



UbiSec&Sens
FP6-2004-IST-4
Contract no.: 26820



UbiSec&Sens

Deliverable <D2.4⁺>

(Specification and Simulation of key pre-distribution schemes) (Extension of the Deliverable D2.4)

Editor:	Claude Castelluccia, INRIA
Deliverable nature:	<P>
Dissemination level: (Confidentiality)	<PU>
Contractual delivery date:	30.06.08
Actual delivery date:	18.06.08
Members of the consortium including te commission services	
Version:	1.0 ⁺
Total number of pages:	59
Keywords:	key pre-distribution (KPD), KPD for concealed data aggregation, KPD for multi-stage WSN.

Abstract

Please note that this Deliverable D2.4⁺ is an extension Deliverable to D2.4. Compared to D2.4, this Deliverable includes the evaluation results of **TAUK** key predistribution scheme with i) physical testbed experiments and ii) with AVRORA simulations. The purpose of this Deliverable is to present our activities on the specification and simulation of key predistribution schemes.

Key management in WSN has generated considerable work in the last years and many key establishment protocols have been proposed. Instead of developing yet another protocol, we have decided to focus our efforts on two topics that have not been extensively studied in the past, but that we consider very important, namely keying for convergecast traffic and key pre-distribution protocols for multi-phase WSN.

This document is composed of 6 main sections. The first section is an introduction of key predistribution protocols in WSN. Section 2 presents existing key predistribution schemes. Section 3 summarizes the motivation of our activities and our main contributions, namely **TAUK** (a new key establishment protocol for convergecast traffic) and **RoK** (a new key pre-distribution protocol for multi-stage WSN). Section 4 and Section 5 details our new protocols, namely **TAUK** and **RoK**. Finally, Section 6 describes the implementation of the **TAUK** and **RoK** protocols under TinyOS.

Disclaimer

This document contains material, which is the copyright of certain UbiSec&Sens consortium parties, and may not be reproduced or copied without permission.

In case of Public (PU):

All UbiSec&Sens consortium parties have agreed to full publication of this document.

In case of Restricted to Programme (PP):

All UbiSec&Sens consortium parties have agreed to make this document available on request to other framework programme participants.

In case of Restricted to Group (RE):

All UbiSec&Sens consortium parties have agreed to full publication of this document. However this document is written for being used by other projects as a survey report.

In case of Consortium confidential (CO):

The information contained in this document is the proprietary confidential information of the UbiSec&Sens consortium and may not be disclosed except in accordance with the consortium agreement.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the UbiSec&Sens consortium as a whole, nor a certain party of the UbiSec&Sens consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.

Impressum

[Full project title] Ubiquitous Sensing and Security in the European Homeland

[Short project title] UbiSec&Sens

[Number and title of work-package] WP2 - Network Security

[Document title] Specification and Simulation of Advanced Concealed Data Aggregation

[Editor: Claude Catelluccia, INRIA]

[Work-package leader: Claude Castelluccia, INRIA]

[Estimation of PM spent on the Deliverable] 11 PMs

Copyright notice

©2007 Participants in project UbiSec&Sens

Optionally list of organisations jointly holding the Copyright on this document

Executive summary

The overall goal of work package WP2 “Network Security” is to design and implement security protocols and mechanisms for WSN networks. This work-package is composed of 4 different tasks: authentication, concealed data aggregation, key predistribution and provable secure routing. The purpose of Deliverable D2.4 is to present the project activities on the specification, simulation and implementation of key predistribution schemes.

Key management in WSN has generated considerable work in the last years and many key establishment protocols have been proposed. Instead of developing yet another protocol, we have decided to focus our efforts on two topics that have not been extensively studied in the past, but that we consider very important, namely keying for convergecast traffic and key pre-distribution protocols for multi-phase WSN.

This document describes two new key pre-distribution protocols, namely **TAUK** (a new key establishment protocol for convergecast traffic) and **RoK** (a new key pre-distribution protocol for multi-stage WSN). **TAUK** is a solution for end-to-end encryption of converge-cast traffic with a simple key pre-distribution scheme causing additional data only logarithmic in the number of sensors. The scheme is robust with respect to unreliable channels, exhausted nodes and routing flexibility. It supports refreshing the keys at the nodes, which has so far been fully neglected. This deliverable also describes **RoK**, a new key pre-distribution scheme that allows sensors of different generations, i.e. deployed at different times, to establish secure channels. We show that a network that is temporarily attacked automatically self-heals, i.e. recovers its initial state when the attack stops. In contrast, with existing schemes, an attacker that corrupts a certain amount of nodes compromises a given fraction of the total number of secure channels. This ratio remains constant until the end of the network, even if the attacker stops its action.

List of authors

Company	Author
<INRIA>	<Claude Castelluccia, Aurélien Francillon>
<NEC>	<Dirk Westhoff, Frederik Armknecht, Osman Ugus>
<BUTE>	<Levente Buttyan>
<Bochum>	<Mark Manulis, Axel Poschman>

Contents

Executive summary	3
List of authors	4
Abbreviations	9
1 Introduction	10
2 Key Pre-Distribution: Deterministic and Probabilistic Approaches	10
Public key based Key Establishment	11
Deterministic Key Pre-Distribution	14
Probabilistic Key Pre-Distribution	14
The basic random key pre-distribution scheme	15
q -composite random key pre-distribution	16
Multipath key reinforcement	17
Random key pre-distribution combined with threshold cryptography	17
3 Summary of our activities and Contributions	18
Keying for Convergecast Traffic	19
Key Pre-distribution Protocol for Multi-Phase Wireless Sensor Networks	20
4 Keying for Convergecast Traffic	21
Bihomomorphic Encryption Transformation	21
TAUK and Bihomomorphic Encryption in a Nutshell	21
Topology Aware Unique Keying and bihomomorphic Encryption	22
Basic idea	22
Initialization phase	23
Aggregation phase	24
Concrete instantiations	26
Encryption scheme	26
Key refreshment	26
Robustness in respect to silent nodes	28
Evaluation	29
Energy consumption	30
Computational effort	30
Transmission costs	30
Security Analysis	33
Security upshot	33
Routing Flexibility	33
5 RoK: A Robust Key Pre-distribution Protocol for Multi-Phase Wireless Sensor Networks	34
A Robust Key Pre-distribution Protocol for Multi-Phase Wireless Sensor Networks	34
Overview	34
Key Pools Generation	34
Key rings assignment	35
Establishing a secure channel	36
Example	36
Security Evaluation	37
Objectives	37
Evaluation by Simulation	38
Simulation Set-up	38
Attacker model and strategy	38
Simulation results	39

Analytical Evaluation	41
R_{active} computation	41
Some numerical results	43
Discussions	44
Counterfeiting protection	44
Controlling the Security of a WSN	45
Comparison with Related Work	45
6 Implementations	47
Implementation details of TAUK	47
Implemented Node addressing model	47
Integration of the proposed addressing model with existing addressing models	48
Data representation	48
Software components	50
Implementation results and evaluation	54
Simulation	55
RoK Protocol Implementation Details	56
Software components	56
Interfaces	58

List of Figures

1	Estimated energy consumption of the EG and the ECDH protocol on a Micaz mote depending of the amount of neighbours. The settings for all three figures are: $nodes = 400$, $kp = 10000$, and $keyring = 250$	12
2	Estimated energy consumption of the EG and the ECDH protocol on a Micaz mote depending on the size of the keyring. The settings for all figures are: $nodes = 400$, $kp = 20000$, and $n_N = 40$	13
3	Estimated energy consumption of the improved EG and the ECDH protocol on a Micaz mote depending on the size of the keyring. The settings for all figures are: $nodes = 400$, $kp = 20000$, and $n_N = 40$	13
4	Distribution phase.	24
5	Aggregation phase at time frame t	25
6	Refreshment phase.	27
7	Comparison of the maximum transmission sizes in respect to the aggregation level for a $h = d = 3$ tree.	32
8	Comparison of the maximum transmission sizes in respect to the aggregation level for a $h = 5$ and $d = 4$ tree.	32
9	Key establishment: A and B share the keys t_1 and t_2	37
10	R_{active} . Adversary compromises nodes at constant rate (constant attacker)	39
11	R_{total} . Adversary compromises nodes at constant rate (constant attacker)	39
12	R_{active} . Adversary compromises from generation 5 to generation 15 (temporary attacker)	40
13	R_{total} . Adversary compromises from generation 5 to generation 15 (temporary attacker)	40
14	Average number of nodes to re-deploy every generation. For every generation is reported the expected number of nodes died in that generation.	43
15	Comparison analysis and simulation results	44
16	Security of RoK with different values of Gw . Ratio between Total-compromised and Total-established channels, for the whole lifetime of the network.	45
17	Security of RoK with different values of Gw . Ratio between Active-compromised and Active channels.	46
18	Node addressing in a tree-shaped network of 6 nodes, whereby both the degree and the height of the tree is 2	47
19	Message flow in TAUK	50
20	Flow chart for the implementation of the TAUK key-distribution	51
21	Graphical representation of the TAUK key-distribution module	52
22	Graphical representation of the radio module for TAUK	53
23	Layout of the sensor testbed comprised of 14 sensor nodes	54
24	Tree-shaped network corresponding to our office setting	54
25	Topology of the simulated network consisting of 84 sensor nodes and the sink	56
26	Graphical representation of the Robust key distribution module	57

List of Tables

1	Used values for estimation	11
2	Comparison of the maximum node transmission size.	31
3	Summary of Notation	35
4	TAUK key distribution latencies in a real world office setting with 14 nodes with respect to different key sizes.	55
5	TAUK key distribution latencies with AVRORA simulation tool with respect to different network sizes.	56

Abbreviations

List of Abbreviations:

CDA	concealed data aggregation
CMT	The Castelluccia-Mykletun-Tsudik Encryption scheme
CRT	chinese remainder theorem
ECC	elliptic curve cryptography
ECDLP	elliptic curve discrete log problem
HbH	Hop-by-Hop Encryption
ID	Identifier
KPD	key pre-distribution scheme
MANET	Mobile Ad Hoc Network
ME	Multiple Encryption scheme (an extension of the CMT scheme)
<i>RoK</i>	Robust Key Predistribution protocol (our new proposal)
<i>PH</i>	privacy homomorphic encryption transformation
<i>PH_s</i>	symmetric privacy homomorphic encryption transformation
<i>PH_a</i>	asymmetric privacy homomorphic encryption transformation
TAGK	topology aware group keying
TAUK	topology aware unique keying
WSN	wireless sensor network

1 Introduction

The purpose of this deliverable is to present our current activities on the specification and simulation of key predistribution schemes. This is the preliminary version of the deliverable D2.4, that is due in Month 24 i.e. in 6 months.

Key management in WSN has generated considerable work in the last years and many key establishment protocols have been proposed. Instead of developing yet another protocol, we have decided to focus our efforts on two topics that have not been extensively studied, but that we consider very important, namely keying for convergecast traffic and key pre-distribution protocols for multi-phase WSN.

This document is composed of 4 main sections. The first section presents existing key predistribution schemes. Section 2 summarizes the motivation of our activities and our main contributions, namely **TAUK** (a new key establishment protocol for convergecast traffic) and **RoK** (a new key pre-distribution protocol for multi-stage WSN).

Section 3 describes **TAUK**, a new key pre-distribution scheme for converge-cast traffic. The use of converge-cast traffic and in-network processing to minimize the amount of transmitted data is a frequently used approach to increase the lifetime of a wireless sensor network (WSN). Consequently when aiming at security for WSNs, one has to focus primarily on protecting this type of traffic. Some recent proposals support the encryption of converge-cast traffic with in-network processing. However, they either require the transmission of the sensors' IDs, creating additional data overhead linear in the number of sensors, or require an elaborate key pre-distribution mechanism. We propose a solution for end-to-end encryption of converge-cast traffic with a simple key pre-distribution scheme causing additional data only logarithmic in the number of sensors. The scheme is robust with respect to unreliable channels, exhausted nodes and routing flexibility. It supports refreshing the keys at the nodes, which has so far been fully neglected.

Section 4 describes **RoK**, a new key pre-distribution scheme that allows sensors of different generations, i.e. deployed at different times, to establish secure channels. In the proposed scheme, the predistributed keys have limited lifetimes and are refreshed periodically. We show that a network that is temporarily attacked automatically self-heals, i.e. recovers its initial state when the attack stops. In contrast, with existing schemes, an attacker that corrupts a certain amount of nodes compromises a given fraction of the total number of secure channels. This ratio remains constant until the end of the network, even if the attacker stops its action. Furthermore, we show that a *RoK* based network that is constantly attacked is much less affected than a network that uses existing key pre-distribution protocols. Our analysis and simulation results demonstrate that our *RoK* scheme outperforms the *RKP* scheme proposed by Eschenauer and Gligor [7]. The number of active compromised nodes can be reduced by a factor of 10. The security of our proposal is based on the assumption that it takes time for an adversary to physically compromise sensors and get their keys. Since in our proposal, the key vulnerability period, i.e. the time that attacker has to corrupt useful keys, is reduced, security is improved significantly. We show that it is possible to control the security of wireless sensor networks by limiting the lifetime of sensors and by deploying new ones periodically. The analytical model, derived in this document, can be used to compute, according to the security objectives and the attacker model, the network parameters, i.e. the sensor lifetime and re-deployment period.

2 Key Pre-Distribution: Deterministic and Probabilistic Approaches

The key management in wireless sensor networks is usually described by the means of a *key pre-distribution* process which requires the upload of some usually secret information into the sensor nodes prior to their deployment within the network. This secret information may either (i) represent a concrete secret key, or (ii) be some auxiliary information which will help nodes to derive the actual secret key through interaction after being deployed in the network. Most of the currently available symmetric key pre-distribution approaches fall into one of the following categories: *deterministic* or *probabilistic*.

However, in the following section we will briefly emphasize, why a standard hybrid approach, i.e. using public key cryptography to establish a private key for further communication, is not an option for most of the foreseen scenarios in WSN.

clock frequency	7.37MHz
supply voltage	3 V
I_{CPU}	12 mA
$E_{CPU,s}$	36 W
$E_{CPU,c}$	4.88nJ
I_{sen}	17.4 mA
I_{rec}	18.8 mA
E_{sen}	209μ Jpb
E_{rec}	226μ Jpb

Table 1: Used values for estimation

Public key based Key Establishment

A natural approach for key-establishing would be to use a standard *hybrid* approach, i.e. to use an asymmetric or public key algorithm to establish a symmetric (or private) key for further encryption, integrity checks, and authentication. This is especially interesting, because the communication overhead with a hybrid approach is very small compared to recently proposed key establishment approaches such as [7]. On the other hand, asymmetric algorithms such as RSA or ECC require huge computational effort.

In the following we will estimate the energy consumption of two key-establishing protocols, namely the basic probabilistic or random protocol [7](EG) and the Diffie-Hellmann protocol based on elliptic curves (ECDH). We assume Micaz motes clocked at 7.37 MHz to be used. At a supply voltage of 3 Volt the Micaz mote draws a current of $I_{CPU} = 12$ mA, when the CPU is operating. From this we can calculate the energy consumption per second $E_{CPU,s} = 12mA * 3V = 36W$ and per clock cycle $E_{CPU,c} = \frac{E_{CPU,s}}{7.37MHz} = 4.88nJ$. Furthermore, we assume its radio device (Chipcon's CC2420) to be operated with 0 dB. From [28] we see that the current consumption for sending and receiving is $I_{sen} = 17.4$ mA and $I_{rec} = 18.8$ mA, respectively. The radio device is compliant with the 802.15.4 standard (Zigbee) and has an effective data rate of 250 kbps. Therefore we can calculate the energy consumption for sending and receiving one bit by $E_{sen} = \frac{17.4mA*3V}{250kbps} = 209\mu Jpb$ and $E_{rec} = \frac{18.8mA*3V}{250kbps} = 226\mu Jpb$, respectively. Table 1 summarizes all these values. In TinyOS by default each message consists of 10 bytes header and 29 bytes payload. ZigBee specifies a maximum packet length of 128 Bytes. Therefore the maximum payload is theoretically 118 Bytes (944 bits). If more than 944 bits are transmitted, the message has to be split into many chunks, such that each of them fits into one packet. Therefore, the overhead for a sent message of any protocol is $80 * \lceil \frac{x}{944} \rceil$, where x denotes the amount of bits.

The basic random key-establishment protocol (EG) is described in detail in 2. Basically it consists of three phases: the *initialization phase*, *direct key establishment phase*, and the *path key establishment phase*. The first phase is performed prior to deployment, hence it is not part of our investigation. For the sake of simplification we assume that all nodes share a key, hence the *path key establishment phase* is not required. The second phase consists of two steps: During the *shared key discovery phase* all nodes broadcast a list of short identifiers of all keys they have in their *key ring*. The length of the short key identifier l_{key} is depending on the size of the *key pool* (kp) and can be derived by the following equation: $l_{key} = \lceil \log_2(kp) \rceil$. Hence, the first broadcast message consists of $kp * l_{key} + 80 * \lceil \frac{kp * l_{key}}{944} \rceil$ bits. Since a node will also receives this message from all its n_N neighbouring nodes, each node will receive $n_N * (kp * l_{key} + 80 * \lceil \frac{kp * l_{key}}{944} \rceil)$ bits.

If two nodes share a common key –which is assumed for the sake of simplification– a challenge-response protocol is invoked. In the following we use the mutual authentication protocol Π_{1-1} , which is also described in detail in the UbiSec&Sens project's Deliverable D2.2. Since Π_{1-1} is a mutual authentication protocol it is only invoked $\lceil \frac{n_N}{2} \rceil$ times. In the Π_{1-1} protocol the first of three sent messages consists of a 64-bit challenge c_s and the short identifier of the common key, hence $80 + 64 + l_{key}$ bits are transmitted and received $\lceil \frac{n_N}{2} \rceil$ times. The second message consists of two identifiers ID_s and ID_r for sender and receiver, two 64 bit challenges c_s and c_r , and the output of a keyed pseudo-random function μ_r . The node identifier have $l_{ID} = \lceil \log_2(n) \rceil$, where n denotes the amount of nodes in the WSN. The last message consists of ID_s , c_r , and μ_s .

The receiver of message one has to compute μ_r and the receiver of message two has to verify μ_r and to compute μ_s , hence the keyed pseudo random function is invoked twice. We assume the AES to be used

as the keyed pseudo random function. The AES can be implemented for the Micaz motes in software to perform one encryption in 14.58 ms, hence the energy consumption for one generation of μ_r or μ_s is $E_{enc} = 14.48ms * E_{CPU,s} = 1.57mJ$.

In the EG protocol all together $E_{EG_{sen}} = E_{sen} * (80 * \lceil \frac{keyring * l_{key}}{944} + keyring * l_{key} + \frac{n_N}{2} * (80 * 3 + l_{key} + 64 * 4 + l_{ID} * 3 + 2 * 64) \rceil)$ bits are sent and $E_{EG_{rec}} = E_{rec} * \frac{n_N}{2} * (80 * 5 + l_{key} * (keyring * 2 + 1) + 4 * 64 + 3 * l_{ID} + 2 * 64)$ bits are received. The computational effort $E_{EG_{CPU}} = 3 * n_N * E_{ENC}$ is very small, because only three times the AES is processed. Note that in our estimations we neglected the computational effort to find a common key in the lists of short key identifiers.

For comparison we chose the Diffie-Hellmann protocol based on elliptic curves (ECDH). The ECDH protocol consists of two steps: first, each nodes broadcasts its public key, and upon receiving of the public key of all the n_N neighbours, subsequently the common key for all neighbours is calculated. The public key PK consists of two coordinates (x,y) of a point on the elliptic curve that are derived by the multiplication of a publicly known base point G with a secret integer k , i.e. $PK = k * G$. When a node S_1 sends its public key $PK_1 = k_1 * G$ to a node S_2 and also receives S_2 's public key $PK_2 = k_2 * G$ the common key $K_{1,2}$ is derived as follows: $K_{1,2} = PK_1 * k_2 = k_1 * k_2 * G = k_1 * PK_2$. That is, only one point multiplication is required to establish a common key between two nodes. Note that an attacker can easily insert malicious nodes into the network if no authenticated key-agreement protocol such as ECMQV is used. However, if we assume a *boot-strapping phase* of the WSN, where we can guarantee that no attacker is in WSN, ECDH is sufficient. For example, such a boot-strapping phase could happen on a secure site prior to deployment.

We chose the *secp160r1* elliptic curve as standardized by the SECG2 consortium, which has a security level that is equivalent to a symmetric algorithm with 80 bits. Since the public key consists of the (x,y)-pair with 160 bits each, in our case the first message of the ECDH protocol consists of 80 + 320 bits. Each node also receives this message n_N times. Hence, the ECDH protocol has an energy consumption of $E_{EC_{sen}} = E_{sen} * (80 + 320)$ for sending and $E_{EC_{rec}} = E_{rec} * n_N * (80 + 320)$.

The point multiplication is very energy demanding on a Micaz mote. The fastest known implementation of a point multiplication for the *secp160r1* elliptic curve on 8-bit micro processors such as Micaz motes is published by SUN [27] and requires 0.81 s. During the UbiSec&Sens project we implemented the fastest known modular multiplication [26], which claims for about 77% of a point multiplication on 8-bit micro processors such as Micaz motes. Together with the figures from SUN, we estimated an optimal point multiplication to still require 0.76 s. Unfortunately, the point multiplication has to be performed for all neighbours once. This leads to a total estimated energy consumption of the ECDH protocol of $E_{EC_{CPU}} = 0.76 * n_N * E_{CPU}$.

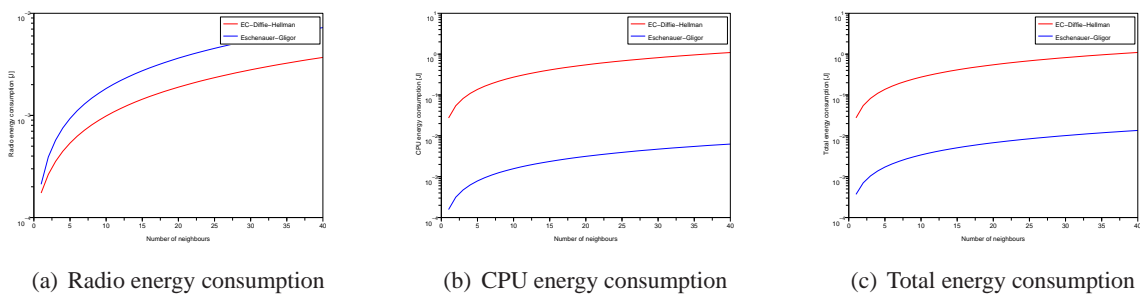


Figure 1: Estimated energy consumption of the EG and the ECDH protocol on a Micaz mote depending of the amount of neighbours. The settings for all three figures are: $nodes = 400$, $kp = 10000$, and $keyring = 250$.

Figure 1 compares the estimated energy consumption of the EG and the ECDH protocol on a Micaz mote depending of the amount of neighbours. The settings for all three figures are: $nodes = 400$, $kp = 10000$, and $keyring = 250$. Figure 1(a) shows that the EG protocol consumes nearly double the energy for radio communication compared to the ECDH and that this difference increases with an increasing amount of neighbours. However, as we can see from Figure 1(b) the ECDH protocol consumes nearly two orders of magnitude higher energy for computation compared to the EG protocol and this difference also increases with an increasing amount of neighbours. Therefore, the total energy consumption of the ECDH protocol is always about two orders of magnitudes higher compared to the EG protocol, independent of the amount of neighbours.

