# The Security Properties of In-Network Aggregation

**Kristján Valur Jónsson**

Doctor of Philosophy

December 2012

School of Computer Science

Reykjavík University

**Ph.D. DISSERTATION**

ISSN 1670-8539

# The Security Properties of In-Network Aggregation

by

Kristján Valur Jónsson

Thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
**Doctor of Philosophy**

December 2012

Thesis Committee:

Ýmir Vigfússon, Supervisor
Assistant Professor, Reykjavik University

Mads F. Dam
Professor, KTH Royal Institute of Technology

Magnús Már Halldórsson
Professor, Reykjavik University

Philippe Bonnet, Examiner
Associate Professor, IT University of Copenhagen

Copyright
Kristján Valur Jónsson
December 2012

# The Security Properties of In-Network Aggregation

Kristján Valur Jónsson

December 2012

## Abstract

In-network aggregation is an important paradigm for current and future networked systems, enabling efficient cooperate processing of aggregate information, while providing sub-linear scalability properties. However, security of this important class of algorithms has to date not been sufficiently addressed.

In this dissertation, we focus on the integrity properties of in-network aggregation algorithms, with emphasis on the sub-goals of correctness and completeness. We propose an efficient solution that provides strong correctness guarantees by ensuring individual node integrity *a priori* by applying the principles of trusted systems. To this end, we propose dedicated trusted sensor and aggregator modules. Trusted modules, in conjunction with cryptographic authentication and transport protocols, are applied to construct trusted aggregation overlays, giving strong guarantees in terms of correctness. We support our findings by a proof-of-concept prototype in a single aggregator model, as well as a design for a hierarchical in-network aggregation system.

Completeness is a more elusive goal than correctness, if only for the fact that drops and message corruptions are a fact of life in distributed systems. Hence, it may not be possible to distinguish between benign and malicious losses. Building on the trusted systems solution for correctness, we propose a protocol that decreases the adversarial influence in a tree-based aggregation network. We exploit the fact that a secure protocol can be executed over a trusted overlay, enabling per-edge fault detection and dissemination of edge ratings. Simulation-based trials suggest that the presented protocol achieves significant reduction in the potential impact an adversary can have on the completeness of aggregate results.

# Öryggiseiginleikar netlægrar samsöfnunar gagna í dreifðum kerfum

Kristján Valur Jónsson

Desember 2012

## Útdráttur

Samsöfnunarreiknirit (e. aggregation algorithms) eru iðulega notuð til að ná yfirsýn yfir mikinn og flókinn flaum upplýsinga í dreifðum netkerfum, svo sem mælinetum. Eftir því sem netkerfin stækka að umfangi reynir æ meira á skölunareiginlega þessara reiknirita og á samvinnu milli tölvanna sem safna og vinna úr upplýsingunum. Þó skalanleg reiknirit af þessu tagi séu til er sá galli á gjöf Njarðar að í mörgum tilfellum geta óheiðarlegir þátttakendur ógnað öryggi þeirra og þannig minnkað traust á útreikningum. Nauðsynlegt er að tryggja gagnaöryggi í slíkum kerfum en það felur í sér ýmis óleyst vandamál.

Í þessari ritgerð er fjallað um öryggiseiginleika dreifðra kerfa sem nota netlæga samsöfnun upplýsinga (e. in-network aggregation), með áherslu á traust sem bera má til reiknaðrar lokaniðurstöðu. Unnið er út frá dreifðum mælinetum þar sem margir þátttakendur vinna saman að mælingum og vinnslu. Sem dæmi má nefna stór skynjaranet og rauntímamælingar á ástandi stórra tölvukerfa. Lagt er til að traust í veitingu og samsöfnun upplýsinga verði tryggt með einingum sem byggja á fræðum traustra kerfa. Sett er fram kerfi sem byggir á traustum skynjurum og vinnslueiningum þar sem hver eining er prófuð og varin á sjálfstæðan hátt. Með öruggum netsamskiptareglum (e. network protocols) mynda þessar einingar síðan heildstætt öruggt dreift mælikerfi. Hönnun kerfisins er sannreynd með frumgerð að traustum skynjara.

