



HAL
open science

Secure aggregation in wireless sensor networks

Wassim Drira, Chakib Bekara, Maryline Laurent

► **To cite this version:**

Wassim Drira, Chakib Bekara, Maryline Laurent. Secure aggregation in wireless sensor networks. [Research Report] Dépt. Logiciels-Réseaux (Institut Mines-Télécom-Télécom SudParis); Services répartis, Architectures, MODélisation, Validation, Administration des Réseaux (Institut Mines-Télécom-Télécom SudParis-CNRS). 2009, pp.58. hal-01360043

HAL Id: hal-01360043

<https://hal.science/hal-01360043v1>

Submitted on 5 Sep 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Secure Aggregation in Wireless Sensor Networks

Logiciels-Réseaux	Wassim Drira Chakib Bekara Maryline Laurent-Maknavicius	09008-LOR 2009
-------------------	---	-------------------

Abstract

The Wireless Sensor Networks (WSNs) are composed of a huge number of sensor nodes. The large number of nodes may lead to a huge amount of data in the network, causing the network to degrade performance and shorten its lifetime. The data aggregation techniques may be a solution to remane the redundancy of data and thereby reduce the number of packets transmitted in the network. Nevertheless, the data aggregation causes some security vulnerabilities: impersonation, denying having received data, dropping packets deliberately, alteration of the sensed readings and aggregation results... In this report, we compare the performances get by two secure aggregation solutions: the Secure Aggregation for Wireless Networks (Hu et al. protocol) and the Secure Aggregation Protocol for Cluster-Based Wireless Sensor Network (SAPC). The analysis shows that Hu et al. protocol gives better performance than SAPC in terms of number of exchanged messages but it is vulnerable to simple attacks. This report describes both protocols and their performances and propose a new protocol based on binary trees and delayed aggregation result checking to improve the performance of SAPC.

Key-words: Security, wireless sensor network, aggregation, TinyOS, authentication.

Résumé

Les réseaux de capteurs sans fil sont composés d'un grand nombre de capteurs avec de faibles ressources en énergie, en mémoire et en calcul. Dans le but d'étendre la durée de vie de ces réseaux, l'agrégation des messages s'avère comme une bonne solution pour diminuer le nombre de messages échangés dans le réseau en éliminant les données redondantes et par suite l'énergie utilisée par les nœuds pour la communication. Mais cette solution peut avoir des conséquences néfastes sur la sécurité du réseau: l'intégrité des messages, l'authentification, la précision des mesures... Dans ce rapport, nous comparons les performances obtenus par deux protocoles d'agrégation sécurisés: Secure Aggregation for Wireless Networks (le protocole de Hu et al.) et le protocole Secure Aggregation Protocol for Cluster-Based Wireless Sensor Network (SAPC). L'analyse montre que le protocole de Hu et al. donne de bons résultats en terme de nombre de messages, mais il est vulnérable à certaines attaques simples. Ce rapport décrit les deux protocoles avec une analyse de leur performances et propose, en vue d'améliorer les performances de SAPC, un nouveau protocole d'agrégation basé sur les arbres binaires et la vérification retardé du résultat d'agrégation.

Mots-clés : Sécurité, réseau de capteurs sans fil, agrégation, TinyOS, authentification.

Wassim Drira
Stagiaire

Chakib Bekara
Doctorant

Maryline Laurent-Maknavicius
Professeur

TELECOM & Management SudParis - Département LOR - CNRS
9 rue Charles Fourier 91011 Evry cedex
{Wassim.Drira|Chakib.Bekara|Maryline.Maknavicius}@it-sudParis.eu

Contents

Introduction	7
1 Wireless sensor networks and security issues	8
1.1 Introduction	8
1.2 Operating Systems	9
1.2.1 TinyOS	9
1.2.2 Contiki	13
1.3 Sensor security problems	13
1.3.1 Very limited resources	13
1.3.2 Unreliable Communication	14
1.3.3 Unattended Operation	14
1.4 Security Goals and Challenges	15
1.5 Attacks on the wireless sensor networks	16
1.6 Conclusion	18
2 Secure data aggregation approaches in Wireless Sensor Networks	19
2.1 Introduction	19
2.2 Secure Aggregation for Wireless Networks	19
2.3 SHIA	22
2.3.1 Query dissemination	22
2.3.2 Aggregation commit phase	22
2.3.2.1 Naive approach	23
2.3.2.2 Improved approach	23
2.3.3 Result-checking phase	25
2.3.3.1 Dissemination of final commitment values	25
2.3.3.2 Dissemination of off-path values	25
2.3.3.3 Verification of inclusion	25
2.3.3.4 Collection of confirmations	25
2.3.3.5 Verification of confirmations	26
2.4 SAPC	26
2.5 A trust based framework for Secure data Aggregation in Wireless Sensor Network	28
2.5.1 System model	28
2.5.2 Threat model	28
2.5.3 Josang's Belief Model	29
2.5.4 A trust based framework against false data injection	29
2.6 Conclusion	31

3	Implementation and Tests	32
3.1	Introduction	32
3.2	Implementation	32
3.3	Tests	33
3.3.1	SAPC	33
3.3.2	Hu et al.	35
3.4	Conclusion	35
4	SeBTIA: Secure binary tree in-network aggregation	37
4.1	Introduction	37
4.2	Commitment tree	37
4.2.1	Notations	37
4.2.2	Ordered tree	38
4.2.3	Formation of the commitment tree (binary tree)	39
4.2.4	Logical representation of the commitment tree	39
4.3	Protocol description	41
4.3.1	Query Dissemination	41
4.3.2	Cryptographic Security	41
4.3.3	Aggregation-Commitment phase	41
4.3.4	Result-checking Phase	45
4.3.5	Elimination of malicious or faulty nodes	46
4.4	Congestion complexity	46
4.5	Implementation and tests	49
4.6	Conclusion	49
	Conclusions & perspectives	50
	Bibliography	51
	Glossary	54
	A Component diagrams	55

List of Figures

1.1	Health care motes	9
1.2	Evolution of motes	10
1.3	Logo TinyOS	11
2.1	Forwarding packets with and without data aggregation	20
2.2	Data aggregation illustration	21
2.3	Aggregation and naive commitment tree in network context	23
2.4	Process of node A (from Figure 2.3) deriving its commitment forest from the commitment forests received from its children	24
2.5	Dissemination of off-path values: t sends the label of u_1 to u_2 and vice-versa; each node then forwards it to all the vertices in their subtrees	25
2.6	Cluster-based wireless sensor network	26
2.7	Abstract architecture of the framework	29
3.1	Number of messages sent by CH	33
3.2	Minimum and maximum number of messages in the network	33
3.3	Calculation time in a normal node	34
3.4	Calculation time in CH	34
3.5	Calculation cost comparison for CH vs simple node	35
4.1	Encoding n-ary trees as binary	39
4.2	Formation of the commitment tree	40
4.3	Logical representation of the commitment tree	40
4.4	An ordered aggregation tree of a network	41
4.5	The aggregate-commit algorithm	44
4.6	Needed labels by node 15 to build the commitment tree	45
4.7	Exploration of tree branches	47
4.8	Comparison of SHIA and SeBTIA messages complexity	48
4.9	Execution time needed to prepare L_1	49
A.1	Component diagram of the SAPC application	56
A.2	Component diagram of the Hu et al. application	57
A.3	Component diagram of the SeBTIA application	58

List of Tables

1.1	The impact of 29-BYTES payload cipher on CPU consumption	14
1.2	The impact of calculating 29-BYTE packet MAC on CPU consumption . .	14
1.3	Sensor network layers and denial-of-service defenses	17
3.1	Memory space occupation in SAPC and Hu et al. protocols	32
4.1	An example of the Aggregation-commitment phase execution in the network shown in figure 4 (“X: Y” signifies that the instruction Y is executed in node X)	43
4.2	Memory space occupation in SeBTIA	49

Introduction

Nowadays, we are overwhelmed with a large number of data. Extracting relevant data and eliminating redundant data is an important concern for researchers. Those techniques are needed in wireless sensor networks, which contain a huge number of sensor nodes, each one generating some sensory readings toward the base station. The data aggregation extends the longevity of the wireless sensor networks as exchange messages are targeted, but it makes the network more vulnerable. Many secure data aggregation protocols have appeared in the literature. Our contribution is to implement and compare two protocols for the secure data aggregation: The Secure Aggregation Protocol for Cluster-Based wireless Sensor network (SAPC) [1] and Hu et al. protocol [2]. Then we propose a solution to improve the SAPC performances.

This report is organized as follows: chapter 1 first introduces the wireless sensor networks and the security threats. Chapter 2 gives the most important solutions for the data aggregation in wireless sensor networks, and then some comparative experimental results between SAPC and Hu et al. protocol. In chapter 4, we present a new aggregation framework based on the binary trees to extend SAPC and improve its performances. Finally we conclude this report with future research directions.

1 Wireless sensor networks and security issues

1.1 Introduction

The Wireless Sensor Network (WSN) is a term used to refer to a wide network composed by a huge number of heterogeneous sensors deployed randomly and that communicate through some wireless technologies. Generally, there is a special node called a base station (BS) in the network that receives the data from the sensors, and forwards them to the network operator. Most of the time, the network topology can not be predicted, and is deployed in hostile or friendly environments. The development of microelectronics and wireless communication technologies are leading to produce tinier and low cost sensors although they still have limited energy, memory and computational resources. Wireless Sensor Networks are widely used in various applications. The most important ones are:

Military: WSNs can be used to supervise the country frontier and enemy movement during a war. As such various sensors can be placed in the network, to detect movement of persons or vehicles, to detect sounds and to get some geographical coordinates. They can be coupled with a camera to take photos and videos when needed. For example, at the frontier between the United States and Mexico, a WSN coupled with cameras, is deployed to detect person's illegal penetration.

Environment: A WSN can be deployed to supervise air pollution, to detect forest fire, and also to detect submarine earthquake to prevent disasters like Tsunamis. It can be also deployed, for example, near a chemical factory to detect any emission of poisonous substances in the air or in a lake.

Buildings: A WSN can be deployed in a building or a factory to manage automatically air conditioning, electric lights. In addition, it is used in dams in Thailand to detect the vibrations and pressure.

Health care: The application can detect and analyze patients health from bio sensors which can collect pulse, temperature and blood pressure of the patient (figure 1.1).

With this application, the doctors can remotely monitor, diagnose and prescribe the patient when an emergency occurs [3].

Supply Chain: WSN can be used to monitor the cold chain, as studied in the project CAPTEURS [4, 5] to which TELECOM SudParis participates in designing a secure solution to supervise the goods temperature along the transportation.

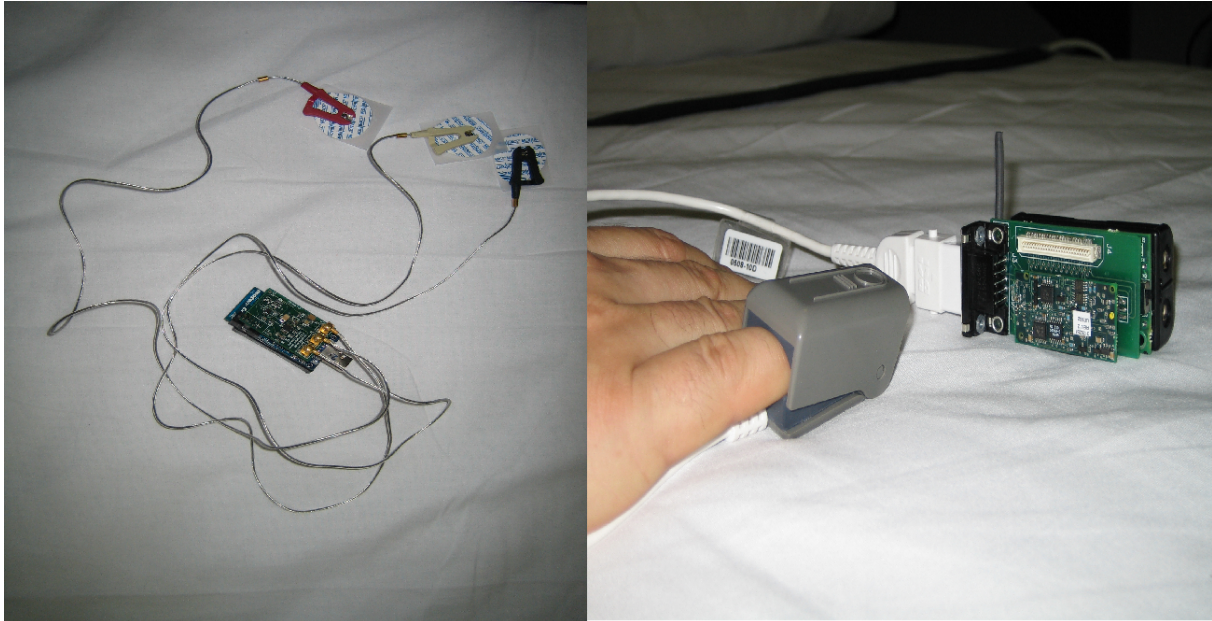


Figure 1.1: Health care motes [6]

A sensor node or a mote is a component mainly composed by a micro-controller, a chip for the wireless communication and one or more sensors to measure the physical quantities (temperature, humidity, light, motion, vibration,...). There are various types of motes in the market, but the most used ones in research are mica, mica2 and tmote. Figure 1.2 shows a comparison between different motes and the evolution of their capacities and resources.

1.2 Operating Systems

There are two main operating systems used in wireless sensor networks: TinyOS and Contiki. We discuss in this section the characteristics of each of them.