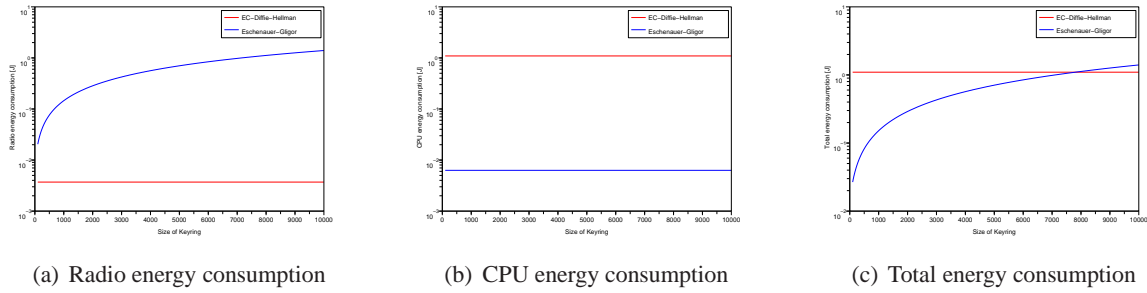


Figure 2: Estimated energy consumption of the EG and the ECDH protocol on a Micaz mote depending on the size of the keyring. The settings for all figures are: $nodes = 400$, $kp = 20000$, and $n_N = 40$.

Figure 2 compares the estimated energy consumption of the EG and the ECDH protocol on a Micaz mote depending on the size of the keyring. The settings for all figures are: $nodes = 400$, $kp = 20000$, and $n_N = 40$. From Figure 2(a) we see that the radio energy consumption of the ECDH protocol stays constant independent of the keyring size, while the energy consumption of the EG protocol increases linearly. From Figure 2(b) we see that the energy consumption of both protocols stays constant, but ECDH consumes more than two orders of magnitude more energy than EG. Figure 2(c) reveals that the EG protocol requires less energy compared to the ECDH protocol as long as the keyring size is below about 7800. With larger keyring sizes ECDH is more energy efficient than EG.

However, the above used, straight-forward implementation of the basic probabilistic scheme (EG) is not optimal. Assume that the key ring of each node is comprised of keys $K_{ID,x}$ that are drawn from the key pool not completely randomly, but with the following algorithm: $K_{ID,x} = hash(ID|r|x)modkp$, where ID denotes the identifier of a node, x denotes the index of the key ring, $hash()$ denotes an arbitrary hash function, r denotes a random 64-bit value, and kp denotes the size of the key pool. Then, each node only has to broadcast its ID and its r , independent of the key ring size. Upon receiving the ID and the r of the neighbouring nodes, each node calculates the key ring index and compares it with its own key ring indices.

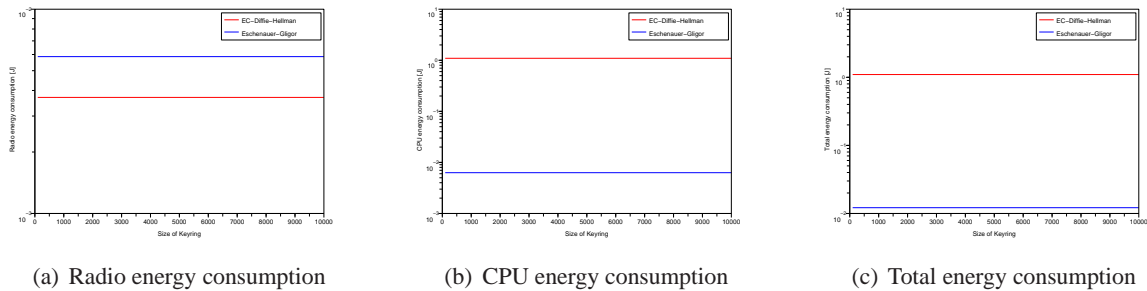


Figure 3: Estimated energy consumption of the improved EG and the ECDH protocol on a Micaz mote depending on the size of the keyring. The settings for all figures are: $nodes = 400$, $kp = 20000$, and $n_N = 40$.

Figure 3 compares the estimated energy consumption of the improved EG and the ECDH protocol on a Micaz mote depending on the size of the keyring. The settings for all figures are: $nodes = 400$, $kp = 20000$, and $n_N = 40$. Since the energy consumption of the improved EG protocol –like the energy consumption of the ECDH protocol– is now independent of the key ring size, both energy consumptions stay constant. Similar to the basic EG protocol, the radio energy consumption of the improved EG protocol is slightly higher compared to the ECDH protocol (see Figure 3(a)). Also the computation energy consumption of the improved EG protocol is two orders of magnitude smaller compared to the ECDH protocol (see Figure 3(b)). Figure 3(c) clearly depicts that altogether the improved EG protocol always requires two orders of magnitude less energy compared to the ECDH protocol.

In most scenarios a keyring size of a few hundred keys is sufficient. Therefore, the EG protocol or any of its derivatives that are discussed in the following section is much more suited for key-establishment than the ECDH

protocol.

Deterministic Key Pre-Distribution

The main advantage of deterministic key pre-distribution solutions is that deterministic processes are used to design the key-pool and the key-chains between sensor nodes to provide better key connectivity. The low-resiliency against node corruptions is usually one of the major drawbacks of such schemes.

- **BROSK** The *BROadcast Session Key* (BROSK) negotiation protocol was proposed by Lai et al. in 2002 [20]. BROSK requires that a master key is shared between all sensors in the network, hence, it offers only low resiliency. Since BROSK requires low computational, low communication, and low storage costs it offers good scalability. Unfortunately, BROSK supports only the establishment of link keys, hence, neither path nor cluster keys can be established with BROSK.
- **CDTKeying** The *Combinatorial Design Theory* based pairwise key pre-distribution scheme (CDTKeying) was proposed by Camtepe and Yener in 2004 [18]. It is based on block designing techniques in combinatorial design theory. The underlying symmetric and generalised quadrangle design techniques require the parameter n to be a prime number. This means that CDTKeying does not support all network sizes for a fixed key-chain size. However, CDTKeying offers both good scalability and resiliency. CDTKeying requires some auxiliary information to be pre-distributed. It supports the establishment of link keys, but unfortunately neither path nor cluster keys can be established. Furthermore, the computational and the communication costs are medium and the storage cost are high.
- **IOS** The *ID based One-way function scheme* (IOS) was proposed by Lee and Stinson in 2004 [21]. It assumes that the WSN can be modeled as an r -regular graph which can be decomposed into star-like subgraphs. Each sensor S_i receives an individual key $\mathbb{K}i$ and link keys $\mathbb{K}i, j = H(\mathbb{K}j|id_i)$ for each of its neighbours S_j of the subgraph. IOS does not require a master key, but individual and link keys to be pre-distributed. It supports the establishment of link keys, but unfortunately it does neither support the establishment of path keys nor the establishment of cluster keys. IOS offers both good scalability and resiliency at medium computational, medium communication, and high storage costs.
- **DMBS** The *deterministic multiple space Blom's* scheme (DMBS) was also proposed by Lee and Stinson in 2004 [21]. It uses a strongly r -regular graph to extend the standard Blom's scheme. DMBS requires an individual key and auxiliary information to be pre-distributed and supports the establishment of link keys, but it does neither support the establishment of path nor cluster keys. It offers good scalability and resiliency at medium computational, medium communication, and high storage costs. DMBS provides better scalability for the cost of decreased resiliency, compared to IOS.

Probabilistic Key Pre-Distribution

The general idea of probabilistic or *random* key pre-distribution can be traced back to the following variant of the birthday paradox [24]: Given a set S of k elements, we randomly choose two subsets S_1 and S_2 of m_1 and m_2 elements, respectively, from S . The probability of $S_1 \cap S_2 \neq \emptyset$ is

$$\Pr\{S_1 \cap S_2 \neq \emptyset\} = 1 - \frac{(k - m_1)!(k - m_2)!}{k!(k - m_1 - m_2)!} \quad (1)$$

For illustration purposes, we plotted the value of expression (1) in Figure ??, where we set $k = 100$ and $m_1 = m_2 = m$. As we can see, the probability of the two subsets intersecting increases rapidly with m , and it reaches $\frac{1}{2}$ when m is around 8. In general, it can be shown that the value of (1) will be close to $\frac{1}{2}$ when k is large and m_1 and m_2 are both close to \sqrt{k} . The paradox is that we would not expect such a high probability of collision when the size of the selected subsets is only the square root of the original set.

[figure]

This result can be used in key pre-distribution to considerably decrease the memory requirements imposed on sensor nodes while still maintaining a rather high probability of any two nodes sharing a common key. For this reason, each node is pre-loaded with a random subset of keys selected from a large key pool. Two nodes

that have a common key in their subsets are able to communicate securely using the shared key. The probability of this event will be rather high when the number of selected keys is in the order of the square root of the pool size. Thus, we expect that large networks can be supported with a rather limited size memory in sensor nodes.

Below, we elaborate on this idea in more detail. First, we describe a basic scheme and some of its straight-forward improvements. Then, we describe an approach to combine random key pre-distribution with threshold cryptography in order to increase the resistance of the scheme to node capture attacks.

The basic random key pre-distribution scheme

The basic random key pre-distribution scheme proposed in [7] works in three phases. In the *initialization phase*, a large pool S of unique cryptographic keys is randomly generated, and then, for each node, m keys are selected randomly from S and pre-loaded into the node. This set of m keys is called the *key ring* of the node. The number k of keys in S is chosen in such a way that any two nodes will have a common key in their key rings with a certain probability p (see analysis below).

After the sensors are deployed, the *direct key establishment phase* is performed. In this phase, the nodes first find out with which of their neighbors they share a common key. Such key discovery can be implemented by assigning short identifiers to each key in S before deployment and by having each node broadcast the set of identifiers that correspond to the keys in the node's key ring. Two neighboring nodes that discover that they share a common key can then verify that they both really possess that key by executing a challenge-response protocol. The shared key is then used to protect the link between the two nodes.

Some pairs of neighboring nodes may not have a common key in their key rings, and therefore may not be able to setup a secure link in the direct key establishment phase. In order to remedy this situation, a *path key establishment phase* is performed. In this phase, neighboring nodes that do not share a key initially establish a shared key through a path of intermediate nodes where each link of the path is already secured in the direct key establishment phase. This will work only if the graph, which consists of the nodes (as vertices) and the secure links created in the direct key establishment phase (as edges), is connected. As we will see below, this can be achieved with high probability by appropriately choosing the parameters of the scheme.

Setting the parameters: We use results from random graph theory to set the parameters of the basic scheme. Although sensor networks are not random graphs, as nodes cannot have communication links with most of the other nodes in the network, using the random graph metaphor is still useful to give us an idea of the order of magnitude of the various parameters.

We know from random graph theory [25] that in order for a random graph to be connected with high probability, the expected degree of the vertices should exceed a certain threshold. More precisely, in order for a random graph to be connected with probability c (e.g., $c = 0.9999$), the expected degree d of the vertices should be:

$$d = \frac{n-1}{n}(\ln(n) - \ln(-\ln(c))) \quad (2)$$

where n is the number of vertices in the graph.

In our case, the edges of the graph correspond to the secure links created between neighboring nodes in the direct key establishment phase. Recall that p denotes the probability that two nodes have a common key in their key rings. In addition, for a given density of node deployment, let n' be the expected number of neighbors of a node. Then, in our graph of secured links, the expected node degree is $d = p \cdot n'$. Thus, we obtain that, in order for the basic scheme to work, the following should hold:

$$p = \frac{d}{n'} \quad (3)$$

where d is defined in (2).

Note that, using (1), we can compute p as follows:

$$p = 1 - \frac{((k-m)!)^2}{k!(k-2m)!} \quad (4)$$

Recall that k is the number of keys in the key pool S , and m is the number of keys in the key rings of the nodes. We can use (4) to determine the values of k and m for a given value of p .

Let us consider a numerical example. Let us assume that there are $n = 10,000$ nodes in the network, and the nodes are deployed in such a way that the expected number of neighbors is $n' = 40$. We want the basic scheme to work with probability $c = 0.9999$. Using (2), we can compute that the expected node degree in the graph resulting after the direct key establishment phase should be $d = 18.42$. From this, we obtain $p = 0.46$ using (3). Finally, we can use (4), to determine the values of k and m . We can check, for instance, that for $k = 100,000$ and $m = 250$, (4) evaluates to approximately 0.5, meaning that a key pool size of 100,000 and a key ring size of 250 would be an appropriate choice. Alternatively, we can use (4) to determine k if m is given due to the memory constraints of the sensor nodes. For instance, if the key ring size is limited to $m = 75$ keys due to memory constraints, then we get from (4) that the key pool size should be $k = 10,000$ to obtain a connected graph after the direct key establishment phase with probability 0.9999.

A brief qualitative analysis: We can see that the basic scheme is quite well-adapted to the special design constraints for key establishment schemes in sensor networks. First of all, the parameters of the scheme can be adapted to support the memory constraints of the sensor nodes. In addition, setting up pairwise keys does not need any intensive computations. Indeed, when the nodes have a common key in their key rings, that common key becomes the shared pairwise key, and no further processing is needed, apart from a simple challenge-response protocol to ensure that the nodes actually possess the key. When two nodes do not have a common key in their key rings, they can establish a shared key through intermediate nodes. This requires some additional processing, because the intermediate nodes must decrypt and re-encrypt the key establishment messages sent between the nodes. However, this must be done only once, at the beginning of the operation of the network. In addition, simulation results in [7] show that the length of the path of the intermediaries is limited to a few hops. This also indicates a moderate communication overhead of the scheme.

The basic scheme does not make any assumptions about the network topology apart from assuming that the expected node degree is known *a priori*. Moreover, the scheme supports the post-deployment introduction of new nodes into the network. For this, the new node must be pre-loaded with its own key ring, and no further action is needed. In particular, the nodes already deployed do not need to be updated, and the new node can use the basic mechanisms (direct and path key establishment) to set up secure links with already deployed nodes.

The disadvantage of the basic scheme is that, by compromising sensor nodes, an adversary obtains keys from the key pool, which may be used to secure links between other, non-compromised nodes. Thus, node capture affects the security of non-captured nodes too. One way to mitigate this problem would be to increase the pool size. In that case, however, the size of the key rings should also be increased in order to ensure the same probability of connectivity of the graph resulting from the direct key establishment phase. The problem is that the size of the key ring is limited by the available memory in sensor nodes, and hence it cannot be arbitrarily increased.

Another related disadvantage is that establishing path keys through captured nodes jeopardizes the secrecy of the recently established key. In order to overcome this problem, compromised nodes must be discovered and excluded from the network rapidly, but discovering that a node is compromised is a very difficult problem in itself.

Finally, yet another disadvantage of the basic scheme is that it does not provide node-to-node authentication. This means that a node can establish shared keys with its neighbors, but it does not know exactly who its neighbors are. Node-to-node authentication would be useful in detecting node replication attacks and in identifying and expelling misbehaving nodes.

***q*-composite random key pre-distribution**

One approach to increase the resilience of the basic scheme against node capture attacks is to use *q*-composite random key pre-distribution as proposed in [8]. The *q*-composite scheme differs from the basic scheme in requiring the nodes to have at least *q* common keys in their key rings in order to be able to establish a pairwise key. The pairwise key is then computed as the hash of *all* shared keys.

Essentially, the *q*-composite scheme degenerates into the basic scheme when $q = 1$. Intuitively, when $q > 1$, the probability that two nodes can directly establish a shared key is smaller than the same probability in the basic scheme for the same values of the parameters k and m , because it is less probable to share at least q keys than to share at least one. Thus, in order to maintain the same expected degree of the nodes after the direct key establishment phase (and hence, to ensure secure connectivity), either the size m of the key rings should

be increased, or the size k of the key pool should be decreased. However, neither of the above two options are desirable: in the first case, the memory use of the sensors is increased, whereas in the second case, an increased fraction of the keys in the pool is compromised by capturing the same number of nodes. It is true, however, that the latter effect (increased fraction of compromised keys) is counterbalanced by the fact that now, in order for the adversary to compromise a link, it must compromise *all* the keys that have been hashed together to obtain the link key.

The simulation results in [8] show that the q -composite scheme offers greater resilience against node capture than the basic scheme does only when the number of captured nodes is small, whereas it tends to reveal larger fractions of link keys when large number of nodes have been captured by the adversary. In effect, by requiring q to be greater than 1, we make it harder for the adversary to obtain sufficient information to compromise links at the beginning when only a few nodes have been captured. But once a certain amount of information is collected by capturing more nodes, it becomes more and more easy to compromise further links. In other words, the q -composite scheme increases the entry cost of a node capture attack. This makes sense, as it is reasonable to assume that it is more difficult to capture a large number of nodes than to capture only a few of them.

Multipath key reinforcement

Multipath key reinforcement [8] is a technique to strengthen the security of a link key by establishing it through multiple disjoint paths. It can be applied in conjunction with the basic scheme to greatly improve its resilience against node capture. The trade-off is that establishing link keys through multiple paths results in a higher communication overhead.

The operation of multipath key reinforcement is the following: Let us assume that the direct key establishment phase of the basic scheme is performed, and two neighboring nodes u and v have discovered that they have a common key K in their key rings. Instead of simply using this key as the link key between u and v , the nodes will establish their link key in the following way. Node u identifies a set of j disjoint paths to v in the graph resulting from the direct key establishment phase, and sends j key shares $\kappa_1, \kappa_2, \dots, \kappa_j$ to v such that each key share is sent through a different path. Each key share is protected during transit hop-by-hop, using the keys that are discovered in the direct key establishment phase. Then, both u and v compute the shared link key as $K \oplus \kappa_1 \oplus \dots \oplus \kappa_j$.

The advantage of multipath key reinforcement is that in order to compromise a link key, the adversary needs to compromise at least one key on every path through which the key shares are transmitted. The simulation results in [8] show that extending the basic scheme with multipath key reinforcement enables it to outperform the q -composite scheme, even when the latter is also extended with multipath key reinforcement. The intuitive reason is that in the q -composite scheme, the trade-off for the increased resilience is the reduced size of the key pool, which undermines the effectiveness of multipath key reinforcement by making it easier for the adversary to build up a critically large collection of compromised keys. As opposed to this, when the basic scheme is extended with multipath key reinforcement, the size of the key pool does not need to be decreased. The cost of the improved resilience in this case is an added overhead in path discovery and key establishment traffic.

Note that multipath key reinforcement can also be used to reinforce path keys that are established between nodes that do not have a common key in their key rings. The operation of the mechanism in this case is similar to the one described above, with the difference that the path key is computed as $\kappa_1 \oplus \kappa_2 \oplus \dots \oplus \kappa_j$. This will further improve the security of the schemes.

Random key pre-distribution combined with threshold cryptography

As we have seen above, the main problem of the basic random key pre-distribution scheme is that if a node is captured, then all its keys become known to the adversary, and as these keys might have been chosen from the pool by other, non-captured nodes too, their compromise affects the security of the non-captured nodes. We would like to extend the basic scheme in a way that minimizes the effect of capturing a node on other non-captured nodes. In particular, if some key material is leaked, it should not be directly usable by the adversary to learn the key material of other nodes. A possible approach to achieve this is to extend the basic scheme with principles borrowed from threshold cryptography.

The general idea of using threshold cryptography is that capturing less than a certain number of nodes is not sufficient for the adversary to learn anything useful. In order to compromise the links of non-captured nodes, the number of captured nodes must exceed a threshold. Detailed descriptions of schemes that are based on this idea can be found in [23] and [22]. Here, we briefly present the scheme proposed in [23]; in effect, the scheme proposed in [22] is analogous.

We start with the description of polynomial-based pairwise key pre-distribution, and show how this can be combined with the basic random key pre-distribution scheme later. Let $f(x, y) = \sum_{i,j=0}^t a_{ij}x^i y^j$ be a bivariate t -degree polynomial over a finite field $GF(q)$, where q is a large prime number, such that $f(x, y) = f(y, x)$. Each node is pre-loaded with a polynomial share $f(i, y)$, where i is the ID of the node. Any two nodes i and j can compute a shared key. For this, node i evaluates $f(i, y)$ at point j and obtains $f(i, j)$; similarly, node j evaluates $f(j, y)$ at point i and obtains $f(j, i) = f(i, j)$.

It can be proven that this scheme is unconditionally secure and t -collision resistant. This means that any coalition of at most t compromised nodes knows nothing about the shared keys computed by any pair of non-compromised nodes. In addition, any pair of nodes can establish a shared key, and this incurs no communication overhead (apart from telling the node IDs to each other). The memory requirement of the nodes is $(t+1) \log(q)$, as each node needs to store a t -degree polynomial over $GF(q)$.

This scheme could be applied in sensor networks, but it has some limitations. In particular, it can only tolerate at most t captured nodes, where the value of t is limited by the memory size of the sensor nodes. This means that t is usually small, and thus the larger the sensor network is, the more likely that the adversary can capture more than t nodes.

In order to overcome this problem, we can use the idea of random key pre-distribution; but instead of a pool of keys, now we have a pool of t -degree polynomials. For each sensor node i , we choose a subset of m polynomials from the pool and pre-load into node i the polynomial shares of these m polynomials computed at point i . Two nodes that have polynomial shares of the same polynomial can establish a shared key as described above. It may happen that two nodes that want to establish a shared key have no common polynomials. In this case, they can establish a shared key through a path of intermediate nodes in the same way as path keys are established in the basic random key pre-distribution scheme.

Combining the polynomial-based key pre-distribution scheme with the basic random key pre-distribution scheme combines their advantages and results in a better scheme. In particular, in the combined scheme there is a unique key between each pair of nodes, thus capturing a node does not directly reveal the shared key of any other pair of nodes. In addition, the storage overhead for each node is $m(t+1) \log(q)$, which differs from the storage overhead of the polynomial based key pre-distribution scheme only in a constant factor m . Although it requires slightly more memory in the sensor nodes, the combined scheme has the advantage that it can tolerate the capture of more than t nodes. The reason is that in order to compromise a polynomial, the adversary needs to obtain $t+1$ shares of that polynomial. However, due to the random selection of polynomials, it is very unlikely that $t+1$ randomly captured nodes have all selected the same polynomial from the pool, and thus collectively have $t+1$ shares of the same polynomial.

It must be noted, however, that once a polynomial is compromised, every pair of nodes that used the shares of that compromised polynomial to set up a secure link is affected. This means that after capturing a critically large number of nodes, the security provided by the system starts decreasing abruptly. The advantage of the combined scheme is that it pushes the threshold where the system becomes insecure much higher than in the basic random key pre-distribution scheme and in any of its straightforward extensions (i.e., the q -composite scheme and multipath key reinforcement).

3 Summary of our activities and Contributions

Key management in WSN has generated a lot of attention and work in the last years and many key establishment protocols have been proposed. Instead of developing yet another protocol, we have decided to focus our efforts on two topics that have not been extensively studied, but that we consider very important, namely group keying and key pre-distribution protocols for multi-phase WSN. The motivations and the contributions of these two activities are described in the rest of this section.

Keying for Convergecast Traffic

Only a few KPD schemes have been proposed that support the encryption of convergecast traffic with in-network processing. All the proposed schemes are closely linked to the concrete encryption instantiation and its combination with a suited key pre-distribution scheme (KPD).