Með traustu mælikerfi má að ákveðnum skilyrðum uppfylltum tryggja að lokaniðurstaða sé rétt, þ.e. að öll gögn sem saman mynda niðurstöðuna séu sannanlega byggð á traustum frummælingum og vinnslu. Þó geta árásaraðilar enn skemmt niðurstöðuna með að fjarlægja réttar upplýsingar áður en þær berast til móttakandans. Til að taka á því vandamáli eru settar fram samskiptareglur sem draga úr þessum áhrifum árásaraðilans. Virkni þeirra er studd með hermunum og benda niðurstöður til að með þeim aðferðum sem lýst er megi draga verulega úr áhrifum slíkra árása.

*To my family, Fjóla, Vala Rún and Jón Valur.*

# Acknowledgements

The work presented in this dissertation could never have been completed but for the assistance of a number of people. First of all, I want to thank my family for giving me the luxury of going back to school and completing my Ph.D. I would never have completed this (sometimes difficult) journey without their patience and support.

I want to thank Reykjavik University and the faculty of the School of Computer science, in particular my advisor Dr. Ýmir Vigfússon, President Ari K. Jónsson, Dean Björn Þór Jónsson, Dr. Magnús Már Halldórsson and Dr. Luca Aceto for their support and assistance. I also want to thank Dr. Mads F. Dam and Dr. Rolf Stadler at KTH Royal Institute of Technology, and my fellow students at Reykjavik University and the TCS and LCN groups at KTH for their support, cooperation and enlightening discussions over the past years. In particular, I thank my co-authors Karl Palmskog, Gunnar Kreitz, Ólafur Ragnar Helgason and Misbah Uddin for their contributions to the work presented here, as well as other projects that we have collaborated on.

The work presented in this dissertation was supported in part by

(i) Rannís (Icelandic Centre for Research) Graduate Studies Grant #080520008.
(ii) The Icelandic Center of Excellence in Theoretical Computer Science (ICE-TCS) and Rannís grant #090032021.
(iii) Funding by Reykjavik University.
(iv) EU FP7 project 4WARD.

Work on the sensor prototype presented in Chapter 3 was supported in part by by a Student Innovations Fund project grant from Rannís (Icelandic Centre for Research) summer 2010. I most gratefully acknowledge the dedication and hard work by the students working on the project, Kristján Rúnarsson and Benedikt Kristinsson.

# Publications

*The following publications contributed to the work presented in this dissertation:*

Helgason, Ó.R. & Jónsson, K.V. *Opportunistic Networking in OMNeT++*. First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTOOLS), OMNeT++ Workshop, Marseille, France. 2008.

Jónsson, K.V. & Dam, M. F. *Towards Flexible and Secure Distributed Aggregation*. AIMS 2010 - PhD Workshop. Zürich, Switzerland. 2010.

Rúnarsson, K., Kristinsson, B. & Jónsson, K.V. *TSense: Trusted Sensors and Support Infrastructure*. Reykjavik University, Rannís NSN project report. September 2010.

Jónsson, K.V. & Vigfússon, Ý. *Securing Distributed Aggregation with Trusted Devices*. NordSec. Tallinn, Estonia. October 2011.

Jónsson, K.V. & Vigfússon, Ý. *Bootstrapping Trust in Networked Measurement Systems with Secure Sensors*. Sensor Applications Symposium (SAS). Brescia, Italy. February 2012. *Top 3 student paper award.*

Jónsson, K.V., Vigfússon, Ý. & Helgason, O.R. *Simulating Large-scale Dynamic Random Graphs in OMNeT++*. SIMUTools, OMNeT++ Workshop. Desenzano, Italy. March 2012.

Jónsson, K.V., Palmskog, K. & Vigfússon, Ý. *Secure Distributed top-$k$ Aggregation*. IEEE International Conference on Communications (ICC). Ottawa, Canada. June 2012.

Jónsson, K.V. & Vigfússon, Ý. *Robust Authentication in Trusted Sensing Networks with Physically Unclonable Functions*. NordSec. Karlskrona, Sweden. October 2012.

# Contents

# List of Figures

# Chapter 1

# Introduction

New networking paradigms, such as wireless sensor networks (Akyildiz et al., 2002) and the *Internet of Things* (Uckelmann et al., 2011), will without a doubt enable new and exciting applications. At the same time, the unprecedented scale of future networked systems and the sheer volume of information produced can be expected to pose new challenges in terms of scalability and security. In this dissertation, we consider a particular class of applications in networked systems, those that employ *aggregation* algorithms to address the scalability challenges involved in gathering information from a large set of contributors and co-operatively produce a series of concise digests representing an approximate global view. In particular, we focus on the security challenges posed by the application of distributed aggregation algorithms in large networked systems.

