J Netw Syst Manage (2012) 20:417–452
DOI 10.1007/s10922-011-9219-8

Configuration Tool for a Wireless Sensor Network Integrated Security Framework

Victor Cionca · Thomas Newe · Vasile Teodor Dădârlat

Received: 28 July 2010 / Revised: 9 October 2011 / Accepted: 28 October 2011 /
Published online: 11 November 2011

Abstract Wireless Sensor Networks (WSNs) will benefit from security frameworks that provide easy to use security while hiding inherent complexity. In comparison to traditional networks, the hardware constraints of wireless sensors demand that network security be configured based on application parameters, in order to provide the desired level of security while efficiently using available resources. Such a security framework is discussed in this paper, with special focus on security configuration, where a configuration methodology is presented in detail. The methodology is implemented as a Java tool with SQLite database backing, and to prove its validity, a sample application is presented and tested. Also, experimental proofs are provided to support the methodology, and the general necessity of configuring security for WSN applications.

Keywords Wireless sensor networks · Security framework · Security configuration

1 Introduction

There are several sides to communication security: safety, usability and configurability. On a basic level, services must be provided to protect communication data from alteration, theft or forgery. These services need to be integrated transparently

V. Cionca (✉) · T. Newe
Department of Electronic and Computer Engineering, University of Limerick, Limerick, Ireland
e-mail: victor.cionca@ul.ie

T. Newe
e-mail: thomas.newe@ul.ie

V. T. Dădârlat
Department of Computer Science, Technical University of Cluj-Napoca, Cluj-Napoca, Romania
e-mail: vasile.dadarlat@cs.utcluj.ro

in the communication stack, to ensure they can be easily used. As applications become more varied, the demands from those security services will differ from scenario to scenario, thus imposing the need to configure security. This is precisely the case for wireless sensor networks (WSNs): there are many possible applications and hardware constraints that require efficient resource usage to prolong network lifetime. This paper presents a user-oriented methodology to determine, for a given application, the security protocols that will provide the required security levels while maintaining efficient resource usage, within the application demands. Several experimental proofs are given to support the necessity of WSN security configuration, and a scenario is used to explain the functionality of the methodology. The scenario application is implemented and tested and the performance results are presented.

The management and configuration of security in sensor networks has not received its due attention in research. The consensus is that a static architecture is used (as proposed in [1]), which consists of a fixed key management system together with authentication, integrity, replay and confidentiality services. However, a static system is not really suitable for sensor networks; if it were used for both a military and a home-oriented application, it would be too weak for the former or too strong for the latter, so it would either lead to insecurity or a waste of power and resources.

Prasad and Alam present in [2] a descriptive approach to select security primitives based on security level, which in turn is determined by the application type. A more in-depth descriptive approach is provided by Law and Havinga in [3]: they analyse application requirements from the point of view of security primitives (confidentiality, authentication, integrity and availability) and recommend protocols or solutions for every network layer, as well as cross-layer solutions. Ransom et al in [4] generate security models for the network defined by the user's parametrised description of the application, and present each model with a list of possible attacks, allowing the user to choose what he thinks is the safest model. Finally, Peter et al present configKit [5], designed to help application developers find the configuration of security modules which resolves the hardware, energy and security requirements of the designed application. A simpler but more practical approach to security configuration is to use a configurable cipher, like RC5, and employ different numbers of rounds or key sizes, yielding different levels of security. This approach is presented in [6].

The papers enumerated in the previous paragraph focus on configuring the security of the sensor network application at development or deployment time. There are approaches that focus on run-time configuration. de Oliveira et al. [7] propose to reduce resource usage by employing security services of different levels in conjunction with intrusion detection. Instead of maintaining a constant high security level throughout the network lifetime, it starts relatively low and is increased if intrusions are detected. An unbalanced approach is used to further alleviate the computational burden placed on individual nodes: the base station centralises intrusion detection reports from the nodes and decides when the security level of the network must be changed. Another approach is by Krontiris et al. [8] where nodes can secure messages with replay protection or authentication or

confidentiality, the security level being indicated by three bits in the message header.

The methodology put forward in this paper is closest to Peter et al.'s approach, but it is more complete as it also handles the usability of the system. It is the authors' opinion that security configuration should be an end-user task, as the security requirements for the application can be changed by making apparently unimportant modifications: if the sensor sampling rate is increased, it might be too high for some cryptographic algorithms; adding or moving nodes might alter network size or topology [9], requiring a change in the key management system. These small changes that are bound to happen in a WSN lifetime require major changes in the application code. It would be impractical to have the user go back to the developer or deployer for each minor change; it could also be error-prone, as repeated changes to an application often introduce errors.

What is required, and what the authors propose, is a complete decoupling of application and security code, which would communicate using a set of generic interfaces; security modules can be developed separately and installed on the application code after application development. Going farther, security protocols can be expressed as sets of network and application parameters; using functions and inference relations, the parameters are reduced to a set that is comprehensible to the user. A security framework with these characteristics allows an end-user to select the most efficient security protocols for an application and install them on the application code. It is also important to state that the parameter values must be expressed in a way that is understood by the user; the authors believe that an issue of Peter et al's approach is the usage of subjective values with no real connection to the measured item, for example: numerical scaling of protocol strength, where it would be difficult for the user to determine whether the protocol is a "seven" or a "six". This is an area that is addressed in the presented methodology.

The paper opens in Sect. 2 by presenting the bigger picture, a security framework that decouples application and security code using a modular architecture, and allows configuration of installed security services, to provide easy to use security for WSN applications. Sections 3 and 4 introduce the security services and protocols considered, and, respectively, the parameters used in the configuration methodology. In Sect. 5 the configuration methodology is explained, showing what protocols, parameters and algorithms are used. A scenario is provided to show how the methodology would be used in an application. A Java implementation of the methodology, and the implementation of the scenario application are presented in Sect. 6 Experimental proofs for security configuration and measurements of the scenario application are presented in Sects. 7 and 8 contains a discussion on security configuration, the presented scenario and the methodology. Finally, the paper concludes in Sect. 9.

2 A Security Framework for Sensor Networks

Many of the security issues of wireless sensor networks have already been addressed and have valid, tested and accepted solutions. There are services offering data

confidentiality and authentication, like TinySec [10], MiniSec [11] or FlexiSec [6]; there is a plethora of key management solutions [12, 13]. Already dedicated chips are appearing that perform some of the cryptographic operations in hardware, thus alleviating the burden of the micro-controller: the CC2420 chip from Chipcon [14] implements the AES (Advanced Encryption Standard) cipher (the Rijndael cipher), and a paper by Hu et al. [15] presents the results for using a *Trusted Platform Module* connected to a WSN node. The building blocks are there to ensure the security of WSN applications, but, from the authors' review of the research, there is no account of fully secured deployments.

Putting together individual security solutions into a robust security architecture is no simple feat. The security must be transparent and must make efficient use of the available resources. But most of all, as the rest of this paper explains, the security should be configurable to meet the specifics of the application. Keeping in mind these requirements, a *security framework* was developed. Its primary goal is to seamlessly install on the application code the most efficient security protocols.

This is not a tool or a collection of software modules that "can" be used by the developer; its a complete software suite that follows the entire life cycle of WSN applications, from development to usage. The functionality and components of the framework are designed taking into account the WSN actors, application developer and application user, and the roles they play. Their tasks are clearly defined and separated based on their knowledge: the developer knows what data is used and exchanged, at what layer in the network and with what purpose; the user decides how the application is used. From a security point of view, the developer can indicate where security might be needed, but only the user can determine which security protocols should be installed. Therefore the framework components must allow delayed installation of security protocols on application code; this requires *decoupling of application and security code*, a way to *identify where security services need to be installed* in the application, and a way to *select individual security services* based on application parameters. The key elements of the framework shown in Fig. 1 are:

- a code repository for security service modules (security protocols, cryptographic algorithms, etc.),

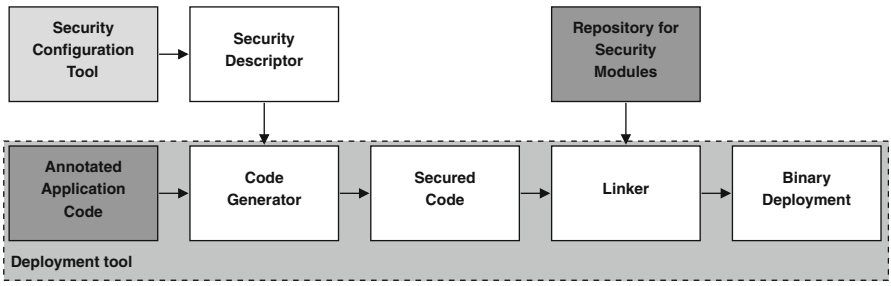


Fig. 1 Security framework for sensor networks: components that require user interaction in *light gray*; components provided by the developer in *dark gray*

- the security configuration tool, presented in this paper,
- a modular security architecture which allows security services to be installed onto application code and integrated with the network stack (although not shown on the diagram, this is the main part of the framework which glues together the individual components),
- a code generator and binary deployment tool.

The modular security architecture provides the required code decoupling, while the security configuration tool allows security protocol selection.

Communication security services are applied by the framework to messages in data streams. The data streams are identified by the following:

- protocol number,
- network layer of origin,
- destination address.

The protocol number differentiates between application on the same network layer that might have different requirements. The network layer where a message originates can be an indication of what keys need to be used. Considering a network stack of three layers, MAC (media access control), routing and application, MAC and routing messages will use link security (the message can and will be decrypted by the next hop) while application messages can also use end-to-end security (the message can only be decrypted by the destination, the last node in the routing chain).

Messages can be secured differently based on their destination address, more precisely whether they are unicast or broadcast, because the same security protocols can't be used for both types of traffic, for example: in unicast authentication, the symmetric key shared by the source and destination can authenticate the traffic, whereas in broadcast an asymmetric scheme would be required, otherwise anyone could impersonate the sender.

All these considerations are taken into account to identify *what* data needs to be secured and *where and how* the security code will be connected to the application code. The framework provides the following steps:

1. For the developer—annotate the application code with security indications.
2. For the user—configure application; internally, this has the effect of selecting security protocols.
3. Internally—install security protocols on application, using the modular security architecture.
4. Internally—compile secured application.
5. Internally—install keys in the application binary.

Finally, the binaries can be installed on motes and deployed.

The framework can be applied on any existing WSN application implemented in TinyOS, provided that it adheres to TinyOS interfaces. The first step has the developer identifying *what* data needs to be secured. The developer must write a configuration file that contains:

- protocol numbers of the data streams that must be secured, and optionally, the type of the data stream (*e.g* broadcast stream);
- the network components that define the layers of the network stack, optionally specifying if default layer security should be used.

When the user performs the security configuration using the methodology and tool presented in this paper, a *security descriptor* is created, which contains the best choices of security services and protocols for the application. The main part of the framework, the modular architecture, uses policies to decide what data must be secured and how. The policies are generated by combining the information provided by the developer (*what*) with the security descriptor (*how*). When the application is deployed, the modular security architecture intercepts messages in transit between network layers. The security policy that describes the data stream is fetched, and the message data is processed by the services and protocols specified by the user through the configuration tool.

This paper focuses on the security configuration part of the framework. The configuration methodology is presented, motivated and validated against a scenario.