1.2.1 TinyOS

TinyOS is an open source operating system for wireless sensor networks, featuring a component-oriented architecture. In addition, it minimizes the code size as required by




Mote Type Year	<i>WeC</i> 1998	<i>René</i> 1999	<i>René 2</i> 2000	<i>Dot</i> 2000	<i>Mica</i> 2001	<i>Mica2Dot</i> 2002	<i>Mica 2</i> 2002	<i>Telos</i> 2004	
									
Microcontroller									
Type	AT90LS8535		ATmega163		ATmega128		TI MSP430		
Program memory (KB)	8		16		128		60		
RAM (KB)	0.5		1		4		2		
Active Power (mW)	15		15		8		33		
Sleep Power (μ W)	45		45		75		75		
Wakeup Time (μ s)	1000		36		180		180		
Nonvolatile storage									
Chip	24LC256			AT45DB041B			ST M24M01S		
Connection type	I ² C			SPI			I ² C		
Size (KB)	32			512			128		
Communication									
Radio	TR1000			TR1000		CC1000		CC2420	
Data rate (kbps)	10			40		38.4		250	
Modulation type	OOK			ASK		FSK		O-QPSK	
Receive Power (mW)	9			12		29		38	
Transmit Power at 0dBm (mW)	36			36		42		35	
Power Consumption									
Minimum Operation (V)	2.7		2.7		2.7		1.8		
Total Active Power (mW)	24			27		44		89	
Programming and Sensor Interface									
Expansion	none	51-pin	51-pin	none	51-pin	19-pin	51-pin	10-pin	
Communication	IEEE 1284 (programming) and RS232 (requires additional hardware)							USB	
Integrated Sensors	no	no	no	yes	no	no	no	yes	

Figure 1.2: Evolution of motes [7]

the severe memory constraints inherent to the sensor networks. TinyOS provides to developers different libraries like the network protocols, distributed services, sensor drivers, and data acquisition tools – all of which can be used as-is or be further refined for a custom application. TinyOS’s event-driven execution model enables the fine-grained power management yet allows the scheduling flexibility made necessary by the unpredictable nature of the wireless communication and physical world interfaces [8].



Figure 1.3: Logo TinyOS

The development of TinyOS began in 1999 in Berkley university. The newest version is TinyOS 2.1 launched in 2008. The system is developed in nesC, a C-like language. Currently, the maintenance and development of the operating system is assured by an international consortium, the TinyOS alliance.

The principal characteristics of the operating system are:

Event-driven: TinyOS is an event-driven operating system. A complete system configuration is formed by 'wiring' together a set of components for a target platform and application domain. Components are restricted objects with well-defined interfaces, internal state, and internal concurrency. Primitive components encapsulate hardware elements, e.g., radio, ADC, timer, or bus. Their interface reflects the hardware operations and interruptions [9].

Components and Bidirectional Interfaces: A component has a set of bidirectional command/ event interfaces implemented either directly or by wiring a collection of subcomponents. The compiler optimizes the entire hierarchical graph, validates that the program is free of race conditions and deadlocks [9].

Split-phase operations: are a typical use of bidirectional interfaces. A higher-level component issues a command to initiate activity in a hardware or software component. The command returns immediately, indicating the status of the request, even though the operation takes some time to complete. When done, the operating component signals an event to the components that will take further action. Meanwhile, the processor may service other tasks and events, or sleep if no tasks are pending. Thus, interleaved execution and power management is provided systematically throughout the entire set of TinyOS components [9].

Hardware Abstraction Architecture: The most important advantages of TinyOS is its compatibility with multiple platforms, at least nine platforms. It is also not that complicated to add or modify the platforms. TinyOS 2.0 uses a three-tier Hardware Abstraction Architecture that combines the strengths of the component model with an effective organization in the form of three different levels of abstraction. The top level of abstraction fosters portability by providing a *platform-independent* hardware interface, the middle layer promotes efficiency through rich *hardware-specific* interfaces and the lowest layer structures access to hardware registers and interrupts [10].

Scheduler: Two types of blocks compose the programs in TinyOS: tasks and event handlers. Tasks are not preemptive but an event preempts the task execution. The semantics of the tasks in TinyOS 2.x are different than those in 1.x. In 1.x, the task queue has a limited length and can include the same task multiple times. In 2.x, each task has its reserved place in the queue so it can have one or none instance there. The scheduler executes the tasks one by one until the queue becomes empty [11].

Timers: TinyOS 2.x offers a rich timer system. Three fundamental properties of timers are precision, width and accuracy. Three precision skills are defined : TMilli (1024 ticks per second), T32khz (32768 ticks per second) and TMicro (1048576 ticks per second). The width for the timer interfaces is 32-bits. The accuracy reflects how closely a component conforms to the precision it claims to provide. Accuracy is affected by issues such as clock drift and hardware limitations. TinyOS defines five types of timer interfaces like Counter, LocalTime and Alarm [12].

Communication: TinyOS 2.1 does not support the TCP/IP stack but the next version will do. The majority of the platforms uses the norm 802.15.4 to define both physical and media access layers. Different network protocols are available like the collection, dissemination and Tymo (adapted version of DYMO).

Power Management: Energy is a critical concern in wireless sensor networks and hence it is a concern for TinyOS. The node goes on a sleep mode when no task is executed; also the TinyOS lets developers to active/disable the radio communication's chip or to use the low power listening. In addition, TinyOS generates a small binary code so it needs low power to memorize it in RAM [13].

nesC language: nesC is an extension to C designed to embody the structuring concepts and execution model of TinyOS [14]. nesC adds support to components, events handling and tasks to the C language.

TOSSIM: TinyOS simulator. TOSSIM allows to compile TinyOS applications into a simulation framework, where developers can perform reproducible tests and debug their code with the standard development tools.

1.2.2 Contiki

Contiki [15] is a small, open source, highly portable, multitasking computer operating system developed for use on a number of memory-constrained networked systems ranging from 8-bit computers to embedded systems on micro-controllers, including sensor network nodes. Contiki provides multitasking and a built-in TCP/IP stack, it implements also a μ IPv6 stack [16].

Contiki supports dynamic loading of programs and multi-threading programming [17]. It comes with a graphical simulator COOJA [18].

1.3 Sensor security problems

The WSNs are different from the computer networks as many constraints make the adaptation of existing security solutions more difficult. So, it is important to understand those constraints to design more efficient security mechanisms [19]:

1.3.1 Very limited resources

The integration of the security mechanisms into the applications generates a memory overhead, computational overhead and more larger packets. This overhead should consider those two constraints:

Limited Memory and Storage Space: As depicted in figure 1.2, the sensor nodes have limited program memory, RAM, and non volatile storage space. So it is necessary to limit the code size of the security algorithm. For example Telos nodes [20] have 60 KB as program memory, 2 KB as RAM memory and 128 KB for non volatile storage.

Power limitation: The energy is a scarce resource for a sensor node since the battery is almost not easily replaced or recharged. Therefore, the battery charge taken with the sensor node to the field must be kept to extend its life time and as a result the entire sensor network longevity. When implementing a cryptographic function or protocol within a sensor node, the energy impact of the added security code must be considered. For example, RC5 (Table 1.1) presented better performance in terms of CPU elapsed time (1.50 ms) while using only 11,059.2 CPU cycles and consuming 36.00 micro-joules to cipher a payload. It (Table 1.2) consumes 49.92 μ J to authenticate 29-BYTE packet.

Algorithm	Time (ms)	CPU cycles	Energy (μJ)
<i>SkipJack</i>	2.16	15,925.2	51.84
<i>RC5</i>	1.50	11,059.2	36.00
<i>RC6</i>	10.78	79,478.7	258.72
<i>TEA</i>	2.56	18,874.4	61.44
<i>DES</i>	608.00	4,482,662,4	14,592.00

Table 1.1: The impact of 29-BYTES payload cipher on CPU consumption [21]

Algorithm	Time (ms)	CPU cycles	Energy (μJ)
<i>SkipJack</i>	2.99	22,044.6	71.76
<i>RC5</i>	2.08	15,335.4	49.92
<i>RC6</i>	15.84	116,785.2	380.16
<i>TEA</i>	5.07	37,380.1	121.68
<i>DES</i>	1,208.00	8,906,342.4	28,992.00

Table 1.2: The impact of calculating 29-BYTE packet MAC on CPU consumption [21]

1.3.2 Unreliable Communication

The sensor network uses wireless communications so it inherits the unreliability of the communications. This technology is characterized by sharing the same media, air, which has a high error rate and a concurrent access, consequently some frequent collisions happen. In addition, the multi-hop nature of the network makes the synchronization between the nodes difficult due to the time needed in each hop to treat the message and forward it. This may be a problem for some security protocols that use a cryptographic key distribution protocol like μtesla [22].

1.3.3 Unattended Operation

Depending on the application of the sensor network, sensor nodes may be left unattended for a long period of time. There are three main caveats to unattended sensor nodes:

Exposure to Physical Attacks The sensor may be deployed in an environment open to adversaries, may be affected by bad weather, and so on. The likelihood that a sensor suffers from a physical attack in such an environment is therefore much higher than the typical PCs, which are located in a secure place and mainly face attacks from the network [19].

Remote Management Remote management of a sensor network makes it virtually impossible to detect physical tampering (i.e., through tamperproof seals) and physical maintenance issues (e.g., battery replacement). Perhaps the most extreme example of this is a sensor node used for the remote reconnaissance missions behind the en-

emy lines. In such a case, the node may not have any physical contact with friendly forces after deployment [19].

No Central Management Point A sensor network is like a distributed network without a central management point. If designed incorrectly, it will make the network organization difficult, inefficient, and weak.

Passive information gathering An intruder can eavesdrop the exchanged messages in the network by positioning malicious nodes or a powerful receiver in the network.

False Node An intruder can maliciously disrupt the network operations for example by injecting a false node so their he is able to drop the packets, inject false data, or modify the contents of a message and hence corrupt the integrity of the message [23].

Node Malfunction A node may become defective for ordinary reasons and, therefore, output wrong data. This can be more dangerous if the node is an aggregator or a cluster head.

Node outage In addition to malfunctioning, a sensor may stop responding completely, for example if its battery is empty.

1.4 Security Goals and Challenges

In order to defend against some attacks, many security mechanisms have been proposed. The goal of each one is to achieve some or all of the following security goals [24]:

Confidentiality or privacy Confidentiality means that only the authorized parties can access the data. For example the confidentiality is a big concern in some cases like military applications or when exchanging the security keys between nodes.

Integrity It means that the transmitted data have not been altered during transit by unauthorized parties.

Authentication It means that the received data are really sent by the claimed sender instead of being injected by someone else.

Availability It means that the network should consistently and continually provide the service that it promises despite the existence of any attacks [23].

Freshness It means that the data are fresh and current. It guarantees that the messages are not replayed or injected by any adversary.

1.5 Attacks on the wireless sensor networks

The attacks, their classes and their classification by the layer are further discussed in [25]. The most important attacks in WSN are:

1. **Passive information gathering** see 1.3.3.
2. **Node subversion** By compromising a node, an attacker can get the program, the secret cryptographic keys thereby it becomes possible to inject false messages from this node. [26] demonstrates that compromising a *Mica2*'s node is done in one minute. For some applications, it becomes necessary to design the tamper resistant nodes. For example, a node should delete its program memory content and its secret keys once the node capture occurs.
3. **Fake node** see 1.3.3.
4. **Node malfunction** see 1.3.3.
5. **Node outage** see 1.3.3.
6. **Message corruption** The integrity of a message is compromised when an attacker modifies its content.
7. **Traffic analysis** Even if the message is encrypted in WSN, an attacker can analyse the communication patterns and detect which nodes are important in the network (e.g. cluster heads, aggregators,...) and focus on them to cause more harm to the network.
8. **Routing loops** An attacker can alter and replay routing information messages, thus some error messages are generated in the network. The routing loops attract or repel the network traffic and increase the node to node latency.
9. **Selective forwarding** If all the nodes participate in routing messages, an attacker can drop certain messages instead of forwarding them all. The percentage of dropped messages and the distance from the malicious node to the base station determine the effectiveness of this attack.
10. **Sinkhole attacks** The attacker places a malicious node in a key point (close to the base station). The node drops all the received messages instead of forwarding them to the base station. The result of this attack is more disrupting if there is a unique BS in the network.
11. **Sybil attacks** In this attack, a node declares multiple illegitimate identities either by forging or stealing the identities of legitimate nodes. This attack can be used against routing algorithms, topology maintenance and voting scheme.

Network layer	Attacks	Defenses
Physical	Jamming	Spread-spectrum, priority messages, lower duty cycle, region mapping, mode change
	Tampering	Tamper-proofing, hiding
Link	Collision	Error-correcting code
	Exhaustion	Rate limitation
	Unfairness	Small frames
Network and Routing	Neglect and greed	Redundancy, probing
	Homing	Encryption
	Misdirection	Egress filtering, authorization, monitoring
	Black holes	Authorization, monitoring, redundancy
Transport	Flooding	Client puzzles
	Desynchronization	Authentication

Table 1.3: Sensor network layers and denial-of-service defenses

12. **Node replication** The attack is made by adding a node, with a replicated (copied) ID of an existing node, to the targeted sensor network. The attacker can copy also cryptographic keys in the malicious node. This attack can result in a disconnected network, false readings
13. **Wormhole** An attacker records the packets at one location in the network, tunnels them (possibly selectively) to another location, and retransmits them into the network. This attack can form a serious threat against the routing protocols and the location-based wireless security systems [27].
14. **Hello flood attack** The malicious node broadcasts a HELLO message with a strong transmission power and pretends that it is coming from the base station. The receiving nodes assume that the malicious node is the closest one to the BS, so they send their data through it. In this attack, the responding nodes to HELLO floods waste their energy.
15. **DoS attacks** DoS appears as any event that diminishes or eliminates a network capacity to perform its expected function (Hardware failures, software bugs, resource exhaustion, environmental conditions, or their combination; or intentional attack). This attack targets the availability (which ensures that the authorized parties can access data, services, or other computer and network resources when requested) by preventing the communication between the network devices or by preventing a single device from sending the traffic [28].

1.6 Conclusion

WSNs become widely used in many applications and the security becomes a key concern to those networks. The traditional security approaches in wirelined and wireless networks are not merely applicable to WSN due to the resource limitation of the nodes and obstacles encountered by those networks.

This report focuses on data aggregation. In the next chapter, we discuss the interest for aggregation technique as well as we list some security protocols supporting the data aggregation.