## Aggregation in Networked Systems

Modern networked systems are capable of generating vast amounts of information about their inner workings and surroundings. This information has the potential to help us visualize systems, aiding us in making intelligent management decisions and learning about an observed environment. However, great volumes of detailed data are not necessarily helpful for this purpose. On the contrary, we may argue that very large data sets or fine-grained streams can occlude system visualization, as one may not see the forest for the trees. Consider the problem of *big data* (Nature, 2008; Horowitz, 2008). Recent advances in computing and research methodologies can generate enormous data sets that challenge traditional information processing techniques. Examples of big data generating sources are diverse and bound to proliferate in the near future, but with respects to the topic of this dissertation, we can consider applications such as sensor networking (Akyildiz et al., 2002; Roush, 2003; Yick et al., 2008), aggregation of incident logs (Slagell & Yurcik, 2005) and cooperative sensing (Kansal et al., 2007), all of which are classes of applications that may be expected to flourish both in number and size in the near future.

*Aggregation* (van Renesse, 2003; Keshav, 2006) is an umbrella term for techniques used to summarize volumes of raw information into a more manageable form. Aggregation can be used to summarize and condense a number of contributions into a *digest* that concisely describes an aspect of the state of a system. For instance, the average ambient temperature over an area can be described as the average of samples taken simultaneously by a number of *motes* in a wireless sensor network. In network management applications, we may want to monitor a system to maintain a running average of packet drops in router queues or detect an increase in the number of abnormal TCP connections. Generalizing, the objective of aggregation in a networked system is to provide a small subset of nodes, which we will call *queriers*, with a bird's eye view of the system. This view is produced by processing and condensing *local inputs* contributed by the node population into a series of manageable digests that suffice to characterize some aspect of a monitored system.

We define networked systems, local inputs and aggregation functions abstractly in this dissertation. Networked systems are a general term applying to a diverse range of systems, from static wireless sensor networks (Madden et al., 2002b) to network management applications in dynamic computer networks and clusters (Stadler et al., 2008). The contributions of the local state of participating nodes are called the local inputs to the protocol. Local inputs and aggregation functions are also broadly defined as the representation of some sort of locally observable and quantifiable phenomenon. Examples include measurement of environmental variables (Yao & Gehrke, 2002), pollution particle count (Paulos et al., 2007) and structural stress (Gomez et al., 2009) in a wireless sensor network, monitoring of performance parameters (Stadler et al., 2008) and efficient meta-data directed searches in information-centric networking (Palmskog et al., 2010).

Our goal in this dissertation is to address a range of problems in the case of aggregation in networked systems in the *general case*, that is, *for arbitrary dynamic network types as well as arbitrary local inputs and aggregation functions.*

## Scalability

Networked aggregation systems have to date been relatively constrained in terms of size, even considering seemingly large-scale examples, such as sensor networks and enterprise-scale computer networks. However, new paradigms, such as shared and cooperative sensing (Deshpande et al., 2003; Kansal et al., 2007) and the Internet of Things (Gershenfeld et al., 2004; Uckelmann et al., 2011), are poised to shatter such bounds, creating large and complex networked sensing, management and control systems. Hence, scalability must be a primary consideration in the design of future systems.

In distributed systems theory, we generally define scalability with respect to the asymptotic relationship of several performance metrics to system size. For instance, one must consider the impact of system size on the maximum memory consumption and CPU load on any single node in a distributed system. The maximum time required to execute a distributed algorithm is also of concern.

In this dissertation, we are primarily interested in messaging complexity, the number of messages generated in course of the execution of a distributed algorithm. Consider a centralized aggregation system in which a querier sends individual requests in a round-robin fashion to managed hosts that in turn report the local state of the requested variables to a central aggregation node (management station). The aggregate is computed in a straight-forward manner by the aggregation node, once all answers have been received. Several problems are immediately apparent with regards to scalability, one being the processing power and memory requirements of the aggregation node. Time complexity, the time required to obtain a snapshot of the entire system, is linear with regards to the system size. Message complexity is also a critical factor. Centralized systems of this sort typically have highly asymmetrical link utilization: a separate end-to-end request/response phase is triggered for each of the hosts queried, leading to exponentially increasing network link utilization in proportion to decreasing distance to the querier. This increase implies potential congestion and resource allocation problems.