Motivation: In [17], the KPD *Topology Aware Group Keying* (TAGK) has been proposed for the usage of a symmetric privacy homomorphic encryption transformation for securing convergecast traffic with in-network processing. TAGK can be classified as a groupwise KPD with keys distributed per "routable" region. The scheme is robust against exhausting nodes and it provides a higher system security compared to single-hop based encryption approaches. However, TAGK is designed to support a symmetric homomorphic encryption transformation which requires the same key for all the encrypting parties like for example the scheme [4]. Therefore, in particular for WSN applications requesting highest system security there is a strong need for conceptual enhancements.

In [3], Castelluccia, Mykletun and Tsudik proposed a key stream generator based CDA scheme which allows to use different symmetric keys per plaintext operand and which is additively homomorphic. By applying this scheme for securing convergecast traffic in WSNs each sensing unit encrypts with a different key. Under the assumption that one-time keys are used the scheme itself is provably secure. In addition, since pairwise keys are used, the proposed scheme provides the highest achievable system security. Also, the required KPD is as simple as possible: keys can be randomly distributed to the nodes and only the sink node needs to store all the keys. Since the storage of keys on the nodes is independent of the final position of the nodes, the KPD reduces to a simple storage of different unique keys before the nodes deployment. Such a simple KPD is nearly perfect for highly self-organizing distributed environments. Also, from the viewpoint of security, a pairwise keying model for convergecast traffic is preferable since it provides a higher system security.

The benefit regarding the overall system security of a KPD supporting CDA with multiple symmetric keys like proposed in [3] comes at the cost of additional overhead. Firstly, the nodes' configuration before node deployment requires a pairwise pairing between each sensor node and the sink node to agree on the shared key. Secondly, although the sink node is considered to be equipped with much more memory than a sensor node, since we are aiming at large scaled sensor networks, the storage of thousands or hundreds of thousands of pairs (ID, key) may turn out to be a serious problem. Thirdly, since we are aiming at security solutions for a highly unreliable medium one cannot ignore the impact of packet loss over the wireless broadcast medium. Revealing per data transmission the key IDs respectively node IDs of all the currently involved nodes becomes mandatory.¹ To fully judge the above requirement, recall that for a static transmission radius the energy consumption for transmission linearly increases with the number of transmitted bits. Since transmission is the pre-dominant energy consuming operation of a sensor node one can roughly say that doubling the data to be transmitted approximately halves the lifetime of a sensor nodes. For example, consider an 4-degree tree shaped WSN of height 4 consisting of $n = \sum_{i=1}^4 4^i = 340$ sensor nodes where each node aggregates and forwards the data coming from its children. Let the sink be at aggregation level 0, its children at aggregation level 1, and so on. Each node has its own ID which has a bit size of $9 = \lceil \log_2(340) \rceil$. That is the nodes at aggregation level 4, where actually no aggregation happens, have to send *in addition* to the ciphertext 9 bits for the ID. At level 3, each node has to transmit up to additional $4 \cdot 9 = 36$ bits, 9 bits for each of its 4 children, plus its own ID, yielding 45 bits which have to be transmitted besides the ciphertext. One sees easily that the amount of data increases, the closer (in terms of hops) a node is to the sink. The nodes at aggregation level 2 have to send up to $4 \cdot 45 + 9 = 189$ bits for all IDs and at aggregation level 1 up to $4 \cdot 189 + 9 = 765$ bits. Let us assume that the sensed values stem from a limited value space of 8 bits size, so that the aggregation of 340 measurements requires $\lceil \log_2(340) + 8 \rceil = 17$ bits. A node at aggregation level 1 has to send $765 + 15 = 782$ bits in total, which is almost 46 times bigger than the size of the ciphertext. Even if we take into account that the transmission overhead on average will be slower in practice as not all nodes might have send some data this example illustrates that a CDA scheme that requires to send the node ID significantly reduces the lifetime of the WSN. Observe that this effect depends only on the size of the WSN and not on the size of the measurements. Therefore, we believe that large scale WSNs require other CDA approaches which work without sending any IDs or sending only few IDs.

Contributions:

¹Alternatively, one could consider an approach where only the IDs of non-participating nodes are transmitted. However, this would pre-suppose a rigid network structure, limiting the routing flexibility.

It is the contribution of the work presented to propose a KPD solution for encrypted convergecast traffic which is

- almost optimal regarding the data overhead. Aggregated and encrypted convergecast traffic should be nearly of the same size than sending aggregated convergecast traffic in plaintext,
- provides reasonable security with respect to an attacker's rational cost window she is willing to invest for breaking the security architecture, and
- is robust against exhausted nodes and an unreliable broadcast medium.

The contribution is especially valuable for large scale WSNs as it does not require the transmission of any node IDs.

Key Pre-distribution Protocol for Multi-Phase Wireless Sensor Networks

Motivation: Key management is a core mechanism to secure wireless sensor networks. The goal of key management is to establish secret keys between sensor nodes that need to communicate securely. One important characteristics of key management protocols is that they must be scalable, CPU and energy efficient. We are considering large networks of battery-operated wireless sensors. We are assuming that the average lifetime of each node is much shorter than the operating lifetime of the overall network. As a result, new nodes are periodically deployed in order to assure network connectivity. Each set of new nodes that join the network in a future time consist of a *node generation*. Of course, new nodes must be able to establish secret keys with previously deployed nodes. Protocols that provide this property are known as *Multi-phase* deployment protocols.

Public key cryptography cost is prohibitive for most WSNs and can rarely be used. Most existing key management schemes are based on symmetric key cryptography. One of the most popular schemes, referred as *RKP* (Random Key Pre-Distribution), was proposed by Eschenauer and Gligor [7] and later on extended by Chan, Perrig and Song [8]. This scheme is distributed and uses a random key pre-distribution approach. In this protocol, each node is configured with a key ring of m sub-keys. These keys are randomly drawn from a large key pool of P sub-keys. Two nodes establish their secret key from the sub-keys they have in common in their key ring. If the parameters m and P are chosen properly, the probability that any two nodes share at least one common sub-key is high. This protocol is scalable, CPU and energy efficient. It also trivially supports node addition: any new deployed node gets configured with m sub-keys from the system key pool. It can then establish a secret key with any previously deployed node.

One important drawback of this key pre-distribution scheme is that an attacker that corrupts several nodes can partially reconstruct, from the compromised nodes key rings, the key pool of system. The more nodes it corrupts the more sub-keys it obtains and the more communications it can eavesdrop. If the attacker is constantly corrupting nodes, it will eventually learn the whole key pool and all newly deployed nodes will establish links that will immediately be compromised. In other words, *the security of the whole network degrades with time*. We believe this is a non-desirable property. A naive solution would be to periodically refresh the key pool, i.e. configure new deployed nodes with fresh sub-keys. Newly deployed nodes would be more secure (since their sub-keys were not exposed previously) but would be unable to establish secure links with previously deployed nodes. Newly deployed nodes would constitute a new network that is unable to securely communicate with the previous one. A novel approach is definitely needed.

Contributions: We have developed a new key pre-distribution scheme, referred as *RoK* (A Robust Key Pre-distribution Protocol for Multi-Phase Wireless Sensor Networks) in the rest of this document, that allows sensors deployed at different times to establish secure links. In this scheme, sub-keys have limited lifetimes and are refreshed periodically. This has the two following positive consequences:

1. A network that is temporarily attacked (i.e. the attacker is active only during a limited amount of time) automatically **self-heals**, i.e. recovers its initial state when the attack stops. With the *RKP* scheme, an attacker that corrupts a certain amount of nodes compromises a given percentage of the total number of secure communications. This percentage remains constant until the end of the network. With our proposal, the percentage of compromised links gradually decreases to 0 when the attack stops. *The security of the network improves with time*.

2. A network that is constantly attacked (i.e. the attacker regularly corrupts nodes of the network, without stopping) is much less impacted than a network that uses the basic key pre-distribution protocol. With the *RKP* scheme, the fraction of compromised links constantly increases until it eventually reaches 1 i.e. until all the links are compromised. With our extension, the proportion of compromised links is limited and constant. *The security of the network is constant and does not degrade with time.*

4 Keying for Convergecast Traffic

Bihomomorphic Encryption Transformation

The basic idea of a keying for convergecast traffic is to apply a specific form of *concealed data aggregation* (CDA) with a specific homomorphic encryption function. Our proposal relies on the use of a homomorphic encryption function, which fulfills the following additional requirements for any α :

$$\mathbb{K}_\alpha := \mathbb{K}^\alpha \quad \text{and} \quad \kappa_\alpha(k_1, \dots, k_\alpha) := k_1 \odot \dots \odot k_\alpha. \quad (5)$$

This yields that the following is true for any $k_1, \dots, k_\alpha \in \mathbb{K}$ and $v_1, \dots, v_\alpha \in \mathbb{P}$:

$$E_{k_1}(v_1) \oplus \dots \oplus E_{k_\alpha}(v_\alpha) = E_{k_1 \odot \dots \odot k_\alpha}(v_1 + \dots + v_\alpha). \quad (6)$$

Thus, the encryption is homomorphic both on the plaintext space *and* on the key space. Consequently, we term such an encryption as *bihomomorphic*.

A simple example for a bihomomorphic encryption transformation is the modulo integer addition. That is for $\mathbb{K} \equiv \mathbb{P} \equiv \mathbb{Z} \pmod{n}$ with an integer $n \geq 1$, $E_k(m) := k + m \pmod{n}$ is a bihomomorphic encryption where both \odot and $+$ being the addition modulo n .

Observe that the encryption with a keystream generator is *not* bihomomorphic according to the definition. As we will show, the encryption is homomorphic in the plaintext and in the keystream elements. However, to be homomorphic in the keys would require for all $k_1, \dots, k_\alpha \in k$ the existence of a key $k \in \mathbb{K}$ such that $kg(k_1, t) \odot \dots \odot kg(k_\alpha, t) = kg(k, t)$ for all t which is very unlikely in the general case.

TAUK and Bihomomorphic Encryption in a Nutshell

Next, we describe the complete key management architecture of the Topology Aware Unique Keying (TAUK) and its application to a bihomomorphic encryption function. Compared to previous solutions like [3], [5] we see substantial advantages in that:

1. an (almost) optimal data overhead during the aggregation phase is ensured. No list of node IDs or, more sophisticated, complementary list of node IDs respectively subset of node IDs needs to be transmitted in each aggregation phase,
2. robustness in respect of silent nodes is provided during the aggregation phase,
3. key refreshment is provided as an essential pre-requisite to support any *deterministic* bihomomorphic encryption function.

The KPD TAUK supports any bihomomorphic encryption function for CDA. It impacts the *initialization phase*, each *aggregation phase* and eventual *refreshment phases*:

- to run TAUK in the *initialization phase*, it is assumed that each sensor node N already knows its direct neighbors $Pred(N)$ and $Succ(N)$. Subsequently, TAUK ensures that each node receives a single symmetric key, whereas all keys from the sensor nodes are derived from a master key, which is solely stored at the sink node. In addition to its key, each node stores encrypted default values. Each of such ciphertexts corresponds to a $N' \in Succ(N)$. We will see that they provide robustness during the aggregation phase. During the initialization phase the system is vulnerable, even to passive attacks. No attacker is assumed to be in place during this phase.

- during an *aggregation phase*, convergecast traffic is encrypted end-to-end from the sensing nodes to the sink node. Each node N applies the bihomomorphic encryption function by encrypting its monitored value with its own unique key and by subsequently summing up the resulting ciphertext to the received ciphertexts from its children $Succ(N)$. Since each node purely stores its own key it can not decrypt the incoming ciphertexts from its children. Only the sink node is enabled to decrypt the final aggregated value by applying the master key to the received ciphertexts. Each intermediate node adds those stored default ciphertexts to the aggregation value which correspond to its silent successors. During an aggregation phase, the system provides reasonable security against passive and active attacks.
- during a *refreshment phase*, each node's key is updated. Unlike during the *initialization phase* where no attacker is assumed to be in place, any information exchanged in this phase might be eavesdropped. Therefore, it must be ensured that no data is revealed that might compromise the security. Note that in particular for any deterministic encryption scheme it is essential to refresh keys, ideally after each single aggregation phase. However, for a realistic setting we propose to find a good balance between the frequency of key refreshment and the required security level.

We will substantiate the statements from this birdsview in the following Section.

Topology Aware Unique Keying and bihomomorphic Encryption

Basic idea

We give a general description of our new CDA scheme before we discuss concrete instantiations. Let in time frame t denote K_N^t the key used by node N and v_N^t the measured value. When the transmitted values are encrypted with a bihomomorphic encryption scheme, S receives at time frame t the value²

$$\bigoplus_N E_{K_N^t}(v_N^t) = E_{\bigodot_N K_N^t} \left(\sum_N v_N^t \right). \quad (7)$$

Thus, S is able to get the sum of all the measurements by decrypting the received value with the key

$$\hat{K}_S^t := \bigodot_N K_N^t. \quad (8)$$

Interestingly, S actually does not need to know the concrete values of the "summands" K_N^t as long as it knows the "sum" \hat{K}_S^t .

This observation is the basic motivation for the CDA scheme we propose. The deployment of a bihomomorphic encryption algorithm allows the aggregation of data encrypted under different keys where only the aggregation of the keys needs to be known to the sink node. For this purpose we assume that for each time frame t every node N has a key K_N^t and the sink node knows $\hat{K}_S^t := \bigodot_N K_N^t$.

At the end of an aggregation process, the sink node receives the encryption of the aggregated sensed values. Due to the bihomomorphic property this corresponds to a ciphertext encrypted with the aggregation of the keys of the responding nodes. However, the keys of the silent nodes are missing. Thus, decrypting with \hat{K}_S^t would lead to a false result. To handle this problem, we propose that each node N additionally stores some values $E_{K_{N'}}(R_{N'}^t)$ for all $N' \in Succ(N)$. The purpose of the values $R_{N'}^t$ is that, whenever a node N does not receive the data from all its successors $Succ(N)$, it replaces the missing ciphertexts by the ciphertexts of dummy values $R_{N'}^t$. Depending on how the dummy values are generated, the sink node has to treat them differently. We will discuss this aspect later.

In principle, the CDA schemes is divided into two different phases:

1. the initialization phase when the data, e.g., keys, are installed on the nodes within the network,
2. the aggregation phase which is divided into time frames where the measurements are taken, aggregated, and forwarded to the sink node.

In the following, we describe both phases in general terms.

²For the sake of simplicity, we assume for the moment that all nodes have participated in the aggregation. The case of silent nodes will be addressed in our scheme later.

Initialization phase

In the initialization phase, the keys and the dummy values are distributed within the network. We assume that each node knows its predecessor and its successors. Furthermore, we assume that no attackers are present in this time period as all secrets are exchanged in clear. Alternatively the nodes may be equipped with a network wide key which is used to secretly exchange the information in this phase and delete it subsequently from the memory. This would protect the initialization phase against passive attackers.

The initialization proceeds top-down from the sink node to the leaf nodes. The sink node chooses one (or, more sophisticated, several) master keys, which will be the aggregation of all individual keys distributed in the WSN. In a nutshell, every node receives at some point in time during the key distribution his share of the master key which it distributes to its successors in a controlled way. At the same time, the dummy values are created and distributed.

Initialization: The sink node S selects a number r which represents how many keys each node will store at the same time. S chooses randomly r master keys $\hat{K}_S^1, \dots, \hat{K}_S^r \in \mathbb{K}$ and dummy values $R^1, \dots, R^r \in \mathbb{P}$.

Distribution: Let N be a node which is not a leaf node and $Succ(N) = \{N_1, \dots, N_m\}$.

1. If N is not the sink node, it receives during the key distribution from its predecessor some data $(\hat{K}_N^1, \dots, \hat{K}_N^r)$ and (R_N^1, \dots, R_N^r) . If N is the sink node, it created these values on its own. Whereas the first is its share of the master keys, the second determines which dummy values will be installed for the successors of N .
2. If N is a leaf node, it solely deletes (R_N^1, \dots, R_N^r) from its memory and defines $K_N^i := \hat{K}_N^i$. Otherwise, the following steps are performed:
 - (a) N splits the key \hat{K}_N^t into $\hat{K}_N^t = \left(\bigodot_{N' \in Succ(N)} \hat{K}_{N'}^t \right) \odot K_N^t$ for $t \in \{1, \dots, r\}$ where the splitting is randomly chosen.
 - (b) Furthermore, N derives from (R_N^1, \dots, R_N^r) appropriate dummy values $(R_{N'}^1, \dots, R_{N'}^r)$ and stores $E_{\hat{K}_{N'}^t}(R_{N'}^t)$ for all $N' \in Succ(N)$. The derivation mechanism depends on the concrete instantiation.
 - (c) Subsequently, N sends to all successors $N' \in Succ(N)$:

$$N \rightarrow N' : (\hat{K}_{N'}^1, \dots, \hat{K}_{N'}^r), (R_{N'}^1, \dots, R_{N'}^r) \quad (9)$$

- (d) Finally, N deletes all values from its memory except of its own keys K_N^1, \dots, K_N^r and the encrypted dummy values $E_{\hat{K}_{N'}^i}(R_{N'}^i)$ for $i = 1, \dots, m$ and $N' \in Succ(N)$. The only exception is in the case of the sink node since S does *not* discard the master keys $\hat{K}_S^1, \dots, \hat{K}_S^r$.

Observe that for each node N , it holds that

$$\hat{K}_N^i = \left(\bigodot_{N' \in ST(N)} K_{N'}^i \right) \odot K_N^i. \quad (10)$$

That is the key \hat{K}_N^i is the aggregation of all keys from the subtree rooted in N . However, as N keeps K_N^i but deletes \hat{K}_N^i , it has no information on the keys in the lower levels. This is in particular true for the case $N = S$ where

$$\hat{K}_S^i = \bigodot_N K_N^i. \quad (11)$$

S knows the aggregation of all keys distributed within the network which is all it needs to decrypt the aggregated data as discussed above. The distribution phase is illustrated in Figure 4.

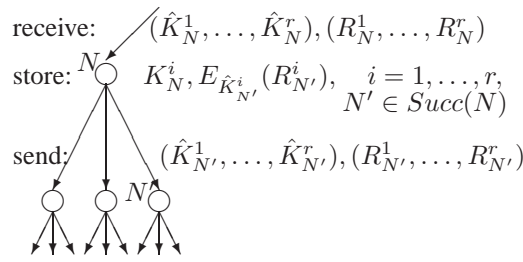


Figure 4: Distribution phase.

Aggregation phase

The aggregation phase is subdivided into discrete time frames in which data is measured, encrypted, aggregated, and forwarded from the leaf nodes to the sink node. Each node receives the encrypted data from (some of its) successors, aggregates them together with its own ciphertext, and subsequently forwards the result to its predecessors. For a detailed description, one has to distinguish between leaf nodes, aggregator nodes, and the sink node.

Leaf nodes:

Let N be a leaf node with key K_N^t and v_N^t denotes its sensed value which is encrypted to $C_N^t := E_{K_N^t}(v_N^t)$. The leaf node sends the ciphertext to its predecessor together with a counter set to zero:

$$N \rightarrow Pred(N) : (C_N^t, 0) \quad (12)$$

Aggregator Nodes:

Let N be a node which is neither a leaf node nor the sink node. If N is not a sensor node, we set $v_N^t := 0$, otherwise let v_N^t denote the measurement made in frame t . Furthermore, it expects to receive data from its successors where it cannot be excluded that some of them remain silent in this frame. Let $Succ(N)_{resp}^t$ denote the nodes from which N got a transmission during frame t and $Succ(N)_{silent}^t$ denote the nodes which remained silent. That is it holds

$$Succ(N) = Succ(N)_{resp}^t \dot{\cup} Succ(N)_{silent}^t \quad (13)$$

for all t . For each $N' \in Succ(N)_{resp}^t$, the node N receives some data $(C_{N'}^t, ctr_{N'}^t)$ where $C_{N'}^t$ is the encrypted aggregated value of the subtree $ST(N')$ and $ctr_{N'}^t \geq 0$ is a non-negative integer which is equal to the number of dummy values contained in the plaintext of $C_{N'}^t$.

N computes the actual ciphertext by

$$C_N^t := \bigoplus_{N' \in Succ(N)_{resp}} C_{N'}^t \oplus \bigoplus_{N' \in Succ(N)_{silent}} E_{K_{N'}^t}(R_{N'}^t) \oplus E_{K_N^t}(v_N^t) \quad (14)$$

and the actual counter

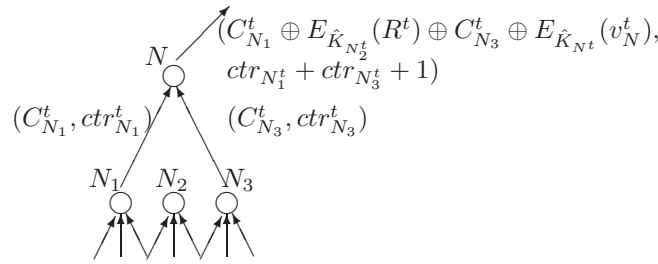
$$ctr_N^t := \sum_{N' \in Succ(N)_{resp}} ctr_{N'}^t + |Succ(N)_{silent}^t|. \quad (15)$$

Let $P_{N'}^t$ denote the plaintext underlying $C_{N'}^t$, that is $E_{\hat{K}_{N'}^t}(P_{N'}^t) := C_{N'}^t$. Then, C_N^t can be rewritten to

$$\begin{aligned} C_N^t &= \bigoplus_{N' \in Succ(N)_{resp}} E_{\hat{K}_{N'}^t}(P_{N'}^t) \oplus \bigoplus_{N' \in Succ(N)_{silent}} E_{\hat{K}_{N'}^t}(R_{N'}^t) \oplus E_{\hat{K}_N^t}(v_N^t) \\ &= E_{\hat{K}_N^t} \left(\bigoplus_{N' \in Succ(N)_{resp}} P_{N'}^t \oplus \bigoplus_{N' \in Succ(N)_{silent}} R_{N'}^t \oplus v_N^t \right). \end{aligned} \quad (16)$$

Thus, any node N sends ciphertext which is encrypted under its key \hat{K}_N^t . The above also implies the following recursive definition for P_N^t where we set $P_N^t := v_N^t$ if N is a leaf:

$$P_N^t := \bigoplus_{N' \in Succ(N)_{resp}} P_{N'}^t \oplus \bigoplus_{N' \in Succ(N)_{silent}} R_{N'}^t \oplus v_N^t. \quad (17)$$

Figure 5: Aggregation phase at time frame t .

Subsequently it sends this tuple to its predecessor (see also Figure 5):

$$N \rightarrow \text{Pred}(N) : (C_N^t, \text{ctr}_N^t). \quad (18)$$

Observe that N knows only the key K_N^t during the initialization phase. Hence it is not able to decrypt the aggregated ciphertexts.

Sink node:

At the end of each time frame, the aggregation of some encrypted measurements reached the sink node while others got lost due to silent nodes. Observe that even if a node N successfully reported its ciphertext to its predecessor $\text{Pred}(N)$, it might be lost at the end if $\text{Pred}(N)$ (or one of its predecessors) was silent in this time frame.