3 Security Protocols

Sensor networks suffer from all the attacks that traditional networks do: impersonation, denial of service, data theft, etc. In addition there are some attacks and threats that are made possible by WSN characteristics: node corruption (installing new software on unsupervised nodes) [16], influencing of sensors (e.g holding a flame near a temperature sensor), network holes (worm holes and black holes), replication, [17] etc. By using traditional security mechanisms, authentication and integrity, confidentiality and key management, most of these attacks can be prevented; the only ones that are left open are worm holes, node corruption and sensor influence [17]. Most of the protocols that are considered here are either classic protocols or ones that are popular in the WSN field; although WSN security literature is rich in protocols, only a few of them are actually implemented or usable. It should be noted that the list given here is not exclusive or comprehensive; it is only used as an example for the presented methodology. Any security protocol for sensor networks can be integrated into the tool by expressing it in terms of the security parameters discussed in the next session.

Data authentication, integrity and confidentiality are usually handled all together by one protocol. The most well-known are:

- TinySec [10], for link-layer security (although stand-alone usage is possible); it achieves increased efficiency by working closely with the radio, which however binds it to certain hardware platforms;
- MiniSec [11], again for link-layer security, is usable with the newer platforms; one of its strong points is the handling of replay prevention for broadcast messages; however, this is not implemented in the released code;
- SPINS [18] is one of the oldest WSN security protocols; it introduced μTESLA, a symmetric broadcast authentication protocol;

- CC2420 hardware security [14] which provides access to high-speed AES (Advanced Encryption Standard, the Rijndael cipher) in counter or counter with CBC-MAC (cipher-block chaining message authentication code) mode, and CBC-MAC authentication; it is provided by the Chipcon CC2420 radio chip.

The management of keys is still one of the most difficult problems of security and which can be said contributed to the transition from symmetric to asymmetric cryptography. The computation and storage constraints of wireless sensing nodes make the use of asymmetric cryptography quite prohibitive. There are reports of successful implementations (mostly using elliptic curve cryptography) [19, 20], but the general consensus is that it's better to use symmetric techniques. Therefore, the problem of key management in WSNs is just as it was in the 1970s for traditional networks. There are two steps that must be completed: predistribution of keys to nodes, following a specific scheme, and establishment or renewal of keys, when their usage is no longer safe.

Key predistribution consists of installing keys on nodes in a specific pattern; traditional schemes are *master*, where all the nodes in the network share a single key, *pairwise*, where every possible pair of nodes shares a key, or *server-based*, where every node shares a key with a server, which acts as a trusted third party. More complex mechanisms have been designed to overcome WSN constraints: probabilistic [21], random [22] or polynomial [23], just to name a few. All start with a large set of keys from which subsets are installed on nodes in a way that ensures a certain probability that any two nodes are securely connected (share a key).

In sensor networks key predistribution is sometimes regarded as more important than establishment, because the latter is assumed to be used only for renewing keys. The lifetime of a key is conditioned by the amount of key freshness data associated with the key, in most cases the initialisation vector (IV) in a block cipher. A common value for key freshness in WSN is of 2^{32} (four byte IV), which considering the average battery lifetime of a sensor node, should be enough to avoid renewing keys. In the results section later in the paper it is proved that it's more efficient to use four bytes of initialisation data for key safety (the counter in counter mode AES), than to reduce that to two and perform more frequent key establishment. However key renewal is not the only usage for key establishment protocols: if new nodes enter the network they need to be able to communicate with other nodes. Network configuration might change, nodes may be reset, keys might be revoked, so it's important to be able to establish new ones. Following an in-depth study of classical key-establishment protocols and their vulnerabilities, the conclusion was reached that it is safer to use well-studied protocols than new ones which are not properly analysed. The following protocols are considered, broken down into five categories:

- centralised key transport with symmetric ciphers: centralised Needham—Schroeder (N-H KDS), Otway Rees (O-R), Wide Mouthed Frog (WMF);
- distributed key transport with symmetric ciphers: Shamir protocol (Shamir);
- key transport with asymmetric ciphers: public Needham-Schroeder (N-H PK), X509;

- key agreement with symmetric ciphers: ZigBee Symmetric Key Key Establishment (SKKE) [24];
- key agreement with asymmetric ciphers: Diffie-Hellman (DH), Matsumoto-Takashima-Imai (MTI) [25], Menezes Qu Vanstone (MQV) [25], Station-to-Station (STS).

If not explicitly cited, a description of the protocol can be found in Clark and Jacob's survey of authentication protocols [26].

4 Application and Security Parameters

The specifics of wireless sensor networks applications are determined by a set of parameters, like number of nodes or network topology. They are parameters whose values *cannot be decided until* the application is deployed and used, and therefore only the user can have full control over them. In a mature WSN life-cycle such details should not be the developer's concern; if the application's behaviour depends on the parameter, the application should be configurable and tools provided to this end. In a similar way, security services and protocols share a set of parameters which can be used to select a security suite that matches certain criteria.

This section presents the application and security parameters that were identified as part of the security configuration methodology. The next section explains the relations that can be established between the two sets (application and security) in order to allow user-based security configuration of the WSN application.

4.1 Application Parameters

When creating configuration tools for the user, it is important to make sure that the parameters selected can be understood by the user and that their values are objective. After an analysis of WSN applications, the following parameters were selected: *application type, number of nodes, topology, communication frequency, expected network lifetime, network dynamics, communication pattern, timing constraints, node performance* and *available power supply*.

Application type specifies the domain of the application. For the present work, the following application types are considered: *military, medical, environment monitoring* and *home*.

Number of nodes specifies the maximum number of nodes envisioned for the network.

Topology specifies the layout of the network, the graph representing actual connections between the nodes. This work supports *centralised, clustered* and *mesh* topologies.

Communication frequency represents the number of packets transmitted per unit of time by each node. In applications that don't use storage or aggregation, this can be equated to the sampling rate.

Expected network lifetime is the amount of time the network should function unattended.

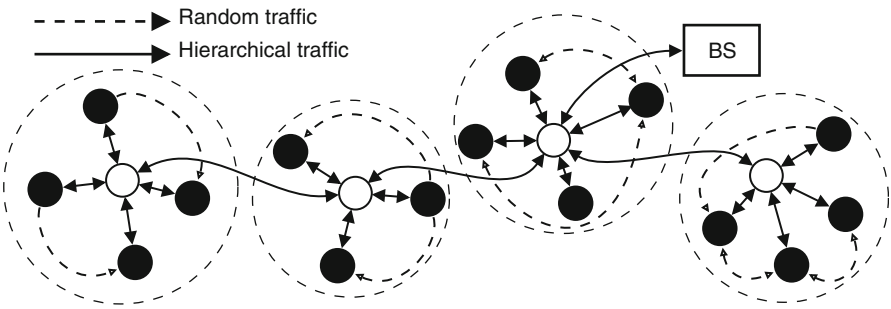


Fig. 2 Communication patterns in WSNs

Network dynamics determines whether the network is static or the topology is likely to change due to node mobility, addition or removal of nodes.

Communication pattern, Fig. 2 shows how data is moved around in the network: within clusters, to facilitate data aggregation, nodes can communicate *randomly* to compare measurements; outside of clusters the data can move *vertically*, or *hierarchically*, since reports are sent to the base station which can send back commands.

Timing constraints indicates whether the application data needs to be communicated to the base station and processed in real-time (*tight constraints*), or the order and processing time of the data is not important (*relaxed constraints*).

Node performance summarises the computational and storage characteristics of the hardware used. To avoid presenting subjective values like "high", "low" and "medium" to the user, a list of known WSN platforms can be used instead, and the translation to a performance level made internally. Good evidence of the computational abilities of different motes is presented in TinyECC [19] for the execution times of public cryptography algorithms, and the results show that there are roughly two classes of performance: high end motes that can perform the algorithm in acceptable time (under one second) and low end motes that can't.

Available power supply is usually determined by the size of the power source connected to the hardware, and most likely specified on the hardware's data sheet.

4.2 Security Parameters

In a similar way, the characteristics of security services and protocols used in WSNs are determined by a set of parameters, which were again selected after analysing the domain. The parameters are: *number of nodes, topology, network dynamics, communication pattern, timing constraints, power consumption per packet, environment security, cryptographic primitives, semantic security* and *attacks*. Some of the parameters are also used for application configuration, and have the same meaning as above.

Power consumption per packet specifies the amount of power or current required to secure and send a packet of data.

Environment security is an indication of the capabilities of the attacker and the likelihood of node compromise. The following values are considered: *trusted* environments, where attacks are unlikely, *public* environments where physical access to the nodes is possible, and finally *hostile* environments where intentional physical attacks should be expected.

Cryptographic primitives are the building blocks used to create security services and protocols; they include symmetric and asymmetric technologies, and for this work the following are considered: Skipjack, RC5 and AES for symmetric cryptography, and ECC (elliptic curve cryptography) for asymmetric.

Semantic security is a requirement for symmetric ciphers which says that an attacker must be unable to retrieve any meaning from secured data. With block ciphers data is considered secure as long as a pair of key and initialisation vector (IV) is not reused. The IV is usually a counter of the same size as a cipher block, and with every sent packet the counter is incremented. As the counter will eventually overflow, a security service can maintain *semantic security* of the data for a number of packets equal to $2^{size\ of\ IV}$ (without updating the key).

Attacks are an indication of the amount of security provided by a service or protocol. By surveying the literature the following attacks were selected, which mostly concern key establishment protocols: forward secrecy (FS), small group (SG), unknown key share (UKS), impersonation by signature forgery (SFImp), multiple session (MS), impersonation by forwarding (FImp), man in the middle (MiM), impersonation (Imp), denial of service (DoS), replay (Rep), type flaw (TF).

5 Methodology

In traditional computer networks, security libraries like OpenSSL [27] offer a unified interface to cryptographic algorithms, so applications can dynamically choose which ones are used when they negotiate the establishment of the secure channel. Generally the strongest (in terms of key and block size) algorithm common to involved peers is chosen. In sensor networks making the choice is more complex because of computation and power consumption constraints not present in traditional networks: the problem shifts from "finding the safest security protocol" to "finding the safest protocol that can provide the required level of security, given a certain set of hardware constraints". Ignoring these hardware constraints can lead to battery exhaustion, communication latency and loss of data.

A security configuration can be defined as a suite of security protocols or cryptographic algorithms which provide for data authentication and integrity, data confidentiality and key management. The suite of security protocols needs to be validated against a set of constraints imposed by the application and the hardware platform. After listing the considered protocols in Sect. 3 and the application and security parameters in Sect. 4, this section presents the main part of the security configuration methodology, namely how the translation is made from the application into the security domain (or from application to security parameters) and finally to security services and protocols.

The methodology consists of several steps:

1. The application parameters specified by the user determine the values of the security parameters. To explain this, Sect. 5.1 first describes how the previously-listed security parameters affect the selection of services and protocols. Then it shows how the security parameters can be determined by the application parameters. Section 5.2 presents the algorithm that performs the translation from application to security parameters.
2. Once the values of the security parameters are known, they are used to select security services and protocols. This is explained in Sect. 5.3
3. If more services or protocols of the same type are selected their attacks and vulnerabilities can be quantified and used as an indication of the provided security. Therefore the protocols can be ordered and the strongest one selected. This is explained in Sect. 5.4.

The section closes with a scenario (Sect. 5.5) which is meant to illustrate the configuration methodology.

5.1 From Application to Security Parameters

The configuration methodology is based on two sets of inference relations: first, relations between the set of security parameters and the set of security protocols; then, relations between the set of application parameters and the set of security parameters. This section presents the relations in a descriptive manner with an illustration in Fig. 3.

First, the impact of security parameters over the selection of security protocols is explained.