2 Secure data aggregation approaches in Wireless Sensor Networks

2.1 Introduction

WSNs are composed by a huge number of sensor nodes. Figure 2.1(a) shows an important number of messages exchanged inside a network in response to a query sent by the BS. The large number of nodes may lead to a huge amount of data in the network, causing the network to degrade its performance and shorten its lifetime. The data aggregation techniques may be a solution to remove the redundancy of data and thereby reduce the number of packets transmitted in the network. For those reasons, some nodes can have the responsibility of aggregating data (messages) before transmitting them to BS. Figure 2.1(b) shows a network where the cluster heads (CH) have the data aggregation capability. We remark that the number of messages is significantly decreased when using the aggregator nodes in the network. Nevertheless, the data aggregation causes some security vulnerabilities: impersonation, denying having received data, dropping packets deliberately, alteration of sensory readings and aggregation results,...

Many security solutions have been proposed in the literature to secure the aggregation process, the most important ones are depicted in this chapter.

2.2 Secure Aggregation for Wireless Networks

In [2], Hu et al. present a secure aggregation protocol for the wireless sensor networks, it is the first article that handles this problem. In this protocol we consider the existence of one powerful node called a base station (BS) which can broadcast messages to all the nodes directly. The other nodes are all identical and organized in a binary aggregation tree rooted by the BS. Some are acting as leaves (nodes A-D in Figure 2.2) and they are responsible for the sensing activities, and the others are acting as intermediate nodes. The aggregation is done only in intermediate nodes and BS. The protocol evolves in two steps:

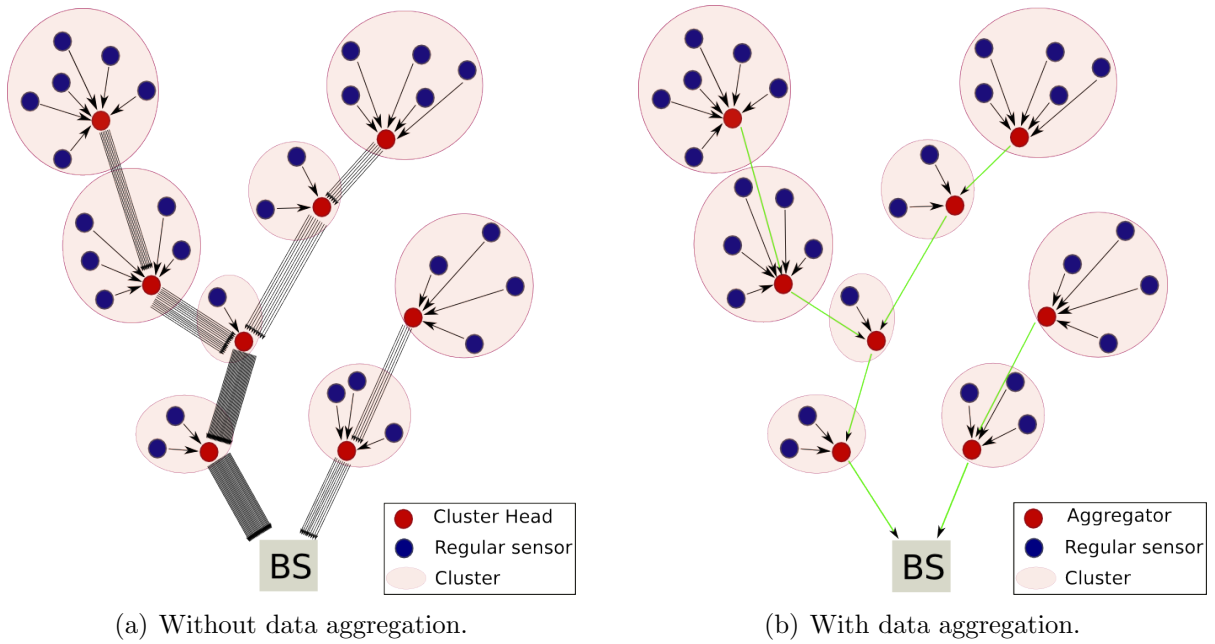


Figure 2.1: Forwarding packets with and without data aggregation

delayed aggregation and delayed authentication. Exchanged messages are authenticated with temporary keys. The temporary key is computed by encrypting a counter value using a key shared between the node and the BS. For example K_{AS} is the key shared between the node A and the BS, and $K_{A0} = E(K_{AS}, 0)$ is a temporary key of A. After the aggregation phase, the BS reveals the temporary keys to enable other sensor nodes to authenticate messages transmitted by nearby sensors. The counter is incremented after each cycle. The processing of aggregation is done as follows (see the network presented in Figure 2.2):

1. Each leaf node N transmits a message to its parent containing its unique identifier and its sensory reading; the message is authenticated by a secret key K_{Ni} only known at that moment N and BS. For example :

$$A \rightarrow E : R_A | ID_A | MAC(K_{Ai}, R_A)$$

2. Upon receiving the messages from its children, the parent node cannot verify their authenticity, so it has to store the messages and verify them later when receiving K_{Ni} . It waits until receiving all its children messages or the waiting timer expiration and then it sends a message to its parent that retransmits that sensory readings and MACs, along with a computed MAC over the calculated aggregate value. For example :

$$E \rightarrow G : R_A | ID_A | MAC(K_{Ai}, R_A) \\ | R_B | ID_B | MAC(K_{Bi}, R_B)$$

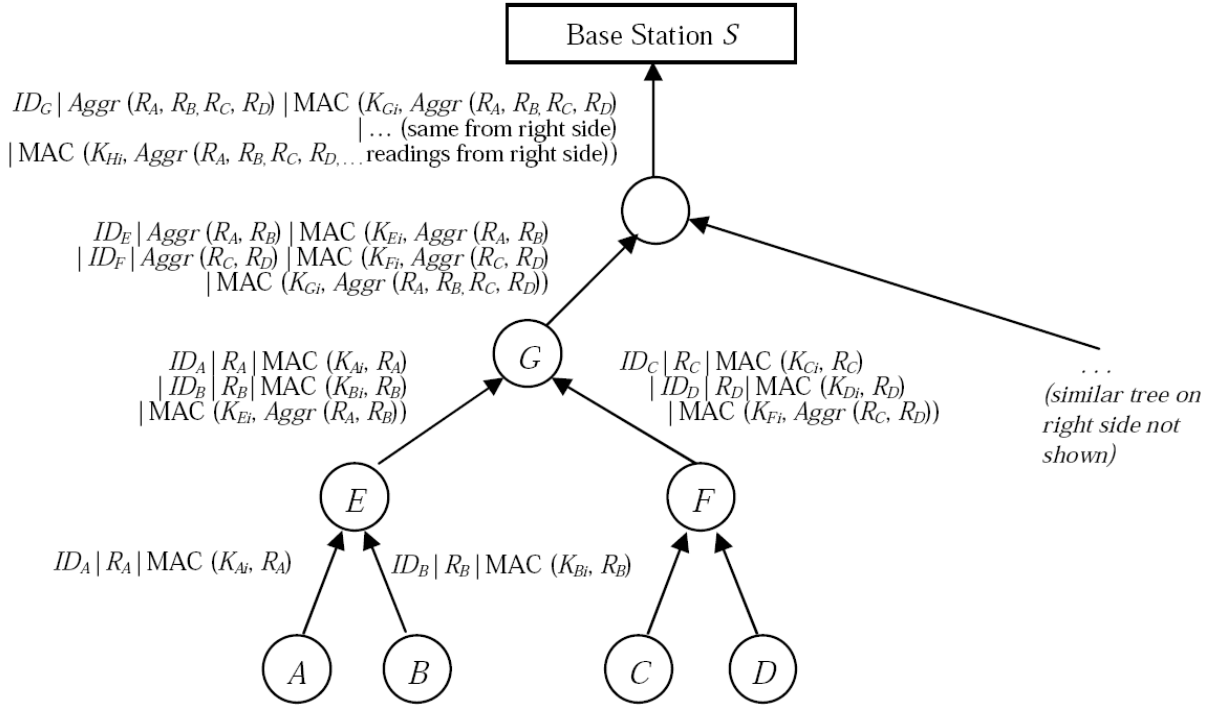


Figure 2.2: Data aggregation illustration

$$|MAC(K_{Ei}, Aggr(R_A, R_B))$$

Note that E does not send ID_E to G since G knows enough about the network topology.

3. Upon receiving the messages from E and F, node G calculates the aggregation result of its grandchildren sensory readings through each child. It then associates the aggregation result of its grandchildren and its children's ID and MAC values in a message; along with a MAC that it computes with its secret key K_{Gi} over the next aggregate result, $Aggr(R_A, R_B, R_C, R_D) = Aggr(Aggr(R_A, R_B), Aggr(R_C, R_D))$, and it sends the message to its parent. For example the message sent from G to its parent is:

$$\begin{aligned}
 &ID_E | Aggr(R_A, R_B) | MAC(K_{Ei}, Aggr(R_A, R_B)) \\
 &| ID_F | Aggr(R_C, R_D) | MAC(K_{Fi}, Aggr(R_C, R_D)) \\
 &| MAC(K_{Gi}, Aggr(R_A, R_B, R_C, R_D))
 \end{aligned}$$

4. The processing described in (3) is recursively repeated upstream until reaching the BS. So, in our case, node H receives the messages from its children, and sends the aggregate message to the BS.

5. Upon receiving the messages from its children, the BS calculates the final aggregation result. Then it broadcasts the authentication keys used by all the nodes in this aggregation round to let them verify the authenticity of the already received messages. Then the nodes verify the MACs and trigger an alarm if it does not match.

This protocol guarantees the authenticity of the messages for the networks where two consecutive nodes are not compromised. So, the compromising of two nodes as closest as possible to the BS, makes the attacker more powerful to falsify the final aggregation result.

We had implemented and tested this protocol, the results of this work are discussed in 3.3.2.

2.3 SHIA

Secure hierarchical in-network aggregation in sensor networks (SHIA) [29] is a secure SUM aggregation protocol. This protocol is useful for sum, count, average and median calculations. It has three main phases : query dissemination, aggregation commit and result checking.

2.3.1 Query dissemination

The base station broadcasts a query to the network. An aggregation tree is established if it is not already present. The query message contains a nonce value N to prevent replay of messages and it is authenticated.

2.3.2 Aggregation commit phase

Sensory data and aggregation results are included in a data structure called label. Label's format is $\langle count, value, complement, commitment \rangle$ where *count* is the number of leaf vertices in the subtree rooted at this vertex; *value* is the SUM aggregate computed over all the leaves in the subtree; *complement* is the aggregate over the COMPLEMENT of the data values; and *commitment* is a cryptographic commitment. The labels are defined inductively as follows: There is one leaf vertex u_s for each sensor node s , which we call the leaf vertex of s . The label of u_s consists of $count = 1$, $value = a_s$ where a_s is the sensory value of s , $complement = r - a_s$ where r is the upper bound on allowable data values, and $commitment$ is the node's unique ID. Internal vertices represent the aggregation operations, and have labels that are defined based on their children. Suppose an internal vertex has child vertices with the following labels: u_1, u_2, \dots, u_q , where $u_i = \langle c_i, v_i, \bar{v}_i, h_i \rangle$. Then the vertex has label $\langle c, v, \bar{v}, h \rangle$, with $c = \sum c_i$, $v = \sum v_i$,

$\bar{v} = \sum \bar{v}_i$ and $h = H[N||c||v||\bar{v}||u_1||u_2||\dots||u_q]$. Two approaches are described in this phase: the naive approach and the improved approach:

2.3.2.1 Naive approach

Leaf nodes in the aggregation tree (e.g. node G in the figure 2.3) send their leaf vertex to their parent. The internal node (like A in figure 2.3) aggregates all the received labels and its leaf vertex, and sends the resulted label to its parent.

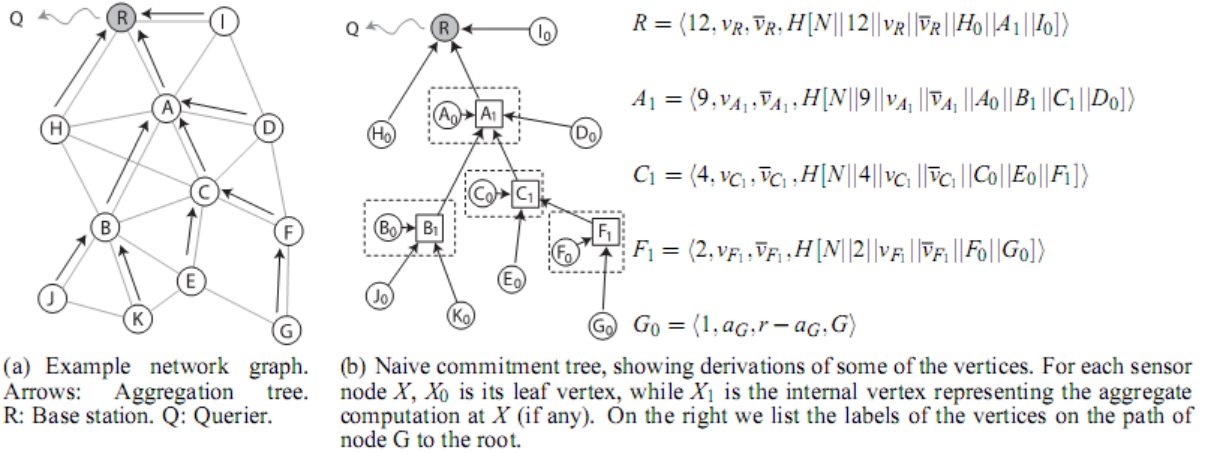


Figure 2.3: Aggregation and naive commitment tree in network context

2.3.2.2 Improved approach

This approach aims to improve the congestion cost in forming balanced aggregation trees. The difference from the last approach is that nodes aggregate only labels with the same depth (count). The leaf sensor nodes in the aggregation tree originate a single label which they then communicate to their parent sensor nodes. Each internal sensor node s originates a similar single label. In addition, s also receives labels from each of its children. s receives one or more label from each of its direct children. It then combines all the labels (its single label and the received ones) to form a new set of labels as follows. Suppose s wishes to combine q labels L_1, \dots, L_q . We let the intermediate result be $SL = L_1 \cup \dots \cup L_q$, and repeat the following until no two labels have the same count in SL : Let c the smallest count such that more than one label in SL has count c . Find two labels L_i and L_j of count c in SL and merge them into a label of count $(2 \times c)$ by creating a new label that is the parent of both L_i and L_j . When no two labels have the same count in SL , node s sends the set of labels SL to its parent. This process in node A is described in figure 2.4.