Let \mathcal{V}_{resp}^t denote the set of all nodes whose ciphertexts are part of the aggregated data that reaches the sink node. As explained, each N forwards the encryption of $P_{N'}^t$ for all $N' \in \text{Succ}(N)_{resp}$, and the encryption of $R_{N'}^t$ for all $N' \in \text{Succ}(N)_{silent}$. If we define

$$\mathcal{V}^t := \bigcup_{N \in \mathcal{V}_{resp}^t} \text{Succ}(N) \quad (19)$$

$$\mathcal{V}_{silent}^t := \mathcal{V}^t \setminus \mathcal{V}_{resp}^t. \quad (20)$$

then the sink node receives the following data:

$$C_S^t = \bigoplus_{N \in \mathcal{V}_{resp}^t} E_{K_N^t}(v_N^t) \oplus \bigoplus_{N' \in \mathcal{V}_{silent}^t} E_{K_{N'}^t}(R_{N'}^t) \quad (21)$$

$$= E_{K_S^t} \left(\sum_{N \in \mathcal{V}_{resp}^t} v_N^t + \sum_{N \in \mathcal{V}_{silent}^t} R_N^t \right) \quad (22)$$

As opposed to the other aggregator nodes, S knows the key K_S^t and can therefore decrypt the ciphertext C_S^t to get

$$D_{K_S^t}(C_S^t) = \sum_{N \in \mathcal{V}_{resp}^t} v_N^t + \sum_{N \in \mathcal{V}_{silent}^t} R_N^t. \quad (23)$$

Observe that the counter ctr_S^t tells how many dummy values are contained in R^t , that is $\text{ctr}_S^t = |\mathcal{V}_{silent}^t|$. Depending on how the dummy values are created and handled, the sink node can cope with $\sum_{N \in \mathcal{V}_{silent}^t} R_N^t$ to get (possibly only an approximation) of $\sum_{N \in \mathcal{V}_{resp}^t} v_N^t$. This will be discussed into more detail in the next Section.

Remark: As it is now, the sink node can recover the sum of the measured and reported values but it does not know how many values have been aggregated in total, that is the size of \mathcal{V}_{resp}^t . Since these information may be relevant for some WSN applications, one should incorporate a second counter, resp^t , which computes the number of responded nodes. The counter resp^t can be handled like ctr^t . The node N receives from each responding node $N' \in \text{Succ}(N)$ a value $\text{resp}_{N'}^t$ and forwards the sum of it to $\text{Pred}(N)$.

Concrete instantiations

Observe that the previous Section provides only a high level description as the exact steps differ depending on the concrete instantiations. For a more concise description, the following building blocks need to be specified:

1. The bihomomorphic encryption mechanism,
2. the refreshing of the keys, and
3. the creation and handling of the dummy values.

Encryption scheme

Recall the definition of a bihomomorphic encryption. To the best of our knowledge, bihomomorphic symmetric encryption schemes have not been studied in open literature. However, a simple example has already been mentioned, namely $E_k(v) := k + v$ if k and v both belong to the same algebraic structure, e.g., \mathbb{Z}_{2^n} .

Key refreshment

For security reasons, keys should be regularly refreshed, at least for deterministic encryption schemes. However, to ensure that the sink node can still decrypt the aggregated values, it is necessary that the sink node knows the aggregation of the modified keys. Thus, either the change of the keys is coordinated by the sink node itself, or it is done following a deterministic rule which allows the sink node to compute the new key autonomously. We refer to this two different approaches by coordinated key refreshment and autonomous key schedule.

Coordinated key refreshment:

In this approach, the keys are refreshed on request of and coordinated by the sink node. This means that some data is distributed from the sink node to the individual nodes which determines how the keys have to be changed.

In the following we propose one concrete coordinated key refreshment mechanism. At this we assume that each node (including the sink node) stores only one key at one point in time, i.e. $r = 1$, which is changed according to data coming from the sink node. In principle the refreshment is similar to the initialization phase. The idea is to modify the keys such that the master key \hat{K}_S^t is replaced by a new master key $\hat{K}_S^{t+1} := \hat{K}_S^t \odot \hat{\Delta}_S^t$ where the sink node S chooses (randomly) the difference $\hat{\Delta}_S^t$. This difference is split and reported to the successors in the same way as the master key in the initialization phase.

More concretely, let N be a node which is not a leaf node. During the key refreshment phase, N receives a value $\hat{\Delta}_N^t$ which will be used to update all keys in the subtree $ST(N)$. For this purpose N splits the given value $\hat{\Delta}_N^t$ into

$$\hat{\Delta}_N^t = \bigodot_{N' \in Succ(N)} \hat{\Delta}_{N'}^t \odot \Delta_N^t \quad (24)$$

and sends $\hat{\Delta}_{N'}^t$ to each N' . The own key K_N^t is changed to

$$K_N^{t+1} := K_N^t \odot \Delta_N^t. \quad (25)$$

Furthermore, N computes for all $N' \in Succ(N)$:

$$E_{\hat{\Delta}_{N'}^t}(0) \oplus E_{K_{N'}^t}(R_{N'}^t) = E_{K_{N'}^t \odot \hat{\Delta}_{N'}^t}(R_{N'}^t) = E_{K_{N'}^{t+1}}(R_{N'}^t) \quad (26)$$

to replace the encryption of $R_{N'}^t$ under the "old" key $K_{N'}^t$ to the encryption under the "new" key $K_{N'}^{t+1}$. At this, we assume that the dummy value $R_{N'}^t$ remains constant over all time frames, that is $R_{N'}^t = R_{N'}^{t+1}$. Observe that the described modifications are possible with neither knowing $R_{N'}^t$, nor $K_{N'}^t$. The whole mechanism is shown in Figure 6.

At the end, any node N has changed its key K_N^t to $K_N^{t+1} = K_N^t \odot \Delta_N^t$. The new keys can be used for the next frame $t + 1$. As each node knows only shares of the difference between the old secret master key and the new one, the confidentiality of the forthcoming data is not compromised.

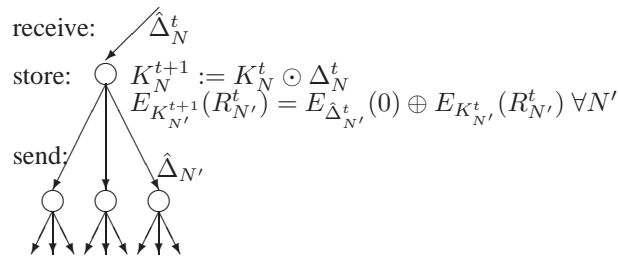


Figure 6: Refreshment phase.

However, it is important that an outsider cannot eavesdrop the differences. Otherwise he could easily compare the values v_N^t and $v_N^{t'}$ from different time frames $t < t'$ by comparing $E_{K_N^{t'}}(v_N^{t'})$ with

$$\bigoplus_{i=t}^{t'-1} E_{\Delta_N^i}(0) \oplus E_{K_N^t}(v_N^t) = E_{K_N^{t'}}(v_N^{t'}). \quad (27)$$

This would render the whole key refreshment needless.

One possibility to conceal these values from outside parties could be that the nodes establish pairwise keys between the nodes and each of their successors during the initialization phase which will be used to encrypt the differences. That is, each node has then two different keys, one used for encrypting the aggregated values, and one for encrypting the data for the key refreshment. For the latter, any encryption scheme can be used, and the keys do not have to sum up to a value known by the sink node.

Key Schedule:

The previously described approach requires the regular transmission of additional data to determine the key change. Depending on the frequency of the key updates and the network structure, this may impose impractical workload and energy consumption. Therefore, we describe as an alternative a key schedule which allows each node to update its key by itself without any further communication with other devices. Recall that during the initialization phase, r different keys K_N^1, \dots, K_N^r have been installed at each node N which are used in the first r time frames. They can be used in a key schedule to determine all subsequent keys. Let φ_N^t denote the operation a node N applies at the end of time frame t to generate the next key K_N^{t+1} . In a similar manner, we refer to the mechanism S uses to update its master key \hat{K}_S^t by $\hat{\varphi}_S^t$. Then it holds

$$K_N^{t+1} = \varphi_N^t(K_N^1, \dots, K_N^r) \quad (28)$$

for all $t \geq 1$. However, recall Equation (11) which states the following basic condition for our scheme:

$$\hat{K}_S^t = \bigodot_N K_N^t \quad \forall t \geq 1. \quad (29)$$

Thus, the key derivation function must be chosen such that this relation remains true for all time frames. This implies

$$\begin{aligned} & \hat{\varphi}_S^t(\bigodot_N K_N^1, \dots, \bigodot_N K_N^r) \\ = & \hat{\varphi}_S^t(\hat{K}_S^1, \dots, \hat{K}_S^r) = \hat{K}_S^{t+1} \stackrel{\text{Eq. (11)}}{=} \bigodot_N K_N^{t+1} \\ = & \bigodot_N \varphi_N^t(K_N^1, \dots, K_N^r). \end{aligned}$$

An obvious choice is to set all functions φ_N^t and $\hat{\varphi}_S^t$ to be equal and to let them be linear in \mathbb{K} with respect to \odot . That is for each t exist coefficients $c_1^t, \dots, c_r^t \in \mathbb{N}$ such that

$$\varphi_N^t(x_1, \dots, x_r) := c_1^t \cdot x_1 \odot \dots \odot c_r^t \cdot x_r. \quad (30)$$

where $c \cdot x$ is defined as $x \odot \dots \odot x$ (repeated c times). Observe that the coefficients c_i^t are the same for each node N and should be determined by some deterministic algorithm. For example one could envision the vector (c_1^t, \dots, c_r^t) to be the actual state of a linear feedback shift register. This would ensure that all possible vectors of coefficients (except the all-zero vector) occur.

Recall that a node must be able to autonomously update (the encryption of) the dummy values $R_{N'}^{t+1}$ as well. This can be achieved by using the same mechanism for key refreshment to modify the encrypted values. For example in the choice described above, one can easily show that

$$\hat{K}_N^{t+1} = \bigodot_{N' \in ST(N)} K_{N'}^{t+1} \odot K_N^{t+1} \quad (31)$$

$$= \bigodot_{N' \in ST(N)} \left(\bigodot_{i=1}^r c_i^t \cdot K_{N'}^i \right) \odot \bigodot_{i=1}^r c_i^t \cdot K_N^i \quad (32)$$

$$= \bigodot_{i=1}^r c_i^t \cdot \left(\bigodot_{N' \in ST(N)} K_{N'}^i \odot K_N^i \right) \quad (33)$$

$$= \bigodot_{i=1}^r c_i^t \cdot \hat{K}_N^i. \quad (34)$$

Thus, if one defines³

$$R_N^{t+1} = \sum_{i=1}^r c_i^t \cdot R_N^i \quad (35)$$

then a node N would compute for each $N' \in Succ(N)$:

$$E_{\hat{K}_N^{t+1}}(R_{N'}^{t+1}) := \bigoplus_{i=1}^r c_i^t \cdot E_{\hat{K}_N^i}(R_{N'}^i) = E_{\bigodot_{i=1}^r c_i^t \cdot \hat{K}_N^i} \left(\underbrace{\sum_{i=1}^r c_i^t \cdot R_{N'}^i}_{=R_{N'}^{t+1}} \right). \quad (36)$$

Thus, this approach allows the autonomous update of both the keys *and* the encryption of the dummy values.

Robustness in respect to silent nodes

As explained above, the (encryption of the) values R_N^t will be used to replace the (encryption of the) missing measurements from silent nodes N . Also here, different possibilities exist on how to implement these which are mainly influenced by the following two issues:

1. How the dummy values are handled by the sink node at the end when the aggregated ciphertext is decrypted (see also Eq. (23)).
2. How the keys are refreshed as each node N needs to store the *encrypted* values $E_{\hat{K}_N^t}(R_{N'}^t)$ for every $N' \in Succ(N)$.

In the previous Section, we distinguished between coordinated key refreshment triggered by the sink node and a key schedule which can be done autonomously by any node. Regarding the first approach, we have shown how the encryption of the dummy values can be updated as well even if the key itself is unknown. Although we assumed that the dummy values themselves remain constant, that is $R_N^t = R_N^1$ for all $t \geq 1$, this is not mandatory. For a more general approach, consider that besides the value Δ_N^t , which specifies the key update, a second value ρ_N^t exists such that $R_N^{t+1} = R_N^t + \rho_N^t$. Then the encryption of the dummy values can be updated without knowing the keys by computing

$$E_{\hat{\Delta}_N^t}(\rho_{N'}^t) \oplus E_{K_{N'}^t}(R_{N'}^t) = E_{K_{N'}^t \odot \hat{\Delta}_N^t}(\rho_{N'}^t + R_{N'}^t) = E_{K_{N'}^{t+1}}(R_{N'}^{t+1}). \quad (37)$$

³Again, we define $c \cdot R$ by $R + \dots + R$.

This shows that the update of the dummy values is completely independent of the key update and can be freely chosen. Although a variety of different possibilities is imaginable, we will describe only two of them. The first is that all dummy values are equal and remain constant. That is a constant $R \in \mathbb{P}$ exists such that $R_N^t = R$ for all time frames t and all nodes N . The advantage of this approach is twofold. Firstly, no additional operations are required to update the dummy value as it remains constant. Secondly, and probably more important, according to (23), the sink node decrypts

$$D_{\hat{K}_S^t}(C_S^t) = \sum_{N \in \mathcal{V}_{resp}^t} v_N^t + ctr_S^t \cdot R. \quad (38)$$

If we require that S knows R and as ctr_S^t is reported to S anyway, the sink node can easily determine the value $\sum_{N \in \mathcal{V}_{resp}^t} v_N^t$. Observe that this approach is efficient and provides utmost accuracy of the result. However, we will see that this compromises the security to some extent.

Therefore, we describe a second alternative approach in combination with the coordinated key refreshment. In this, the dummy values are no random values anymore but defined by the last successfully reported value. More formally, assume that a node N received during time frame t the ciphertext $C_{N'}^t = E_{K_{N'}^t}(P_{N'}^t)$ from one of its successors $N' \in Succ(N)$ where $P_{N'}^t$ is the aggregation of one or several values. However, in the next time frame $t + 1$, node N' remained silent, for whatever reason. Then, in this approach N simply updates the encryption of (the unknown value) $v_{N'}^t$, to re-use it for the actual time frame. More precisely, N computes

$$E_{\hat{\Delta}_{N'}^t}(0) \oplus C_{N'}^t = E_{\hat{\Delta}_{N'}^t}(0) \oplus E_{K_{N'}^t}(m_{N'}^t) = E_{K_{N'}^t \odot \hat{\Delta}_{N'}^t}(m_{N'}^t) \quad (39)$$

$$= E_{K_{N'}^{t+1}}(m_{N'}^t) =: C_{N'}^{t+1} \quad (40)$$

and uses the result for the aggregation. This means in principal that N pretends that N' has re-send the same value $P_{N'}^t$, which it did send in the previous frame t . Of course the final result is not accurate anymore as N' might have measured another value in this time frame. Therefore, we assume that this approach has value in dedicated use cases. However it illustrates the flexibility our scheme provides.

This flexibility might no longer be provided if a key schedule is used. As we explained for the concrete proposal, if we define $R_N^{t+1} = \sum_{i=1}^r c_i^t \cdot R_N^i$, a node is able to update the encryption of the dummy values on its own, even without knowing the keys. However, the dummy values cannot longer be freely modified but have to follow the given rule. This prohibits for the example the approach described above of re-using previous (unknown) values $P_{N'}^t$. Yet, what still works is to let the dummy values in one time frame to be equal to allow the sink node to take them off from the final decryption. More precisely, assume that there exist some constant values $R^1, \dots, R^r \in \mathbb{M}$ such that $R_N^i = R^i$ for any node N and $1 \leq i \leq r$. This immediately implies $R^t = \sum_{i=1}^r c_i^t \cdot R^i$ which can be computed by the sink node if it knows the values R^1, \dots, R^r . Furthermore, the sink node S decrypts at the end of a time frame the following:

$$D_{K_S^t}(C_S^t) = \sum_{N \in \mathcal{V}_{resp}^t} v_N^t + ctr_S^t \cdot R^t. \quad (41)$$

Thus, again S is able to derive the accurate result $\sum_{N \in \mathcal{V}_{resp}^t} v_N^t$.

Evaluation

In this section, we are going to evaluate the proposed scheme and compare it to previous proposals in respect to communication costs, security, and flexibility. As already pointed out, a variety of different instantiations are thinkable which may differ in the communication costs and/or the provided security. In practice the choice should be made depending on the aimed use scenario. However, we will concentrate on only one concrete variant to make the comparison more simple and illustrative. Stimulated by the fact that the lifetime of a WSN has probably the highest market relevance, we have chosen a variant which has minimum communication costs and still a reasonable security level. This means that only the aggregated ciphertexts and few additional data need to be forwarded whereas still a reasonable security level is provided for a huge class of applications. More concretely, we focus on the following instantiation:

Encryption: The message space \mathbb{P} is $\{0, \dots, 2^\mu - 1\}$ and both the ciphertext space \mathbb{C} and the key space \mathbb{K} are equal to $(\mathbb{Z}_p, +)$ where p is a prime number $\geq 2^{\mu+b_{\#sensors}}$, $b_{\#sensors}$ the logarithm of the number of

sensors in the network taken to the base of 2, and "+" denotes the usual modulo addition. Observe that \mathbb{P} can be easily embedded in \mathbb{K} (resp. \mathbb{C}). The encryption is defined by $E_k(m) := m + k$.

Key refreshment: For the key encryption we consider the key schedule where the keys are defined by the first r keys. The coefficients $c^t = (c_1^t, \dots, c_r^t) \in \{0, 1\}^r \in \mathbb{Z}_p^r$ correspond to the internal state of a linear feedback shift register (LFSR) of length r with a primitive minimal polynomial and a non-zero initial state.

Robustness: For handling silent nodes, we suppose the variant that at each time frame t , every node stores the same dummy value R^t .

For comparison, we consider the following CDA schemes:

- **HbH** (*Hop by Hop encryption*): we use the simple hop by hop encryption and aggregation scenario as the basis for our comparison.
- **TAGK** (*Concealed data aggregation, [17]*): encryption using the Domingo-Ferrer encryption scheme.
- **CMT** (*the scheme proposed by Castelluccia, et al. [3]*): encryption by adding the output of a keystream generator.
- **ME** (*Multiple Encryption, [3, 5]*): an extension of CMT. This class refers to the proposals described in [3, 5] which minimize the node ID transmission overhead problem in schemes where unique keys are used.
- **TAUK** (*TAUK with modular addition and deterministic key schedule*): the proposal described in this paper.

Energy consumption

We analyze and compare the cost of our proposal with previously proposed solutions in terms of energy consumption. For this purpose, we differentiate between computational effort and transmission costs.

Computational effort

Regarding the computational effort, the schemes **CDA**, **ME**, and **TAUK** all encrypt by aggregating some keys to the plaintexts, which in most cases will mean modular addition. Thus, the effort for one encryption is comparatively the same. However, CDA and TAUK encrypt each plaintext only once, whereas in ME a purely sensing node encrypts d times before transmission, and an aggregator node decrypts and encrypts $d + 1$ times, resulting in a somewhat higher effort.

TAGK uses besides modular addition also modular *multiplication*, making it more elaborate than the previous ones. For **HbH**, any appropriate encryption scheme can be used. Thus, the encryption effort might be smaller compared to the other schemes. However, each aggregator node has to decrypt all incoming ciphertexts and to encrypt the result again. Therefore, we expect that the total cost are rather higher compared to CMT, ME, and TAUK.

Transmission costs

Compared to the energy consumption for CPU processing, the energy consumption for data sending or receiving is in the order of 10^2 higher. Therefore, transmission of messages represents the pre-dominant energy consuming process. We will study only the sending costs although there is also a corresponding receiving cost associated to every transmission.

First of all, equal for all schemes is that every node (except the sink) forwards during each time frame some ciphertext. However, schemes may use different encryption schemes which vary in their data size. Another difference lies in the handling of the identifiers. Let b_{data} denote the bit size of the data sensed by the sensors and similarly let b_{ID} be the bit size of a node's identity. For the sake of simplicity, we assume these values to be constant throughout the whole network. Furthermore, let $b_{\#sensors}$ be the logarithm of the number of

Scheme	$N \rightarrow Pred(N)$
HbH	$(b_{data} + b_{\#sensors}) + b_{\#sensors}$
CDA	$(b_{data} + b_{\#sensors}) \cdot s \cdot \#splits + b_{\#sensors}$
CMT	$(b_{data} + b_{\#sensors}) + b_{ID} \cdot \#ST(N)$
ME	$(b_{data} + b_{\#sensors}) + b_{ID} \cdot \#ST_{\#hops}(N) + b_{\#sensors}$
TAUK	$(b_{data} + b_{\#sensors}) + 2 \cdot b_{\#sensors}$

Table 2: Comparison of the maximum node transmission size.

sensors within the network taken to the base 2. Thus, the aggregation of all $2^{b_{\#sensors}}$ measurements takes at least $b_{\#sensors} + b_{data}$ bits, being a lower bound for the ciphertext size. Besides, we assume that the sink needs to know the number of aggregated values in the result it receives. Thus, if no identifiers are reported, at least one counter of size $b_{\#sensors}$ has to be included within the transmissions.

In the (**HbH**) case, the aggregator simply decrypts any incoming ciphertexts, aggregates the resulting plaintexts, and forwards the encryption of the result. No identities need to be transmitted. For encryption, any encryption scheme can be used. Therefore we assume the minimal length of $b_{\#sensors} + b_{data}$ for a ciphertext.

In **CDA**, an encryption function taken from [4] is considered where two parameters influence the size of the ciphertext: $\#splits$, the number of times the plaintext message is split, and m , the module used which determines the size of each part. For this purpose, m has to be chosen such that there exists a divisor m' which needs to be at least as big as the plaintext range and $s := \log_{m'}(m)$ specifies the security parameter. Thus, we can express the ciphertext length by $\#splits \cdot s \cdot (b_{\#sensors} + b_{data})$. As each node uses the same key, no identifiers need to be transmitted but one counter.

CMT must transport all the IDs which were involved in the encryption process. At each aggregator N this results in the concatenation of the ID list within $ST(N)$. The usage of a keystream generator allows to the ciphertext to be of minimal size $b_{\#sensors} + b_{data}$.

ME based mechanisms reduce the effect of transporting the IDs by performing $\#hops$ -hop encryption and decryption. The problem of carrying the involved IDs is limited to the IDs of the nodes which are at most $\#hops$ hops after an aggregator N . We denote this subtree of $ST(N)$ by $ST_{\#hops}(N)$. The encryption is again done by a keystream generator. As the IDs are not transported up to the sink, a counter is necessary here.

The scheme proposed in this paper requires only the transmission of the aggregated ciphertexts and two counters. It is therefore almost as data efficient as simple hop-by-hop encryption while requiring less operations (only aggregation instead of encryption and decryption) and providing a higher security (aggregator nodes are no longer able to decrypt the incoming ciphertexts).

Table 2 shows the parametrized cost caused by a transmitting node during the aggregation phase. For the sake of simplicity, we assume that every node is a sensor node. While some schemes have a constant data overhead for every aggregation level, the effort of other schemes like CMT heavily depends on the network structure.

To give an impression on how much the considered schemes differ in the message size, we illustrate this on some concrete examples. Similar to the example discussed in the motivation, we consider networks with a balanced tree structure. That is the network has h aggregation levels and each node (except of the leaves) has an outdegree of d . To examine how much the message sizes can maximally vary, we assume that no nodes are silent.