The *number of nodes* in the network is influential in the selection of key predistribution protocols, since it determines the number of keys that must be stored on a node; this is limited by the available memory space. Therefore, pairwise predistribution is suitable for smaller networks; master keys can be used in larger

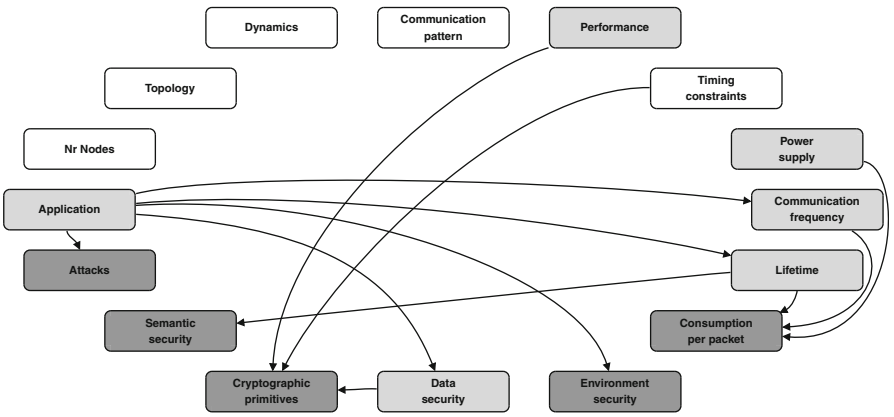


Fig. 3 Configuration parameters

networks, but the larger the network the higher the possibility of node compromise which would lead to total security failure. So master key should also be used for smaller networks. Server shared keys can be used independently of network size, same as probabilistic schemes.

Network topology determines the selection of key predistribution schemes and key establishment protocols. Networks with centralised topology should make use of a server shared key and of centralised key distribution protocols using symmetric cryptography. Clustered topologies can use group or master keys within the clusters and server shared keys between cluster heads and server. Mesh networks demand the use of pairwise or probabilistic key predistribution but master keys can also be used, followed by key establishment, if the devices have sufficient computation power.

Network dynamics determines key predistribution: dynamic networks cannot use pairwise keys as for every change the entire network must be updated.

The *communication pattern* determines key management: random communication should use pairwise, probabilistic or master keys, otherwise the server would be a bottleneck. Also, it should use distributed key establishment or key agreement. Master and server shared keys and centralised key distribution are suitable for hierarchical patterns.

Timing constraints are used to differentiate between key establishment protocols based on the number of exchanged messages: the more messages in the protocol, the larger the latency in sending data. Protocols which were deemed capable of maintaining tight timing constraints are WMF, D-H and MQV (see Sect. 3) The rest are only suitable to applications with relaxed constraints.

Environment security is a measure of the likelihood of node compromise, which usually aims at retrieving secure data. The key predistribution scheme decides what and how many keys are stored per node and by extension, the amount of information the attacker can gain from a single compromise. This must be limited:

- in hostile environments attacks will be intentional so pairwise keys or server shared keys should be used;
- in public environments nodes can be stolen or damaged by accident so probabilistic schemes can also be used;
- finally master keys should only be used in trusted environments where there is no danger of node compromise.

The *cryptographic primitives* determine which data authentication and security as well as key establishment protocols will be used, since all define their preferred ciphers. TinySec and MiniSec both use the Skipjack cipher, SPINS uses RC5 and the CC2420 hardware security implements AES. The keying protocols based on asymmetric cryptography use ECC.

The *attacks* provide information about the strength and security of a protocol or service. Section 5.4 will explain how they are used.

Some security parameters were deliberately left out as they are computed from application parameters and therefore make more sense to be explained in that context.

Before moving on to the relations between application and security parameters, a special parameter needs to be explained. *Data security level* is an *intermediate*

parameter: it does not have direct influence on security services and protocols and it is not within the user's application knowledge. However it had to be introduced as the *application* → *security* relation doesn't cover the whole set of security parameters. Its values are *low* and *high security*, and it influences the selection of the cryptographic primitives: Skipjack and RC5 provide relatively low security, while AES and ECC based protocols provide high security.

The section now continues with the description of how security parameters can be determined from application parameters.

Application type determines the data and environment security level, as shown in Table 1.

Performance together with *timing constraints* define the ciphers (cryptographic primitives) used. Symmetric ciphers Skipjack, RC5 and AES can be performed on low end devices as determined in [28] while ECC, used in key establishment protocols, should only be performed on high end devices. The timing constraints define the same classification for tight and relaxed constraints (tight constraints—symmetric cryptography, relaxed—asymmetric).

The *available power supply*, with the *communication frequency* and the *network lifetime* define the *maximum power consumption per packet* with the formula

$$Max\_cons\_per\_packet = \frac{power\_supply}{communication\_frequency * lifetime}$$

This value is then compared to measured or reported *consumption per packet* values for the data security and authentication protocols to determine if a protocol can be used in the defined circumstances without exhausting the power supply ahead of time.

As explained in Sect. 4.2, *semantic security* indicates the number of packets that can be secured with a cipher and one key, before the initialisation vector (IV) of the block cipher mode of operation, or other key freshness mechanism, will expire and reduce the security. The parameter is equal to the maximum amount of key freshness available to a security service, so, in most cases, $2^{sizeof(IV)}$. The suitability of security services can be determined by comparing the maximum number of packets expected to be sent within the network's lifetime, (*communication_frequency* * *lifetime*), with the value of a service's semantic security: if a service doesn't include key update mechanisms and doesn't use enough semantic security to cover the entire network lifetime, it should not be used; if the service does use key update then it will be selected only if it provides the most semantic security (this is to reduce the number of key updates).

Table 2 presents the complete relations from application to security parameters.

Table 1 Influence of application type on data and environment security level

Application type	Military	Medical	Env. monitoring	Home
Data security	High	High	Low	Low
Environment security	Hostile	Public	Public	Trusted

Table 2 Relations from application to security parameters

Parameter	Relation		
Number of nodes	<100	→	Key_predist={pairwise,master,server-shared}
	>100	→	Key_predistribution={server-shared,prob}
Topology	Centralised	→	Key_predist={server-shared}
		and	Key_est={SymmCentrDistr}
	Clustered	→	Key_predist={server-shared}
		and	Key_est={SymmCentrDistr}
	Mesh	→	Key_predist={pairwise,master,prob}
		and	Key_est={SymmDistDistr,SymmDistPK,SymmAgr,PKAgr}
Dynamics	Dynamic	→	Key_predist={master,server-shared,prob}
Comm pattern	Random	→	Key_predist={pairwise,master}
		and	Key_est={SymmDistDistr,SymmDistPK,SymmAgr,PKAgr}
	Hierarchical	→	Key_predist={master,server-shared,prob}
		and	Key_est={SymmCentrDistr}
Performance	LowEnd	→	Crypto_primitives={Skipjack,RC5}
Timing constraints	Tight	→	Crypto_primitives={AES,Skipjack,RC5}
		and	Key_est={WMF,D-H,MQV}
	Relaxed	→	Key_est={N-S sym,O-R,Shamir,N-S pub,X509,SKKE,STS}
Data security	Low	→	Crypto_primitives={Skipjack,RC5}
	High	→	Crypto_primitives={AES,ECC}
Application type	Military	→	Data_sec={High}
		and	Env_sec={Hostile}
	Medical	→	Data_sec={High}
		and	Env_sec={Public}
	Env. mon	→	Data_sec={Low}
		and	Env_sec={Public}
	Home	→	Data_sec={Low}
		and	Env_sec={Trusted}

prob = Probabilistic key predistribution, SymmCentrDistr = centralised key transport with symmetric ciphers, SymmDistDistr = distributed key transport with symmetric ciphers, SymmDistPK = key transport with asymmetric ciphers, SymmAgr = key agreement with symmetric ciphers, PKAgr = key agreement with asymmetric ciphers, Env. mon = environment monitoring, Env_sec = environment security

5.2 Parameter Translation Algorithm

Before determining the most appropriate suite of security protocols for an application, the information provided by the user (in terms of application and hardware parameters) needs to be translated into the security knowledge domain. The parameter relations are used first to translate from user parameters to the intermediate (indirect) parameters, and then from this set to the security parameters. The set of parameters that the value of another parameter depends on is termed *dependencies*, and determining the value of a parameter is termed *resolving* the parameter.

Listings 1 Steps for translation algorithm

```
set the values for all the protocol parameters that are also user
  parameters
LP = list of parameters
LNV = list of unresolved parameters
while LNV is not empty do
P = first(LNV)
    LNV = LNV \ ⟨P⟩
    SP = set of parameters that influence P's value
    if ∀p ∈ SP, resolved(p) then
        resolve P
    else
        LNV = LNV ∪ ⟨P⟩
    end if
end while
```

Listings 2 Resolving list parameters

```
PV ← possible_values_for_parameter_P
for all p ∈ dependencies(P) do
    IV  ←  constrained_values_for_p
    PV = PV ∩ IV
end for
return PV
```

To allow the configuration methodology to be extended with new protocols and parameters, a generic algorithm is used, which assumes any number of indirect parameter sets and relations. The algorithm is similar to planning algorithms: it starts from the *goal* (the set of protocol parameters) and tries to resolve every parameter (to translate it in user parameters) using the relations. The algorithm is presented in listing 1.

Depending on the value type of a parameter there are two ways to resolve it. If the values form a discrete set, all the values that are not constrained by the values of the influencing parameters will be removed from the list. This is presented in listing 2. If the value of a parameter is computed, a formula solver for simple expressions has been implemented to handle basic arithmetic with round brackets. It processes the formula as a string where it first replaces the variables with the values of the parameters indicated by the variable list. Then the expression is solved mainly through string processing.

5.3 Selecting Protocols

Protocols must be represented as sets of ⟨*parameter, value*⟩ pairs. Once the security parameters are all resolved the protocols that satisfy the user application configuration can be selected. Protocol selection is based on the following definition of protocol constraints: protocols are considered to satisfy a constraint if

they match the values of the security parameters; protocols that don't give values to certain parameters are considered to *not be constrained*, therefore they also satisfy the constraint. For example, if a key establishment protocol does not define a preferred cipher, it can be used with any of them. For a protocol p, parameter r and value v, a constraint c is represented as follows:

$$c(p,r,v) = \begin{cases} 1 & \text{if } p \text{ has value } v \text{ for } r \\ 0 & \text{if } p \text{ has different value than } v \text{ for } r \\ 1 & \text{if } p \text{ doesn't have a value for } r \end{cases}$$

To select the protocols that validate the constraint, from the full set of protocols in a category the ones that provide different values for the parameters are removed.

5.4 Quantifying Protocol Strength Based on Attacks

The security configuration methodology should yield a single protocol or service for every security category (confidentiality, authentication, key management). It is however possible that more than one protocol in a category satisfies the application configuration. Ideally the most secure protocol would be selected to be installed on the application. Quantifying the robustness and security of a service or protocol is no easy thing, and sometimes the solution implies subjective classifications, like "more secure" or "less secure". In this methodology, the security of two protocols is compared based on the set of attacks they are vulnerable against.

The attacks presented in Sect. 4.2 were ordered according to the effort placed on the attacker. The list in ascending order is: forward secrecy (FS), small group (SG), unknown key share (UKS), impersonation by signature forgery (SFImp), multiple session (MS), impersonation by forwarding (FImp), man in the middle (MiM), impersonation (Imp), denial of service (DoS), replay (Rep), type flaw (TF). Every attack is tagged with a number that indicates it's position in the list, and is an estimate of its likelihood: FS is 1 (most difficult attack, since it requires the compromise of an old key) and TF is 11 (easiest attack that only requires reordering or replacing message fields). To determine the security of a protocol (and allow comparison), the values for all the attacks the protocol is vulnerable against are summed, which results in the total vulnerability: the lower this value, the more secure the protocol. Table 3 shows the vulnerabilities for the considered key establishment protocols.

