



# Encryption and Key Exchange in Wireless Sensor Networks

Erwin\*

January 15, 2013

Encryption and key distribution are important primitives to build secure Wireless Sensor Networks (WSN). Different block ciphers were proposed in literature to provide encryption in resource constraint distributed networks. A large amount of different key distribution schemes were implemented, targeting different types of WSNs. These schemes face issues with respect to their requirements, implementations, and theoretic foundations.

In this paper we provide an overview of selected encryption schemes and introduce so called fault attacks targeting hardware implementations of block ciphers. Selected key pre-distribution schemes, like random key pre-distribution and hierarchical key distribution, are presented and depicted. Memory efficient Elliptic Curve Cryptography (ECC) introduces feasible public key cryptography schemes for WSNs. A summary of selected ECC based schemes is given with a focus on Identity Based Encryption (IBE), which is especially useful in scenarios with a trusted sink node.

## 1 Introduction

Wireless Sensor Networks (WSN) are increasingly deployed in environments where data integrity, confidentiality and authentication of senders becomes an important requirement. Considering a WSN forming a Body Area Network (BAN), several low-cost nodes are attached to the human body to collect sensor data. In most body

---

\*Name has been changed for editorial reasons

networks, this data is periodically transferred to a sink node for further processing. Sink nodes are also often designed to work as gateways to transfer data to eHealth systems residing in the cloud. These BANs could be deployed in hospitals or at homes for ambient and assisted living monitoring elderly. Patient data is often classified as high privacy information that must be transferred in an encrypted form, without revealing sensitive contents. Security implementations in BANs must also ensure that nobody can inject faked data as this can have serious impacts, e.g., wrong prescription of drugs. Thus data packets need to be authenticated to allow a verification of their origin. While an attacker could have state-of-the-art hardware to perform attacks, sensor nodes are constraint embedded platforms, which results in an unusual challenge regarding encryption and key exchange. To keep energy consumption low, nodes have limited computing power, small RAM, and low storage capacity. Thus, classic public key cryptography based on RSA trapdoor function is not suitable due to its high computational overhead. The complexity of RSA operations does not scale linear with their key size [1], which makes them too costly when initialized with parameters defined as sufficiently secure by the National Institute of Standards and Technology (NIST) [2] or other standard institutions<sup>1</sup>. As today's key management schemes heavily rely on public key cryptography, researchers have proposed several lightweight alternatives to RSA based key management. Once keys are exchanged and authenticated, efficient block ciphers are required to encrypt network communication in real time. Besides block ciphers, which were designed for a small memory footprint and smaller block size, modern microcontrollers used in sensor nodes come with implementations of the Advanced Encryption Standard (AES) [3].

In our scenario, where a WSN is attached to a body and collects patient data, special issues need to be resolved. These WSNs face interoperability issues and disruptive connections between its nodes, meaning that an end-to-end connection is not always available. Especially in the case of data transmission between meeting humans, a connection is only available ad hoc at random meetings or at specific meeting times. Therefore approaches exist to deploy network stacks implementing a Delay-Tolerant Network (DTN) [4].

In section 2, we provide an overview of software implementations for encryption. We study software modules for TinyOS<sup>2</sup> [5] and Contiki<sup>3</sup> [6] and look at their choice of block ciphers and operation modes for authenticated encryption. Furthermore, precautions regarding hardware implementations are presented and an introduction into fault attacks is given. Besides presenting the challenges of using block-ciphers, the following sections mainly focus on key management as it often introduces new and interesting problems. An overview of key pre-distribution schemes is given in section 3, where random key pre-distribution and hierarchical group key management is presented in detail. In section 4, we look at schemes using public key cryptography based on Elliptic Curve Cryptography (ECC). As part

---

<sup>1</sup>A comparison of recommendations for key sizes can be found at <http://www.keylength.com>.

<sup>2</sup>TinyOS website: <http://www.tinyos.net>

<sup>3</sup>Contiki website: <http://www.contiki-os.org>

of ECC schemes, modern research on Identity Based Encryption (IBE) is presented, before concluding the seminar paper.

## 2 Encryption

Because of RAM and performance constraints, block ciphers need to be optimized regarding memory consumption and execution time when implemented for sensor nodes. Also energy consumption and protocol overhead have to be kept small for better adaption into these distributed networks.

We argue that the selection of appropriate block ciphers should not compromise on security, as attackers conceivably are equipped with powerful hardware. It is also imaginable that attackers capture encrypted traffic for later analyzation, which contradicts the argument that sensor networks are short lived and thus would require only small key sizes. Hence, we have a critical look at statements regarding security trade offs.