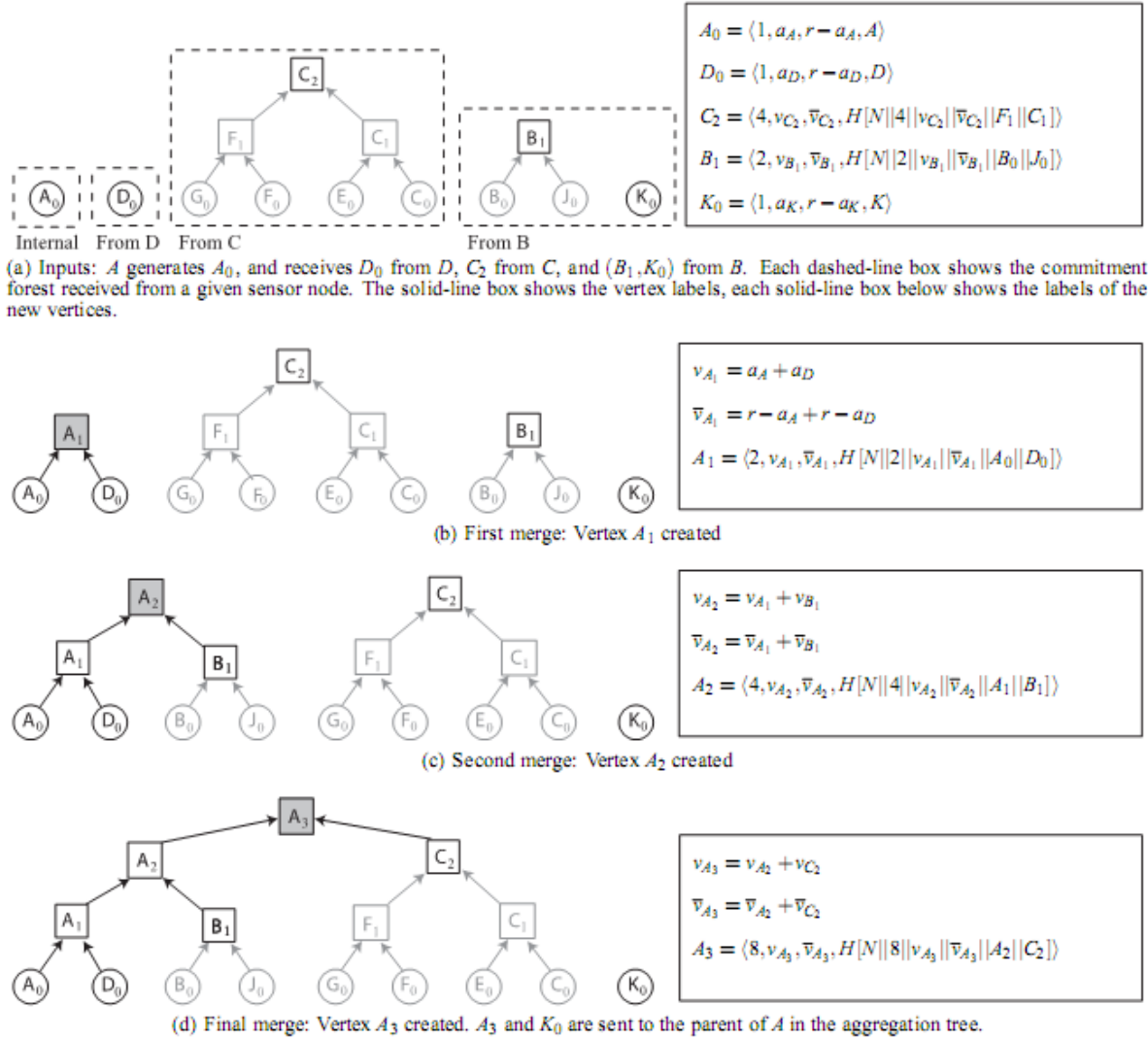


Figure 2.4: Process of node A (from Figure 2.3) deriving its commitment forest from the commitment forests received from its children

2.3.3 Result-checking phase

This phase lets each sensor s verify that its sensory data a_s were added into the SUM aggregate and the complement $(r - a_s)$. Each sensor verifies that its label was included in the calculation of the final root label. Thus by inspecting the inputs and the aggregation operations in the commitment tree. This phase is split into five steps:

2.3.3.1 Dissemination of final commitment values

After the BS has received the final set of labels, it sends each of these labels to the entire sensor network using an authenticated broadcast.

2.3.3.2 Dissemination of off-path values

To enable verification, each leaf vertex must receive all its off-path values. Each internal node sends any labels received from its parent to all its children. It sends also the set of labels received from its child u to all its other children (Figure 2.5).

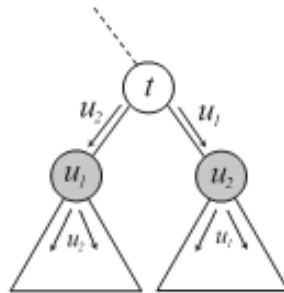


Figure 2.5: Dissemination of off-path values: t sends the label of u_1 to u_2 and vice-versa; each node then forwards it to all the vertices in their subtrees

2.3.3.3 Verification of inclusion

When the node u_s receives all the labels of its off-path vertices, it may then verify the labels until obtaining the root set of labels and verify that the received from the BS and the calculated ones are equal.

2.3.3.4 Collection of confirmations

After the successful verification of inclusion, each sensor node s sends an acknowledgment to the BS with form $MAC_{k_s}(N||OK)$ to the BS. Those messages are XOR-ed hop by hop. Nodes with failed verification of inclusion send nothing.

2.3.3.5 Verification of confirmations

When the BS receives the XOR-ed acknowledgments, it calculates an XOR-ed acknowledgment and verifies that the received and the calculated ones are equal.

2.4 SAPC

In the Secure Aggregation Protocol for Cluster-Based wireless Sensor network (SAPC) [1], nodes are organized into disjoint cliques (clusters), thanks to Sun et al. protocol [30], where each node is one-hop away from the remaining nodes of the cluster. Then nodes in each cluster elect a Cluster-Head (CH) from them to act as a CH and an aggregator, to communicate with the BS, and roots other CHs messages to BS. The organization of the network is described in figure 2.6. In addition, the nodes share a secret key with the BS, initially loaded before deployment. Also, each pair of nodes within the same clique shares a pairwise key to authenticate their messages. To authenticate its locally broadcast messages, each node u generates a one-way key chain $\{K_u^n\}$, and u sends the commitment key of the key chain to each neighbor, authenticated with the pairwise key.

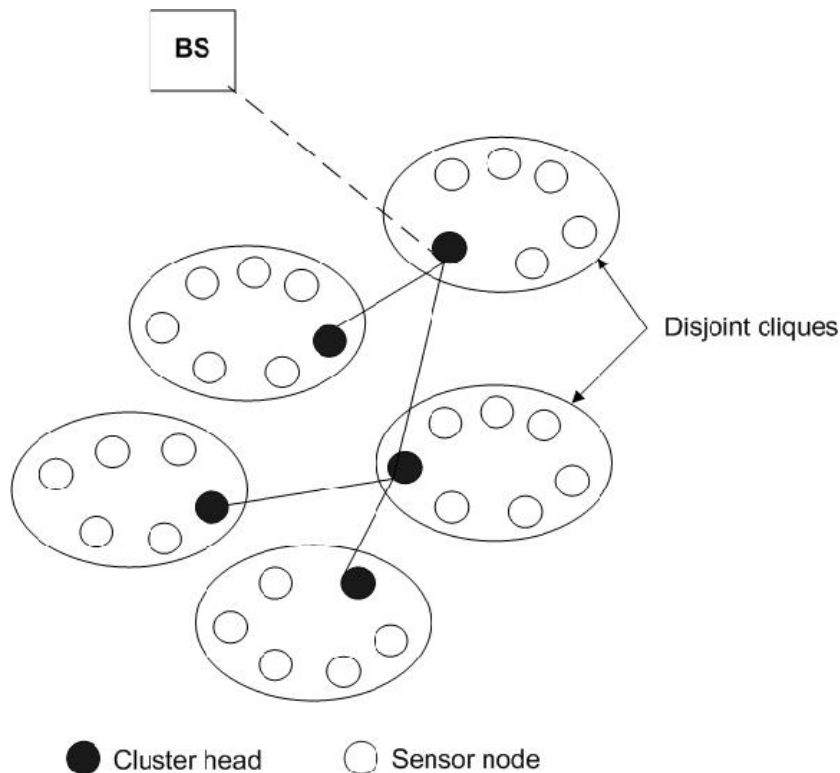


Figure 2.6: Cluster-based wireless sensor network

The aggregation is done in each cluster, and all the nodes participate in its calculation. The BS, which knows the list of sensors per cluster, can check whether the aggregation

result of a cluster was approved by cluster members or not. An aggregation round is described by four main steps, the l^{th} aggregation round in the cluster C headed by node CH_i (C_{CH_i}) is done as follows :

1. Each node u in the cluster broadcasts its sensory reading R_u authenticated with the current key K_u^j of its key chain

$$u \rightarrow * : R_u || MAC_{K_u^j}(R_u) || K_u^j$$

2. Each node $v \in C_{CH_i}$ receives all the broadcasted reading messages of u ($\forall u \in C_{CH_i} \setminus \{v\}$). For each received message, v verifies it by following those steps :

- (a) verify the authenticity of the key K_u^j ($K_u^{j-1} = H(K_u^j)$), if succeeded then
- (b) verify the MAC field

If the two conditions are accepted, node v stores the key K_u^j (to use it to verify the u 's reading message in the next round). Note that this key is used once, so it is never used to authenticate another message.

After receiving all the reading messages, the node calculates the aggregation result : $AGR_v = f(R_u / \forall u \in C_{CH_i})$.

Then node v prepares a double authenticated message, the first MAC is generated by the pairwise key $K_{BS,v}$, shared between node v and BS, over the aggregation AGR_v and counter C_v where C_v is a counter shared between node v and BS and it is used to protect BS from replay attacks. The second MAC is calculated over AGR_v and the first MAC with the pairwise key shared between node v and CH_i . Node v sends this message to CH_i :

$$v \rightarrow CH_i : \overbrace{AGR_v || MAC_{K_{BS,v}}(AGR_v, C_v)}^1 || MAC_{K_{CH_i,v}}(1)$$

3. The CH_i verifies all the received messages using the secret pairwise keys. Unauthenticated messages are ignored. Logically all the messages contain the same aggregation result, but it might exist some malicious or faulty nodes (less than the majority in the cluster), or some nodes which have not received all the reading messages due to collisions. Anyway, the cluster adopts the majority aggregation result AGR, it XORs the MACs for BS from the messages with the aggregation result equal to AGR, and finally it authenticates the message with its shared pairwise key with BS and sends it to BS. Node IDs with different aggregation results are included in the message.

$$CH_i \rightarrow BS : \overbrace{AGR \parallel \bigoplus_{v \in C_{CH_i}} MAC_{K_{BS,v}}(AGR_v, C_v)}^2 \parallel MAC_{K_{BS,CH_i}}(2)$$

4. Upon receiving the message sent by CH_i , the BS verifies its authenticity using K_{BS,CH_i} . If authenticated, BS proceeds in the verification of the XOR-ed MAC by calculating a set of MACs using the set of its shared keys with cluster nodes and then XOR them. If the calculated and received MACs are equal, the result is authenticated, otherwise it is rejected (Note that BS excludes the MACs of the nodes which sent a different aggregation result to CH_i).

2.5 A trust based framework for Secure data Aggregation in Wireless Sensor Network

2.5.1 System model

In this framework [31], we consider those assumptions:

- a static sensor network composed of a large number of densely deployed sensors which are organized into clusters
- in the network model, all the nodes are similar but they have three different behaviors :
 - sensor nodes send their sensory reading values to an aggregator
 - aggregator collects readings, aggregates them and sends a report to CH
 - CH aggregates the received reports from the aggregators and forwards the aggregated results to the BS (sink)
- in one cluster, all the sensor nodes including the CH and aggregators are physically close to each other and hence their sensory data are highly correlated
- every sensor node has a pairwise key with its one-hop neighbors, and uses it in a MAC for authentication.

2.5.2 Threat model

In this model, the attacker can inject/replay messages, compromise a sensor node either physically by capturing the node or by spreading a malicious code, and obtain all the secret materials.

2.5.3 Josang's Belief Model

Josang's model [31] proposes a belief metric to express an opinion about an aggregation result. It defines a quadruple $w = (b, d, u, a)$ where b is belief, d is disbelief, u is uncertainty, a is relative atomicity; $a, b, d, u \in [0, 1]$ and $b + d + u = 1$. The opinion O is calculated as

$$O = E(w) = b + au \quad (2.1)$$

This model is suitable to capture the uncertainty in data aggregation in WSNs since the sensory data and aggregation results are infiltrated with uncertainties due to the unavoidable sampling errors, false data injected by either compromised source nodes or aggregators.