5.5 Scenario

Configuring security for a sensor network is explained in the following scenario: a modern hospital uses wireless sensors to monitor patient status, so that every room

Table 3 Attacks (second row) and total vulnerability (third row) for key establishment protocols

NS sym	OR	WMF	Shamir	NS PK	X509	SKKE	DH	MTI	MQV	STS
FS	TF	MS	Imp	FImp	SFImp	FS, UKS	Imp, MiM	UKS, SG	UKS	UKS
1	11	5	8	6	4	4	15	5	3	3

has a network of less than one hundred nodes. Data from every sensor is reported to a central computer every second, and the maximum time a patient is considered to stay in the hospital under monitoring is one month. The nodes are cheap and the hardware is low end, they are powered by two AA batteries giving roughly 3,000 mAH each, so a total of 6,000 mAH. The nodes should be mobile to allow patients to move around, and common sense dictates that a clustered topology should be used, with each patient being a cluster of sensors.

The security configuration is resolved as follows. Since this is a medical application a high level of security is required and the environment is public. The public environment eliminates master keys, and the clustered topology, hierarchical communication and node mobility restrict the choices to using server shared keys. Key establishment protocols are restricted to centralised distribution, and the safest protocol is found to be Wide Mouthed Frog, vulnerable only to multi-session attacks. Next, for data security, the ciphers are determined from the performance, timing constraints and security level: public algorithms are eliminated due to low end hardware, and as medical applications require a high level of security we are left with AES. Therefore, we can choose CC2420 security, as the others use either RC5 or Skipjack. The AES cipher is used in CCM (counter with cipher-block chaining MAC) mode thus also providing authentication. For broadcast authentication a version of μTESLA working with the AES cipher would be needed.

If the network were deployed with the same requirements but used for environment monitoring a low security level would be acceptable therefore the security tool would be able to select RC5 and Skipjack too. The choice between CC2420, TinySec, MiniSec and SPINS would be made on the maximum consumption per packet and on the semantic security. The expected number of packets sent by a node is at least 2,678,400 (one packet per second a month), which eliminates TinySec whose IV of 16 bits overflows after 65,536 packets and would require more frequent key establishments. The maximum allowed consumption per packet is 0.00224 mAH. Since MiniSec has a packet consumption of 0.000167 mAH and SPINS 0.000188 mAH, both protocols can be used. They are both resilient to replay attacks so the tool will recommend both.

6 Implementation

6.1 Java Application

The security configuration methodology has been implemented as a Java application. Security protocols and parameters are stored in an SQLite database which is interfaced with the application using the appropriate JDBC (Java Database Connectivity) driver. The database schema is presented in Fig. 4, and the main application interface in Fig. 5; these are the two windows that the user will primarily interact with, the application definition window, where parameters are set, and the security suite window, where the tool recommends security protocols to install. Additional interfaces are used to create or modify parameters and protocols.

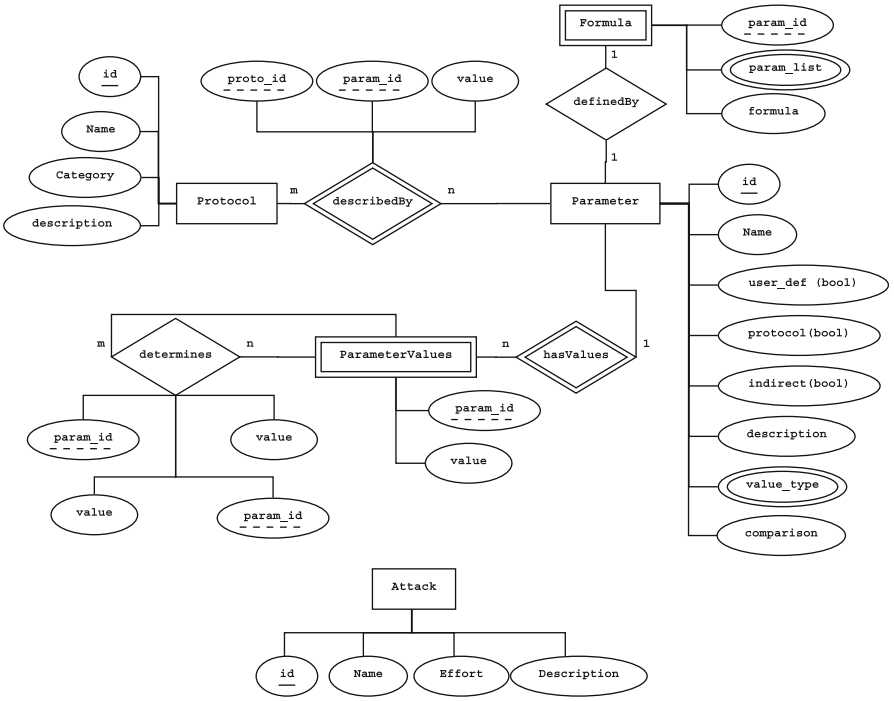
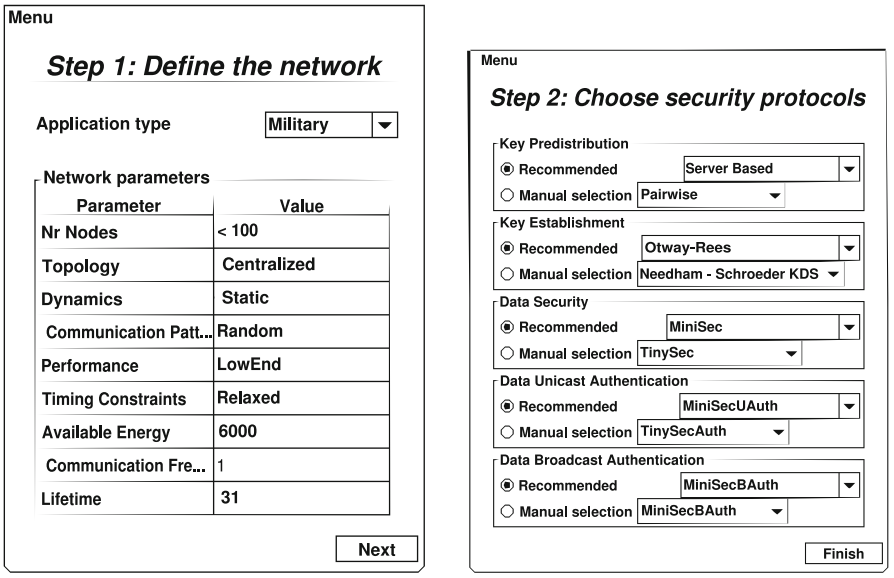


Fig. 4 Database schema



(a) Application definition **(b)** Results

Fig. 5 Primary user interaction with configuration tool

The tool goes through the following steps:

1. in the first window (Fig. 5a), the user selects application parameters, then clicks the Next button;
2. the tool determines the values for the security parameters using the translation algorithm, given previously in listing 1;
3. the security protocols are determined through a database search;
4. if more than one protocol of a certain type is present, they are ordered based on vulnerability (see discussion in Sect. 5.4) and the safest is selected; if for some type, there is no available protocol, all protocols of that type are returned, ordered on safety, and the user can select one;
5. the final set of protocols is presented to the user, as shown in Fig. 5b.

The application also provides a graphical database interface which allows management of protocols and parameters and inference relations.

6.2 Scenario Application

The application described in the scenario in Sect. 5.5 has been implemented as closely as possible in TinyOS [29] on the Shimmer platform [30], an MSP430-based wireless sensing node that uses the CC2420 radio chip for 802.15.4 compatible communication. The Shimmer platform is used in clinical trials and research for digital health and ambient-assisted living [31], therefore it is likely to be used in hospitals in the future.

In the application, whose structure is presented in Fig. 6, nodes will exchange data over a secure channel every second. As per the recommendations of the configuration tool, the channel is secured using the CC2420 implementation of AES

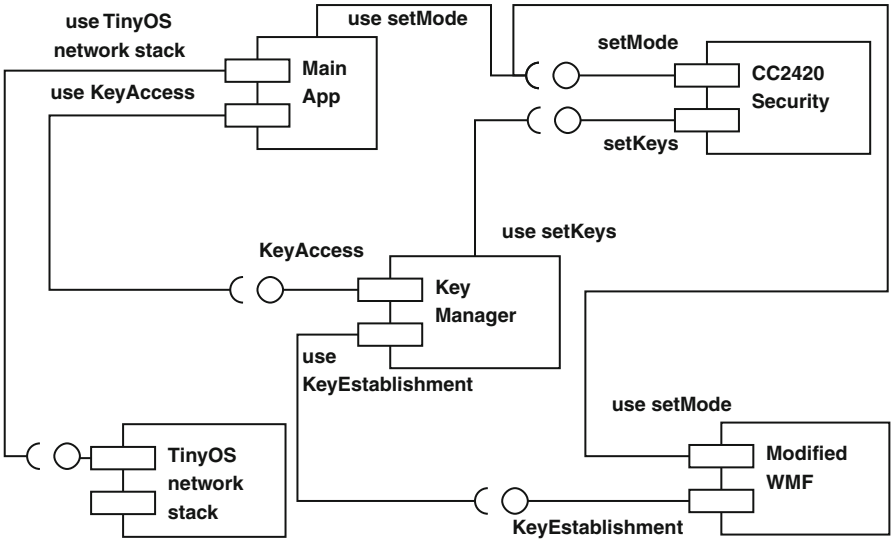


Fig. 6 Components of the scenario application

in CCM mode. Also, every node initially shares a key with the server and for two nodes to communicate they need to establish a key, which is achieved through the Wide-Mouthed Frog (WMF) protocol [32]. Traditional WMF only requires two messages which makes it one of the shortest and safest key transport protocols. It operates as follows:

1. A generates key to use with B;
2. A sends to server: A, B, timestamp_A, key encrypted with the key shared by A with the server;
3. The server sends to B: A, B, timestamp_server, key encrypted with the key shared by B with the server.

The validity of each message is provided by timestamp verification. The protocol is known to only be vulnerable to multisession attacks.

This version of WMF is not really suitable for sensor networks due to the difficulty of preserving tight synchronization between nodes. In [33] the timestamps are replaced with a nonce selected by the receiver (node B). This version was further enhanced by the authors to provide key confirmation, and is preceded by a HELLO-PRESENT (discovery) stage, to make sure that the receiver is ready to establish a key with the initiator. The final version is presented in listing 3. Every received message is verified thoroughly to prevent type flaw attacks; only if a message is valid will the protocol move to the next step. The state machine for the protocol initiator is displayed in Fig. 7.