### 2.1 Available Software Implementations

In the following, available software modules for encryption are presented and evaluated regarding their choices of block ciphers and authentication.

#### 2.1.1 Implementations for TinyOS

TinySec [7] for TinyOS 1.x was one of the first implementations of link layer security in wireless sensor networks. It ensures low power requirements, small latency and low bandwidth overhead and supports two different modes of operation: authenticated encryption (TinySec-AE) and authentication only (TinySec-Auth) [7]. CBC mode with an 8 byte Initialization Vector (IV) is used together with a block cipher. Stream ciphers were excluded as they are vulnerable to repeating IVs and an addition of more bytes to a packet would violate their premise of keeping the packet sizes small. To overcome the problem of data leakage in CBC mode using repeated IVs, they propose to pre-encrypt the IV before use. As an appropriate energy-efficient block cipher, they chose Skipjack due to its simple implementation and efficiency. To ensure message integrity CBC-MAC is deployed.

Even from yesterdays perspective, Skipjack was a bad choice as it is a subsequently declassified algorithm, designed behind closed doors at the NSA, vulnerable to several cryptanalysis attacks [8]. *“In a footnote, which was added later to their paper [7], they themselves had to admit that AES would also be a viable choice with similar performance like Skipjack. Even TinySecs successor MiniSec [9] and another implementation called TinyKey [10] are based on Skipjack instead of widely accepted and standardized block ciphers, like AES.”* [11]

### 2.1.2 Implementations for Contiki

ContikiSec [12] provides security for the operating system Contiki with the following modes of operation:

**ContikiSec-Enc** Providing encryption only using AES with an 2 byte IV, ContikiSec-Enc uses Cipher Block Chaining-Ciphertext Stealing (CBC-CS) mode. This mode is utilized to prevent expansion of the produced ciphertext, when messages are not entirely separable into input blocks. This method requires a single network wide key, pre-distributed on all devices or any method providing a session key resulting from other protocols.

**ContikiSec-Auth** Authentication only is provided by producing a MAC using the standardized CMAC algorithm.

**ContikiSec-AE** AE mode provides both authentication and encryption by deploying AES in Offset Codebook Mode (OCB), which generates an authenticated cipher stream.

All choices are well made as the authors chose well-studied algorithms mostly standardized by the NIST. No practical attacks against AES itself is known, also CBC-CS mode for encryption only and CMAC for authentication only are sufficiently secure regarding their requirements. OCB mode for combined authentication and encryption is also the best choice for this purpose, because formal analysis has shown its security and superior performance compared with for example AES-GCM [13]. What should be noted is that OCB mode is patented and its authors allows its use only free of charge when implemented in software licensed under the GNU GPL, non-commercial software, or government software<sup>4</sup>.

## 2.2 Hardware Implementations

Since the NIST standardized the Advanced Encryption Algorithm (AES), manufactures have implemented it in hardware, providing extended instruction sets for software modules [3]. Many improvements were made to use less hardware for AES specific operations. The radio controller Atmel AT86RF231, for instance, provides instructions to setup AES using Electronic Codebook (ECB) mode or Cipher Block Chaining (CBC) mode. Ghaznavi et al., for example, redesigned the MixColumns function to use less hardware. Others optimize regarding parallelizing the algorithm in hardware [15]. Consequently, it is not necessary to implement the block cipher algorithm in software. However, this does not mean that no mistakes can be made. The provided implementations have to be used cautiously to prevent for example key exposure, injection/fault attacks, or side-channel attacks.

In the following, we want to focus on selected challenges and attacks regarding these hardware implementations.

---

<sup>4</sup>Regarding the usage of OCB: <http://www.cs.ucdavis.edu/~rogaway/ocb/offer.htm>

### 2.2.1 About the Utilization of Hardware Instructions

When implementing encryption using microcontroller instruction sets, e.g., instructions provided by Atmel AT86RF231 [3], some precautions have to be taken. ECB mode must not be used because of its insecurity against replay attacks and preservation of message pattern due to its non-chaining structure. As AES in CBC mode itself does not provide any authentication, additional software implementations are needed to provide tamper resistant ciphertexts. CCM mode (counter with CBC-MAC) or AES-GCM can provide authenticated encryption, standardized in RFC 5084 [16].

### 2.2.2 Fault Attacks against AES Hardware Implementations

Considering an attacker who has physical access, fault attacks become possible. Fault attacks influence hardware components by changing their environment to cause faults in their calculations. Common methods of inducing faulty behavior by external stimulation according to *The sorcerer's apprentice guide to fault attacks* [17] are:

**Variation of power supply** Changing the voltage that is supplied to a microcontroller can cause glitches, like misinterpretation or skipping of instructions. This attack is only possible with direct physical access.