2.5.4 A trust based framework against false data injection

The aggregation process in the protocol is as follows:

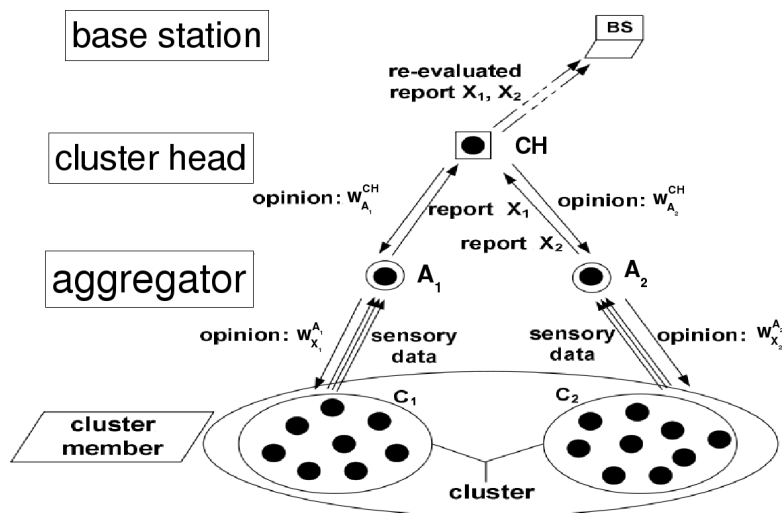


Figure 2.7: Abstract architecture of the framework [31]

- Each sensor node reports its sensory data to its corresponding aggregator. The sensory data should be protected by a MAC with the pairwise key shared between the node and its aggregator.
- Upon receiving reports from sensor nodes, the aggregator does those steps :
 - It eliminates sample values significantly deviated from the median
 - The sensing data of the sensor nodes follow a normal distribution : $N(\mu, \sigma)$, where μ is the mean and σ is the standard deviation. In a long run, if the sampling is independent between each round, the probability of one node's

sensory data falling in $[\mu - \sigma, \mu + \sigma]$ should be 0.68, this is *ideal node frequency* (f_{ideal}) distribution. Some aggregators calculate the parameters μ and σ , then they update each *actual node frequency* (f_a).

- The aggregator calculates the *Kullback-Leiber* (KL) distance between the ideal and actual distributions for each node. The KL distance D is calculated as follows :

$$D = (1 - f_a) \log_2 \left(\frac{1 - f_a}{1 - f_{ideal}} \right) + f_a \log_2 \left(\frac{f_a}{f_{ideal}} \right)$$

- Aggregator calculates the reputation of each sensor node as $\frac{1}{1 + \sqrt{D}}$
- Aggregator dynamically classifies sensor nodes, according to their reputation, into K groups with the K-Means partition algorithm.
- After classifying the nodes, the aggregator calculates the average (\bar{x}) of sensory values of nodes in the highest reputation group as well as the standard deviation (σ) as its aggregation result in this round.
- The aggregator counts the number of nodes having their sensory values within $[\bar{x} - \sigma, \bar{x} + \sigma]$ and treats those nodes as trustworthy for this aggregation round.
- The aggregator A formulates its opinion $w_X^A = \{b^A, d^A, u^A, a^A\}$ about the aggregation result X as follows:

- * b^A : percentage of nodes fall in $[\bar{x} - \sigma, \bar{x} + \sigma]$
- * $u^A = 1 - b^A$
- * a^A : average reputation of the uncertain nodes
- * $d^A = 0$
- * the expectation of the aggregator's opinion about aggregation result X is $O_X^A = b^A + a^A u^A$

- The aggregator sends its report, composed by aggregation result and opinion, to the cluster head.
- Upon receiving reports from aggregators, cluster head compares its own sensing data with the received aggregation results. It formulates an opinion about aggregators on two steps:

- It opts for the majority value and considers nodes with this value as *honest* and treats the rest as *dishonest*.
- The number of times the aggregator A is considered as honest by the cluster head (H) is noted by k_A^H ; and l_A^H is the contrary. The cluster head opinion about aggregator A, $w_A^H = (b_A^H, d_A^H, u_A^H, a_A^H)$ is obtained by:

$$b_A^H = \frac{k_A^H}{k_A^H + l_A^H + 2}, d_A^H = \frac{l_A^H}{k_A^H + l_A^H + 2}, u_A^H = \frac{2}{k_A^H + l_A^H + 2}$$

O_A^H like defined in equation (2.1).

- This step is known as *belief discounting*, in which the cluster head formulates an opinion about each aggregator report X. This by discounting of w_X^A (the aggregator A opinion about its aggregation result X which is included in its emitted report) by w_A^H (opinion of cluster head H about aggregator A) to obtain $w_X^{HA} = (b_X^{HA}, d_X^{HA}, u_X^{HA}, a_X^{HA})$ where

$$\begin{aligned} b_X^{HA} &= b_A^H \times b_X^A, & d_X^{HA} &= b_A^H \times d_X^A, \\ u_X^{HA} &= d_A^H + u_A^H + b_A^H \times u_X^A, & a_X^{HA} &= a_X^A. \end{aligned}$$

The expectation of the cluster head's opinion about the aggregator's report is $O_X^{HA} = b_X^{HA} + a_A^H \times u_X^{HA}$.

- Finally, as the cluster head receives a report from each aggregator, suppose that they are two aggregators, the final aggregation result is calculated by $X = \omega_1 X_1 + \omega_2 X_2$, where ω_1 and ω_2 are weighting factors defined as

$$\omega_1 = \frac{O_X^{HA_1}}{O_X^{HA_1} + O_X^{HA_2}}, \omega_2 = \frac{O_X^{HA_2}}{O_X^{HA_1} + O_X^{HA_2}}$$

The cluster heads send its report to the base station.

In addition to its role of sensing and reporting data to its aggregator, the sensor node overhears the transmitted reports by aggregators and cluster head to update their reputations. The cluster members can reselect an aggregator or cluster head either if their reputation drops below a certain threshold or to balance energy consumption between nodes by rotating periodically the roles.

2.6 Conclusion

In this chapter, we show different approaches to secure data aggregation in WSNs. In the next chapter we compare two protocols SAPC and Hu et al.

3 Implementation and Tests

3.1 Introduction

Many secure aggregation protocols are proposed in the literature, as described in the previous chapter. It exists two types of protocols: some protocols perform local aggregation in the clusters [1, 31] and the others perform hop-by-hop data aggregation [2, 29]. We selected two protocols from the two different types for implementing and testing which are SAPC [1], the protocol proposed by our laboratory, and Hu et al. [2] protocol. The evaluation of the performance is based on two criteria: the message complexity and the computation time. The message complexity is defined as a measure where the overhead of the protocol is measured in terms of number of messages needed to satisfy the protocol's request [32]. The computation time is measured on a tmote sky [33] sensor node which uses a 16-bit, 8MHz Texas Instruments MSP430 microcontroller with only 10 KB RAM, 48KB Program space, 1024 KB External flash, and which is powered by two AA batteries. Results of this work are the concern of this chapter.

3.2 Implementation

We implemented SAPC and Hu et al. protocols in TinyOS 2.x to test and analyze them. The memory spaces occupied in a tmote sky sensor by a node (not a BS) program are described in the table 3.1. In this implementation, we use CBC-MAC RC5 to authenticate the messages as it requires minimum CPU cycles and execution time [21].

Memory space	SAPC	Hu et al.
<i>ROM (bytes)</i>	29124	26364
<i>RAM (bytes)</i>	3308	3215

Table 3.1: Memory space occupation in SAPC and Hu et al. protocols

3.3 Tests

3.3.1 SAPC

Let's start with the analysis of the number of messages sent in the SAPC protocol. The node in the cluster should send at least two messages in each round of the aggregation and the CH should broadcast one message containing its reading and another to the BS containing the aggregation result, but the CH should also forward the aggregation messages of other CHs to the BS. So in the optimized case, a CH sends two messages when it is a leaf in the routing tree or when all CHs are one hop away from the BS. The figure 3.1 compares the maximum and minimum numbers of messages sent or forwarded by a CH. For example in a network of 400 nodes, the maximum number of messages is 37. If we calculate the total number of messages in the network by adding the number of messages sent or forwarded by each node in one round, the maximum number is 1430 messages and the minimum is 800 (Figure 3.2).

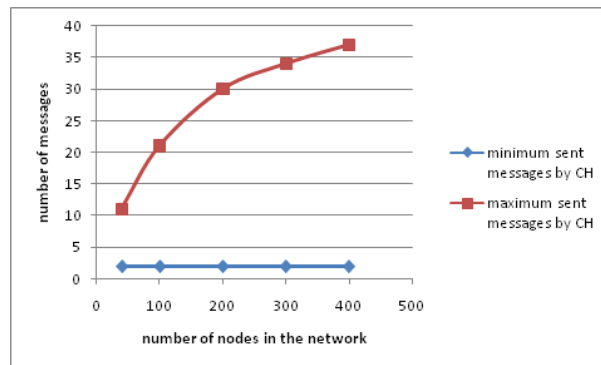


Figure 3.1: Number of messages sent by CH

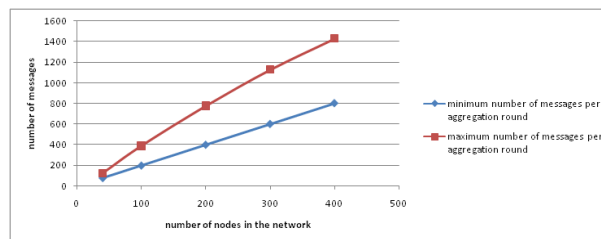


Figure 3.2: Minimum and maximum number of messages in the network

Now let us evaluate the calculation time needed by a simple node (neither CH nor BS) to accomplish its tasks (figure 3.3), i.e. to:

- prepare a message containing the sensed reading and authenticate it

- receive other reading messages, verify the authentication key ($K_u^j = H(K_u^{j-1})$) and then verify the MAC
- calculate the aggregation result and authenticate this message with two MACs

For this node, it takes 146.4 ms to accomplish those tasks in a cluster formed of 11 nodes. The distribution of this time between different tasks is described in figure 3.3.

The CH, in addition to the previous tasks has to receive the aggregation messages, verify them and calculate the final aggregation result. The final aggregation result computation includes XORing the MACs of all the nodes having the same result, authenticating the message and finally sending it to the BS. To accomplish those tasks in a cluster formed of 11 nodes, it takes a total calculation time of 187,5 ms in a CH. The distribution of this time between different tasks is described in figure 3.4.

Figure 3.5 compares the calculation cost for a CH vs a simple node, according to the total number of nodes in the network. Note that the two calculation costs are close although a simple node implements fewer functions than a CH. We can deduce that the position of the node in the hierarchy has little impact on the calculation cost.

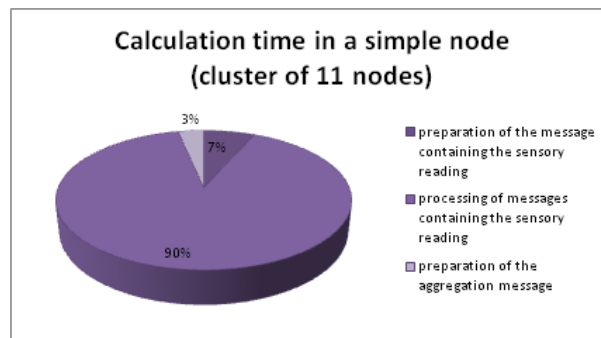


Figure 3.3: Calculation time in a normal node

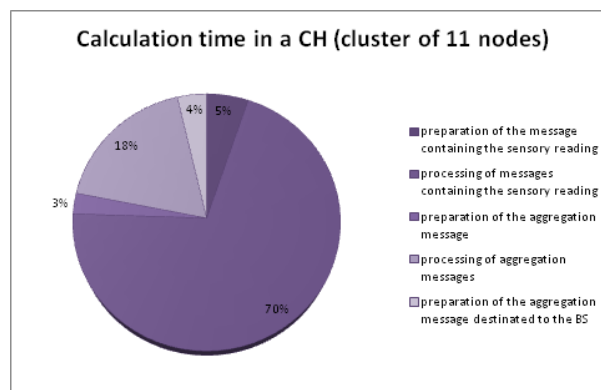


Figure 3.4: Calculation time in CH

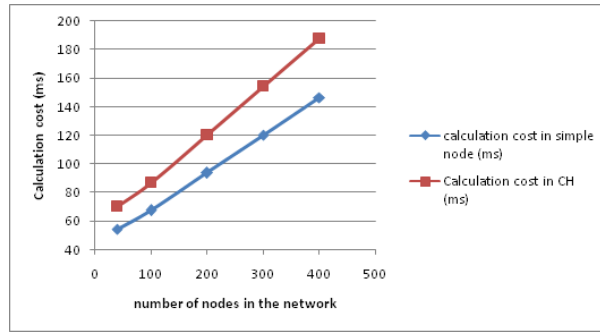


Figure 3.5: Calculation cost comparison for CH vs simple node

3.3.2 Hu et al.

In this protocol we can differentiate between three kinds of nodes: (1) leaf nodes, (2) aggregator nodes with leaf nodes descendants and (3) aggregator nodes with aggregators descendants.

The leaf node generates a message containing its reading (sensored data) and sends it to its parent. The aggregator of types (2) or (3) receives two messages from its descendants and then it sends a message to its parent. In fact, each node in this protocol sends one message but the aggregators should receive two others; this is independent of the number of nodes in the network and if we suppose that our network is organized into a binary tree.

In our implementation, each node uses its authentication key once and sends it with the message. The authenticity of the key is verified before the verification of the authenticity of the message.

A node of type (1) needs 4,0283 ms to prepare the message containing the sensed data and to perform the authentication. Nodes of type (2) need 19,683 ms to verify the authenticity of the keys and received messages, to calculate the aggregation result and to authenticate it. Nodes of type (3) need 44,616 ms to verify the authenticity of the three keys and the three MACs in each received message, to calculate the aggregation result and authenticate it.

3.4 Conclusion

We remark that in the Hu et al protocol, the number of messages is constant, and equal to the number of nodes in the network. Each node sends a unique message whatever its type. In SAPC, the number of messages depends on the number of nodes, the position and role of the node. So, it is constant for a simple node and equal to 2; but it is variable for the CH; it is in the range of 2 to 11 in a 40 node network and in the range of 2 to 37 in a 400 node network. The length of the data messages used in the implementation is

52 bytes in Hu et al, and 30 bytes in SAPC.

The verification of the reading messages consumes most of the time in SAPC. It takes 90% of the total execution time in a simple node and 70% in the CH. An attack with replayed or forged reading messages might cause an important calculation overhead in the sensor node. The security of the Hu et al. aggregation result can be altered if there are two consecutive malicious nodes not being detected.

In the next chapter, we present a new aggregation protocol to enhance the performances of SAPC by aggregating the exchanged messages. In addition to the local cluster aggregation, we propose a solution to do the hop by hop aggregation based on the binary distributed commitment tree.