In Figure 7, we display the results for the case $h = d = 3$ (39 sensor nodes). It shows that the size of the transmissions made in TAUK is only slightly bigger compared to HbH but about 4 times smaller at aggregation level 1 and about 3 times smaller at aggregation level 2 compared to CMT.

Figure 8 shows for the case of a tree of height $h = 5$ and outdegree $d = 4$ (1364 sensor nodes), that the difference in the transmission sizes are even more extreme. The number of bits which need to be send in CMT are of magnitudes bigger compared to TAUK.

Concluding, we see that the use of CMT or other schemes which need to send the node IDs are rather impractical for large scale WSNs. In these cases, schemes with a low data overhead like our proposed scheme are more promising.

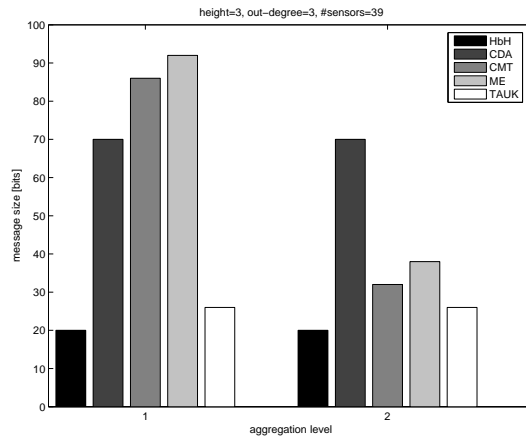


Figure 7: Comparison of the maximum transmission sizes in respect to the aggregation level for a $h = d = 3$ tree.

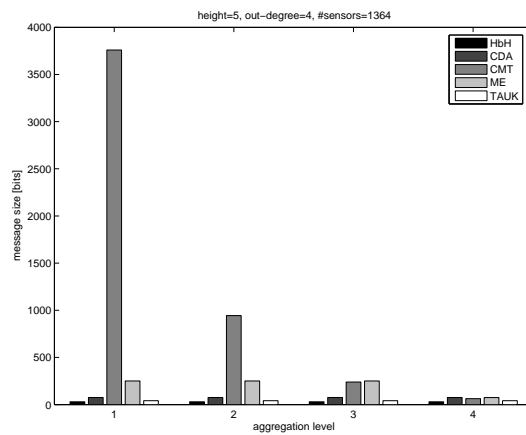


Figure 8: Comparison of the maximum transmission sizes in respect to the aggregation level for a $h = 5$ and $d = 4$ tree.

Security Analysis

We consider attackers who aim to gain knowledge of the aggregation of as many measurements as possible made within the network. Attackers can control the communication in the network. She can eavesdrop, stop, and alter the outgoing messages of any node but not for all nodes at the same time. We refer to this kind of attackers as cryptographic attackers. In addition to controlling the communication, an attacker might physically capture some nodes to read out the confidential data stored on them. We name such attackers physical attackers.

For a security analysis, one can ignore the attacker's capability of changing messages as this does not yield any new information. More formally, assume that a node N sends at time frame t a message C_N^t , that is a ciphertext, towards its predecessor N' but this message is changed to \tilde{C}_N^t by the attacker. N' receives \tilde{C}_N^t (instead of C_N^t), aggregates it with the other ciphertexts, and forwards the result $\tilde{C}_{N'}^t$ (instead of $C_{N'}^t$) to its predecessor. Because of $\tilde{C}_{N'}^t - C_{N'}^t = \tilde{C}_N^t - C_N^t$ the impact of changing C_N^t has on other messages yields no new information to the attacker.

For the comparison we assume that the deployed cryptographic primitives like the encryption function in HbH and the keystream generators used in CMT and ME are secure.

Security upshot

The security analysis shows that the considered instantiation is vulnerable against some attacks which do not apply for the other schemes. Obviously it does not reach the same security level as its competitors. However, recall that the primary goal for the considered instantiation was to increase the lifetime while still providing some reasonable security. Some of the weakness are consequences from these choices, e.g., to use a network wide dummy values R^t to allow key refreshment without any additional data exchange.

If the provided security level meets the requirements of the concrete scenario depends on the attacker cost model (Section ??). Recall the arguments that an attacker will likely not invest more time and resources into attacking the network than it would require to build her own. In this context, we stress that none of the attacks provide a total break "for free". The ciphertext-only attack is certainly the cheapest one but provides only the aggregation of plaintexts over several time frames, making the value for an attacker unclear. The known-plaintext attack requires to eavesdrop *each* node at least r times to reveal the corresponding plaintexts. In particular the second condition could have the practical implication that an attack at reasonable costs requires to attack the network level by level, starting from the leaves. The physical attack requires the corruption of only the highest level (plus one successor) to break the whole network. Thus, in scenarios where corrupting nodes is feasible *and* the specified nodes are easily reachable, we would not recommend to use TAUK without any further protection. In other scenarios, the provided security level might be enough to fulfill the requirements, especially when the user is aiming for a long-lived WSN.

Routing Flexibility

Next, we discuss the flexibility and robustness properties of the available CDA solutions and their recommended key pre-distribution schemes.

A WSN representing the network model which we introduced requires some flexibility with respect to changing routes for the transmission of convergecast traffic. This comes mainly due to the need of i) exhausting nodes, as well as the conceptual mean of ii) aggregator node election per time frame.

In case of using **HbH** we differentiate between HbH with pairwise keys and HbH with groupwise keys. In the first case the architecture provides no routing flexibility at all. With groupwise keys HbH encryption achieves routing flexibility at the cost of almost no system security. Regarding the degree of routing flexibility we observe that **CMT** with a randomized pairwise key pre-distribution as well as the single symmetric key based **TAGK** are the most promising ones. However, they either come at the costs of reduced system security or an unacceptable data overhead during the aggregation phase.

Since both **ME** and **TAUK** require some keys stored on the path to the sink, this approach loses routing flexibility and ends up in the requirement of static routing. These restrictions can be relaxed to some extent by storing the particular keys at several nodes at the same aggregation level. However, a more detailed analysis is out of scope of this work.

5 RoK: A Robust Key Pre-distribution Protocol for Multi-Phase Wireless Sensor Networks

A Robust Key Pre-distribution Protocol for Multi-Phase Wireless Sensor Networks

This section describes, in detail, our novel Key Pre-distribution Protocol for Multi-Phase Wireless Sensor Networks proposal. It explains how nodes get configured and how they establish secure channels. We then evaluate the security of our scheme and compares it with the security of the RKP scheme. This evaluation is performed by simulations and analytically. It also discusses some issues such as counterfeiting protection. We also compare our proposal to existing schemes.

Overview

Because a sensor is battery-powered, it has a limited lifetime. Very often, the sensor's lifetime is much smaller than the lifetime of the whole network. In order to assure a good network connectivity, new sensors need to be deployed periodically to make up for disappearing ones.

In our scheme, we assume that new sensors are deployed at regular epoch that we call *generations*. The time between two consecutive generations is called *generation period*. We also assume that a sensor lifetime is bounded by a given number, Gw , of generations. We refer to Gw as the *generation window*. In other words, a sensor deployed at generation j will run out of power before generation $j + Gw$. For simplicity, we assume in the rest of this document that the generation period is the time unit, i.e. that the generation period is set to 1.

Newly deployed nodes should be able to establish secure links with previously deployed nodes otherwise connectivity will not be provided. In our scheme, a node deployed at generation j can establish a secret channel with any other sensor deployed in the time period $[j - k, j + k]$, where k is a integer. Note that we assume, for simplicity, in this document that $k = Gw$. Therefore, since all nodes deployed before generation $j - Gw$ have died, our newly-deployed sensor can establish a key with any other node of the network. Note that, in some scenario, it might make sense to choose a value of k that is smaller than Gw . In this case, the newly deployed node will only be able to establish a secure channel with a subset of the network sensors.

Our scheme is similar to the key pre-distribution scheme described by Eschenauer and Gligor [7]. However as opposed to this scheme, we assign to each sensor A two different key rings: FKR_A and BKR_A , where a key ring is a subset of a pool of keys, namely $FKR_A \subset FKP$ and $BKR_A \subset BKP$.

These key rings are respectively called the *forward* and the *backward* key ring. Similarly, the key pool are called the *forward* and the *backward* key pool. The pools of keys are refreshed at every generation.

The rest of this section describes how the two key pools FKP and BKP are built and refreshed, how the key rings are assigned to nodes and how nodes establish secret keys.

Key Pools Generation

In the *RKP* scheme [7], the key pool is composed of random keys that do not evolve with time. In contrast, in our scheme, the key pools evolve with time: they are updated at each generation.

The Forward key pool

The forward key pool is initiated with $P/2$ random keys. At each generation, each key is updated by hashing the current key with a secure hash function, $h : \{0, 1\}^* \rightarrow \{0, 1\}^{160}$, such as SHA1 [19].

More precisely, the forward key pool at generation 0 (i.e. when the network is first deployed) is defined as follows:

$$FKP^0 = \{fk_1^0, fk_2^0, \dots, fk_{P/2}^0\}$$

where each fk_i^0 is randomly generated. We call f keys the keys of the forward key pool.

At generation $j + 1$, the f keys are refreshed as follows:

$$FKP^{j+1} = \{fk_1^{j+1}, fk_2^{j+1}, \dots, fk_{P/2}^{j+1}\}$$

Table 3: Summary of Notation

Gw	generation window
n	last generation of the network
A	sensor A
$kr_A^j = (FKR_A^j, BKR_A^j)$	key ring of A at gen. j
FKR_A^j	Forward key ring of A at gen. j
BKR_A^j	Backward key ring of A at gen. j
m	key ring size
FKP^j	Forward key pool at gen. j
BKP^j	Backward key pool at gen. j
P	key pool size
$fk_t^j \in FKP^j$	t-th fkey at gen. j
$bk_t^j \in BKP^j$	t-th bkey at gen. j
h	secure hash function $h : \{0, 1\}^* \rightarrow \{0, 1\}^{160}$
f	hash function $f : \{0, 1\}^* \rightarrow \{0, 1\}^{\log_2(P)}$
RKP	key management defined in [7]
RoK	our scheme

where $fk_t^{j+1} = h(fk_t^j)$, and

$$FKP^j = \{fk_1^j, fk_2^j, \dots, fk_{P/2}^j\}$$

defines the forward key pool at generation j.

The Backward key pool

The backward key pools, composed of backward keys, are generated using a hash-based Lamport chain [10]. If we assume that the network is deployed at generation 0 and will last, at most, until generation n , we start by generating the key pool of generation n . The bkeys (backward keys) at generation n , i.e. the last generation of the network, are initialized with random values i.e.:

$$BKP^n = \{bk_1^n, bk_2^n, \dots, bk_{P/2}^n\}.$$

At generation j, the bkeys are refreshed as follows:

$$BKP^j = \{bk_1^j, bk_2^j, \dots, bk_{P/2}^j\}$$

where $bk_t^j = h(bk_t^{j+1})$ and

$$BKP^{j+1} = \{bk_1^{j+1}, bk_2^{j+1}, \dots, bk_{P/2}^{j+1}\}$$

defines the backward key pool at generation j + 1.

That is, the bk_t^j key of generation j is obtained from the key bk_t^{j+1} of generation j + 1, using the hash function h .

Key rings assignment

This section describes how the backward and forward key rings are assigned to each sensor.

Each node is configured with $m/2$ subkeys from the backward and forward key pools. The subkeys assigned to a given node A deployed at generation j are selected using a pseudo-random function. More specifically, the first subkey of the forward key ring will be the subkey with index $f(id_A||1||j)$ of the forward key pool, where $f(\cdot)$ is for example a hash function with range $[1; m]$. Similarly, the first subkey of the backward key ring will be the subkey with index $f(id_A||1||j)$ of the backward key pool. The second subkey of the forward key ring will be the subkey with index $f(id_A||2||j)$ of the forward key pool. Similarly, the second subkey of the backward key ring will be the subkey with index $f(id_A||2||j)$ of the backward key pool and so on.

More formally, node A is configured with a key ring, $kr_A^j = (FKR_A^j, BKR_A^j)$, defined as follows:

$$FKR_A^j = \{fk_t^j \mid t = f(id_A \parallel i \parallel j), i = 1, 2, \dots, m/2\}$$

$$BKR_A^j = \{bk_t^{j+Gw-1} \mid t = f(id_A \parallel i \parallel j), i = 1, 2, \dots, m/2\}$$

and m is the size of the whole key ring.

Note that all the key rings are strictly bound to the deployment generation of the owner sensor. As a result, nodes need to be loosely time-synchronized.

Note that node A can only update its key ring kr_A^j , for the generations i between j and $j + Gw$. In fact since it cannot compute the b keys for the generations after generation $j + Gw$, it cannot update kr_A beyond generation $j + Gw$. Furthermore, since it cannot recover any f keys for the generations before generation j , it cannot compute kr_A for the generation before j . As a result, the lifetime of its key ring is limited. As we will see later, this is an essential aspect of our scheme since it considerably limits the power of the attacker: an attacker that corrupts a node will only be able to use its keys for a limited period of time.

Establishing a secure channel

When a sensor A is deployed at generation j , it initiates the neighbors discovery procedure by broadcasting a message that includes its identifier, id_A , and its generation.

All the nodes that receive its message are able to reconstruct the list of the key indexes in kr_A and identify the keys that they have in common.

For example, a neighbor node B can construct the set $\{t \mid t = f(id_A \parallel l \parallel j), l = 1, 2, \dots, m/2\}$ of the key indexes and check if it shares at least one index t with A . The B node then replies with its identifier, id_B , and its generation. As a result, A identifies the list t_1, t_2, \dots of all the key indexes in common.

Both A and B then calculate their *overlapping generations*, that is the set of the generations in which they can communicate. Formally, if A is deployed at generation j and B was deployed at generation i with $i \leq j$, their overlapping generations will be all the generations in the interval $[j, i + Gw]$.

Notice that if A and B have in common the key indexes t_1, t_2, \dots, t_z , then both of them compute all the f keys $\{fk_\tau^j \mid \tau = t_1, t_2, \dots, t_z, j = j, \dots, i + Gw - 1\}$ and all the b keys $\{bk_\tau^\gamma \mid \tau = t_1, t_2, \dots, t_z, \gamma = j, \dots, i + Gw - 1\}$.

A and B can then compute their secret key as follows:

$$k_{AB} = h(fk_{t_1}^j \parallel bk_{t_1}^{i+Gw-1} \parallel fk_{t_2}^j \parallel bk_{t_2}^{i+Gw-1} \parallel \dots \parallel fk_{t_z}^j \parallel bk_{t_z}^{i+Gw-1})$$

In this formula, the forward keys fk_x^j provide forward security: The forward keys of corrupted nodes, that happen to have the same key indexes that those used to compute k_{AB} , are only useful if they are corrupted before generation j . The backward keys bk_x^{i+Gw-1} provide backward security: The backward keys of corrupted nodes, that happen to have the same key indexes that those used to compute k_{AB} , are only useful if they are corrupt after generation i .

The key k_{AB} can then be used to established a secure channel between sensors A and B .

At each new generation, each node should erase their forward keys of the previous generation. By doing this, the f keys of the previous generations not yet retrieved by the adversary cannot be compromised any more: all nodes will have the f keys of the new generation and none of the nodes that can be captured will maintain a copy of the previous f keys. As a result, all the secure channels established with f keys of previous generation j cannot be compromised anymore. As we will see in Section 5, this significantly increases security.

Example

This section illustrates our protocol with an example (see Figure 9). Let's assume two sensors A and B , deployed respectively at generation j and $j+2$, with a generation window Gw of 4. The overlapping generations are therefore $j + 2, j + 3$ and $j + 4$. Let's also assume that A and B share the keys t_1 and t_2 (that is $t_\tau = f(A \parallel l_{A\tau} \parallel j) = f(B \parallel l_{B\tau} \parallel j + 2)$ for some $l_{A\tau}$ and $l_{B\tau}$, with $\tau = 1, 2$).

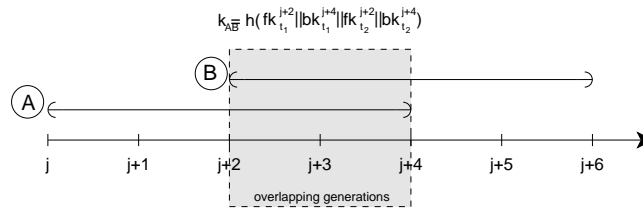


Figure 9: Key establishment: A and B share the keys t_1 and t_2

A and B can then compute the common secret key

$$k_{AB} = h(fk_{t_1}^{j+2} || bk_{t_1}^{j+4} || fk_{t_2}^{j+2} || bk_{t_2}^{j+4}).$$

When generation $j + 3$ arrives, A and B update their forward key ring by hashing all their f keys. They must also erase from memory their forward keys of the generation $j + 2$. Note that the keys used by node A are only valid between generation j and $j + 4$. Similarly, the keys used by node B are only valid between generation $j + 2$ and $j + 6$.

Security Evaluation

Objectives

Our scheme improves the security of the RKP scheme by limiting the lifetime of the key pools and refreshing the pool subkeys. With RKP , an attacker that corrupts enough sensors can reconstruct a large part of key pool and compromise many of the established or upcoming links. Since the key pool does not change, newly deployed nodes are configured with some corrupted subkeys. In other words, the security degrades with time.

In contrast, in our scheme, the key pools evolve with time. As a result, since newly deployed nodes are configured with fresh subkeys, an attacker has to constantly corrupt nodes and be very aggressive if he wants to eavesdrop on the established communications. If the attacker operates only during a limited period of time, the network will automatically self-heal when the attacker stops compromising nodes.

The goal of this section is to evaluate the security of our proposal and compare it with the security of the RKP scheme.

We evaluate the security of these two schemes by evaluating the number of channels that get *indirectly* corrupted when x nodes are compromised (and their keys are disclosed). A channel, between nodes A and B , is said to be indirectly corrupted when neither A nor B have been corrupted, but the adversary has collected all the backward and forward sub-keys that A and B have in common. These sub-keys have been collected by compromising other nodes.

We evaluate the two following ratios:

1. R_{active} is the ratio of the number of *indirectly* corrupted *active* channels over the total number of active channels. A channel is active when both of its ends are still alive. An attacker that corrupts an active channel can decrypt all the messages that are sent of the link, and can also inject bogus messages by impersonating one of the communication nodes.
2. R_{total} is the ratio of the total number of *indirectly* corrupted channels over the total number since the beginning of the network, over the total number of channels that have been established since the birth of the network. An attacker that corrupts a non-active channel, can decrypt the messages that were sent over these channels, if these messages were recorded. However, these messages are old and might not be valuable anymore. Furthermore, an attacker that corrupts a non-active node cannot inject bogus messages, since the secret key it has recovered has expired and cannot be used anymore to send new messages. For these reasons, we believe the R_{active} is more important. However, for completeness, we evaluate both ratios.

We first evaluate these ratios for both schemes, RKP and RoK , by simulation. This allows us to consider elaborate and complex scenarios. As detailed in the following section, we consider different types of attackers. We then compute analytically the ratio R_{active} . We believe this is also useful for a network designer to

understand and control the security of its network. He can, for example, compute for some sets of parameters, the security of its network (i.e. fraction of corrupted links). Alternatively, he can set the maximum acceptable fraction of corrupted links, and use our formulas to derive the network parameters (i.e., for example, the sensor node lifetime).

Evaluation by Simulation

Simulation Set-up

We have simulated our scheme, referred to as *RoK*, and the Eschenauer and Gligor scheme, referred to as *RKP*.

To simplify the security analysis, we modeled the network as a grid of sensors of size 400. We assumed that the number of neighbors, i.e. of nodes in the communication range, of each sensor is constant and equal to four.

When nodes are deployed, they establish a secure channel with their four neighbors, using all the sub-keys they share. If the intersection of the key rings of two neighboring nodes, *A* and *B*, is empty, node *A* (resp. *B*) checks whether any of its 2-hop neighbors shares at least a subkey with *B* (resp. *A*). If this is the case, this 2-hop neighbor is used as a proxy. If not, *A* extends its search to its 3-hop neighbors. This algorithm proceeds until a node that shares a subkey with *B* (resp. *A*) is found. This node is then used as a proxy.

In order to be fair, we used the same memory requirement for both schemes. The *RKP* scheme used a key pool of size P and each node was configured with a key ring size of m . In contrast, the *RoK* scheme used a forward key pool of size $P/2$, a backward key pool of size $P/2$ and each node was configured with two key rings, each of size $m/2$. Since the connectivity (the probability of two neighbors to establish a key) depends on the size of the key pool, the connectivity performance of our scheme, that uses a key pool of size $m/2$, is weaker than the connectivity achieved with the *RKP* scheme. However since our scheme uses proxies, the overall network connectivity achieved with these two schemes are similar. In our simulations, P was set to 20000 and m to 500. These parameters are similar to the ones used in [7] and [8].

We assumed that each generation is composed of 10 time slots. At generation 0, N nodes are deployed. We simulated nodes expiration by assigning to each node a random expiration date, chosen according to a Gaussian distribution with mean $\frac{Gw}{2}$ and with standard deviation $\frac{Gw}{6}$. Where not differently specified, we used a generation window $Gw = 10$. In order to simplify the security analysis, we assumed that the network topology does not change over time: At each generation, expired nodes are replaced with new ones, configured with fresh keys. The new nodes establish secure channels with their four neighbors, using the common keys of the least and the greatest overlapping generations (respectively for the *f*keys and for the *b*keys).

Attacker model and strategy

It is important to underline that the *f*keys and *b*keys can be computed separately. For a captured node of generation j , *f*keys with the same subscripts of all future generations after j (even after $j + Gw$) and *b*keys with the same subscripts of all generations before $j + Gw$ can easily be computed, stored and exchanged among captured nodes. In this way, the attacker may create a table of keys that belong to various generations. This table of keys might be used to compromise some additional secure links of various generations (including the past ones).

In our simulations, the attacker uses the strategy described above: at each time slot, he corrupts x active nodes and updates the previously described table with their backward and forward keys. He then uses this table to corrupt links (alive or past).

We considered two different types of attackers: the *eager*, and the *temporary* attackers. An eager attacker keeps compromising nodes at constant rate, from the deployment of the first generation of sensors to the end of the network. In contrast, temporary attacker compromises nodes during a limited period of time, from generation 5 to generation 14 in our simulations.

We then counted, at each time slot, the number of compromised links (i.e. links that were secured using forward and backward compromised keys that are known to the attacker), and computed the ratios R_{active} and R_{total} , described in Section 5.