**Temperature** Hardware must operate in a specific temperature range, as manufactures do not guarantee correct viability outside these thresholds. Because read and write temperature thresholds on non-volatile memories vary, attacks can be mounted to heat chips until read operations fail, but write operations work according to the specification.

**White light** The photoelectric effect causes microcontrollers to be sensitive regarding white light.

**Laser** With similar effects as white light, lasers can be targeted much more precisely to influence specific areas of a chip.

Hagai Bar-El and Whelan have done experimental evaluations to show that instructions are skipped, when the power supply of their microcontroller was dropped from  $V_{cc}$  to 0 V and later resumed [17]. As shown in Figure 1a, a selective execution of instructions can be achieved. When executed at exact moments, this attack can cause serious security implications for the implemented encryption scheme. The induced instruction glitches can reduce the number of loops in a block-cipher to produce an easy to break single-round variant. Besides reducing loops, data can be manipulated when the voltage supply is controlled precisely (cf. Figure 1b).

Because many smart-cards are vulnerable to these types of attack, much research is being done to find new attacks and countermeasures. Roche et al. [18] attack AES implementations based on a new fault model. Fault attacks combined with

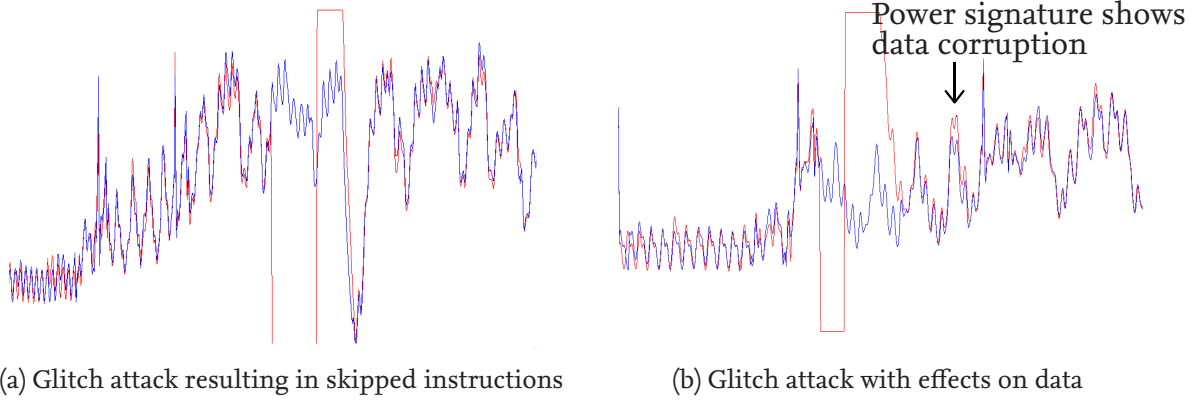


Figure 1: Experimental fault attacks by dropping voltage supply, blue curve depicts normal execution, red curve depicts execution with  $V_{cc}$  glitch (Source: [17])

physically leaked side-channel information, e.g., electromagnetic radiations, are used to recover the key. The attack will induce faults in the last round of AES key-scheduling. Their simplest attack using fault injection on pre-computed key schedules will change the last two round keys in the following way:

$$\begin{aligned}\tilde{K}^9 &= K^9 \oplus E^9 \\ \tilde{K}^{10} &= K^{10} \oplus E^{10}\end{aligned}$$

$K$  denotes the original key,  $E$  is the value of an occurring error, and  $\tilde{K}$  the resulting faulty key. Their attack works by letting the device encrypt  $N$  messages  $P^1, \dots, P^N$  two times. The second time they inject an error while the key schedule is computed, which produces  $N$  faulty ciphertexts  $\tilde{C}^1, \dots, \tilde{C}^N$ , instead of  $C^1, \dots, C^N$ . By using simple statistic analysis of the ciphertext pairs  $(C^1, \tilde{C}^1), \dots, (C^N, \tilde{C}^N)$ , they try to find the specific error and subkey bytes. This basic attack is developed further and combined with side-channel analysis to lower the requirements regarding the attack surface. Their more sophisticated attacks also works against non pre-computed key schedules [18].

### 2.2.3 Countermeasures

Countermeasures against fault attacks can take place on hardware and/or software level. Sensors can be implemented to actively detect malicious induced effects and classify them as attacks. This includes light sensor, frequency detectors, and supply voltage detectors. Hardware redundancy, based on duplicated circuits and comparison of results, can make attacks much harder. These are one of the most implemented and effective countermeasures when done the right way [17, 18]. All hardware countermeasures could also be implemented in software, but would double the computing time in most cases, as parallelization is not as easy as in hardware implementations [17].