Access to keys is mediated by the *Key Manager*; this provides generality and portability to the code. The key manager delegates key generation and

Listings 3 Modified wide-mouthed frog with HELLO-PRESENT phase

$A \rightarrow B$ **HELLO, A, B**
$B \rightarrow A$ **PRESENT, A, B, nonce**
$A \rightarrow server$ **Enc(A,B,nonce,Key)** $_{K_{AS}}$
$server \rightarrow B$ **Enc(A, B, nonce, Key)** $_{K_{AS}}$
$B \rightarrow A$ **Enc(PRESENT, A, B, nonce)**$_{Key}$

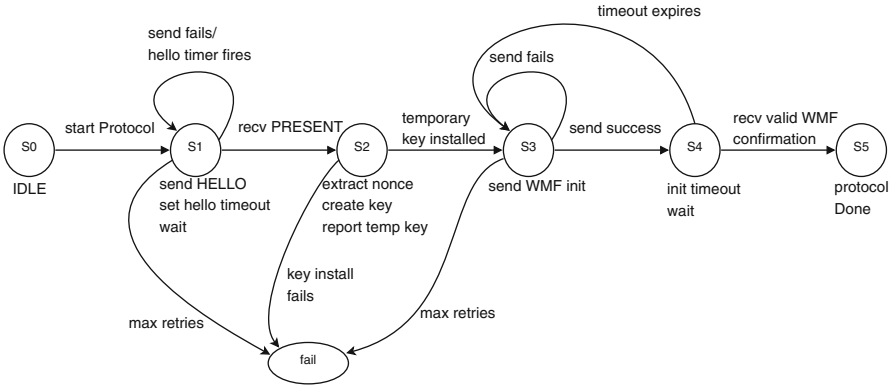


Fig. 7 Protocol state machine for WMF initiator

establishment, monitors key usage, notifies when keys need to be renewed and revoked. The keys themselves are stored in a database inside the key manager, together with information needed to handle them (*e.g* to determine key establishment protocol). It is possible to integrate any key establishment protocol with the key manager, provided the *Key Establishment* interface is used. A typical key manager interaction is as follows, with the interfaces presented in Fig. 8:

- application needs key to secure channel, requests key using *KeyAccess.getKey*
  - if key exists in database it is returned immediately by key manager
- if there is no key, or it must be renewed, the key manager returns a *WAIT* status, and asks the proper key establishment protocol to establish a key (with the specified peer), using the *KeyEstablishment.startProtocol* command
- when the key is ready, the manager is notified with *KeyEstablishment.protocolDone*
  - to support key confirmation the key establishment protocol uses a *TemporaryKey* interface, provided by the manager
- in turn, the manager will notify the application and provide the new key with *KeyAccess.foundKey*.

In the current application the key manager works closely with the CC2420 security mechanism. The chip provides only two slots for keys, and for two nodes to be able to communicate securely, they need to be using the same key slot. The key manager therefore needs to constantly install and remove keys from the chip RAM (Random Access Memory) if there are more than two nodes involved; this happens in the WMF key establishment, as there is no way to use two key slots and have all three parties (initiator A, receiver B and server) communicating. The convention used in the application is the following:

- in the first part of the protocol the server keys have priority, so A will share key slot 0 with the server and B will share slot 1 (this is dictated by the server slots);
- when A creates the session key it installs it in slot 1;
- once B receives A's key, it will switch slots and install the new key in the server's slot (1) so that it can use it to communicate with A.

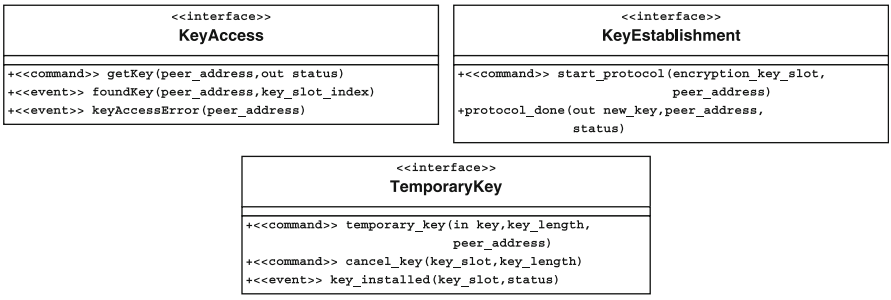


Fig. 8 Key handling interfaces in the scenario application

This is an inconvenience of the CC2420 implementation of the 802.15.4 standard.

The implemented scenario application is validated in Sect. 7.2

7 Tests

This section provides test results which support the idea that wireless sensor networks need security configuration. The scenario application is validated against the scenario requirements presented in Sect. 5.5

7.1 Motivation for Security Configuration

Ideally all applications would use the strongest cryptography possible. However, doing so would always cause one or more of the application parameters to go outside of the allowed boundaries. The memory might not be enough to store all the keys, or the hardware not fast enough to maintain network latency within the required bounds. The size of initialisation vectors and MACs could push the packet size beyond the maximum packet size allowed by the physical network layer, or the amount of network traffic created by key management schemes could be so large as to quickly deplete the available power supply of the motes. Therefore a methodology such as presented in Sect. 5 to determine the ideal suite of security protocols using the parameters listed in Sect. 4 is required.

The relations between application and security parameters, which are the main part of this methodology, are all explained where they are defined. In what follows some experimental proofs are given to support the necessity of security configuration for WSNs. The following are evaluated:

- the impact of key predistribution scheme on the amount of network traffic in multi-hop networks,
- the impact of key predistribution scheme on the memory consumption,
- the impact of security overhead on the size of a packet and current required to send it.

The key predistribution scheme must set up keys between nodes in a way that follows the network topology and the planned communication pattern. With hierarchical network traffic and a clustered or centralised topology, every node should share a key with the cluster head or base station. With multi-path routing or random network traffic and a mesh topology, nodes should share keys directly with other nodes. Choosing an inappropriate predistribution scheme can have a negative impact on either the amount of network traffic required to set up keys or the memory space required to store them. There is a trade-off between the two, as some schemes create less traffic but use more memory while others store few keys but require complex protocols with more messages to set up new keys. This trade-off is explored in the first two tests.

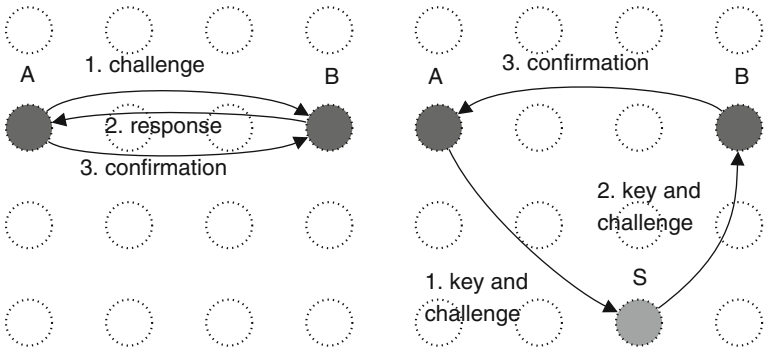
The following scenario is evaluated: a network organised in a mesh, with random traffic, where any two nodes should be able to communicate securely. In the first test

pairwise key predistribution is used, and in the second server-shared. To establish secret keys nodes use a key update (KU) protocol in the first case, and the Wide-Mouthed Frog (WMF) protocol in the second case. The key update protocol, Fig. 9a, has nodes performing a challenge response secured with the pre-shared key (set up during pairwise predistribution), followed by a key confirmation. In WMF, Fig. 9b, the nodes will use a trusted third-party, the server, as a mediator to exchange the secret key and in the end perform a key confirmation. Considering the scenario, from the point of view of network traffic, the first case uses the correct keying scheme while the second uses an inappropriate one.

Observe that for the first case, as the nodes are already sharing a key, a key derivation could also be performed, which would require the exchange of less messages. The key update protocol is instead used as it employs the same number of messages (three) as WMF and therefore provides better comparison.

The protocols were simulated over an ideal routing protocol that always returns the shortest path between two nodes. The application was set up to have all the nodes establish keys with each other. The number of packets sent through the network, during key establishment was measured. Only the actual key establishment packets were counted, including the ones that are forwarded in case multi-hop communication is used. Any additional control traffic needed to determine the routing path was ignored. The size of the network was increased from ten to ninety nodes in increments of ten. The graph in Fig. 10 shows the average number of packets sent by both protocols as a function of the network size. It is clear that the use of server-shared keys and the WMF protocol in the mesh incurs more network traffic than the use of pairwise keys and the key update protocol. This is directly proportional to the amount of power consumed throughout the network. The results of using server-shared keys is *a more aggressive shortage of the entire network lifetime*.

The WMF protocol causes more network traffic as the key establishment is mediated by the server. The farther (in terms of hops) the server is from the initiator



(a) Key update **(b)** Wide Mouthed Frog

Fig. 9 Message exchange for both key establishment protocols in a mesh network

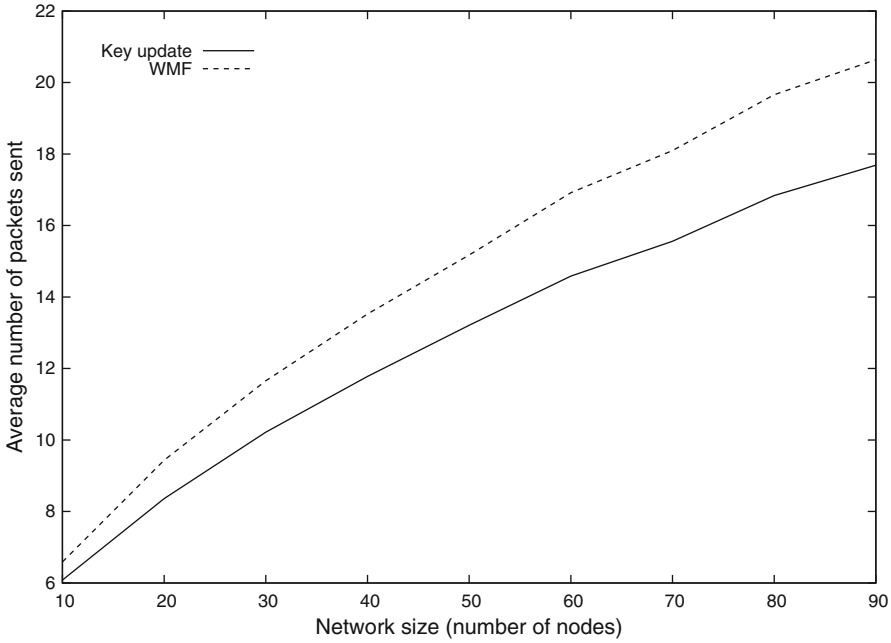


Fig. 10 Average number of packets sent through the network when any two nodes establish a key

and receiver, the more traffic. This is pointed out on the graph in Fig. 11. The graph shows protocol runs grouped by the distance between the protocol initiator (A) and receiver (B). The X axis represents the ratio between the distance from A to B through the server and the distance between A and B. As the traffic generated by the key update protocol does not depend on the server's position, its values are constant within groups, while those of WMF increase. The graph also indicates that the two protocols create the same amount of network traffic when the path A-server-B is twice as long as A-B, after which WMF values are larger.

The choice of key predistribution also has an obvious impact on the memory consumption. The reverse of the previous scenario, using pairwise keys in a centralised network will prove inadequate from this point of view. More explicitly, as shown in Fig. 12, memory space required to store pairwise keys increases linearly with the network size, as each node must store keys for the rest of nodes in the network. It is better to use other schemes:

- in server shared and master schemes, there is only one key stored per node; however, the former is more resilient against node compromise;
- in probabilistic schemes the memory space is still a function of the network size, but it varies sub-linearly.

In Sect. 5.1, the reason for choosing 100 nodes as the limit for using pairwise keying for mesh networks is that with an approximate key size of 100 bits (most ciphers use 128 bits, some still use 64 or 96) about 10 kilobytes of memory would be occupied only by keys. That is the maximum memory size for most low performance motes.