4 SeBTIA: Secure binary tree in-network aggregation

4.1 Introduction

A node in a wireless sensor network sends its measurements to a BS and it can route the messages of other nodes. If it has not direct communication with the BS, it sends them to an intermediate node. So the network is organized like a tree: BS is the root, and the sensor node is a leaf if it does not route other nodes messages otherwise it is an intermediate node. Each node knows its direct children and its parent which is BS or an intermediate node. A distributed algorithm can be used to build this tree like the described one in [34]. The BS knows the organization of the tree, and each rearrangement should be authorized by the BS. We call this tree an aggregation tree.

The hop-by-hop data aggregation is an interesting way to reduce the communication overhead and energy expenditure of the sensor nodes. Although, the data aggregation can let malicious or faulty nodes deviate the result, we propose in this chapter a protocol that lets the sensor nodes and the BS verifying the final aggregation result to eliminate the malicious nodes. The protocol has two phases: an aggregation-commitment phase to collect sensory data and aggregate them hop-by-hop, and a verification phase in which the sensor nodes verify that their sensory data have been included in the aggregation result. If there is a negative acknowledgment, the BS eliminates the suspected malicious nodes.

4.2 Commitment tree

4.2.1 Notations

- BS : Base Station
- O_t : Ordered aggregation tree
- B_t : commitment tree or binary tree
- $|D(O_t, A)|$: number of vertex A 's children in O_t

- $D(O_t, A, i)$: the child number i of A in O_t (children are numerated from left to right)
- $P(O_t, A)$: A 's parent in O_t
- L_i^A : label i of node A . Its form is organized in $\langle N, A, i, n_A, v_A, commitment \rangle$
 - N : a value to guarantee the freshness of the messages, it can be a nonce value included in the query disseminated by the BS or a counter incremented in each round of aggregation. It is a unique value used by all the nodes during the aggregation round.
 - A : the node's identifier
 - i : a value in the range $[0..2]$
 - n_A : the number of nodes in the subtree rooted by A
 - v_A : the aggregation result of the subtree
 - commitment: cryptographic value. This field is noted next as h_A
- f : is the aggregation function (max, min, mean, ...) with two parameters as inputs for function like max or min or with four parameters as inputs for functions like mean ($f(v_B, v_A, n_B, n_A) = \frac{v_B \times n_B + v_A \times n_A}{n_B + n_A}$)
- H : the hash function used to calculate the commitment value
- $F: L \times L \rightarrow L$

$$L_{i+1}^A = F(L_j^B, L_i^A)$$

$$= \langle N, A, i + 1, n_B + n_A, f(v_B, v_A), H(N, A, i + 1, n_B + n_A, f(v_B, v_A), h_B, h_A) \rangle$$

where L is the set of all possible labels

- K_{A-B} : a pair wise key used to authenticate messages between nodes A and B
- K_A : a pair wise key used to authenticate the messages between node A and the BS

4.2.2 Ordered tree

The obvious method that a vertex can use to order its children is based on their identifier. In our protocol, we are going to use three parameters. First, the vertex orders them decreasingly based on their number of descendants. This order is not total; the vertex can have some children with equal number of descendants. Second, it can rearrange them increasingly based on the depth of the subtree rooted by those nodes. This order is not total too; finally children with equal numbers of descendants and depth are ordered based on their unique identifier.

When each node in the aggregation tree orders its children, we obtain an ordered aggregation tree.

4.2.3 Formation of the commitment tree (binary tree)

There is a one-to-one mapping between the general ordered trees and binary trees, which in particular is used by Lisp to represent general ordered trees as binary trees [35]. Each node N in the ordered tree corresponds to a node N' in the binary tree; the left child of N' is the node corresponding to the first child of N , and the right child of N' is the node corresponding to N 's next sibling (the next node in order among the children of the parent of N). For example figure 4.1, in the left tree, A has the 6 children B,C,D,E,F,G . It can be converted into the binary tree on the right.

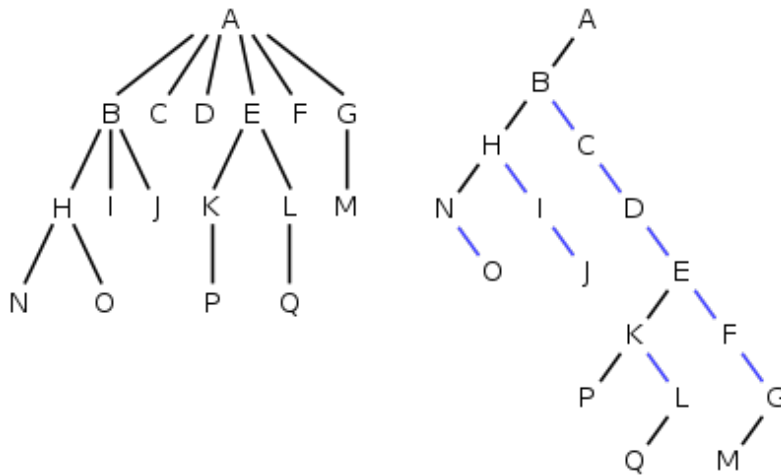


Figure 4.1: Encoding n-ary trees as binary

We apply this algorithm on an ordered aggregation tree as defined in paragraph 4.2.2 to form a commitment tree. This formation is distributed in each intermediate node, which knows only its children, and is able to organize them and send the result to its parent. So the real structure of the tree is not known by any node, only the BS knows the organization of nodes and then of the commitment tree. An example of a commitment tree is the right tree in figure 4.2, the left tree is an ordered (routing) tree. Before describing the commitment tree formation process in section 4.3, we describe how to represent the tree nodes and labels in a graph.

4.2.4 Logical representation of the commitment tree

In the commitment tree, a node can at most have three links. Each link means a label calculation. The logical representation of the commitment tree is represented in figure 4.3.

- ID: Identifier of the node, for example the identifier of the highlighted node in bold is 23
- L_0^{23} : Label 0 of node 23 represents the initial node label $\langle N, 23, 0, 1, v_{23}, h_{23} \rangle$

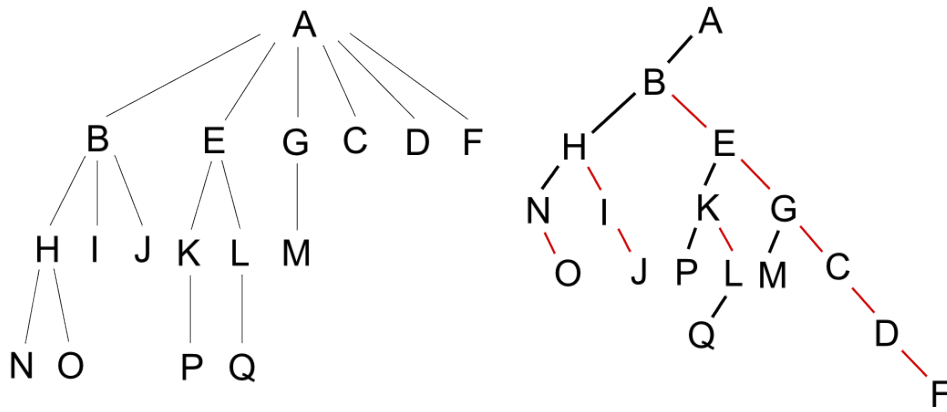


Figure 4.2: Formation of the commitment tree

- L_1^{23} : Label 1 of node 23 is the result of function F applied on L_2^{38} and L_0^{23} . $L_1^{23} = F(L_2^{38}, L_0^{23})$. The node sends Label 1 to its parent. If a node has no child in the routing tree, its Label 1 will be equal to its Label 0.
- L_2^{23} : Label 2 of node 23 is the result of function F applied on L_2^{15} and L_1^{23} . This Label is calculated by the parent of node 23 in the routing tree. If a node has not a right child in the commitment tree (e.g. has not a right sibling in the ordered routing tree), its Label 2 is equal to its Label 1.

Note that all the labels of a leaf node (in the commitment tree) are equal.

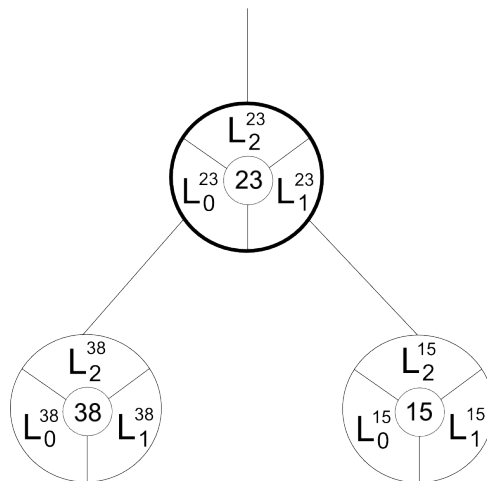


Figure 4.3: Logical representation of the commitment tree

4.3 Protocol description

4.3.1 Query Dissemination

Our protocol may be used first to aggregate the nodes responses to a query disseminated by the BS. This query can be authenticated by using an authentication protocol like μ -tesla [22] or H2BSAP [36]. Second, if nodes should periodically send a report containing their measurements of some physical phenomenon to the BS, those reports can be aggregated thanks to our protocol.

4.3.2 Cryptographic Security

Exchanged messages between nodes are authenticated by a pairwise key. When a node receives a message, it verifies that N' used in the label is equal to N used in the current aggregation round. Then it verifies the number of descendants (each node knows the number of descendant of its children). Finally it verifies the message authentication code with the corresponding pairwise key.

4.3.3 Aggregation-Commitment phase

The algorithm of this phase is described in figure 4.5. In the beginning, each node prepares its initial label L_0 .

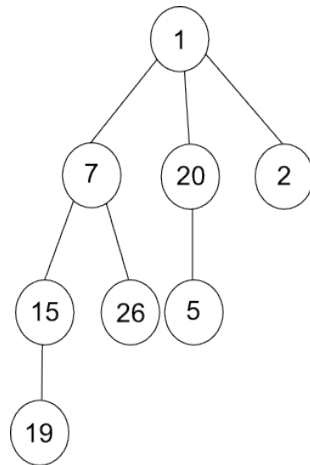


Figure 4.4: An ordered aggregation tree of a network

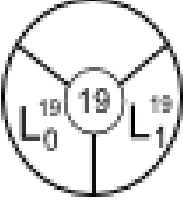
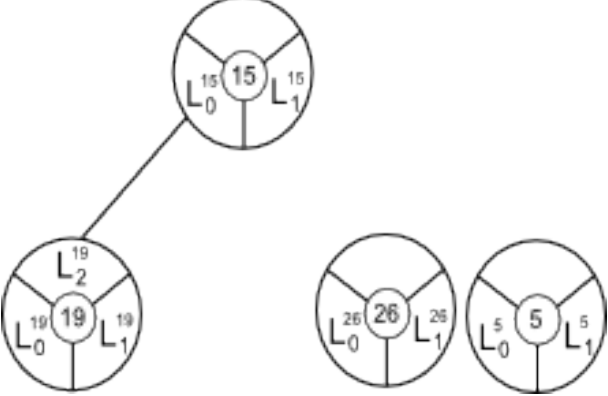
If node M has no child in the routing tree, its labels L_1 and L_0 are equal. M sends L_1 to its parent.

If M has one or more child, it waits until receiving all its children labels or until the expiration of a waiting timer. If timer expires before receiving all labels, identifiers of

nodes missing labels are included in the message routed to the BS; this parent and its children are marked by BS.

Based on the ordered aggregation tree defined in paragraph 4.2.2; the last M's child on the right has not right child in the binary tree, so its label L_2 is equal to its label L_1 . If M has more than one child, it calculates label L_2 of next children one by one from the left to the right until calculating L_2 of the first child on the left. After the calculation of label L_2 of the M's first child on left, or if M has only one child: M calculates its label L_1^M and sends it to its parent if it has a parent otherwise it is the root (BS).

An execution sample of this algorithm in the network presented in figure 4.4 is illustrated in Table 4.1.

	$19: L_1^{19} = L_0^{19}$
$19 \rightarrow 15 : L_1^{19}, MAC_{K_{19-15}}(L_1^{19})$	
	$15: L_2^{19} = L_1^{19}$ $15: L_1^{15} = F(L_2^{19}, L_0^{15})$ <hr/> $26: L_1^{26} = L_{26}^0$ <hr/> $5: L_1^5 = L_0^5$
$15 \rightarrow 7 : L_1^{15}, MAC_{K_{7-15}}(L_1^{15})$	
$26 \rightarrow 7 : L_1^{26}, MAC_{K_{7-26}}(L_1^{26})$	
$5 \rightarrow 20 : L_1^5, MAC_{K_{20-5}}(L_1^5)$	

	$7: L_2^{26} = L_1^{26}$ $7: L_2^{15} = F(L_2^{26}, L_1^{15})$ $7: L_1^7 = F(L_2^{15}, L_0^7)$ <hr/> $20: L_2^5 = L_1^5$ $20: L_1^{20} = F(L_2^5, L_0^{20})$ <hr/> $2: L_1^2 = L_0^2$
$7 \rightarrow 1: L_1^7, MAC_{K_{7-1}}(L_1^7)$ <hr/> $20 \rightarrow 1: L_1^{20}, MAC_{K_{20-1}}(L_1^{20})$ <hr/> $2 \rightarrow 1: L_1^2, MAC_{K_{2-1}}(L_1^2)$	
	$1: L_2^2 = L_1^2$ $1: L_2^{20} = F(L_2^2, L_1^{20})$ $1: L_2^7 = F(L_2^{20}, L_1^7)$ $1: L_1^1 = F(L_2^7, L_0^1)$ $1: L_2^1 = L_1^1$
$\Rightarrow L_2^1 \text{ is the final label of the aggregation}$	

Table 4.1: An example of the Aggregation-commitment phase execution in the network shown in figure 4 (“X: Y” signifies that the instruction Y is executed in node X)