### 3 Key Pre-distribution

To allow the use of previously discussed block ciphers, encryption keys need to be pre-distributed among the devices that want to communicate with each other. Algorithms for deploying these pre-shared keys need to be implemented, which must support different requirements based on the setup of its WSN. According to Xiao et al., key distribution schemes should mainly provide authenticity to provide a way to identify nodes, scalability, and flexibility to allow adding of new nodes at every time [19]. We omit confidentiality and integrity because they were discussed in Section 2. Authenticity, scalability, and flexibility lead to several basic attack scenarios key distribution schemes have to protect against [19]. An attacker could, for example, compromise certain parts of the network and replicate those nodes to take over the entire network. Key distribution schemes should resist against those attacks by preventing replication. When bad behavior of single nodes is detected, those nodes should be excluded, by revoking them using the key management scheme. Furthermore, it should be possible to deploy new nodes into existing key management infrastructures and to retain secrecy of benign nodes, when single nodes are taken over (Resilience).

Several key pre-distribution schemes were proposed to fulfill these requirements, which are based on differing assumptions regarding the sensor network. We will present fundamental problems, the most common schemes, and show possible weaknesses.

#### 3.1 Fundamentals

The simplest key pre-distribution form is the pre-deployment of a single network wide key on all nodes in the network [19]. Besides small storage requirements, this scheme apparently does not prevent any of the three discussed attacks as a single compromised node results in a compromised network.

In a pairwise key distribution scheme [19], a node is deployed with  $n - 1$  keys to communicate with every other node. This offers pairwise encryption and authentication as every key belongs uniquely to two nodes. Pairwise pre-distribution also complies with Resilience, because a take over of nodes does not reveal secret information about other nodes due to differing keys. This scheme only makes sense in small sensor networks as each new node requires a new key to be stored by all other nodes, resulting in a high memory usage. A superior sink node with a larger memory and higher transmission rates can be used as Key Distribution Center (KDC). A KDC can be queried for session keys used one time for a specific communication between nodes.

#### 3.2 Random Key Pre-distribution

Eschenauer and Gligor presented a random key pre-distribution protocol to overcome the problem of storing  $N - 1$  keys on every participant. The distribution

scheme works as follows [19, 20]:

1. A large pool of  $|S|$  key-identifier pairs is generated.
2.  $K$  key-identifier pairs are randomly taken out of this pool, where  $K \ll N$  and  $N$  is the number of nodes in the WSN and saved as a keyring. Every node gets its own keyring assigned.
3. Based on the identifiers, two nodes can agree upon one shared key, while the probability is sufficiently high that they both share at least one matching key. If no shared key is available the nodes perform path-key discovery. This means a third party node will be searched that shares keys of both communicating nodes.

The scheme successfully optimizes regarding storage space and is also scalable, as the size of the underlying key pool and the number of chosen keys can be adaptively chosen according to the size of the network. The downsides of this schemes are that it provides no form of multicasting messages because between the sender and every recipient a different key has to be negotiated. There is also no method that would provide strong revocation of compromised nodes and freshness of keys.

### 3.3 Schemes Based on Random Key Pre-distribution

Several more complex protocols were proposed that are based on the random pre-distribution model [19, 20]. Du et al. published a model extending the concept using Blom's key matrix instead of individual keys [21]. A matrix  $M_{n \times n}$  is generated, where nodes can select keys from by looking at specific cells. If node  $i$  and  $j$  want to communicate,  $i$  looks at  $M_{ij}$  and  $j$  looks at  $M_{ji}$  to select the same shared key. In this scheme  $n$  matrices are generated and on each node only randomly selected matrices are stored, like in the basic random key pre-distribution scheme. Their scheme offers more robustness and security, because more nodes need to be compromised compared to the basic random pre-distribution scheme. This advantages comes to the cost of more computational overhead and memory consumption.

Even more complex protocols like LEAP and SHELL were published [19, 20]. They introduce different key types, like individual, group, cluster, and pairwise keys, where each key type is distributed and used differently. They fulfill most requirements and are flexibel, but require complex implementations. Consequently much memory space is needed for their implementation and keys. Besides their memory usage, their implementation will likely contain bugs due to their high complexity.