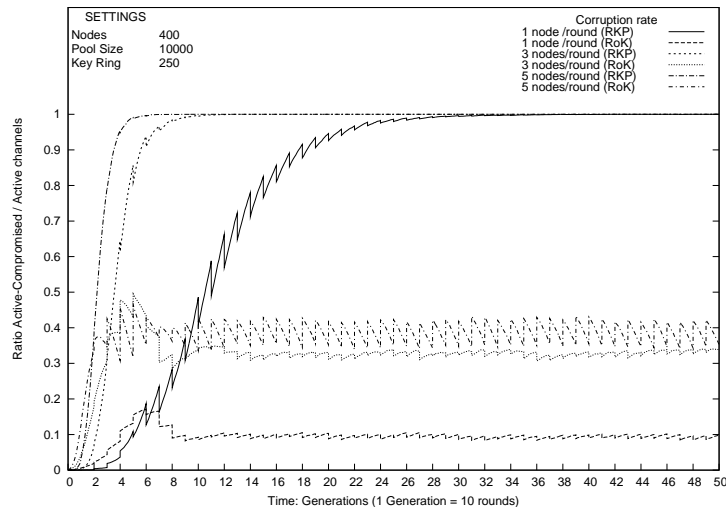


Figure 10: R_{active} . Adversary compromises nodes at constant rate (constant attacker)

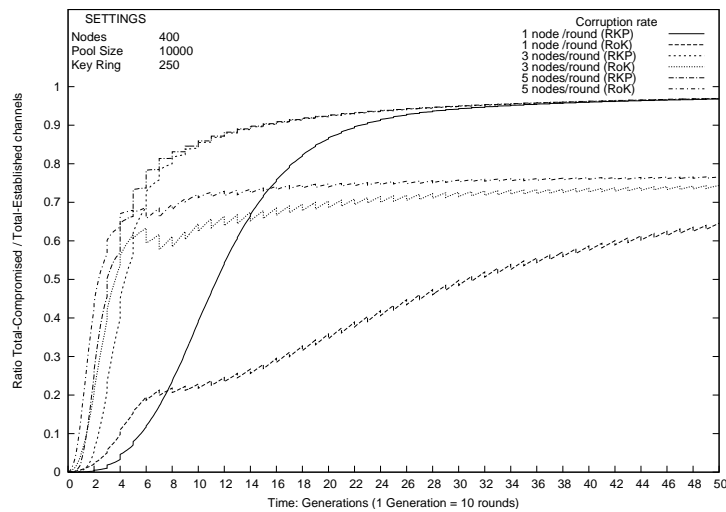


Figure 11: R_{total} . Adversary compromises nodes at constant rate (constant attacker)

In the next section we report the results of our simulation. All the simulations were repeated 25 times and the results report the average values.

Simulation results

This section presents the results of our simulations for the different types of attackers.

Eager Attacker model: Figure 10 and 11 display, respectively, the R_{active} and R_{total} ratios with an eager attacker. These figures present the simulation results for three different corruption rates: 1, 3 and 5 nodes at every time slot, i.e. respectively 10, 30 and 50 nodes per generation. Notice that we only consider *indirectly* corrupted channels, i.e. channels established between non-compromised nodes.

The main observation is that while the *RKP* scheme has ratios of compromised channels with an almost constant growth, the ratios obtained with the *RoK* scheme are bounded by a threshold value, that depends on the value x . Secondly, from Figure 10 it can be observed that with *RKP*, the R_{active} ratio reaches 1 in a really short time (around 30 generations for corruption rate 10 and 8 generations for corruption rate 50), while with *RoK* the ratios are much smaller and always below 1/2.

The oscillations in the plots are due to newly-deployed nodes: the new nodes establish secure channels with keys not yet compromised by the adversary, suddenly reducing the ratio compromised/active channels. In the

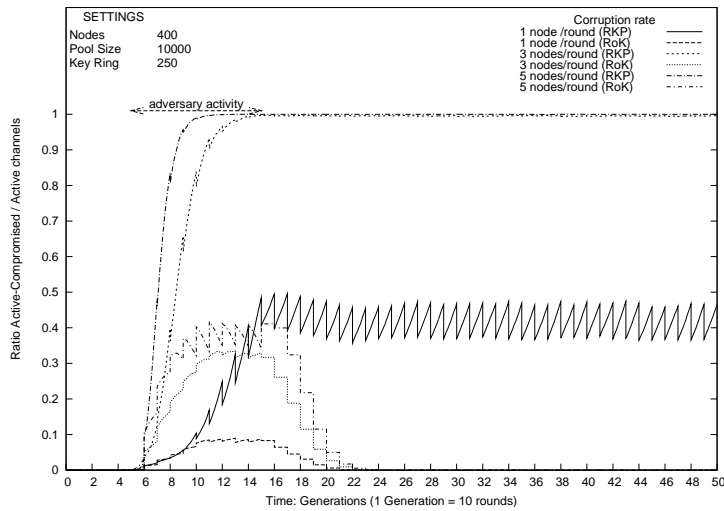


Figure 12: R_{active} . Adversary compromises from generation 5 to generation 15 (temporary attacker)

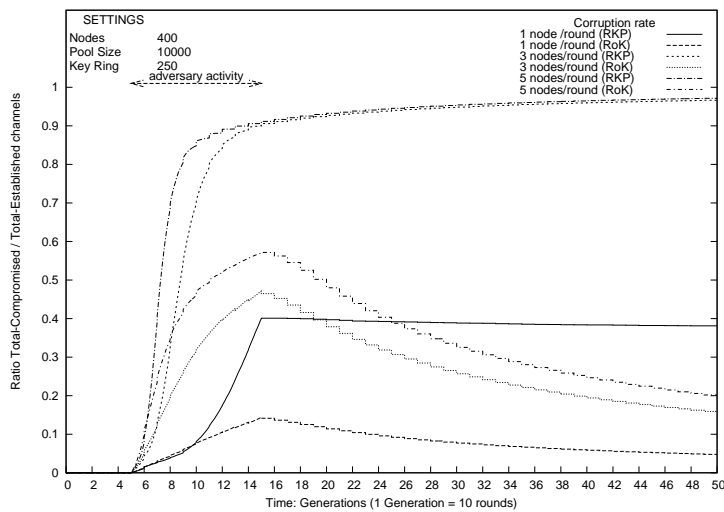


Figure 13: R_{total} . Adversary compromises from generation 5 to generation 15 (temporary attacker)

RKP scheme, the oscillation amplitude reduces as the adversary keeps acquiring keys. With the *RoK* scheme the amplitude is almost constant, for the whole life of the network, since keys are refreshed at every generation.

We can also observe that the plots for the *RoK* scheme in Figure 10 have an increasing phase followed by a decreasing one. Furthermore, for all three corruption rates, the plots reach their maximum around the 5-th generation. This apparently unexpected behavior is due to the proximity of the deployment time: at generation 5, almost half of the nodes of the network were deployed at the first generation and almost all the keys that the adversary has collected can be used to exploit the channels of those nodes (since $Gw = 10$, the average lifetime of a node is $Gw/2 = 5$). From the 6-th generation on, most of the nodes are “young” and the channels were established with keys that were not compromised yet. As soon as the population of the nodes of the network stabilizes, the ratio of compromised channels also stabilizes.

Note that corruption ratio of the total number of established channels shown in Figure 11 never reaches 1 because of the directly corrupted channels that are not taken into account.

It can be observed in Figure 12 and Figure 13, that *RKP* outperforms our scheme in the first generations (i.e. from generation 0 to 5, when the network is mainly composed of nodes deployed at the first generation). In fact, since the connectivity achieved with our scheme is smaller, each link is secured with less sub-keys than with the *RKP* scheme and can therefore be compromised more easily. However, as new nodes are deployed in the network and new channels are established with fresh sub-keys, the performance of our scheme significantly increases with time.

Temporary Attacker Model: The results for the temporary attacker are collected in Figure 12 and in Figure 13. The interval of action of the attacker (from generation 5 to generation 14) is denoted with the label “adversary activity”.

We simulated a network with the same settings as the network used for the eager attacker, and the three corruption rate of 10, 30 and 50 nodes per generation.

Both figures illustrate the *self-healing* property of the proposed scheme: as soon as the adversary stops its activity, the ratio of the compromised channels starts decreasing as new generations of nodes are deployed. Figure 12 in particular shows that the network starts recovering as soon as the attack stops. It takes 10 generations for the network to fully recover.

The *RKP* scheme, instead, keeps a ratio of compromised channels greater than 0, even when the adversary stops its activity. Moreover, when the attacker corrupts nodes with a rate greater than 30, the ratio approximates 1 and never decreases. Figure 13 shows that with *RoK* the ratio R_{total} decreases when the adversary stops; for the *RKP* scheme the ratio keeps increasing toward 1.

In summary, these simulation results show that our *RoK* scheme outperforms the *RKP* scheme. The number of active compromised nodes can be reduced by a factor of 10. For example, when 10 nodes are compromised at each generation, the ratio of active compromised channels is reduced from 100% to 10%. The security of our proposal is based on the assumption that it takes time for an adversary to physically compromise sensors and get their keys. Since in our proposal, the key vulnerability period, i.e. the time that attacker has to corrupt useful keys is reduced, security is improved significantly.

Analytical Evaluation

R_{active} computation

In this section, we compute analytically R_{active} , the fraction of active indirectly compromised links when x nodes get compromised. We assume that most of the links are established directly, i.e. each node shares at least one sub-key with each of its neighbors.

As explained in [8], the probability that two nodes share i sub-keys is defined by:

$$p_i = \frac{\binom{P}{i} \binom{P-i}{2(m-i)} \binom{2(m-i)}{(m-i)}}{\binom{P}{m}^2} \quad (42)$$

where m is the key ring size and P the key pool size.

Therefore the probability that two nodes share at least one sub-key is defined by $1 - p_0$ i.e:

$$p_{connect} = 1 - \frac{\binom{P}{2m} \binom{2m}{m}}{\binom{P}{m}^2} \quad (43)$$

We need to choose m and P such that $p_{connect} \sim 1$, i.e., for example, $m = 250$ and $P = 10000$.

Furthermore, a secure link will be created by, on average, nk sub-keys, where nk is defined as follows:

$$nk = \sum_{i=1}^{+\infty} p_i \cdot i \quad (44)$$

Note that for $m = 250$ and $P = 10000$, $nk \sim 6$.

We assume in our scheme that n nodes are deployed at bootstrap time (generation 0, G_0). Nodes that die during generation G_i are immediately replaced at generation G_{i+1} . The time between two deployments is called generation *period*.

We also assume that each sensor has a lifetime that follows a distribution, for example Gaussian, with a density probability $f(x, \mu, \sigma)$, where μ defines the mean lifetime (the unit is the generation period) and σ the standard deviation. The density probability $f(x, 3, 1/6)$ indicate that the mean lifetime of a sensor is 3 generations and its standard deviation is 1/6 generation. The lifetime of a node is bounded by Gw , i.e if a node is deployed at generation G_j , it will, for sure, be dead after generation $G_j + Gw$.

It can easily be shown that the number of nodes that died at generation G_{j-1} , and redeployed at generation G_j , with $j > 1$, is defined by:

$$X_j = \sum_{i=1}^{Gw} X_{j-i} \int_{i-1}^i f(x, \mu, \sigma) dx \quad (45)$$

with $X_0 = n$.

If the sensor lifetime follows a Gaussian distribution $f(x, \mu, \sigma)$, we have:

$$X_j = \sum_{i=1}^{Gw} X_{j-i} \cdot (F(i; \mu, \sigma) - F(i-1; \mu, \sigma)) \quad (46)$$

where

$$F(x; \mu, \sigma) = \frac{1}{2} \left[1 + \operatorname{erf} \left(\frac{x - \mu}{\sigma\sqrt{2}} \right) \right]. \quad (47)$$

We can demonstrate that, for j enough large, X_j becomes constant: the network stabilizes and, then, the average number of nodes to re-deploy is constant.

If we denote, $X^{(i)}$ the number of active nodes deployed i generations ago, we can have:

$$X^{(i)} = X_j \int_i^{Gw} f(x, \mu, \sigma) dx \quad (48)$$

$$= X_j (F(Gw; \mu, \sigma) - F(i; \mu, \sigma)) \quad (49)$$

From Formula 49 we can compute that, on the average, a node picked at random from the network has age i , with $0 < i < Gw$, with probability:

$$p(i) = \frac{1}{n} \cdot X^{(i)} \quad (50)$$

The average age $E[\alpha]$ of nodes is therefore defined as:

$$E[\alpha] = \sum_{i=0}^{Gw} i \cdot \frac{1}{n} X^{(i)} = \frac{1}{n} \sum_{i=0}^{Gw} i X^{(i)} \quad (51)$$

Let the average number of captured nodes be x during a period. Since each node contains m keys, the probability that a given key has not been compromised is $(1 - \frac{m}{P})^x$. Therefore if a given link was secured with i sub-keys, the probability that this link is compromised is $(1 - (1 - \frac{m}{P})^x)^i$.

According to [8], we defined the average probability that a link is **indirectly**⁴ corrupted when x nodes are corrupted as:

$$plc = \sum_{i=1}^m \left(1 - \left(1 - \frac{m}{P} \right)^x \right)^i \quad (52)$$

Therefore, with the *RKP* scheme, the probability that an active link is compromised at generation j is defined as follows:

$$plc_{RKP}(j) = \sum_{i=1}^m \left(1 - \left(1 - \frac{m}{P} \right)^{j \cdot x} \right)^i p_i \quad (53)$$

with p_i defined by Formula 42. Note that this probability increases with time.

⁴ Remember that a link is indirectly corrupted when none of its end-points have been corrupted, but the adversary collected all the keys used to establish the link, recovering those keys from compromised nodes

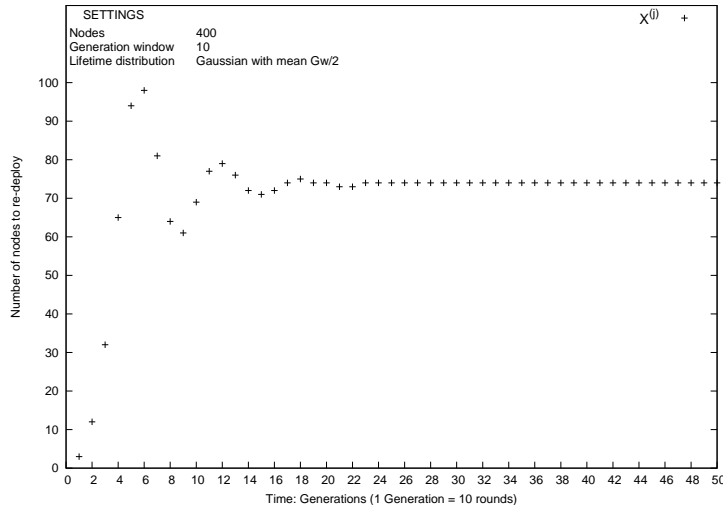


Figure 14: Average number of nodes to re-deploy every generation. For every generation is reported the expected number of nodes died in that generation.

With *RoK*, the probability that an active link is compromised at generation G_j depends on the generation of the “oldest” node that established the channel. So, we have to evaluate the average age of the oldest between two random nodes of the network. Let be X and Y two independent random variables with the same distribution defined by Formula 49. Let be $Z = \text{MAX}(X, Y)$. The expected value of Z is the average time at disposal of the adversary to indirectly corrupt a random active channel.

The probability that Z assumes the value j , where $0 < j < Gw$ is defined by:

$$P\{Z = j\} = P\{(X = j \wedge Y \leq j) \vee (X \leq j - 1 \wedge Y = j)\} \tag{54}$$

$$= p(j) \cdot \sum_{k=0}^j p(k) + \sum_{k=0}^{j-1} p(k) \cdot p(j) \tag{55}$$

$$= p(j)^2 + 2 \left(p(j) \cdot \sum_{k=0}^{j-1} p(k) \right) \tag{56}$$

Therefore, $E[Z]$ can be computed as follows:

$$E[Z] = \sum_{j=0}^{Gw} j \cdot P\{Z = j\} = \tag{57}$$

$$= \sum_{j=0}^{Gw} j \left[p(j)^2 + 2 \left(p(j) \cdot \sum_{k=0}^{j-1} p(k) \right) \right] \tag{58}$$

Finally, the probability that an active link is compromised at generation j with our scheme is defined as:

$$pl_{CRoK}(j) \sim \sum_{i=1}^m \left(1 - \left(1 - \frac{m}{P} \right)^{x \cdot E[Z]} \right)^i p_i \tag{59}$$

Note that this probability is constant, since it does not depend on j .

Some numerical results

If $n = 400$, $m = 250$, $P = 10000$, $Gw = 10$, $x = 10$, $ord = 4$, the total number of links is around 800. Evaluating Formula 44, we obtain that $nk \sim 6$.

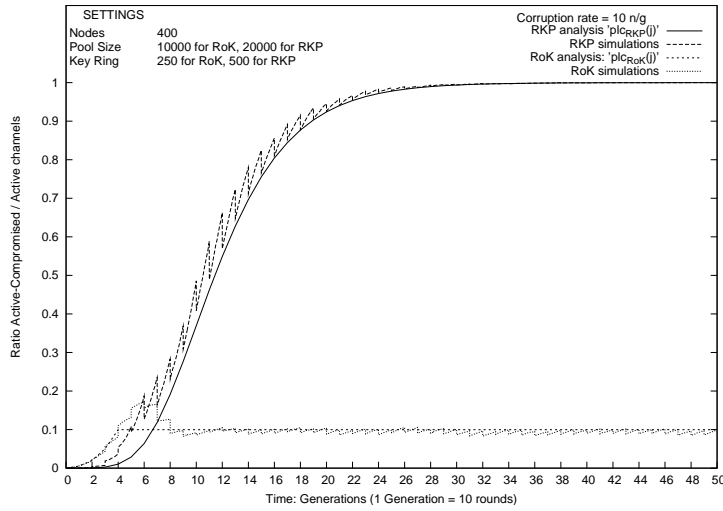


Figure 15: Comparison analysis and simulation results

Formula 46 shows that the average number of nodes to re-deploy every generation converges. This function is plotted in Figure 14. With the previous settings, the function converges to the value 72.

The distribution of the age of a node can be evaluated via Formula 50.

The expected age (Formula 51) is 2.5, while the expected age of the oldest between two random nodes (Formula 58) is 3.6.

Using Formulas 53 and 59, we obtain that $plC_{RKP}(15) \sim 0.75$, $plC_{RKP}(30) \sim 0.993$ and $plC_{RKP}(50) \sim 0.9999$. Furthermore, $plC_{RoK}(10, 30, 50) \sim 0.10$ (and 83 links are directly compromised). These results demonstrate that the *RoK* scheme reduces the numbers of compromised channels by a factor of 10. They also show that, with *RoK*, the ratio of compromised channels remain constant with time. In contrast, the number of compromised channels increases with *RKP*, i.e. security degrades with time.

Figure 15 compares the results obtained analytically with formulas 53 and 59 with the results obtained by simulations. This figure demonstrates that the results obtained analytically and by simulations are very similar. Note that in the simulations, it is assumed that the attacker corrupts nodes regularly within a generation. In the analytical model, we assume, for simplicity, that the attacker corrupts all the nodes at once i.e. at the beginning of each generation. This explains why the oscillations that are visible in the simulation results do not appear in the analytical results.

Discussions

Counterfeiting protection

Our scheme not only improves the security of deployed nodes but also makes node counterfeiting very difficult. Counterfeiting happens when an attacker generates a new node from the sub-keys it has compromised. With the *RKP* scheme, the attacker has to find an *id* such that it has compromised all the corresponding sub-keys (or at least the sub-keys it has in common with the nodes it wants to establish a secret channel with). If the attacker has compromised enough nodes, it will know many sub-keys and this operation might not be too difficult.

With our scheme, since sub-keys have limited lifetimes, node generation is much more difficult. As explained in section 5, two sensors *A* and *B*, deployed respectively at generation *i* and generation *j*, compute their secret key as follows:

$$k_{AB} = h(fk_{t_1}^j || bk_{t_1}^{i+Gw-1} || fk_{t_2}^j || bk_{t_2}^{i+Gw-1} || \dots || fk_{t_z}^j || bk_{t_z}^{i+Gw-1})$$

If the attacker wants to generate a new node *X*, let's say deployed at generation *x*, where $x < j$, in order to establish a secure link with *B*, it can only use the nodes corrupted in $j - x$ generations (i.e. from generation *x* to generation *j*) to perform his attack. The lifetime of the established link will then be $Gw - (j - x)$. There is

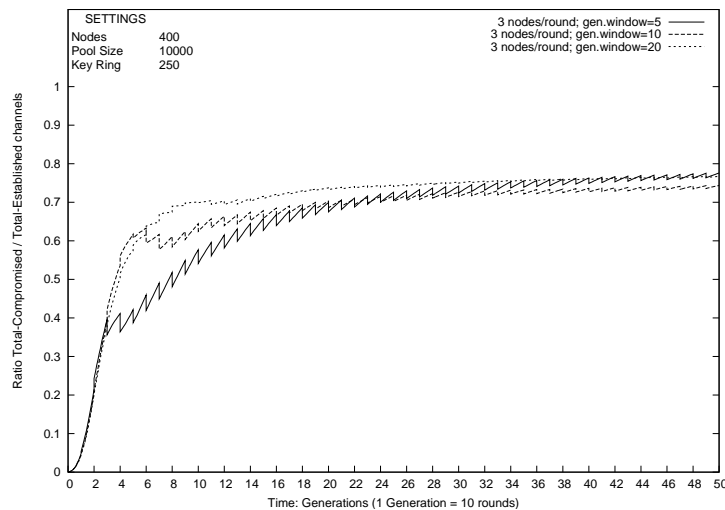


Figure 16: Security of *RoK* with different values of Gw . Ratio between Total-compromised and Total-established channels, for the whole lifetime of the network.

a clear trade-off here between the time at disposal of the attacker to perform his attack and the benefit it gains (i.e. the lifetime of the resulting established channels).

If $j - x$ is small, then the attacker has very limited time to collect the correct sub-keys, but the lifetime of the link might be large i.e. $Gw - (j - x)$. However if the attacker needs more time, i.e. if it decreases x , the duration of the established lifetime will be smaller. We believe that this is a nice feature of our scheme. An attacker needs to be very aggressive and corrupts many nodes very quickly if it wants to benefit from its attacks. Performing such aggressive attacks might not always be possible and requires very strong attackers.

Controlling the Security of a WSN

Formula 59 clearly shows that the security provided by our scheme depends on the value of Gw , or in other words, the refresh period of the sub-keys. This formula can be used to evaluate the security of a WSN that has predefined parameters. On the other hand, it can also be used to specify the network parameter Gw according to the administrator security goals and the attacker model. For example, the above example shows that if the attacker can corrupt up to 10 nodes per period (period being the unit of time), the network is composed of 400 nodes, each configured with 250 sub-keys out of 10000 sub-keys, and that the security goal is to make sure that less than 33% of the network is corrupted, then Gw must be set to 10 periods. In other words, the maximum lifetime of each node must be set to 10 periods (and the average lifetime will be 5) and new nodes must be regularly deployed in order to replace dying nodes. Similarly, if the security goal is less than 20% of corruption, Gw must be set to 5 periods. Equation 46 shows that, with the previous parameters, ~ 80 nodes die at every period and need to be replaced. An idea of the impact of the parameter Gw can be had from the Figures 16 and 17, where we compared the ratios introduced in Section 5, for a fixed and constant rate of corruption and varying Gw .

Comparison with Related Work

Key management is one of the core mechanism to provide security to WSN. Even though public key based solutions are being investigated in the literature ([11, 12, 13]), these solutions suffer from the CPU, memory and energy limitations of sensors. As a result, most of the existing proposals consider only the use of symmetric key cryptography.