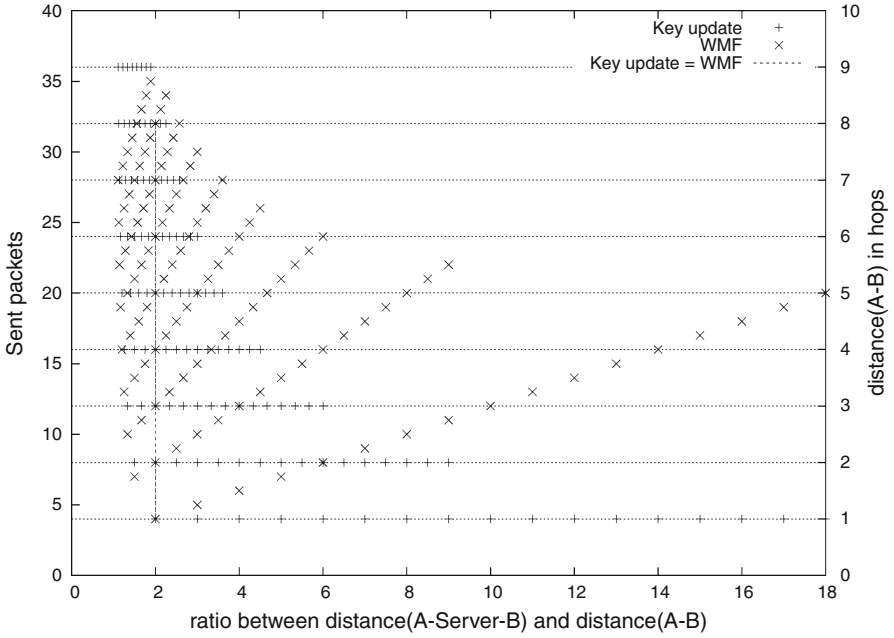


Fig. 11 Influence of server's position (relative to A–B path) on overall network traffic

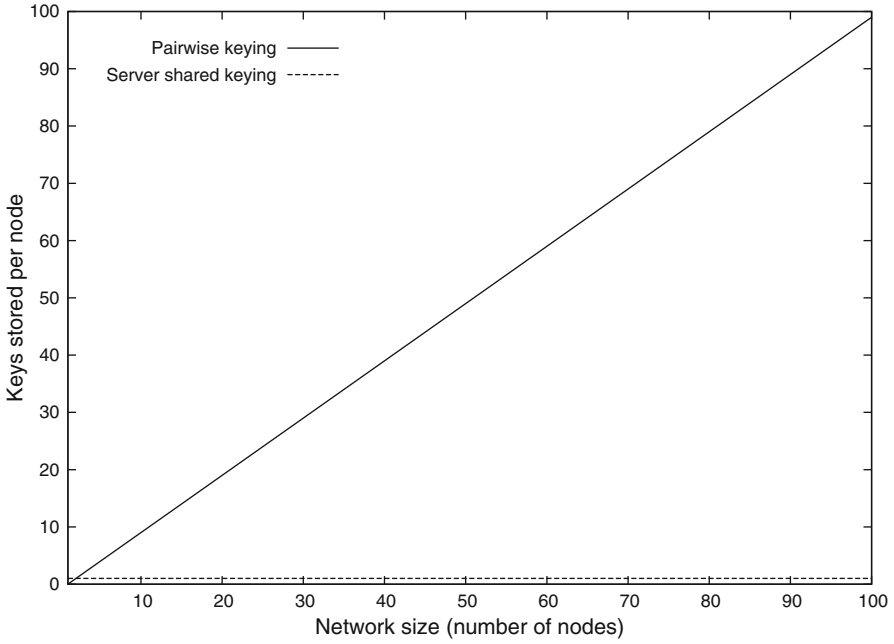


Fig. 12 Key predistribution impact on memory consumption

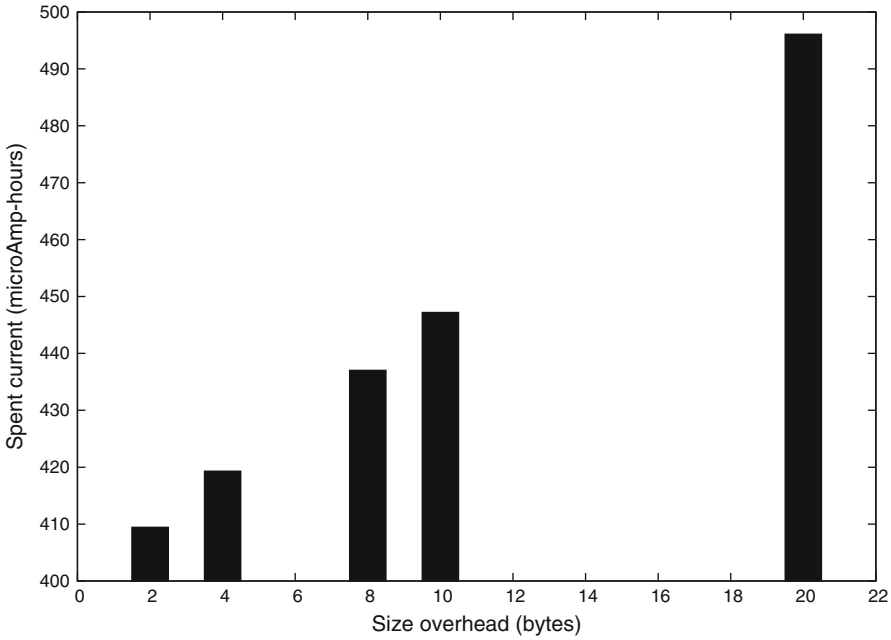


Fig. 13 Security data added to messages causes a linear increase in the current required for transmission

Of course, the case could me made that keys can also be stored in ROM (Read Only Memory) or external storage, but this gives rise to other issues like key access times or memory compromise.

Another parameter which should influence the selection of security services is the amount of security data which is added to packets. This concerns data confidentiality and authentication services, which add initialisation vectors, authentication and replay protection data to messages. It also concerns the size of keys or key establishment messages which are exchanged during key establishment protocols. The problem is that the size of the sent message is directly proportional to the current spent to send it. As the size of the packet increases, the decrease in the power available to the node will be faster, thus shortening the node's lifetime. This was evaluated by sending payloads of various sizes and measuring the current consumption. Measurements were performed with an Agilent 66321D DC source over batches of 10000 packets. The results are shown in Fig. 13. The increase in amount of spent current is linear with the size of the packet.

Keeping packets as small as possible is a well-known requirement in WSN applications. Most security services will construct the IV, which doesn't need to be secret, from other fields in the packet, like source and destination address, adding a small bit of information to make the IV unique to every sent message. The message authentication code can't follow the same approach, so its overhead is unavoidable. Using small MACs is dangerous as it reduces the attacker's effort in forging packets with valid authentication data. The authors of the TinySec paper [10] defend their

usage of a four byte MAC (which the attacker could forge in 2^{31} attempts) with the observation that the attacker, unable to know if a forgery is successful, would have to send 2^{31} packets and analyse the behaviour. That amount is more than a sensor network could support.

The intended message of the previous paragraph is to state that the amount of added security is already quite low: MiniSec adds 3 bytes [11], TinySec adds 5 bytes [10] and SNEP 8 bytes [18]. The results in Fig. 13 show variations of only tens of micro Ampere-hours, which could cause major differences in the node's lifetime only for tens of thousands of packets. Given the low sampling rates of long-life networks, this apparently makes the size overhead inoffensive. However, with applications that use on demand routing protocols, a single application packet could cause a lot of network traffic.

Consider the following scenario: an application uses security at link layer to protect the routing protocol and prevent attackers from joining the network. A first test was set up to measure the amount of control packets sent by the routing protocol when searching for a route. The Tymo routing protocol [34] was selected as the only unicast routing protocol properly supported by TinyOS. Networks of different sizes were tested with one of the nodes sending ten packets to every other node. The nodes were set up in a single line, to encourage multi-hop routing. A sniffer was used to count the total number of packets sent throughout the network, that is, actual data packets plus multi-hop forwarding packets and routing control packets. The results are shown in Fig. 14. It is clearly visible that the overhead increases aggressively

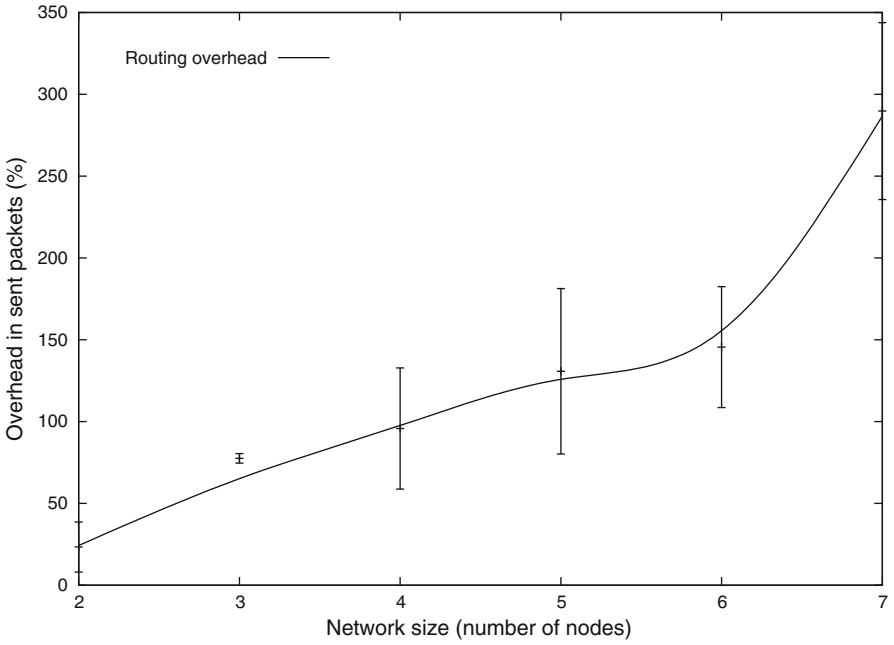


Fig. 14 Overhead in network traffic caused by routing control packets and network instability

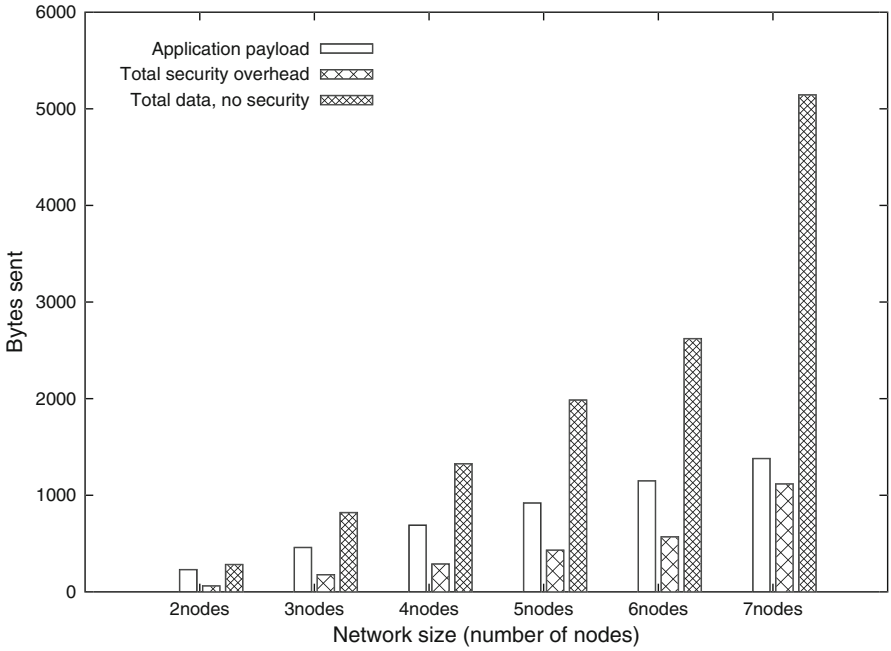


Fig. 15 Multi-hop network with on-demand routing. Comparison of application payload and total security overhead (from forwarding and control data) at routing layer

with the network size. The explanation is that routing queries are flooded through the network every time a path must be updated. This happens whenever a message is not acknowledged, which is common, given the unstable nature of WSNs, well-documented in [35]. This is also causes the large standard deviation.