### 3.4 Hierarchical Group Key Management Schemes

Panja et al. were the first who proposed a hierarchical group key management [22]. A hierarchical management scheme consists of groups sharing the same group



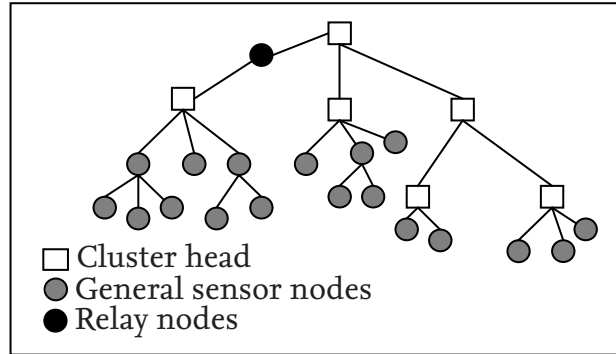


Figure 2: Hierarchical structure proposed by Panja et al. (Derived from [22])

key, ordered hierarchically based on their computing power or other factors like access rights. The scheme consists of cluster heads, having many general sensor nodes as children, arranged in a tree structure (cf. Figure 2). A partial key for each sensor node is computed as follows [22]:

1. All leaf nodes generate their keys randomly.
2. The parents of these leaf nodes compute their keys based on the child nodes and the given function  $f(k_1, k_2) = \alpha^{k_1 \oplus k_2} \bmod p$ , where  $k_1, k_2$  are keys and  $p$  is prime with  $k < p$  for both  $k$ .
  - a) When the corresponding tree is binary, a partial key is calculated by the following equation, where the index of  $K$  defines the position in the tree by depth and row.

$$K_{l,v} = f(K_{l+1,2v}, K_{l+1,2v+1})$$

- b) In situations where the tree is not binary a slightly modification is added, where  $m = n_{\text{siblings}} - 2$  and  $n_{\text{siblings}}$  defines the number of siblings in this row.

$$K_{l,v} = f(K_{l+1,2v+m}, K_{l+1,2v+1+m})$$

Group key computation is done by utilizing multi-party Diffie-Hellman and TGDH. A cluster head node can initiate a group key computation, where every all partial keys are gathered bottom-up and collected by the cluster head.

1. All leaf nodes generate a partial key  $g^S$ , where  $\langle g \rangle = \mathbb{Z}_p^*$ ,  $p$  prime, and  $S$  is a randomly generated number. These partial keys are send to their parent nodes.
2. Nodes, which are not cluster heads and not leafs, receive partial keys from all their  $n$  children and sums them up together with their own partial key  $S$  to generate an intermediate key  $IK$ .

$$IK = g^{S_1} + \dots + g^{S_n} + g^S = g^{S_1 \dots S_n \cdot S}$$

3. Finally, the cluster head receives all intermediate keys and sums them up again together with its own partial key  $S$ , resulting in the group key  $K$ .
4. This group key is encrypted with a one-time symmetric key, only available in deployment phase, and broadcasted to all nodes of this group. Afterwards the symmetric key is discarded and the group key is used.

A new group key is calculated, when a cluster head broadcasts an encrypted authenticated message containing the instruction to refresh the group key. This could also contain the instruction to remove a compromised node, which implements revocation.

In addition to these algorithms the authors differentiate between intra-cluster group keys and inter-cluster group keys. When the cluster head broadcasts the key the first time, blind factors are added, which are unique numbers assigned to individual nodes [22]. When distributing the group key, each node can recognize its own blind factor and replace it with their own real factor. This prevents attackers from eavesdropping the group key in deployment phase.

This scheme is flexible, because it is possible to refresh group keys and revoke nodes. However, revocation only works to a certain degree, because cluster heads can only revoke leafs or complete branches. The downside is that it assumes that a cluster head does not get compromised, which makes it a worthy target. It is also quite elegant and simpler to implement than over-engineered schemes like SHELL [19]. Mortazavi et al. [23] improved upon such schemes to reduce the rekeying overhead.

## 4 ECC Based Key Management

Key pre-distribution schemes do not require large amount of processing power or RAM, but are much less flexible than schemes based on public key cryptography. In the past, most sensor networks were deployed with pre-distributed keys, lacking important requirements like fast revocation of keys and secure integration of newly deployed nodes. RSA based public key algorithms were omitted due to their seemingly large power consumption while performing underlying multiplication methods. New implementations of Elliptic Curve Cryptography (ECC) were done to provide usable public key cryptography with low requirements. ECC is based upon the Elliptic Curve Discrete Logarithm Problem (ECDLP), which states that it is hard to find a discrete logarithm of an elliptic curve element with a publicly known base point. According to *Understanding cryptography: a textbook for students and practitioners* [24], the basic definitions are as follows.