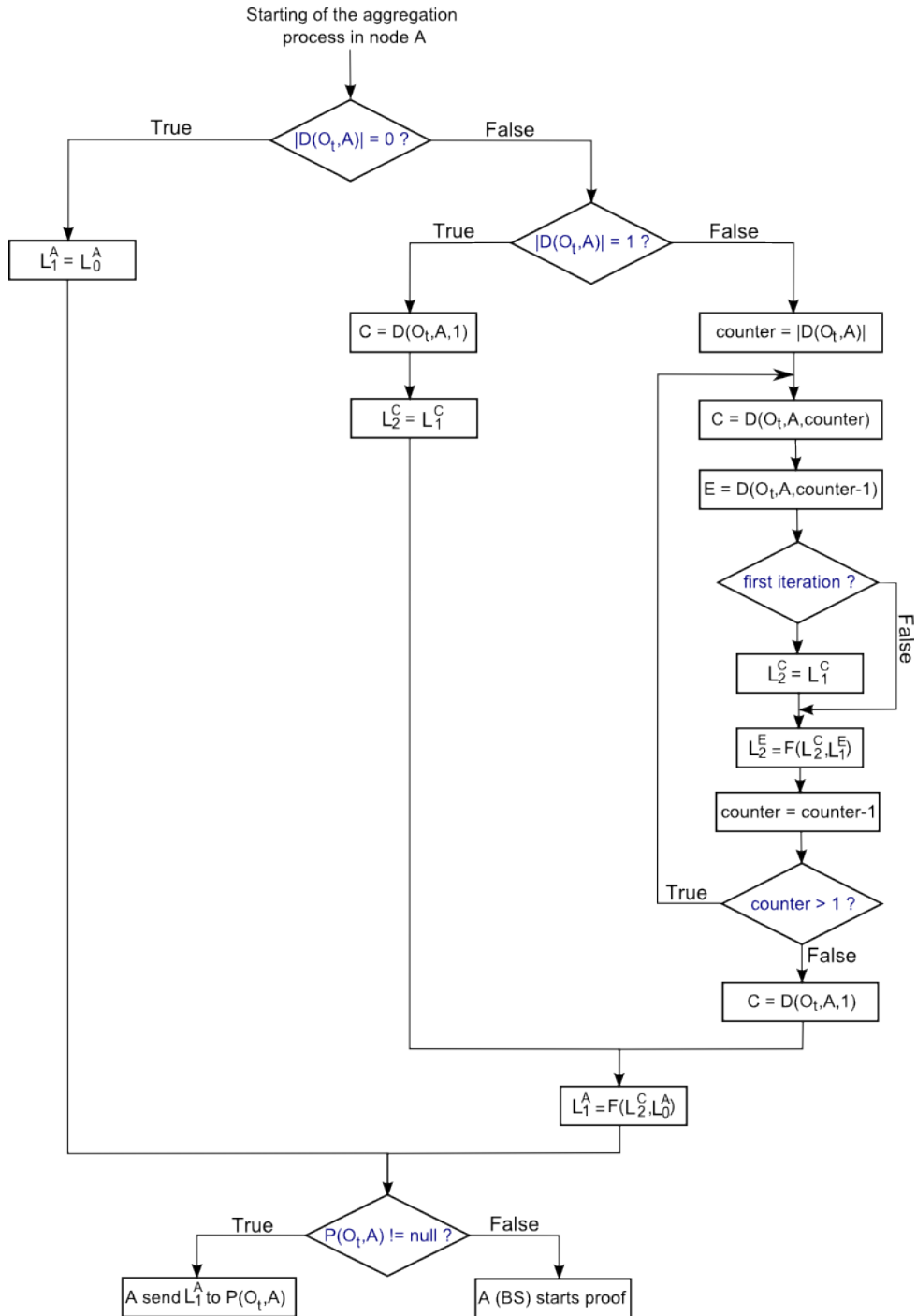


Figure 4.5: The aggregate-commit algorithm

4.3.4 Result-checking Phase

The BS broadcasts securely the final label in the network using a secure broadcast protocol.

To let each node verify that its label has been included in the aggregation process, it needs some labels to build the aggregation tree until obtaining the root label and verifying that the received and calculated labels are equal. For example in the aggregation tree, figure 4.6, the root label is L_2^1 . The node 15 needs some labels, which are highlighted on green, to build the commitment tree.

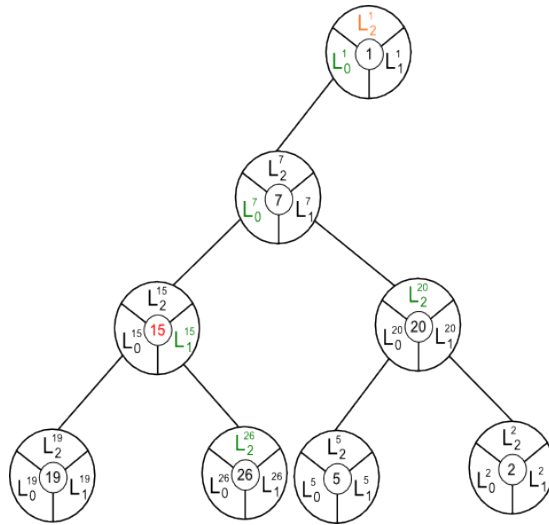


Figure 4.6: Needed labels by node 15 to build the commitment tree

So, each internal node A in the aggregation tree should send some information to its children:

- All labels that A receives from its parent
- its label 0 (L_0^A)
- If A has more than one child, it sends to its first child B on the left in O_t , the label 2 of B's next sibling in O_t
- To its child B, the Labels 1 of all B's left siblings and Label 2 of B's next sibling if B is not the last.

When the node receives all the necessary data to build the commitment tree, it verifies that its label has been included in the aggregation process by verifying that the received and the calculated final labels are equal.

Then each node sends an authentication code to the BS:

- if the verification is successful: $MAC_{K_s}(N||Ok)$; let us refer to as $OK - ACK$

- otherwise $MAC_{K_s}(N||No)$; let us refer to as $NO - ACK$

Ok and No are unique message identifiers; and K_s is a pairwise key shared between the BS and node s . To minimize the amount of data sent to the BS, each node XOR its authentication code with its children and sends the result to its parent.

The BS verifies the consistency of the aggregation result by calculating

$$MAC_{K_1}(N||Ok)XOR \cdots XOR MAC_{K_n}(N||Ok)$$

and then it verifies that the calculated and the received values are equal.

4.3.5 Elimination of malicious or faulty nodes

In this protocol we proceed differently from the method proposed in [37]. In the latter case, they use two adversary localizer schemes to mark and eliminate misbehaved nodes. In result-checking phase, they use an XORed MAC for each level in the commitment tree and in the second scheme they use big encrypted messages. The major weakness of this protocol is big messages which are very dependent on the network size and architecture.

In our protocol we proceed by branch in the aggregation tree so we explore branches where there is a NO-ACK. By exploring the branch, we can find the node N that has NO-ACK so we mark it and its parent M . If M is not already marked, it becomes a leaf. Its descendants (S : the set of M 's descendants) become roots of their subtree. An algorithm to rearrange the tree is triggered. The BS requests nodes in S to seek for a new parent; a HELLO message is sent by each node in S and add responding nodes to the list of its possible parents (nodes in S should not respond to the HELLO message). Then each node sends its list to the BS; the BS reorganizes the tree by choosing the new parent of the nodes in S . Different and not marked parents are the most preferred.

Nodes marked twice and those which do not send the potential parent list are eliminated from the network. The children of eliminated nodes are added to S .

For example in figure 4.7, A is the BS. A receives XORed MAC from all its children, it verifies it and it finds that in E 's branch there is a NO-ACK. So the BS demands from E its Ack and the XORed MACs of its children. E has an OK-ACK, so BS asks the E 's children to send ACKs and the XORed MACs of their children. A discover that K has a NO-ACK so it marks E and K . E becomes a leaf, K and L seeks for a new parent that is preferably not already marked.

4.4 Congestion complexity

In the aggregation commitment phase, each node sends one label to its parent for whatever the number of its children. In the result-checking phase, each node needs some off-path

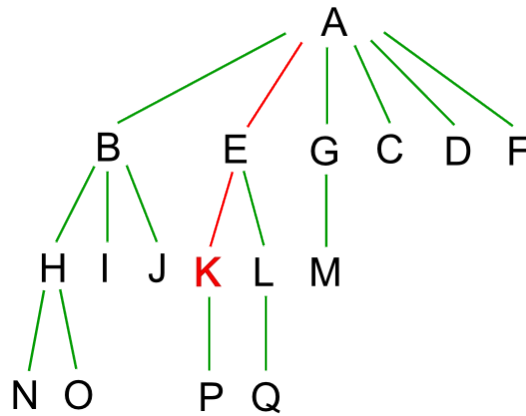


Figure 4.7: Exploration of tree branches

labels to generate the aggregation tree and obtain the final label. Let's define the depth of a node in the commitment tree by the minimum number of hops from it to the root plus one. So the root depth is one. Node 15 depth, in figure 4.6, is three. The number of labels needed by each node, except the root, is in the range $[n-1, 2(n-1)]$, where n is the node depth.

Proof: The extreme number of messages in the range $[n-1, 2(n-1)]$ are reached when the commitment tree is a full binary tree.

- **Depth $n=2$:** the tree is composed of one root and two child like figure 4.3.
 - node 38 needs L_0^{23} and L_2^{15} : $2 \times (2 - 1) = 2$
 - node 15 needs only L_{23}^1 : $2 - 1 = 1$
- Suppose that a node in extreme left needs $2(n-1)$ labels and a node in extreme right needs $(n-1)$ labels.
- Demonstrate it for nodes in depth $n+1$ (range $[n, 2n]$)
 - As described in 4.3.4, a node A in extreme left with depth $(n+1)$ has a parent B in extreme left too with depth n . B sends to A the labels that it has already received, L_0^B and label 2 of A sibling. So A receives $2 \times (n - 1) + 2 = 2 \times n$ labels
 - As described in 4.3.4, a node C in extreme right with depth $(n+1)$ has a parent D in extreme right too with depth n . D sends to C the labels that it has already received, L_1^D . So C receives $(n - 1) + 1 = n$ labels.

Analysis and comparison

As we see, each node sends a unique label in each aggregation round; but it needs between $(n - 1)$ and $2(n - 1)$ labels to verify the aggregation result authenticity, where n is the node depth in the binary tree.

The positive point of this protocol SeBTIA is: the aggregation result is received rapidly by the BS because the transmission of one message is pretty expensive with 12ms duration and the number of messages sent in the aggregation-commitment phase is optimized. The BS obtains the aggregation result and then it verifies its authenticity. If malicious nodes are detected, they are eliminated by executing the algorithm described in section 4.3.5.

Let N be the number of nodes in the network, and Δ be the maximum degree of any node in the aggregation tree [29]. In the aggregation commitment phase, each node using SHIA can send more than one label. Moreover, SHIA uses SUM and COMPLEMENT aggregation functions which make labels voluminous. In the result checking phase, SHIA induces $O(\Delta \log^2(N))$ maximum node congestion in the aggregation tree [29]. Suppose that the maximum node congestion in SeBTIA is $2(N - 1)$ (that can not happen). To compare the two congestion functions: $O(\Delta \log^2(N))$ and $2(N - 1)$, let's do that for different values of Δ according to the total number of nodes in the network. We remark in figure 4.8 that for a value of Δ , there is a limit N_1 where $\forall x < N_1, 2(x - 1) < \Delta_1 \log^2(x)$.

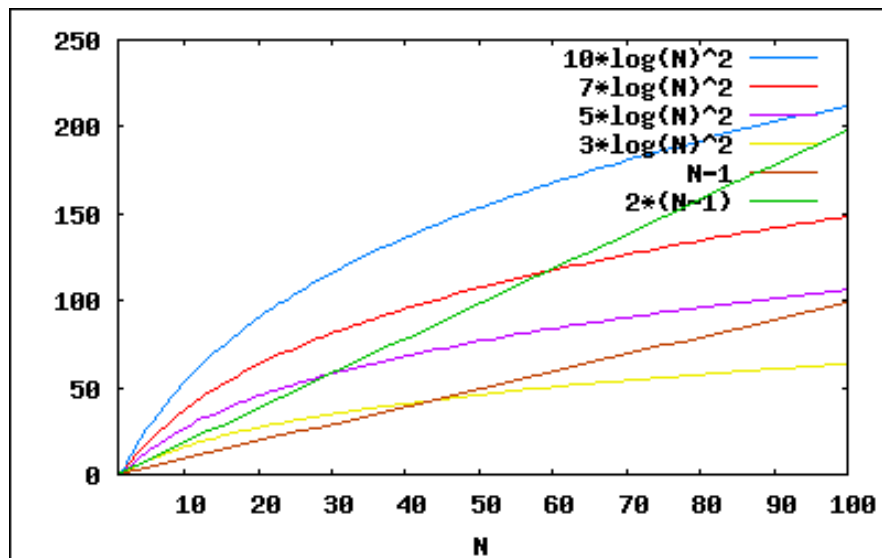


Figure 4.8: Comparison of SHIA and SeBTIA messages complexity

SeBTIA becomes more efficient when we limit the number of N . That can be done by associating SeBTIA to a local aggregation protocol like SAPC which aggregates the messages locally in the cluster. Thus the CH aggregates the data of the cluster nodes, then it aggregates the messages that it receives from other CHs. Finally, it sends a unique message to its parent, the BS or another CH, in the aggregation tree.

4.5 Implementation and tests

We implemented the aggregation protocol of SeBTIA in TinyOS 2.x to test and analyse it. The memory space occupied in a tmote sky sensor by the program is described in the table 4.2:

	Memory space
<i>ROM (bytes)</i>	31928
<i>RAM (bytes)</i>	2572

Table 4.2: Memory space occupation in SeBTIA

The authenticity verification of each received label takes 3,68ms. The generation of L_0 in each node takes also 3,68ms. The preparation of L_1^A depends on the number of A 's children. The figure 4.9 describes the execution time needed to prepare label L_1 by a node according to the number of its children.