One of the most popular approach, referred as *random key pre-distribution approach*, was proposed by Eschenauer and Gligor [7]. In this scheme, each node is configured with a key ring randomly selected from a larger pool of symmetric keys: they propose a model that relies on probabilistic key sharing among the nodes of a random graph. In [14], Di Pietro et al. provide some new results to study the scheme, providing a new model instead of the traditional one based on the Erdős-Rényi random graphs. The Random Key Pre-distribution

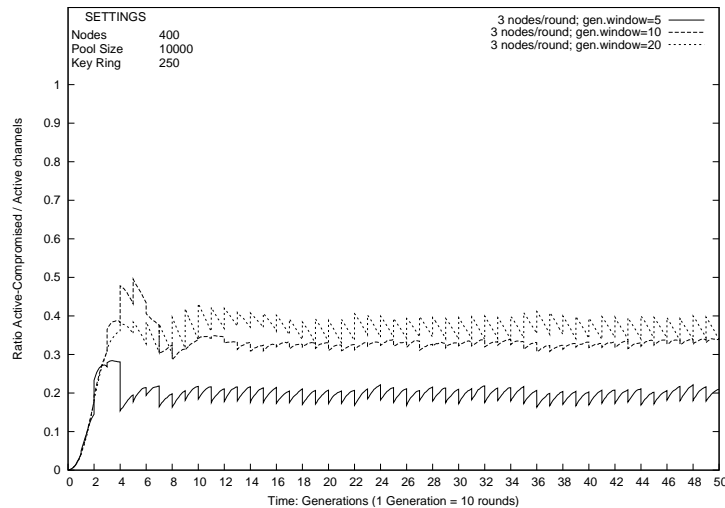


Figure 17: Security of *RoK* with different values of Gw . Ratio between Active-compromised and Active channels.

Scheme [8] of Chang, Perrig and Song is an extension of [7] and presents three proposals to strengthen security and improve resilience of the established link keys. In [15, 16], a cooperative pair-wise key establishment protocol is proposed, where a link between two nodes is reinforced using the cooperation of the common neighborhood.

While random key pre-distribution schemes are efficient and promising, the security that they provide degrades with time. Our proposal, *RoK*, solves this problem by periodically refreshing the key pool.

One work that takes in account the challenge to protect wireless sensor networks that can be deployed in different phases is [6]. The considered model is very similar to the one considered in this document: sensors are deployed in successive generations, when previously deployed sensors fail or when the capability of the existing network is determined to be insufficient. All the deployed nodes are able to establish secure channels with nodes of the next n generations. Unlike the previous works, this protocol is not based on a random pre-distribution of the keys. Every node is deployed with exactly one *generation key* that binds them to a specific generation; for all the other generations it can communicate with, it maintains a secret derived from an own unique random value and the corresponding generation key. Formally, if node A is deployed at generation j , it is configured with a unique random value R_A , the generation key gk_j and a set of secrets $S_{A,j+1}, S_{A,j+2}, \dots, S_{A,j+n}$, one for every of the n following generations; each secret is constructed as follows: $S_{A,i} = G_{gk_i}(R_A)$, where G is a keyed one-way hash function and gk_i is the generation key of generation i . Clearly sensor A cannot recover gk_i from $S_{A,i}$ and R_A . A node B of generation $j' \leq j + n$, upon receiving R_A will be able to construct $S_{A,j'}$, since it knows the key $gk_{j'}$. Nodes A and B , then, share the secret $S_{A,j'}$ that can use to establish a session key. Since each node only knows the generation key corresponding to its own generation, it is not able to construct the secrets for an arbitrary generation. Moreover, it cannot tamper with the communication between nodes of a different generation, and cannot impersonate a node Z of generation i , since it cannot compute $G_{gk_i}(R_z)$.

The security of the whole protocol is based on the assumption that it takes time for an adversary to physically compromise sensors and get the stored keys. Since the whole security of the scheme relies on the generation key gk_j , all sensors of generation j must erase this key as soon as the key-establishment protocol is over.

The security provided by this protocol is rather weak: since all the nodes of generation j have a copy of gk_j , it is sufficient for the adversary to compromise *one* of those nodes to corrupt all the communications of that generation.

With the *RoK* scheme, discovering the whole traffic of the network is a very difficult task, since the adversary needs to reconstruct the whole key pools. Moreover, *RoK* is more secure since the corruption of a single node has only a very limited impact on the security of the whole network.

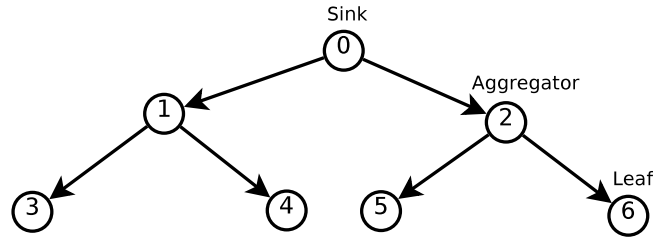


Figure 18: Node addressing in a tree-shaped network of 6 nodes, whereby both the degree and the height of the tree is 2

6 Implementations

Implementation details of TAUK

We implemented the TAUK for initial key-distribution both on the MicaZ and TelosB motes, which are a typical platform for WSNs. The MicaZ mote is equipped with 8-bit processor, while the processor employed in TelosB mote is 16-bit. The operating system employed in the implementation was TinyOS-2.0.2. The realization of TAUK requires a communication between the sensor nodes. Thus, there is a need for a well defined network model for a successful implementation of the TAUK. In the following subsection we briefly explain the network model, e.g. a model for node addressing, employed in this implementation.

Implemented Node addressing model

The key-distribution in the TAUK proceeds top-down from the sink node to the leaf nodes: The sink node selects a master key and distributes it to its successors such that the aggregation of the keys stored on the successors sum up to the master key. Since a tree-shaped network best suits to implement such a network traffic, we had to agree on a common addressing model that is suitable for a tree-shaped network structure. Please note that the tree-shaped network structure makes the modeling of the communication for key-distribution easier, but it is not necessary in general for a proper realization.

In this addressing model, the nodes are addressed by a unique tauk ID. The tauk ID of the sink node is set to constant 0, while the IDs of the aggregators and the leaf nodes are assigned such that each node has a unique tauk ID and the difference between two subsequent IDs is 1. Figure 18 shows an example of a tree-shaped network with 6 sensors with the described addressing model.

The main advantage of such an addressing model is that each node in the network can calculate the address of its direct neighbors only by using its own ID. The reason for this is that due to the regularity in the assigned addresses, the tauk IDs in the network may be represented in an integer sequence. This makes the additional communication overhead between the sensor nodes needless which would be required to discover the addresses of the neighbors, since TAUK assumes that each sensor node already knows its direct neighbors in the initialization phase. In this model, one may use the following functions to calculate the IDs of the direct neighbors of the sensor node N :

$$Pred(N) := \lfloor \frac{N_{ID} - 1}{h} \rfloor \quad (60)$$

$$Succ(N) := \{ID' | (N_{ID} \cdot d) + 1 \leq ID' \leq (N_{ID} + 1) \cdot d\}. \quad (61)$$

Thereby, N_{ID} denotes the tauk ID of a node N , while the degree and the height of the tree is denoted by d and h , respectively. For example, in the case of Figure 18, $Pred(6) := \lfloor \frac{6-1}{2} \rfloor = 2$ and $Succ(1) := \{3, 4\}$. In order to decide, if a sensor node is a leaf node, one can use of the IDs of the left and the right most sensor nodes of the tree. More precisely, if the ID of the node N is between the IDs of the left and right most leaf nodes, the node N is also a leaf node. Let assume that ID_{LN} and ID_{RN} denote the IDs of the right and the left most sensor nodes in the tree, respectively. One may use the following algorithm to find out, if the node N is a leaf

node or not:

$$isLeaf(N) := \begin{cases} 1 & \text{if } ID_{LN} \leq ID_N \leq ID_{RN} \\ 0 & \text{otherwise} \end{cases} \quad (62)$$

whereby

$$ID_{LN} := \lfloor \frac{d^h - 1}{d - 1} \rfloor \quad (63)$$

and

$$ID_{RN} := \lfloor \frac{d \cdot (d^h - 1)}{d - 1} \rfloor. \quad (64)$$

Integration of the proposed addressing model with existing addressing models

Within the UbiSec&Sens tool box several software components were developed, which may be used to build a full function WSN applications. In those components the roles and the routing paths are defined with a unique identifier that is the node ID in general. Since the addressing model defined in the previous subsection requires an ID for the realization as well, using the the same node ID may lead to inconsistencies. To solve this integration problem, we define a new tauk node ID being initialized during the compile time and used only during the boot strapping of a WSN for the key-distribution. As explained in the following subsection we employ for this a virtual tauk ID. Once the key-distribution is finished, tauk node IDs may be replaced by the real node IDs that are necessary for usage of other software components such as routing protocols or the identity manager.

The TAUK key-distribution is performed to provide sensor nodes with a unique key that is employed in the bi-homomorphic concealed data aggregation (CDA). A CDA based on TAUK requires a storage of some encrypted data, i.e. encrypted dummies, on distinct sensor nodes towards the sink. These encrypted dummies are necessary for a successful decryption of the concealed data on the sink in case of silent or exhausted sensor nodes and are different for each node. Thus the aggregation from sensors towards the sink should follow a certain path in the WSN. This means that the routing flexibility is restricted. However, this restriction may be relaxed by storing the encrypted dummies on several nodes which are on the possible paths to the sink. For this document the routing flexibility is out of scope. Thus, we point out that the specific CDA derivation based on double homomorphic encryption requires a static routing which may be achieved with the proposed addressing model and the existing routing protocols such as TinyLUNAR.

Data representation

Prior to explaining implementation details we introduce the type definitions and macros used in the implementation. The following types and macros are employed in this implementation:

- **TREE_DEPTH:** This macro is used to define the height of the tree, which is needed in the employed addressing model e.g. for determining the IDs of the right and left most sensor nodes in the tree.
- **TREE_DEGREE:** This macro is used to define the degree of the tree, i.e. the number of the successors of each sensor node.
- **TAUK_DATA_SIZE:** This macro defines the length of the plain texts in bytes to be encrypted. The length of the plain texts is needed to determine the length of the cipher texts and, thus, to determine the size of the individual keys need to be derived from a master key on each sensor node. Note that this implementation assumes that the size of the plain texts is smaller than 7. The reasons for this restriction are:
 - The events such as light, temperature, humidity, etc. that are monitored by the sensors of the nodes is representable in 2-Bytes and the employed aggregation operation keeps the size of the produced cipher texts constant independent from the number of the sensors in the network.

- Keeping the data size smaller than 7 makes the use of standard mathematical operations such addition, subtraction, division, etc. possible. This decreases the resource consumption in the implementation, as the implementation of a dedicated multi-precision integer arithmetic becomes unnecessary.
- **TREE_SIZE():** This *inline function* calculates the size of the network, which is the number of the sensor nodes except the sink node. It is necessary for determining the key and ciphertext sizes.
- **TAUK_KEY_SIZE:** This macro contains the length of the keys of the sensor nodes in bytes. It is calculated as follows:

$$TAUK_KEY_SIZE := \begin{cases} TAUK_DATA_SIZE + 2 & \text{if } TREE_SIZE() > 256 \\ TAUK_DATA_SIZE + 1 & \text{otherwise} \end{cases} \quad (65)$$

One may easily realize that the calculation of the key size in the aforementioned way yields a correct result, only if the size of the network is smaller than 65793 nodes. On the other hand, it is unlikely that a WSN application with more than 65793 nodes will ever be deployed.

- **neighbor_id_t:** This data type is used to represent the 16-Bit tauk ID of the sensor nodes. The zero is reserved for the sink node and defined as a constant in TAUK_SINK_ID.
- **tauk_modulus_t:** This data type is used to represent the *modulus* used in the key derivation, encryption, and decryption. Since it does not decrease the provided security level, the modulus in this implementation is always set to $2^{(TAUK_KEY_SIZE)-8} - 1$. Setting the modulus to a constant value improves the efficiency of the protocol, as the transmission of the modulus is not required any more.
- **tauk_dummy_t:** This data type is used to represent the *dummies* defined in the protocol description.
- **tauk_key_t:** This data type is used to represent the *keys*.
- **tauk_cipher_t:** This data type is used to represent the *cipher* of the bi-homomorphic encryption. It is needed for storing encrypted dummies.
- **tauk_key_msg_t:** This structure defines the messages changed between sensor nodes in the key-distribution phase.

```
typedef nx_struct tauk_key_msg{
    nx_uint8_t type;
    nx_uint8_t dummy[TAUK_KEY_SIZE];
    nx_uint8_t key[TAUK_KEY_SIZE];
} tauk_key_msg_t;
```

Depending on the platform encoding of the data may be different, namely, little or big endian. This is especially problematic, if a WSN consists of heterogeneous sensor nodes. In order to overcome this incompatibility problem, one should use independent data types. In TinyOS, independent data types are defined by the `nx_` prefix and the `tauk_key_msg_t` is defined in this work as an independent data type.

The `type` variable defined in this structure is used to denote the type of the message, which can be:

- tauk key message
- acknowledgment message.

A tauk key message, denoted by `TAUK_KEY_MSG`, is employed for disseminating the keys in the network. An acknowledgment message is sent from a sensor node to its predecessors on receiving a tauk key message and is denoted by `TAUK_ACK_MSG`. The field `dummy` is used to represent the dummies to be sent successor nodes. Finally, the `key` contains the master key, from which the keys for the sub-tree of the recipient are derived.

All of these macros and type definitions are defined in the `Tauk.h` file.

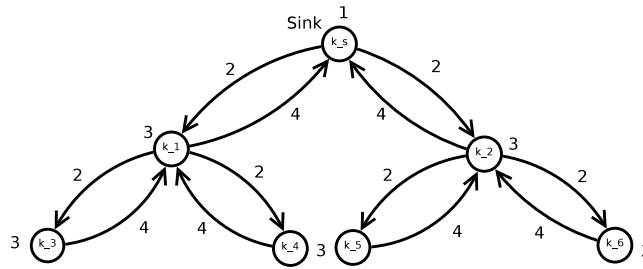


Figure 19: Message flow in TAUk

Software components

To make the roles of the software components used in this implementation more understandable, we first summarize the message flow in our implementation of TAUk briefly. An example message flow for TAUk in a tree-shaped network of 6 sensors is shown in Figure 19:

- Step-1: The sink node selects a master key k_s and splits it into d keys (1) such that $k_s = (k_1 + k_2 + \dots + k_d) \bmod P$, whereby d is the degree of the tree-shaped network and P is the modulus. For example, in case of Figure 19, the key k_s is splitted in two keys such that $k_s = k_1 + k_2 \bmod P$ holds.
- Step-2: The sink node transmits d splitted keys to its successors (2) and starts waiting for the commitment messages from them (4) to signal that the key distribution is completed.
- Step-3: Upon a received key from the predecessor, the sensor node n checks if it is a leaf node (3). If the node n is a leaf node, it stores the received key in its memory and sends a commitment message to its predecessor (4). If the node n is not a leaf node, it performs the following operations (3):
 - node n splits the received key k_r in $d + 1$ keys such that $k_r = (k_n + k_1 + k_2 + \dots + k_d) \bmod P$ and stores k_n in its memory to use in the homomorphic encryptions, while deleting the received key k_r from its memory.
 - node n transmits the d splitted keys to its successors and starts waiting the commitments from them (2).
 - after receiving a commitment from all of its successors, the node n sends a commitment message to its predecessor (4).

The implementation of TAUk mainly consists of three components providing the functionalities summarized above such as deriving individual keys from a master key (3), disseminating the splitted keys to the successors (2), and keeping the track of the direct neighbors. The operations required for splitting the master key is implemented in the `TaukUtilSM` module. The `NeighbourTableM` module is responsible for providing the information about the sensor node's location in the network and the addresses of the successors and the predecessor. The transmission of the keys to the successors and the transmission of a commitment message for a successful key generation to the predecessor (4) are realized in the `TaukRadioM` module. In addition, the `TaukRadioM` is responsible for forwarding the messages received over Radio to the responsible modules for further processing. Finally, the `TaukUtilSM` module implements the TAUk by means of the aforementioned modules.

In the following subsections, we explain the software components used in the implementation in detail.

- **TaukM**

As shown in Figure 21, the `TaukM` is the module implementing the protocol functionality with the help of other modules listed and described below. The execution of the TAUk differs slightly depending on the role of the sensor nodes:

- **Sink node:**

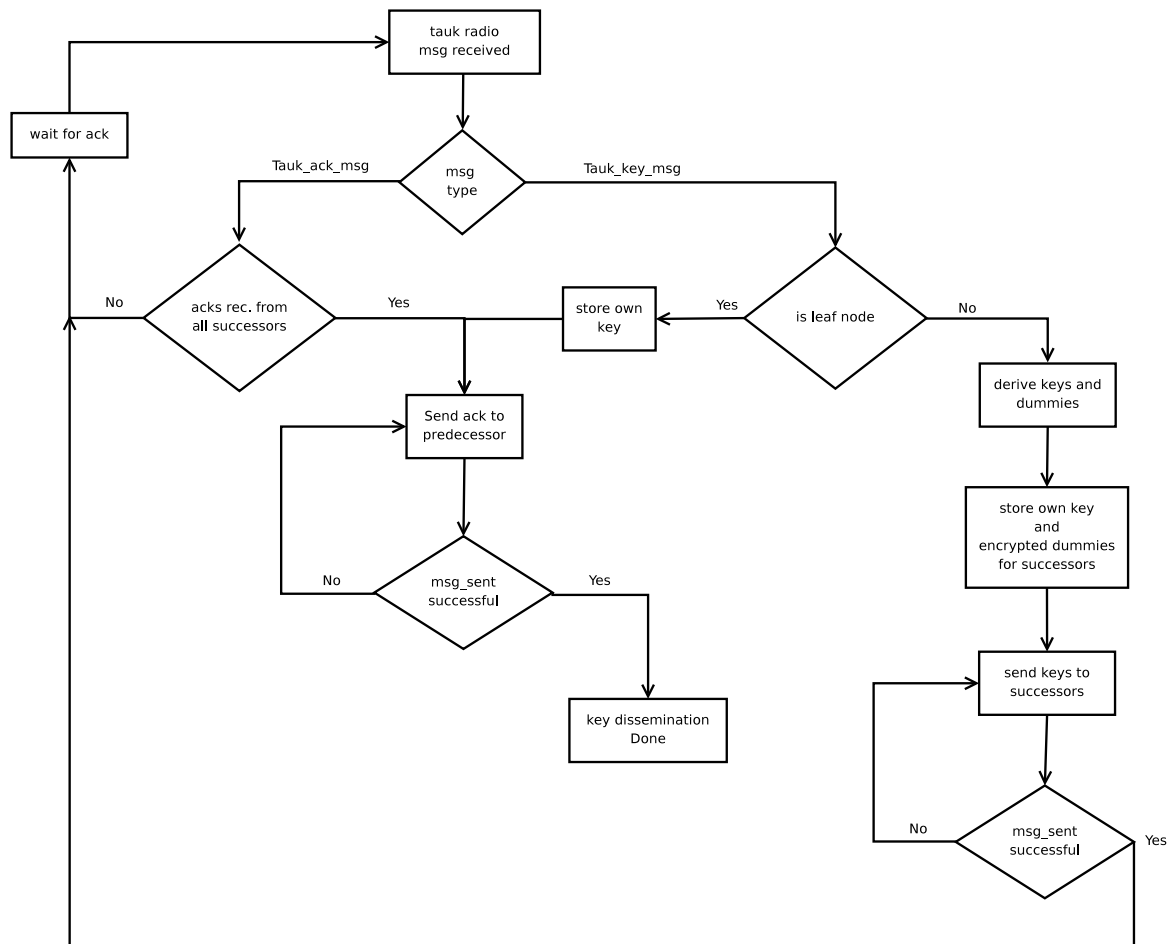


Figure 20: Flow chart for the implementation of the TAUk key-distribution

The key-dissemination is started by calling the command `startTauk()`. Once this command is executed successfully, the sink node initializes its neighbor table and selects a random key and a dummy value to be disseminated in the network. After splitting them regarding to the number of neighbors, it sends the splitted keys and dummies to corresponding neighbors and starts waiting the acknowledgment messages (`acks`). Once it receives `acks` from its all neighbors, the key-distribution is finished.

– **Aggregator node:**

Once an aggregator receives a `tauk_key_msg_t` message from the sink or from an another aggregator, it first derives the keys and dummies for itself and its neighbors. After this it encrypts the dummies of the neighbors with their keys and stores them together with its own key by using the `KeyProvider` interface provided by `Identity` module. It transmits then the derived keys to the neighbors and starts to wait the `acks` from them. Once it receives `acks` from its all neighbors, it sends an `ack` message to its predecessor and fires the the key-distribution is finished event.

– **Leaf node:**

Leaf nodes receives only `tauk_key_msg_t` messages. Once a leaf node receives a key message, it stores the key and sends an `ack` message to its predecessor. Afterwards it fires its the key-distribution is finished event.

The implementation of the TAUk key-distribution for an aggregator and a leaf node is summarized in a flow chart, see Figure 20, in a simplified way. To achieve the aforementioned functionalities, the `TaukM` module implements the following commands and events:

– **command error_t startTauk():**

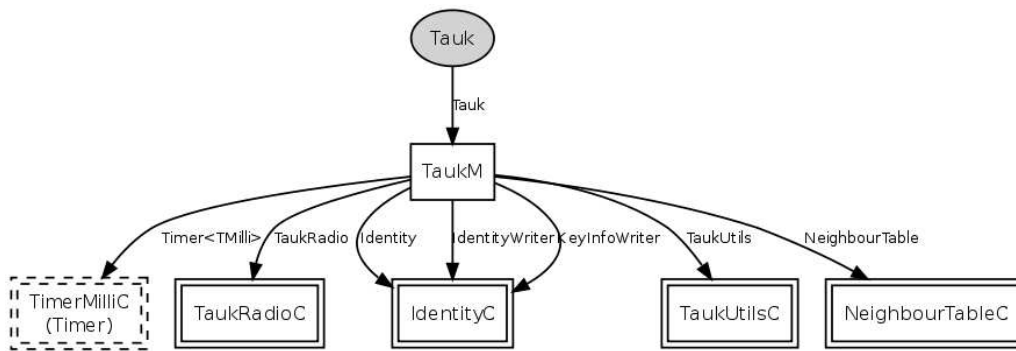


Figure 21: Graphical representation of the TAUk key-distribution module

This command initializes the components used by the `TaukM` module, such as `NeighbourTable` and `TaukRadio`. Moreover, if the node is the sink, it starts the key-distribution as described above. If the node is an aggregator or a leaf node, it changes its state for waiting messages from its predecessor.

– **event void startTaukDone(error_t err):**

This event fires after the `startTauk()` is executed. The `err` is set to `SUCCESS`, if the operation was successful, otherwise `FAIL`.

– **command error_t disseminateKeys(uint8_t* key_msg_to_disseminate, uint8_t length):**

This command is executed only on the sink and aggregator nodes. It transmits the keys and dummies, which are contained in the `key_msg_to_disseminate`, to the successors of the node, after processing, e.g. splitting, them. The `length` denotes the size of the structure `key_msg_to_disseminate`. Please note that the `key_msg_to_disseminate` message is generated on the sink in the beginning of the key-distribution process, while it is received over radio for other nodes. The leaf nodes do not need to execute this command. Because they do not have any successors and therefore further processing of the received message is not necessary. They only need to store the received key in its memory.

– **event void disseminateKeysDone(error_t err):**

This event fires, if the `disseminateKeys` command has distributed the keys successfully. More precisely, this event is fired, if the keys and dummies derived from the `key_msg_to_disseminate` are sent to the successors without failure and the `acks` from all successors were received.

• **TaukRadioC**

This component, see Figure 22 is responsible for sending and receiving the messages related to TAUk key-distribution. It implements the following commands and events:

– **event error_t initTaukRadio():**

This command is responsible for the initialization of the radio components and the other software modules employed in the `TaukRadioC` component.