**Discrete logarithm problem** The discrete logarithm problem is to compute  $l$  such that  $\beta = \alpha^l$ .  $\alpha$  is a generator for a multiplicative group  $\langle \alpha \rangle = \mathbb{G}$  and  $\beta \in \mathbb{G}$ .

**Elliptic Curve Discrete Logarithm Problem** Compute  $l \in [0, n - 1]$  such that  $Q = lP$ . Let  $P$  be a point of order  $n$  and  $Q \in \langle P \rangle$

Gura et al. have done implementations of public key cryptography for RSA and ECC for sensor nodes based on ATmega128 and CC1010 microcontroller [1]. Their implementations are highly optimized regarding memory access, as this is the most limiting factor on these small processors. The key size for ECC keys can be much lower than RSA based keys, e.g., a 1024 bit RSA key pair provides as much security as an 160 bit ECC key pair [2]. This is due to the fact that RSA trapdoor function is based on the hardness of integer factorization and ECC is based on its ECDLP. While sub-exponential algorithms exist to solve integer factorization, known algorithms to solve ECDLP scale exponentially [1]. Their ECC point-multiplication using 160 bit, implementing SECGs standardized elliptic curves in conjunction with their optimized multiplication, is by a factor of two faster than RSA-1024. As an example by absolute numbers, RSA-1024 private key operation needed 10.99 s while a 512 bit montgomery exponentiation for ECC needed 5.37 s. ECC-240 outperforms RSA-2048 by a much more higher degree as shown in their paper [1].

Their evaluation has shown that it is indeed possible to use public key algorithms without hardware acceleration on microcontrollers intended for sensor nodes. As recommended key sizes by NIST [2] for 2011-2030 are RSA-2048 and ECC-240, ECC should be favored due to their small key sizes and low runtime of its algorithms.

Szczechowiak et al. [25] improved the work of Gura et al. Besides implementing a hybrid multiplication algorithm, they chose Koblitz curves as these are less expensive due to their appearance without point doubling. Their implementation is done for TinyOS and written in C/nesC with some optimizations using assembly, generated by utility programs for better portability. It is based on MIRACL<sup>5</sup>, a library that provides all necessary primitives for ECC and was optimized for embedded platforms. The ECC implementation is based on NIST recommended k163 Koblitz curves over  $GF(2^{162})$  and  $y^2 = x^3 - 3x + 157$  with  $p = 2^{160} - 2^{112} + 2^{64} + 1$  over  $GF(p)$ . To speed up the point multiplication in elliptic curves, which are defined as  $sP$  with  $s$  as an integer and  $P$  a point of the curve, additional memory is utilized. Their implementation was evaluated on two sensor nodes, the MICA2 (ATmega128L) and Tmote Sky (MSP-430). Computations for the binary field were between 1.04 s and 2.16 s and 0.72 s to 1.27 s in the prime field. Thus, NanoECC demonstrates a efficient implementation usable in practice.

## 4.1 Routing-Driven Key Management

Du et al. [26] proposed a routing-driven key management using ECC. They state that sensor nodes often only communicate with their communication-neighbors to route data to the sink node. A node  $v$  is a communication-neighbor ( $c$ -neighbor) of a node  $u$  if  $v$  is on a route from  $u$  to the sink node (cf. Figure 3). Because of this many-to-one communication pattern, nodes do not need to obtain shared keys for all other nodes in the network. Thus, while in key establishment phase, shared keys only need to be generated for every node and its neighbors. They

---

<sup>5</sup>MIRACL can be found on <http://www.shamus.ie/index.php?page=elliptic-curves>

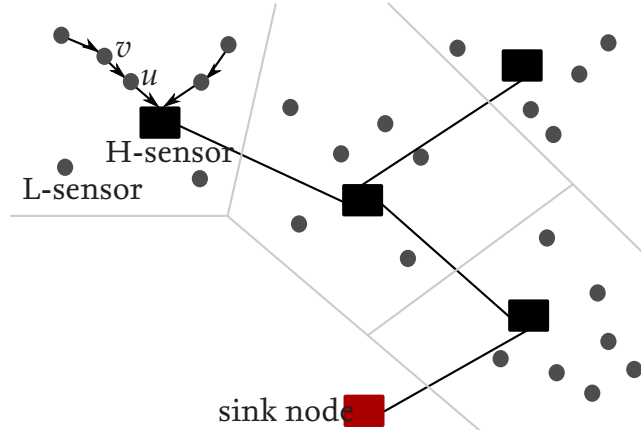


Figure 3: Routing-driven key management: Clusters with many low-end sensors (L-sensors), controlled by single high-end sensors (H-sensors).

differentiate between high-end (H-sensors) and low-end sensors (L-sensors) with different hardware configurations. A small number of H-sensors exist in a network, which are equipped with tamper-resistant hardware, preventing an adversary to extract keys, data, or stored code. Whereas L-sensors are low cost devices without such protection. Both sensor types are location aware. As depicted in Figure 3, H-sensors form a basis station for clustering many L-sensors together, and are used as gateways to route traffic to the sink node of the whole WSN. We will introduce the centralized key establishment described in the paper and omit the distributed scheme.

