4. Perrig, A., et al., *SPINS: security protocols for sensor networks.* Wireless Networks, 2002. **8**(5): p. 521 - 534.
5. Ekonomou, E. and K. Booth. Securing data transfer in Wireless Sensor Networks. in Seventh annual postgraduate symposium (PGNet). 2006. Liverpool / UK: Liverpool John Moores University.
6. Pietro, R.D., L.V. Mancini, and A. Mei, Energy efficient node-to-node authentication and communication confidentiality in wireless sensor networks Wireless Networks, 2006. **12**(6): p. 709-721.
7. Sencun, Z., S. Sanjeev, and J. Sushil, LEAP+: Efficient security mechanisms for large-scale distributed sensor networks. 2006, ACM. p. 500-528.
8. Du, W., et al. A pairwise key pre-distribution scheme for wireless sensor networks. in Conference on Computer and Communications Security. 2003. Washington D.C., USA: ACM Press.
9. Yang, H., et al. Toward resilient security in wireless sensor networks. in Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing. 2005. Urbana-Champaign, IL, USA.
10. Atmel, ATmega 128 & ATmega 128L. 8-bit AVR microcontroller with 128K Bytes in-system programmable flash. 2004.
11. Kömmerling, O. and M.G. Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. in USENIX Workshop on Smartcard Technology. 1999. Chicago, Illinois, USA.
12. Hess, E., et al. Information Leakage Attacks Against Smart Card Implementations of Cryptographic Algorithms and Countermeasures. in EUROSMART Security Conference. 2000.
13. Skorobogatov, S.P., Semi-invasive attacks - A new approach to hardware security analysis, in Computer Laboratory. 2005, University of Cambridge: Cambridge.
14. Karlof, C., N. Sastry, and D. Wagner. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. in Second ACM Conference on Embedded Networked Sensor Systems (SensSys 2004). 2004.
15. Crossbow, Crossbow Technology Inc.: http://www.xbow.com. 2004.
16. Li, T., et al., *SenSec Design. Tech. Rep.* 2005, Institute for Infocomm Research: Singapore. p. 15.
17. Luk, M., et al. MiniSec: a secure sensor network communication architecture. in Proceedings of the 6th international conference on Information processing in sensor networks 2007. Cambridge, Massachusetts, USA: ACM.
18. Dworkin, M., *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*. 2005, National Institute of Standards and Technology. http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf.
19. Karlof, C. and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. in First IEEE International Workshop on Sensor Network Protocols and Applications. 2003.
20. Wheeler, D. and R. Needham., *TEA, a tiny encryption algorithm*. 1994.
21. National Institute of Standards and Technology, *SKIPJACK and KEA Algorithm Specifications*. 1998.