– **event void initTaukRadioDone(error_t err):**

This event fires after the initialization of the `TaukRadioC` component is finished. The `err` is set to `SUCCESS`, if the initializations were successful, otherwise `FAIL`.

– **command error_t sendSuccKeys(uint8_t* keysToSuccessors, uint8_t keySize):**

This command transmits the keys stored in the `keysToSuccessors` array to the successors of this node. The list of the successors are received over the `NeighbourTable` interface from the `NeighbourTableC` component. Please note that it may happen that depending on the application

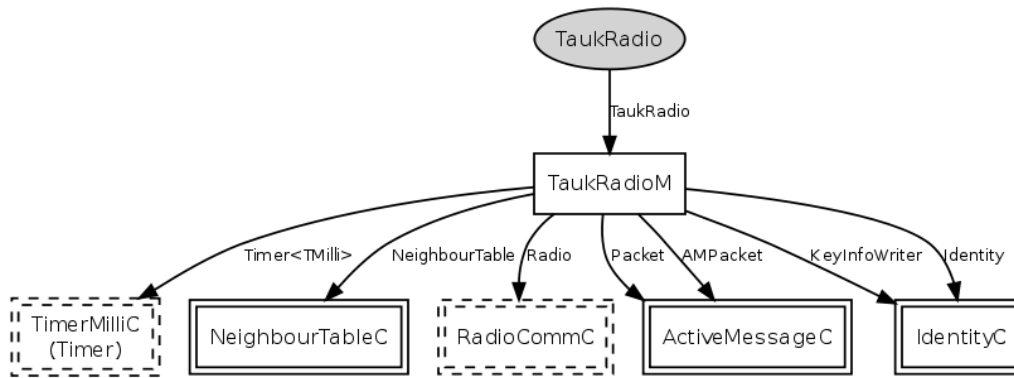


Figure 22: Graphical representation of the radio module for TAUk

the number of the nodes are not equal to the calculated `TREE_SIZE`. In this case one can not form a fully balanced tree with the virtual tauk node IDs, resulting in a problem that an aggregator node trying to send a key to a non-existing successor. To solve this problem, we have a radio component implemented such that it tries to send a key to a successor only a user definable number of times. If the transmission still fails after that limited number of times, an aggregator node assumes that the successor is not available and accordingly re-refresh the keys to be sent to the successors and ignore that successor.

– **event void sendSuccKeysDone(error_t err):**

This event is fired, if the transmission of the keys with the command `sendSuccKeys` was successful. Here, successful means that an aggregator node has received the acks from all of its successors.

– **event void receiveMsg(void* data, uint8_t size):**

This event is fired, once an aggregator receives a new key message from its predecessor. The received message is then forwarded to the `TaukM` module for further processing.

• **TaukUtilsC**

This module implements the arithmetic operations necessary in TAUk key-distribution.

– **command void splitKey(uint8_t* splittedKeys, int8_t* keyToSplit, uint8_t keySize, uint8_t numOfSplits):**

This commands splits the key `keyToSplit` of size `keySize` into `numOfSplits` keys and stores the results in the `splittedKeys` array.

– **command uint8_t* enc(uint8_t* plainText, uint8_t plainTextSize, uint8_t* key, uint8_t keySize, uint_t* p):**

This commands encrypts the data `plainText` of the size `plainTextSize` with the key `key` of size `keySize` and returns the address of the resulted cipher. As proposed in the protocol, $enc(plainText) = plainText + key \bmod p$, whereby p is the modulus.

– **command void genRandom(uint8_t* rnd, uint8_t randSize):**

This command generates a random number of the size `randSize` and stores it in the `rnd`. This method is required for splitting the keys.

• **NeighbourTableC**

The `NeighbourTableC` component is responsible for providing the information about the sensor node's location in the network and the addresses of the successors and the predecessor. It implements the addressing model proposed in Subsection 6. The following commands are provided by the `NeighbourTableM` module:

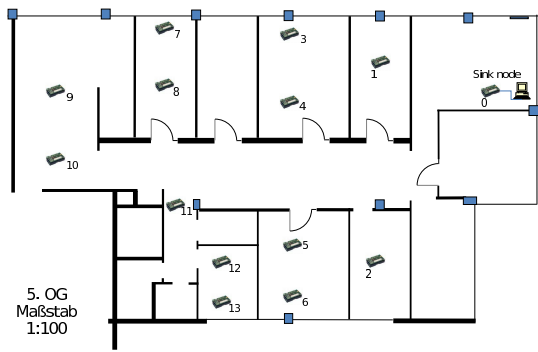


Figure 23: Layout of the sensor testbed comprised of 14 sensor nodes

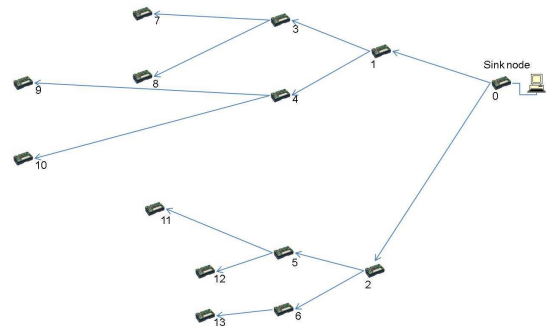


Figure 24: Tree-shaped network corresponding to our office setting

- **command neighbor_id_t getPredecessor(neighbor_id_t nodeID):**
This command returns the address of the predecessor of a given node with the address nodeID.
- **command neighbor_id_t* getSuccessors(neighbor_id_t nodeID):**
This command returns the list of the successors of a given node with the address nodeID.
- **command bool isLeafNode(neighbor_id_t nodeID):** This command returns TRUE, if the node nodeID is a leaf node, otherwise FALSE.

• IdentityC

This is the general identity management module for the UbiSec&Sens software components. In this implementation, it is needed to store the keys and to manage the tauk and node IDs.

Implementation results and evaluation

We use a physical sensor tested to evaluate the implementation of TAUk and to demonstrate its implementability. Our tested evaluation includes the measurement of code size, memory usage, and the dissemination latency.

In addition to our evaluation based on the physical sensor tested, we simulate our implementation by AVRORA, too. The results of our AVRORA simulations allow us for further evaluation with regard to dissemination latency as the network size and the number of hops increase.

Testbed Our tested is comprised of 14 TelosB motes in an office setting (See Figure 23). The TelosB motes are equipped with 16-Bit CPU and have 10KB of RAM and 48KB of ROM. The nodes communicate using IEEE 802.15.4 compatible radio interfaces with a transmission capacity of up to 250KBits/s.

We used the serial port over USB as a communication channel for debugging and gathering data from sensor nodes. Sensor nodes utilize the USB channel to send debugging outputs to the external PC. This debugging information is mainly comprised of the key received from upper level sensor nodes, the keys to be sent to the lower level sensor nodes and the key to be stored [1]. In each of our tested experiment, a single node is designed as the sink node. The sink node is used to initiate the key distribution process. At the end of each key distribution process, the dissemination latency is computed at the sink node and is sent to the external PC over the serial port.

To show the multi-hop capability of our implementation, we chose a tree-shaped network topology consisting of 14 nodes. Even though the degree and the height for the tree-shaped network topology can be chosen freely, we set the degree and the height of the tree to 2 and 3, respectively. This is due to the fact that this configuration suits best to our office layout and to the number of the sensor nodes available in our tested. The locations of the sensor nodes in the tested and the resulted tree-shaped network are shown in Figure 23 and in Figure 24, respectively.

Some of sensor nodes in a WSN setting may run out of battery along the life time of the deployment. Based on this observation, we implemented the communication module of TAUk such that the sensor nodes can dynamically adapt themselves to the case where some sensor nodes are disappeared. To show this capability of our implementation, we employed in our testbed 14 sensor nodes instead of 15 by removing one of the

Key size	Maximum	Minimum	Average
3 bytes	2608ms	1098ms	2102ms
4 bytes	2649ms	1673ms	2611ms
5 bytes	2675ms	1681ms	2363ms
6 bytes	2648ms	1708ms	2462ms
7 bytes	2492ms	1792ms	2032ms

Table 4: TAUk key distribution latencies in a real world office setting with 14 nodes with respect to different key sizes.

neighbors of the node 6. In this setting (see the network model depicted in Figure 24), the sensor node 6 realizes that the sensor node 14 is missing (simulating a non-responding node due to e.g., depleted power). Therefore, the node 6 stops to trying to communicate with node 14 to orient itself to its absence and to update its own key. In order to keep the implementation complexity at minimum, we utilized a simple trick to detect the absence of any sensor node: If a sensor node still does not get an ACK message from a destination node after requesting a certain number of times, the sender assumes that the destination node is missing. Once a sensor node identifies the non-responding sensor nodes from its neighborhood, it updates its own key by adding the sum of the keys of missed sensor nodes to it.

Code size and memory consumption On the TelosB sensor mote, our current implementation of TAUk consumes roughly 26.5KB ROM and 1.2KB RAM. In fact, one may argue that the ROM consumption is roughly the half of the ROM size available which is relatively high. The reason for this is that our implementation includes all components shown in Figure 21, hence the received ROM consumption is actually the sum of the ROM consumptions for all those components. Since most of the included components, e.g., radio, timer, etc. are common for general WSN applications, the use of TAUk in a complete WSN application will increase the overall ROM consumption not in a marginal way.

Dissemination latency Table 4 shows TAUk's latency in our office setting relative to different key sizes being disseminated. In order to increase the measurement precision for the average case, each key distribution process was performed 51 times and the average dissemination latency was taken as a result. Since all of the tested key sizes can fit in a single radio packet, one may expect that the dissemination latencies should be roughly the same. The results shown in Table 4 confirm this assumption. Moreover, the similar latencies for different key sizes imply that the performance of TAUk's core functionality (e.g., splitting the keys, performing modular operations, etc.) is affected slightly as the key sizes increase. The minimal differences between the latencies are mainly due to multiple re-transmissions of some radio packets due to packet collisions.

Every sensor node in our indoor setting was in the communication range of the other nodes. Obviously, this increases the number of packet collisions, hence the number of the packet re-transmissions. Since the number of the packet collisions will be smaller, one may expect a better performance of our implementation in the field deployment of real WSN applications.

Simulation

We use AVRORA to evaluate the scalability of our current implementation. AVRORA is a precise simulation and analysis tool for Mica2 and MicaZ sensor nodes. For the simulation, we run a test application for different number of sensor nodes (See Table 4), but with a fixed key size of 5 Bytes. The reason why we chose a fixed key size is the experience what we got from running of our testbed. The results of our office tests with real sensor nodes show that the key size does not influence the dissemination latency. Since AVRORA does not support the TelosB platform and the support for the MicaZ platform was still not completed while we were writing this document, the Mica2 platform is selected as a reference platform for our simulations in the following.

Code size and memory consumption On the Mica2 sensor platform, our current implementation of TAUk consumes roughly 26.9KB ROM and 1.4KB RAM. The code size and the RAM consumption on the Mica2

Network size	Maximum	Minimum	Average
3 nodes	860ms	375ms	627ms
7 nodes	1350ms	866ms	1122ms
15 nodes	2352ms	1710ms	2040ms
31 nodes	4133ms	2004ms	3732ms
63 nodes	TBD	TBD	TBD
85 nodes	TBD	TBD	TBD

Table 5: TAUk key distribution latencies with AVRORA simulation tool with respect to different network sizes.

platform is slightly higher than on the TelosB platform. This may be an indicator that the compiler used for the TelosB platform can optimize our implementation better than the compiler used for the Mica2 platform.

Dissemination latency Table 5 shows TAUk's latency in our AVRORA simulation relative to different network sizes and a fixed key size of 5 Bytes. In order to increase the measurement precision, each key distribution process was performed 10 times and the average dissemination latency was taken as a result. As expected, the dissemination latencies generally increase with linear scalability with respect to the number of sensor nodes. This is mainly due to the required number of hops increasing as the network size grows.

The simulation results for 15 sensor nodes are very close to the results what we received from our in the office experiment. This verifies the precision of AVRORA simulation tool and allows us for relying on the results for other simulation results. An example network topology used in our simulation for 85 nodes is depicted in Figure 25.

RoK Protocol Implementation Details

We designed the specification and interfaces of Rok scheme (Robust key predistribution), and will implement the functionality. The Rok scheme interfaces are implemented for TinyOS 2.0.1 and Nesc 1.8. Implementation is compatible with Micaz as well as Telosb motes and is integrated with other modules present in UbiSec&Sens toolbox such as *HashChainC*, *IdentityC*, *EpochCounterC* and UbiSec&Sens Neighbour management (*NeighbourC*).

Software components

From the figure 26 we can see Rok 3 main modules :

- **RokP** is the module implementing the protocol fonctionnality

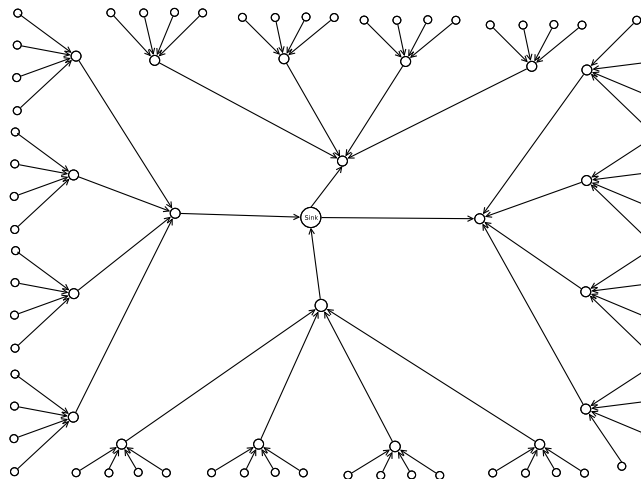


Figure 25: Topology of the simulated network consisting of 84 sensor nodes and the sink

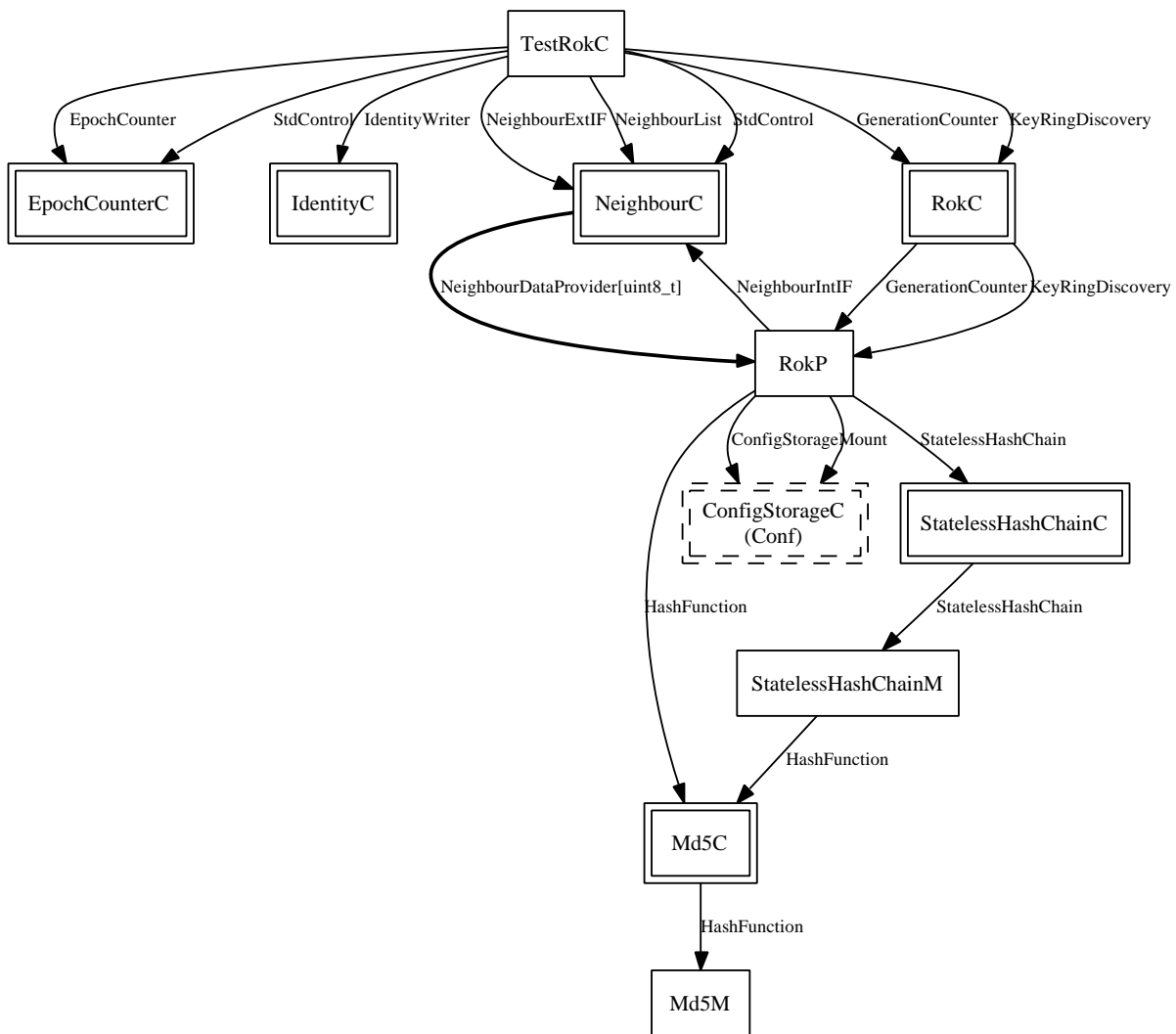


Figure 26: Graphical representation of the Robust key distribution module

- **RokC** is the configuration wiring RokP with components used by RokP.
- **TestRokC** is a demo application to test the functionality of Rok module

We can then see that RokP depends on the following modules :

- **HashChainC** is the generic component which is used to manage the backward and forward hash chains (implemented by *HashChain_Forward* and *HashChain_Backward*), it relies on *Md5C* component.
- **EpochCounterC** provides the time synchronization as well as firing events at each new epoch such that the *RokP* module is able to perform a new keying at each epoch.
- **Md5C** is used through the HashFunction interface, it's used during establishment of the secret key between nodes.
- **ActiveMessageC**, **AMSenderC** and **AMReceiverC** are used for general message communication with neighbours.
- **NeighbourC** component is used to get the list of current neighbours and provides events when availability of a neighbour node change.
- **IdentityC** Is the general identity management module of UbiSec&Sens platform. Rok access it mainly through the *KeyProvider* Interface to update the key for a neighbour node. This update is performed

once the key establishment is complete. The new key is therefore available for general use by any of the *UbiSec&Sens* applications.

Interfaces

The Rok software component does not define any new interfaces as it's not meant to be used by other components. It's included in the program by top level configuration. It then executes independently from other modules and registers keys to the identity module once they are available. The Identity module makes eventually the keys available to other modules. Rok component provides only a **SplitControl** interface to be used for starting and stopping the Rok module.

References

- [1] F. Armknecht, D. Westhoff, A. Hessler, and J. Girao, A lifetime-optimized end-to-end encryption scheme for sensor networks allowing in-network processing. Special Issue 'on Algorithmic and Theoretical Aspects of Wireless Ad Hoc and Sensor Networks', Elsevier Computer Communications, 2007. 54
- [2] F. Armknecht, D. Westhoff, Key Agreement for W-BANs under submission, 2007.
- [3] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, San Diego, CA, USA, July 2005. 19, 21, 30
- [4] J. Domingo-Ferrer. A provably secure additive and multiplicative privacy homomorphism. In *Information Security Conference (ISC'02)*, Springer LNCS 2433, pages 471–483, 2002. 19, 31
- [5] M. Önen and R. Molva. Secure data aggregation with multiple encryption. In *Proceedings of 4th European Conference on Wireless Sensor Networks (EWSN 2007)*. 21, 30
- [6] B. Dutertre, S. Cheung, and J. Levy, "Lightweight key management in wireless sensors networks by leveraging initial trust," SRI International, SDL Technical Report SRI-SDL-04-02, April 2004. 46
- [7] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and communications security, CCS, Washington, DC, USA*, November 2002. 10, 11, 15, 16, 20, 34, 35, 38, 45, 46
- [8] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003. 16, 17, 20, 38, 41, 42, 46
- [9] D. Eastlake 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," , United States, 2001. 34
- [10] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, 1981. 35
- [11] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs." in *CHES: Cryptography Hardware and Embedded Systems*, August 2004. 45
- [12] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *PERCOM '05: Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications*, March 2005. 45
- [13] D. Malan, M. Welsh, and M. Smith, "A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography," in *Proceedings of 1st IEEE International Conference on Sensor and Ad Hoc Communications and Network, Santa Clara-CA, USA*, October 2004. 45
- [14] R. Di Pietro, L. V. Mancini, A. Mei, A. Panconesi, and J. Radhakrishnan, "Connectivity properties of secure wireless sensor networks," in *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks, SASN, Washington DC, USA*, October 2004. 45

- [15] R. Di Pietro, L. V. Mancini, and A. Mei, "Random key assignment for secure wireless sensor networks," in *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks, SASN, Fairfax-VA, USA*, October 2003. 46
- [16] M. Conti, R. Di Pietro, and L. V. Mancini, "ECCE: Enhanced cooperative channel establishment for secure pair-wise communication in wsn," in *Journal of AdHoc Networks*, Elsevier, Ed., January 2007. 46
- [17] D. Westhoff, J. Girao, and M. Acharya. Concealed data aggregation for reverse multicast traffic in wireless sensor networks: Encryption, key pre-distribution and routing. In *IEEE Transactions on Mobile Computing*, October 2006. 19, 30
- [18] S.A. Camtepe and B. Yener, "Combinatorial Design of Key Distribution Mechanisms for Wireless Sensor Networks", in *Proceedings of the 9th European Symposium on Research Computer Security*, Springer, 2004. 14
- [19] D. Eastlake 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," , United States, 2001. 34
- [20] B. Lai, S. Kim and I. Verbauwhede, "Scalable Session Key Construction Protocol for Wireless Sensor Networks", in *Proceedings of the IEEE Workshop on Large Scale Real-Time and Embedded Systems (LARTES)*, 20 14
- [21] J. Lee and D.R. Stinson, "Deterministic key predistribution schemes for distributed sensor networks", *Lecture Notes in Computer Science*, vol. 3357, pp. 294–307, Springer, 2004 14
- [22] W. Du, J. Deng, Y. S. Han and P. K. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks", in *Proceedings of the 9th ACM conference on Computer and communications security, CCS, Washington, DC, USA*, November 2003. 18
- [23] D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and communications security, CCS, Washington, DC, USA*, November 2003. 18
- [24] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1997. 14
- [25] P. Erdős and A. Rényi, "On the Evolution of Random Graphs," in *Publications of the Math. Institute of the Hungarian Academy of Sciences*, vol. 3, 1960. 15
- [26] L. Uhsadel, A. Poschmann and C. Paar, Enabling Full-Size Public-Key Algorithms on 8-bit Sensor Nodes in *Proceedings of ESAS 2007*, LCNS, Cambridge. 12
- [27] Gura, N. and Patel, A. and Wander, A. and Eberle, H. and Shantz, S.C. "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs" in *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*. 12
- [28] Crossbow Technology Inc., "MPR-MIB Users Manual - Revision A". available via http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf June 2007 11