#### 4.1.1 Centralized Key Establishment

1. A server generates public/private key pairs  $(K_u^R, K_u^U)$ . Every L-sensor is preloaded with its private key, whereas every H-sensor also contains the public keys of all L-sensors.
2. Each H-sensor stores a common ECC public/private key pair  $(K_H^U, K_H^R)$  for authenticating broadcasts using Elliptic Curve Digital Signature Algorithm (ECDSA). The public part is also stored on every L-sensor to verify these broadcasts. Additionally, H-sensors are deployed with a symmetric key  $K_H$  used for communicating among H-sensors.
3. Inside a cluster, each L-sensor sends key-request messages to their H-sensor, which includes their unique ID and location. The H-sensor generates shared-keys  $K_{u,v}$  for all L-sensors in its cluster for sensor  $u$  and its  $c$ -neighbor  $v$ , using its public key  $K_u^R$ . This shared-keys are encrypted and send by unicast to  $u$ .
4.  $u$  decrypts the message and stores the shared key for communication between  $u$  and  $v$ .

#### 4.1.2 Remarks

Revocation is done by H-sensors, which should also detect malicious nodes. A revocation message is signed using ECDSA and contains the ID of the compromised node. This message is sent to all L-sensors of this cluster, which can check whether they communicate with a malicious node or not.

The scheme implements revocation and resilience in a feasible way. Their performance evaluation also looks promising considering the amount of L-sensors involved in the setup [26]. This key management requires a well thought out network structure, location aware nodes, and H-nodes with tamper-resistant hardware, which are quite high requirements.

## 5 Conclusion

In this seminar paper, we presented selected security primitives for wireless sensor networks. Software implementations of encryption modules and fault attacks against hardware implementations were discussed. Key pre-distribution schemes were depicted with their advantages and disadvantages. As a promising public key technique, ECC could solve some problems pre-distribution schemes could not solve perfectly and is thus viewed as WSNs future key distribution implementation.

Several hard issues remain unsolved. The problem of adding nodes to an existing network is difficult, even with public key cryptography, as every node in this distributed network has to be informed about newly added and accepted public keys. This can partially be solved by defining the sink node as a trusted third party, as done in Identity Based Cryptography. Revocation is even harder, because of several pitfalls. Firstly, a node needs to be identified as a malicious node, possibly overtaken by an adversary. Only a few papers deal with this issue, as it is impossible to detect malicious nodes, when they behave correctly and just capture traffic. Secondly, the revocation packet needs to be routed to every node quickly, although malicious nodes could work as black holes, preventing the distribution. These problems exist due to the distributed nature of wireless sensor networks. Furthermore, physical attacks against hardware implementations, as discussed in section 2.2, need to be taken more seriously. Many WSN are deployed in areas without any additional surveillance, which makes fault attacks possible and leave them undetected.

## References

- [1] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs". In: *Lecture Notes in Computer Science* 3156 (2004), pp. 119–132.
- [2] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. "Recommendation for key management". In: *NIST Special Publication 800-57 Part 1 Rev. 3* (May 2011).
- [3] *AT86RF231/ZU/ZF datasheet*. Atmel Corporation.