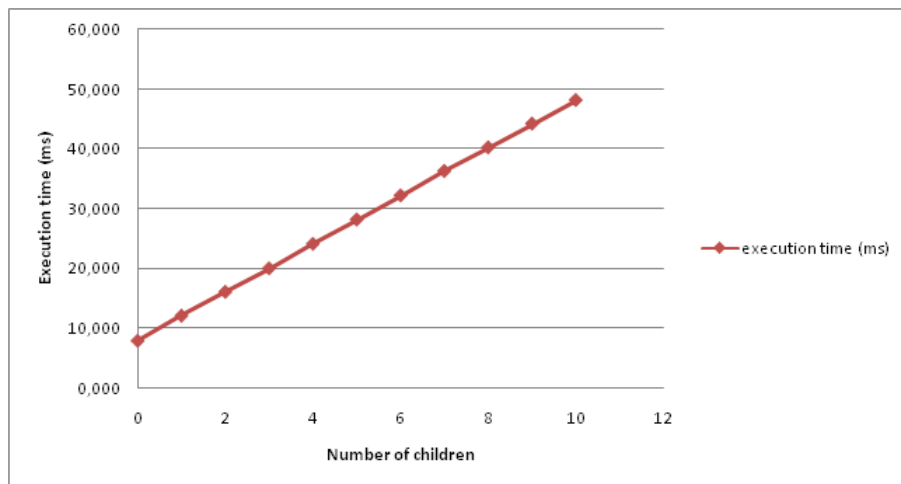


Figure 4.9: Execution time needed to prepare L_1

4.6 Conclusion

In this chapter, we discussed a new aggregation protocol that enhances the security along the aggregation phase while minimizing the number of labels needed to check the aggregation result. Compared with SHIA, our protocol is independent from the aggregation function, it generates smaller labels and each node sends a unique label in the aggregation phase.

Conclusions & perspectives

In this report, we focus on securing data aggregation in wireless sensor networks from outsider attacks. We compare two protocols, SAPC and Hu et al.; and we propose a new aggregation protocol SeBTIA. It enhances the performance of SAPC and it is more scalable and efficient than SHIA in several aspects. It is also applicable on random network topologies.

There are other solutions proposed in the literature to secure the data aggregation which are not based on cryptographic functions. As surveyed in chapter 2, in [31], the security is assured mainly by the probabilistic functions and trust. Thus it can reduce the calculation cost in the sensor nodes.

The tests of SAPC, in chapter 3, show that it is time consuming especially for the verification of reading messages. It causes also the exchange of many messages inside the cluster. We plan to further investigate the mathematical functions to enhance the performances in the cluster local aggregation. *Game Theory* is a possible way to accomplish security without cryptographic functions. In fact, the aggregation process in the cluster can be viewed as a game between the cluster members.

In chapter 4, we describe the SeBTIA protocol which is a secure and rapid solution for hop-by-hop data aggregation. It uses a delayed verification of the aggregation result and it gives a mechanism to detect and eliminate the malicious nodes. The association of this protocol to a more adapted local aggregation protocol will enhance its performances. We plan to investigate this track by associating it to a local aggregation protocol based on the Game Theory.

TinyOS 2.x does not support an advanced query language, secure query dissemination and user rights. Those are also possible subjects for future works.

Bibliography

- [1] Chakib Bekara, Maryline Laurent-Maknavicius, and Kheira Bekara. Sapc: A secure aggregation protocol for cluster-based wireless sensor networks. In Hongke Zhang, Stephan Olariu, Jiannong Cao, and David B. Johnson, editors, *MSN*, volume 4864 of *Lecture Notes in Computer Science*, pages 784–798. Springer, 2007.
- [2] Lingxuan Hu and David Evans. Secure aggregation for wireless networks. volume 0, page 384, Los Alamitos, CA, USA, 2003. IEEE Computer Society.
- [3] Saeyoung Jeong. Rfid + wsn application. <http://tooth2.blogspot.com/2008/01/rfid-wsn-application.html>, 01 2008.
- [4] Laurent-Maknavicius Maryline, Bekara Chakib, and Drira Wassim. Light and simple security solution for cold chain supervision. In *EUNICE 2008 copyright IFIP 2008*. A. Gravey, Y. Kermarrec, X. Lagrange (Eds.), September 2008.
- [5] Drira Wassim, Bekara Chakib, and Laurent-Maknavicius Maryline. Sécurité dans les réseaux de capteurs sans fil : conception et implémentation. Technical Report 08012-LOR, TMSp, 2008.
- [6] Stankovic John A. Dust to doctors: Wsn for assisted living. Microsoft Research - Faculty Summit 2007, 2007.
- [7] Polastre Joseph, Szewczyk Robert, Sharp Cory, and Culler David. The mote revolution: Low power wireless sensor network devices. *HOT CHIPS 16*, 2004.
- [8] TinyOS. <http://www.tinyos.net>.
- [9] David E. Culler. Tynyos: Operating system design for wireless sensor networks. <http://www.sensorsmag.com>, May 2006.
- [10] Handziski Vlado, Polastre Joseph, Hauer Jan-Hinrich, Sharp Cory, Wolisz Adam, Culler David, and Gay David. Hardware abstraction architecture. Technical Report TEP102, TinyOS Doc., June 2008.
- [11] Levis Philip and Sharp Cory. Schedulers and tasks. Technical report, TinyOS Doc., 2007.
- [12] Sharp Cory, Turon Martin, and Gay David. Timers. Documentary 102, TinyOS Doc., 2007.
- [13] Szewczyk Robert, Levis Philip, Turon Martin, Nachman Lama, Buonadonna Philip, and Handziski Vlado. Microcontroller power management. Technical Report TEP:112, TinyOS Doc., 2005.

-
- [14] Gay David, Levis Philip, Culler David, and Brewer Eric. *nesC 1.1 Language Reference Manual*, May 2003.
- [15] <http://www.sics.se/contiki>. Contiki website.
- [16] <http://en.wikipedia.org/wiki/Contiki>. Contiki operating system.
- [17] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. pages 455–462, 2004.
- [18] Fredrik Österlind. A sensor network simulator for the contiki os. Master’s thesis, UPPSALA UNIVERSITET, 2006.
- [19] Walters John Paul, Liang Zhengqiang, Shi Weisong, and Chaudhary Vipin. *Security in Distributed, Grid, and Pervasive Computing*. Auerbach Publications, CRC Press, 2006.
- [20] Moteiv Corporation. *Telos - Rev A datasheet*, 2004. Rev A (Low Power Wireless Sensor Module).
- [21] Guimarães Germano, Souto Eduardo, Sadok Djamel, and Kelner Judith. Evaluation of security mechanisms in wireless sensor networks. *IEEE Proceedings of the 2005 Systems Communications (ICW’05)*, 2005.
- [22] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. Spins: Security protocols for sensor networks. *Mobile Computing and networking*, 2001.
- [23] Khalifa Tarek. *Secure Data Aggregation Protocol with Byzantine Robustness for Wireless Sensor Networks*. PhD thesis, University of Waterloo, 2007.
- [24] Wei Zhang. *Secure data aggregation in Wireless Sensor Networks*. PhD thesis, THE UNIVERSITY OF TEXAS AT ARLINGTON, May 2008.
- [25] Tanveer Zia and Albert Zomaya. Security issues in wireless sensor networks. *IEEE Systems and Networks Communications, 2006. ICSNC ’06*, 2006.
- [26] Carl Hartung, James Balasalle, and Richard Han. Node compromise in sensor networks: The need for secure systems. Technical report, Department of Computer Science University of Colorado at Boulder, January 2005.
- [27] Yih-Chun Hu, A. Perrig, and D.B. Johnson. Wormhole attacks in wireless networks. *Selected Areas in Communications, IEEE Journal on*, 24(2):370–380, Feb. 2006.
- [28] Oya Şimşek. Denial of service attacks (dos) in wireless sensor networks (wsns). CS 432/532 - Computer and Network Security, 2008.
- [29] Haowen Chan, Adrian Perrig, and Dawn Song. Secure hierarchical in-network aggregation in sensor networks. In *CCS ’06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 278–287, New York, NY, USA, 2006. ACM.
- [30] Kun Sun, Pai Peng, Peng Ning, and Cliff Wang. Secure distributed cluster formation in wireless sensor networks. In *ACSAC ’06: Proceedings of the 22nd Annual Computer Security Applications Conference*, pages 131–140, Washington, DC, USA, 2006. IEEE Computer Society.

- [31] Wei Zhang, S.K. Das, and Yonghe Liu. A trust based framework for secure data aggregation in wireless sensor networks. In *Sensor and Ad Hoc Communications and Networks, 2006. SECON '06. 2006 3rd Annual IEEE Communications Society on*, volume 1, pages 60–69, Sept. 2006.
- [32] Sang-Chul Kim and KeeHyun Shin. A performace analysis of manet multicast routing algorithms with multiple sources. In *Fifth International Conference on Software Engineering Research, Management and Applications*, 2007.
- [33] Joe Polastre. Designing low power wireless systems telos / tmote sky. Presentation. Moteiv Corporation.
- [34] Kui Wu, Dennis Dreef, Bo Sun, and Yang Xiao. Secure data aggregation without persistent cryptographic operations in wireless sensor networks. *Ad Hoc Networks*, 5(1):100 – 111, 2007. Security Issues in Sensor and Ad Hoc Networks.
- [35] http://en.wikipedia.org/wiki/Binary_tree. Binary tree.
- [36] C. Bekara, M. Laurent-Maknavicius, and K. Bekara. H2bsap: A hop-by-hop broadcast source authentication protocol for wsn to mitigate dos attacks. pages 1197–1203, Nov. 2008.
- [37] Parisa Haghani, Panos Papadimitratos, Marcin Poturalski, Karl Aberer, and Jean-Pierre Hubaux. Efficient and Robust Secure Aggregation for Sensor Networks. In *The 3rd workshop on Secure Network Protocols (NPSec)*, 2008.

Glossary

BS	Base Station, 6, 15, 17, 35, 44, 46
CBC-MAC	Cipher Bloc Chaining Message Authentication Code, 30
CH	Cluster Head, 17
ID	IDentifier, 19
MAC	Message Authentication Code, 19
NO-ACK	Negative ACKnowledgment, 44
OK-ACK	Positive ACKnowledgment, 44
SAPC	Secure Aggregation Protocol for Cluster based wireless sensor network, 5, 23, 29, 30, 34, 46, 48
SeBTIA	Secure binary tree in-network aggregation, 46, 48
SHIA	Secure hierarchical in-network aggregation in sensor networks, 20, 46–48
TOSSIM	TinyOS SIMulator, 11
WSN	Wireless Sensor Network, 6, 7, 11, 14, 15, 17, 29

A Component diagrams

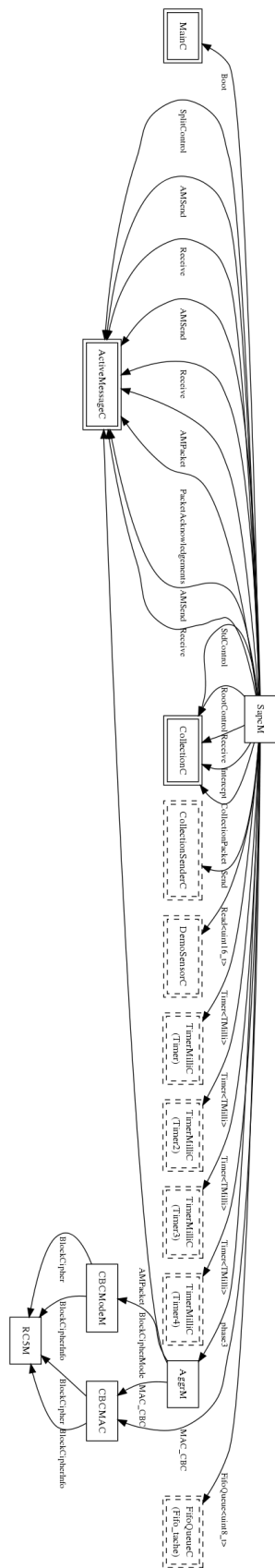


Figure A.1: Component diagram of the SAPC application

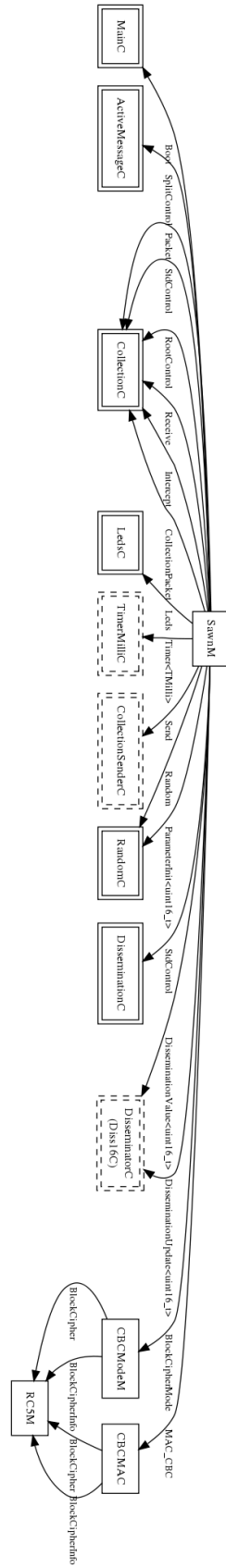


Figure A.2: Component diagram of the Hu et al. application

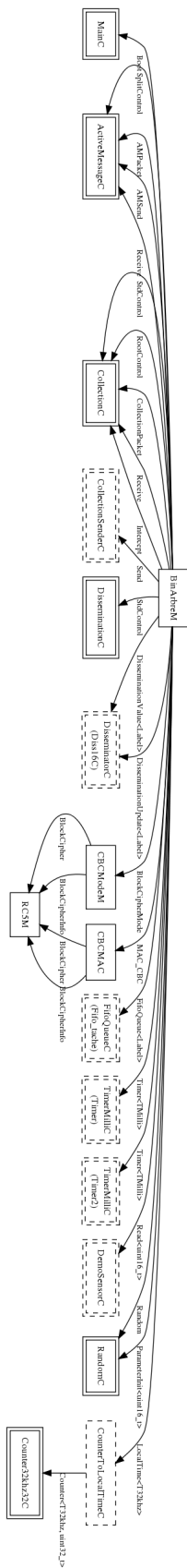


Figure A.3: Component diagram of the SeBTIA application