Coming back to the scenario, if the link layer is secured then every routing control packet is accompanied by authentication and confidentiality data. To send the additional security data current is needed that could be used to send more packets, therefore all the security overhead reduces the amount of packets that the network can transport during its lifetime. To show the extent of the reduction, the following estimation is performed: the TinyOS default message payload is of 28 bytes. Suppose that out of that, 23 bytes are used for actual data and the other five for security information. The application from the previous scenario is used, a multi-hop network with the Tymo protocol for routing, and the application that has a source node sending ten messages to all other nodes. The link layer is secured and the values for routing overhead are taken from Fig. 14. In the graph in Fig. 15 the following are compared:

– the number of bytes sent at application layer, that is, ⟨number of nodes⟩x⟨10 messages per node⟩x⟨23B per message⟩;
– the number of bytes *actually* transmitted through the network (forwarded application data plus routing control) broken down into security only and data only.

The graph shows that as the routing overhead increases drastically for larger networks, the *overall* security overhead becomes almost as large as the data sent by the node at application layer. That amount of data is subtracted from the overall network lifetime.

In conclusion, there must be a balance between the amount of security provided by a service or protocol and the resources it consumes. Of course it would be easy to use the best solution all the time, but that can exhaust the available resources or violate network requirements for some applications. Therefore, the strength of the key management scheme must balance the memory size and the overhead in network traffic. The level of security provided by a cipher must match the performance of the sensor node, to stay within the desired network latency. And the amount of security data added to a message must balance the available power supply and expected network lifetime.

7.2 Scenario Application: Test Results

The scenario application has been implemented with the following goals in mind:

- a version of the WMF protocol adapted to wireless sensor networks is implemented and tested;
- the configuration tool is validated in a real world application.

Three tests were performed: the first two measure the current consumption and efficiency of the WMF key establishment protocol. The last test determines whether the application can be used in the medical scenario presented in Sect. 5.5, in terms of current consumption and network life time. All the measurements were performed using an Agilent 66321D DC source.

Current consumption measurements were done for three versions of the application, with the results displayed in Fig. 16:

- data exchange without encryption, *solid black* on the graph;
- encrypted data exchange *with preinstalled keys*; this means that key establishment is not used, *dashed black* on the graph;
- encrypted data exchange *without preinstalled keys*, which requires key establishment, *solid gray* on the graph.

The keys used are shared between a node and the server. The three plots are aligned at the start of the data exchange session to help compare the current consumption values, and also clearly indicate the HELLO-PRESENT discovery stage of the modified WMF key establishment protocol. The current consumption difference between encrypted (solid gray and dashed black) and non-encrypted (solid black) data exchange is visible on the plot. The messages exchanged *during* the key establishment protocol are only *indicated* in this figure, as they are too close together to be distinguished individually.

The exact message exchange for the protocol initiator in WMF is shown in Fig. 17. The protocol receiver will have roughly the same activity, with one message less, as the receiver does not send HELLO messages. The average current during the

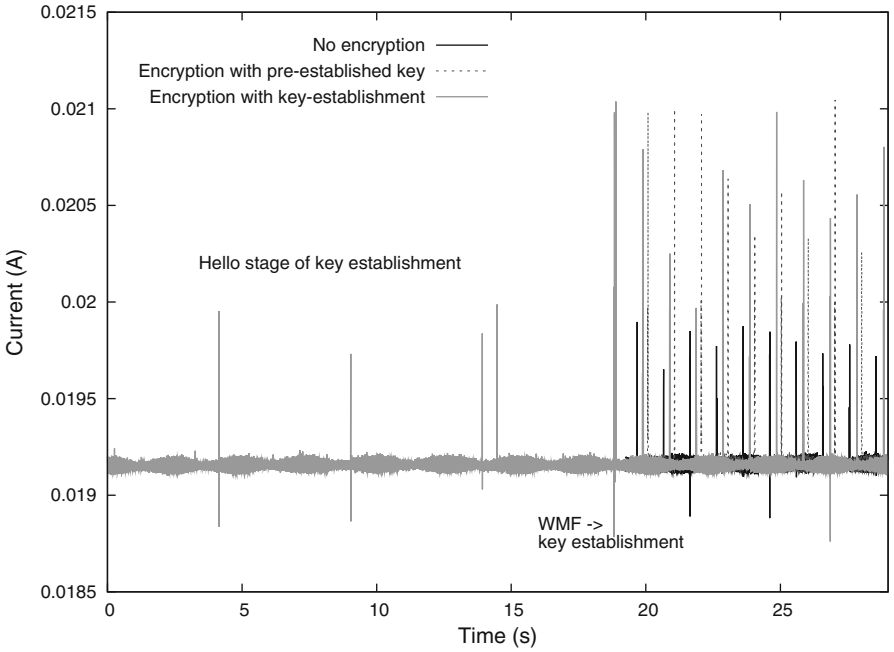


Fig. 16 Difference in current consumption for three versions of the application

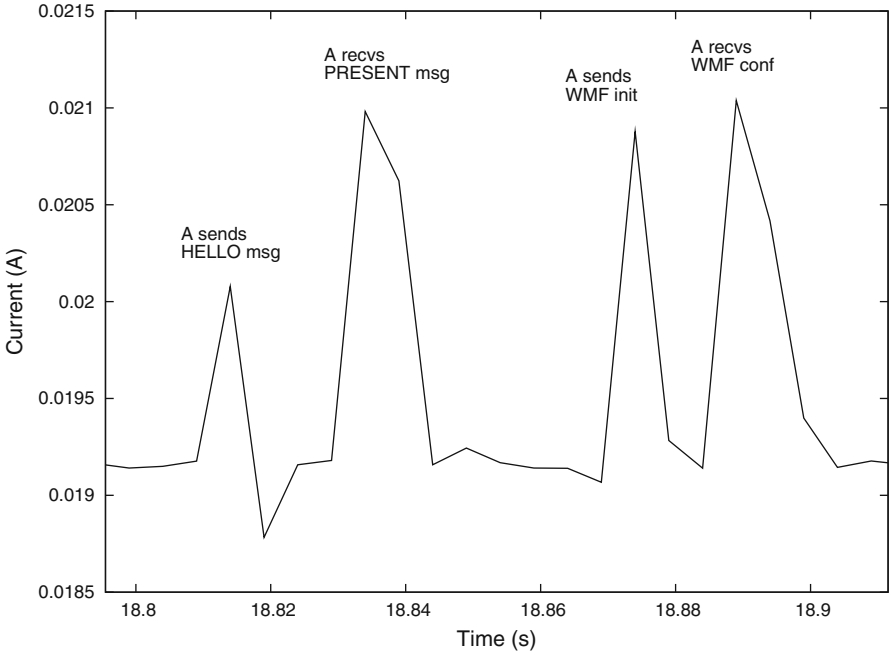


Fig. 17 Message exchange during the WMF key establishment protocol

exchange was measured at 19.5 mA; the average duration of the exchange is 120 ms, which translates into a 0.00065 mAH current consumption. The unencrypted hello messages sent by the initiator need an average of 19.3 mA. With a duration of 25 ms per packet, this translates into 0.00013 mAH for each additional hello packet.

The TinyOS driver for the CC2420 hardware security uses a four byte counter to ensure key freshness (or semantic security, as explained in the parameter Sect. 4.2). This implies that after sending 2^{32} secured packets the key must be renewed. Otherwise, the generated keystream of the counter mode of AES will be the same as 2^{32} packets earlier, meaning that an attacker can perform the following operation:

- $C_i = Ks_i \oplus P_i; C_j = Ks_i \oplus P_j$, where $j = i + 2^{32}$, C stands for ciphertext, P for plaintext and Ks for keystream;
- $C_i \oplus C_j = P_i \oplus P_j$, and an experienced attacker could deduce the value of P_i and P_j.

A question that was asked as part of this work is whether there is any improvement in the current consumption if the counter (which is transmitted in the packet) is reduced to two bytes, which leads to more frequent execution of the key establishment protocol. In other words, if the current overhead of sending two bytes over 2^{16} packets is greater or less than the current required to perform one key establishment. The CC2420 driver was modified to reduce the counter size and five current measurements were taken for packets with four byte counter, and five for packets with a two byte counter. In both cases, the AES cipher was used in counter mode, with no authentication. The average current consumption determined for the four byte counter packet is 22.268, and 22.266 mA for the two byte counter packet. The determined difference of 2μA amounts to 0.000353 mAH for 2^{16} packets of roughly 10 ms each. This value is smaller than the current needed for a WMF key establishment (0.00065 mAH), therefore it was established that *there is no gain from reducing the counter size from four to two bytes*.

As previously mentioned, the application was also developed to test the validity of the configuration scenario, most important, to see whether it would match the constraints imposed by the application and the user. In terms of security, AES with 128 bit keys is considered safe until 2030 [36], and the CC2420 hardware implementation is fast, which accounts for the timing constraints of the application. The main question is whether, considering the specified power supply of 6,000 mAH, the nodes can function unattended for thirty days, as required by the user. It is well known that the principal culprit for power consumption in sensor networks is the radio system. According to the CC2420 specifications [14], in receiving mode the radio drains 19.7 mA. However, if the radio is idle, it only requires 426μA. It is a common practice to perform duty cycling on the radio, and turn it off when it is not needed. The *LowPowerListening* capability of the CC2420 radio was used in the application, with a duty cycle of one second. As a result, the average current was measured at 6.274 mA. The 6,000 mAH supply of current could support the network for approximately forty days, which is more than demanded in the scenario, and could be extended with more aggressive power saving, like duty cycling the sensors or even the entire microcontroller.

8 Discussion

8.1 On Security Configuration

In the current state of things, the security configuration of wireless sensor networks is not 100% necessary. Some of the results presented in this paper, as well as those presented by other authors mentioned throughout the paper, show only slight differences in timing and power consumption between ciphers or protocols.

- The security provided by the available ciphers is only *relatively* high or low, as usually mounting a cryptanalysis attack is the last resort for the attacker.
- An additional two or three messages for key establishment will not cause a big difference in the power consumption of a single node.
- Using asymmetric cryptography techniques, with an overhead of a few seconds for ECC is possible for most applications, provided that they are not too frequent.

Of course there are conditions, like memory space or network latency, that must be respected when choosing security protocols. In networks requiring real-time data processing, the fastest ciphers and protocols will be used. And if the memory space available isn't enough for a certain key predistribution scheme, another one should be selected. But one could say that with the current WSN hardware, security configuration is only *borderline* required. If the hardware starts increasing in performance (and cost) then it might not be necessary anymore. If WSN nodes start getting smaller and with lower power, relying more and more on power harvesting techniques, then that available supply will need to be heavily rationed and the security protocols extremely efficient. In any way, just the fact that WSNs can be used in so many different applications with such varied requirements should be an incentive to provide good methodologies for choosing the right suite of security protocols.

8.2 On the Scenario

The scenario details are purely illustrative. Although most research platforms available to the authors use AA batteries, it is unlikely that the same would be used in a hospital as they would be uncomfortable to wear. Button cell batteries are of more suitable size, but they usually provide less than 500 mAH. This reduces the network lifetime, and would probably require periodic recharge of the nodes. A rotation of two groups of nodes per patient is envisioned, with one group active while the other one is recharging. This complicates the problem, because while nodes are recharging they are in a *hostile* environment as an attacker could compromise them (say, while the patient is asleep), extract the key and gain full control over patient data when they become active again. Hostile environments require frequent key updates and protocols with forward and backward key security, to restrict the impact of node compromise.