- [4] K. Fall. "A delay-tolerant network architecture for challenged internets". In: *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM. 2003, pp. 27–34.
- [5] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. "Tinyos: An operating system for sensor networks". In: *Ambient intelligence* 35 (2005).
- [6] A. Dunkels, B. Gronvall, and T. Voigt. "Contiki-a lightweight and flexible operating system for tiny networked sensors". In: *29th Annual IEEE International Conference on Local Computer Networks*. IEEE. 2004, pp. 455–462.
- [7] Chris Karlof, Naveen Sastry, and David Wagner. "TinySec: a link layer security architecture for wireless sensor networks". In: *Proceedings of the 2nd international conference on Embedded networked sensor systems, SenSys '04*. Baltimore, MD, USA: ACM, 2004, pp. 162–175.
- [8] E. Biham, A. Biryukov, O. Dunkelman, E. Richardson, and A. Shamir. "Initial observations on the skipjack encryption algorithm". In: *SAC'98*. 1998.
- [9] M. Luk, G. Mezzour, A. Perrig, and V. Gligor. "MiniSec: a secure sensor network communication architecture". In: *6th International Symposium on Information Processing in Sensor Networks, IPSN 2007*. IEEE. 2007, pp. 479–488.
- [10] R. Doriguzzi Corin, G. Russello, and E. Salvadori. "TinyKey: A light-weight architecture for Wireless Sensor Networks securing real-world applications". In: *Eighth International Conference on Wireless On-Demand Network Systems and Services, WONS*. 2011, pp. 68–75.
- [11] Felix Büsching, Andreas Figur, Dominik Schürmann, and Lars Wolf. "Utilizing Hardware AES Encryption for WSNs". In: *Proceedings of the 10th European Conference on Wireless Sensor Networks, EWSN 2013*. 2013, pp. 33–36.
- [12] L. Casado and P. Tsigas. "Contikisec: A secure network layer for wireless sensor networks under the contiki operating system". In: *Identity and Privacy in the Internet Age (2009)*, pp. 133–147.
- [13] T. Krovetz and P. Rogaway. "The software performance of authenticated-encryption modes". In: *Fast Software Encryption*. Springer. 2011, pp. 306–327.
- [14] S. Ghaznavi, C. Gebotys, and R. Elbaz. "Efficient technique for the FPGA implementation of the AES MixColumns transformation". In: *International Conference on Reconfigurable Computing and FPGAs, ReConFig '09*. 2009, pp. 219–224.
- [15] Chih-Peng Fan and Jun-Kui Hwang. "Implementations of high throughput sequential and fully pipelined AES processors on FPGA". In: *International Symposium on Intelligent Signal Processing and Communication Systems, ISPACS 2007*. 2007, pp. 353–356.
- [16] R. Housley. *Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)*. RFC 5084 (Proposed Standard). Internet Engineering Task Force, Nov. 2007.
- [17] Hamid Choukri David Naccache Michael Tunstall Hagai Bar-El and Claire Whelan. *The sorcerer's apprentice guide to fault attacks*. Cryptology ePrint Archive, Report 2004/100. 2004.
- [18] Thomas Roche, Victor Lomné, and Karim Khalfallah. "Combined fault and side-channel attack on protected implementations of AES". In: *Smart Card Research and Advanced Applications*. Ed. by Emmanuel Prouff. Vol. 7079. Lecture Notes in Computer Science. Springer, 2011, pp. 65–83.
- [19] Yang Xiao, Venkata Krishna Rayi, Bo Sun, Xiaojiang Du, Fei Hu, and Michael Galloway. "A survey of key management schemes in wireless sensor networks." In: *Computer Communications* 30.11-12 (Nov. 2, 2007), pp. 2314–2341.

- [20] J.C. Lee, V.C.M. Leung, K.H. Wong, Jiannong Cao, and H.C.B. Chan. “Key management issues in wireless sensor networks: current proposals and future developments”. In: *Wireless Communications, IEEE* 14.5 (2007), pp. 76–84.
- [21] Wenliang Du, Jing Deng, Yunghsiang S. Han, Pramod K. Varshney, Jonathan Katz, and Aram Khalili. “A pairwise key predistribution scheme for wireless sensor networks”. In: *ACM Trans. Inf. Syst. Secur.* 8.2 (May 2005), pp. 228–258.
- [22] B. Panja, S.K. Madria, and B. Bhargava. “Energy and communication efficient group key management protocol for hierarchical sensor networks”. In: *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*. Vol. 1. 2006, 8 pp.
- [23] S.A. Mortazavi, A.N. Pour, and T. Kato. “An efficient distributed group key management using hierarchical approach with Diffie-Hellman and Symmetric Algorithm: DHSA”. In: *International Symposium on Computer Networks and Distributed Systems, CNDIS*. 2011, pp. 49–54.
- [24] C. Paar and J. Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer, 2010.
- [25] Piotr Szczechowiak, Leonardo B. Oliveira, Michael Scott, Martin Collier, and Ricardo Dahab. “NanoECC: Testing the limits of Elliptic Curve Cryptography in sensor networks”. In: *Wireless Sensor Networks*. Ed. by Roberto Verdone. Vol. 4913. Lecture Notes in Computer Science. Springer, 2008, pp. 305–320.
- [26] Xiaojiang Du, Mohsen Guizani, Yang Xiao, and Hsiao-Hwa Chen. “A routing-driven Elliptic Curve Cryptography based key management scheme for Heterogeneous Sensor Networks.” In: *IEEE Transactions on Wireless Communications* 8.3 (Nov. 15, 2009), pp. 1223–1229.