8.3 On the Tool's Precision

Lastly, a few things can be said concerning the precision of the configuration tool as far as power consumption and network life-time are concerned. Calculations in the configuration tool take into account only the current consumption for security related operations, sending a secured packet or performing key establishment. No assumption is made about the average current consumption for the considered application and platform (if radio communication is excluded). Whether the application performs any kind of power saving and duty cycling can make a big difference, as seen at the end of Sect. 7.2 with low power listening. With the current configuration methodology it is recommended that developers create power-efficient applications; in the future, the methodology will be modified to take into account a minimum and maximum average current consumption for each platform, when estimating the network lifetime for the secured application.

9 Conclusions

Configuring security in wireless sensor networks is not an easy task, but it is a necessity if WSNs are to reach maturity. The best way to handle security configuration is through a security framework integrated in the sensor network life cycle, from development to deployment. The framework is designed to demand from the WSN actors, developer and user, only information which is within their knowledge domains: the developer indicates *where* security is needed, while the user determines *what* security protocols will be used. Work on the security framework will be the subject of future publications.

The security configuration methodology, which is the main focus of this paper, hides the inherent complexity of communication security and places the controls of security configuration in the hands of the user. By creating a correspondence between user knowledge (application) domain and the security domain, the configuration methodology is able to recommend the most efficient security protocols for the user's application.

The paper presents several experimental proofs of how choosing a security service or protocol can have an impact on the various parameters of a WSN. The test results support the necessity of configuring security for WSN applications.

The configuration methodology is implemented as a Java tool, and further exemplified using a scenario. The application described in the scenario is implemented as closely as possible in TinyOS and the security protocols recommended by the Java tool are installed on the application. The implementation and tests performed on the sample scenario application provide important, previously unpublished values: the current consumption for the Wide-Mouthed Frog protocol (as modified by the authors to fit WSN usage), as well as the trade-off between reducing the key freshness (counter size) and an increase in key establishment frequency. More important, the measurements show that the secured application should meet the requirements specified by the user during the security

configuration, especially network lifetime. This provides validation for the decisions made by the security configuration tool.

Acknowledgments The authors wish to thank the following for their financial support: the Embark Initiative and Intel, who fund this research through the Irish Research Council for Science, Engineering and Technology (IRCSET) postgraduate Research Scholarship Scheme. Also, many thanks go to the anonymous reviewers for their most valuable comments that helped complete this paper.

References

1. Zhu, S., Setia, S., and Jajodia, S.: LEAP: efficient security mechanisms for large-scale distributed sensor networks. In: CCS '03: Proceedings of the 10th ACM Conference on Computer and Communications Security, pp. 62–72. ACM, New York, NY, USA (2003)
2. Prasad, N.R., Alam, M.: Security framework for wireless sensor networks. Wirel. Pers. Commun. **37**, 455–469 (2006)
3. Law, Y.W., Havinga, P.J.M.: How to secure a wireless sensor network. Intelligent sensors, sensor networks and information processing conference, 2005. In: Proceedings of the 2005 International Conference on Dec., pp. 89–95 (2005)
4. Ransom, S., Pfisterer, D., Fischer, S.: Comprehensible security synthesis for wireless sensor networks. In: MidSens '08: Proceedings of the 3rd International Workshop on Middleware for Sensor Networks, pp. 19–24. ACM, New York, NY, USA (2008)
5. Peter, S., Piotrowski, K., Langendorfer, P.: In-network-aggregation as case study for a support tool reducing the complexity of designing secure wireless sensor networks. In: Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on, pp. 778–785 (2008)
6. Jinwala, D., Patel, D., Dasgupta, K.: Configurable link layer security architecture for wireless sensor networks. J. Inform. Assur. Sec. **4**(4), 582–603 (2009)
7. de Oliveira, S., de Oliveira, T.R., Nogueira, J.M.: A policy based security management architecture for sensor networks. In: Integrated Network Management, 2009. IM'09. IFIP/IEEE International Symposium, pp. 315–318 (2009)
8. Krontiris, I., Dimitriou, T., Soroush, H., Salajegheh, M.: WSN link-layer security frameworks. In: Lopez, J., Zhou, J. (eds.) Wireless Sensor Network Security, Chap. 6, pp. 142–163. IOS Press (2008)
9. Carman, D.W., Kruus, P.S., Matt, B.J.: Constraints and approaches for distributed sensor network security (final). DARPA Project report (cryptographic technologies group, trusted information system, NAI Labs) (2000)
10. Karlof, C., Sastry, N., Wagner, D.: TinySec: a link layer security architecture for wireless sensor networks. In: SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, pp. 162–175. ACM, New York, NY, USA (2004)
11. Luk, M., Mezzour, G., Perrig, A., Gligor, V.: MiniSec: a secure sensor network communication architecture. In: IPSN '07: Proceedings of the 6th International Conference on Information Processing in Sensor Networks, pp. 479–488. ACM, New York, NY, USA (2007)
12. Singh, K., Muthukkumarasamy, V.: Analysis of proposed key establishment protocols in multi–tiered sensor networks. J. Netw. **3**, 13 (2008)
13. Camtepe, S.A., Yener, B.: Key distribution mechanisms for wireless sensor networks: a survey, pp. 05–07. Rensselaer Polytechnic Institute, Troy, New York (2005) (technical report)
14. Chipcon, CC2420 datasheet rev 1.2. Tech. rep. (2004)
15. Hu, W., Corke, P., Shih, W., Overs, L.: secfleck: A public key technology platform for wireless sensor networks. Wireless Sensor Networks Book Series. Lecture Notes in Computer Science, vol. 5432, pp. 296–311. Springer, Berlin/Heidelberg (2009). ISBN: 978-3-642-00223-6
16. Becher, E., Benenson, Z., Dornseif, M.: Tampering with motes: real-world physical attacks on wireless sensor networks. In: Proceeding of the 3rd International Conference on Security in Pervasive Computing (SPC), pp. 104–118 (2006)
17. Karlof, C., Wagner, D.: Secure routing in wireless sensor networks: attacks and countermeasures. In: Proceedings of 2003 IEEE International Workshop on Sensor Network Protocols and Applications, Anchorage, Alaska, pp. 113–127 (2003)
18. Perrig, A., Szewczyk, R., Tygar, J.D., Wen, V., Culler, D.E.: SPINS: security protocols for sensor networks. Wirel. Netw. **8**, 521–534 (2002)

19. Liu, A., Ning, P.: TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In: Information Processing in Sensor Networks, 2008. IPSN '08. International Conference, pp. 245–256 (2008)
20. Gupta, V., Millard, M., Fung, S., Zhu, Y., Gura, N., Eberle, H., Shantz, S. C.: Sizzle: a standards-based end-to-end security architecture for the embedded internet. In: Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference, pp. 247–256 (8–12 March 2005)
21. Eschenauer, L., Gligor, V.D.: A key-management scheme for distributed sensor networks. In: CCS '02: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 41–47. ACM, New York, NY, USA (2002)
22. Chan, H., Perrig, A., Song, D.: Random key predistribution schemes for sensor networks. In: SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy, p. 197. IEEE Computer Society, Washington, DC, USA (2003)
23. Liu, D., Ning, P.: Establishing pairwise keys in distributed sensor networks. In: CCS '03: Proceedings of the 10th ACM Conference on Computer and Communications Security, pp. 52–61. ACM, New York, NY, USA (2003)
24. ZigBee Alliance, Zigbee specification v1.0 (2004)
25. Law, L., Menezes, A., Qu, M., Solinas, J., Vanstone, S.: An efficient protocol for authenticated key agreement. Des. Codes Cryptogr. **28**(2), 119–134 (2003)
26. Clark, J., Jacob, J.: A survey of authentication protocol literature. http://cs.york.ac.uk/jac/papers/drareview.ps.gz (1997)
27. OpenSSL website. http://www.openssl.org/
28. Law, Y.W., Doumen, J., Hartel, P.: Survey and benchmark of block ciphers for wireless sensor networks. ACM Trans. Sen. Netw **2**, 65–93 (2006)
29. Levis, P., Gay, D., Handziski, V., Hauer, J.-H., Greenstein, B., Turon, M., Hui, J., Klues, K., Sharp, C., Szewczyk, R., Polastre, J., Buonadonna, P., Nachman, L., Tolle, G., Culler, D., Wolisz, A.: Berlin, Technische Universität, Crossbow Inc., Arched Rock Corporation, T2: A second generation OS for embedded sensor networks. Tech. rep. (2005)
30. Realtime Technologies LTD, Shimmer—wireless sensor platform. http://www.shimmer-research.com/wp-content/uploads/2010/08/Shimmer-2R-Capabilities-Overview.pdf (2010)
31. Dishongh, T.J., McGrath, M.: Wireless sensor networks for healthcare applications, chap. 7. Artech House Publishers (2010)
32. Burrows, M., Needham, R.: A logic of authentication. ACM Trans. Comput. Syst. **8**, 18–36 (1990)
33. De Decker, B., Piessens, F.: Cryptolog: a theorem prover for cryptographic protocols. In; DIMACS Workshop on Design and Formal Verification of Security Protocols, Rutgers University, New Jersey, USA, September (1997)
34. Thouvenin, R.: Implementing and Evaluating the Dynamic Manet On-demand Protocol in Wireless Sensor Networks. Master's thesis, University of Aarhaus (2007)
35. Ganesan, D., Krishnamachari, B., Woo, A., Culler, D., Estrin, D., Wicker, S.: Complex behavior at scale: an experimental study of low-power wireless sensor networks. UCLA Computer Science Technical Report UCLA/CSD-TR 02-0013 (2002)
36. NIST (National Institute of Standards and Technology), Special publication 800-57 part1. Tech. rep. (2007)

Author Biographies

Victor Cionca received his B.Eng. degree from the Computer Science department of the Technical University of Cluj-Napoca, Romania. He is studying towards a Ph.D. degree at the University of Limerick, designing resource-efficient security architectures for wireless sensor networks. He did a 6 month internship with the Digital Health Group of Intel Ireland, developing firmware for the Shimmer platform. His research interests are communication and security protocols for wireless sensor networks, operating systems and cryptography.

Thomas Newe born in Ireland in 1967. He received his Honours Degree in Computer Engineering in 1991. Following this he worked as a software engineer where his main area of expertise was embedded system design and programming. He was awarded his PhD in 2003 from the University of Limerick for work on Formal Verification Logics for use in security protocol design. He is currently a Lecturer in the

Department on Electronic and Computer Engineering in the University of Limerick and a member of the Optical Fibre Sensors Research Centre (http://www.ofsrc.ul.ie). He has graduated a number of PhD students in the broad area of network security. His students are funded from a variety of sources including: SFI, IRCSET and industrially funded. His active areas of research include: wireless sensor networks, network and data security, embedded systems and formal verification methods.

Vasile Teodor Dădârlat (M'05) received the Diploma (M.Sc.) degree in computer science from the University "Politehnica" of Bucharest, Romania, in 1980, and the Ph.D. degree in computer science from the Technical University of Cluj Napoca, Romania, in 1995. He holds the Chair of Computer Networks and Communications in the Department of Computer Science, Technical University of Cluj Napoca, Romania. His current research interests lie in communications protocols and security for hybrid networks, as well as modeling and performance evaluation for network protocols and design of integrated circuits. In 2011 he served as Technical Programme chair for the 10th International Symposium on Parallel and Distributed Computing.