## In-Network Aggregation

To address the scalability issues associated with large-scale future systems, we may expect a paradigm shift from the predominance of relatively straight-forward client/server-based aggregation systems, e.g. SNMP (Case et al., 1990), to distributed architectures. This shift has already begun with distributed aggregation algorithms being common in wireless sensor networks (Madden et al., 2002b), as well as being introduced in network monitoring applications (Birman & van Renesse, 2002). The class of *in-network aggregation protocols*, in which participants co-operatively compute the aggregate, is of particular interest. These protocols are efficient, with regards to both time and message complexity, typically sub-linear with regards to the system size. Further, the per-edge messaging complexity is generally of a constant order, irrespective of the system size.

Considerable work has already been carried out in the field of in-network aggregation, most prominently for sensor networks (Intanagonwiwat et al., 2000; Krishnamachari et al., 2002; Madden et al., 2002b; Shrivastava et al., 2004; Rajagopalan & Varshney, 2006; Upadhyayula & Gupta, 2007), but also in network monitoring and management systems (van Renesse, 2003; Keshav, 2006; Stadler et al., 2008). In-network processing has also been proposed as an approach to process big data (Costa et al., 2012).

In-network aggregation protocols can be roughly categorized into families. Protocols which aggregate over a spanning-tree overlay are amongst the most efficient ones, in terms of message and time complexity (Madden et al., 2002b; Dam & Stadler, 2005; Lim & Stadler, 2005). Gossip-based protocols (Kempe et al., 2003; Jelasity et al., 2005; Boyd et al., 2006) are based on randomized "rumor spreading" amongst neighbors on a network overlay and can converge quite quickly to a close approximation of the true system state. Consensus propagation (Moallemi & Van Roy, 2006; Aurell & Pfitzner, 2009) is yet another technique, particularly applicable to aggregation of Gaussian data.

## Security

The inherent scalability attributes of in-network aggregation algorithms makes them an attractive choice for data processing applications in large-scale networked systems. In this dissertation, we focus on the integrity properties of in-network aggregation, but open issues abound, for instance regarding data confidentiality and integrity, entity authentication, key distribution and management, availability and privacy (Wood & Stankovic, 2002; Karlof & Wagner, 2003; Perrig et al., 2004; Shi & Perrig, 2004; Roosta et al., 2006).

Research on distributed aggregation algorithms has historically regarded the population of nodes as uniformly benign. Security considerations have mostly been confined to an adversarial model in which nodes are honest but edges may be corrupt, giving the adversary the ability to view, drop and inject messages, akin to the classical Dolev-Yao model (D. Dolev & Yao, 1983). Under this model, standard cryptographic primitives and protocols, such as SSL/TLS (Dierks & Rescorla, 2008), suffice to neutralize the adversary by confining communications to trusted channels. However, allowing for the eventuality of node compromise paints a different picture. Let us now assume the existence of an *insider adversary* able to corrupt one or more nodes participating in the system. Corruption of nodes gives the adversary access to a subset of the system secrets, such as cryptographic keys. For instance, the SSL security model provides strong security guarantees against outsiders, as long as the integrity of participating nodes remains intact. However, all bets are off with regards to the security of future communications if an attacker is able to compromise one of the node participating in such an exchange (Moehrke, 2011).

The insider adversary poses an acute threat to aggregation algorithms based on the principle of in-network aggregation. Under this model, we are forced to place considerable trust in the individual contributing nodes, not only in terms of their data production but also their data processing. In fact, even a single corrupt insider is capable of introducing arbitrary bias into the computation (Wagner, 2004), presenting an acute threat to the integrity of the aggregate data. Furthermore, this malfeasance is in general hard to detect.

Information is the primary product of a networked aggregation system, whose trustworthiness is, hence, of prime concern. Yet, security has too often been neglected when designing in-network aggregation protocols and systems. In our opinion, applications which can tolerate arbitrary data can only be used for the most trivial of applications. Examples of applications in which aggregate accuracy and authenticity is of concern include military systems (Arora et al., 2004; He et al., 2006), public safety command and control (Gomez et al., 2009) and nuclear plant monitoring (Barbarán et al., 2007). We can also consider applications where money is at stake, such as distributed monitoring of electricity usage in a smart grid system (Amin & Wollenberg, 2005). The expected importance of distributed aggregation system and the importance of reliable data motivates our focus on their integrity properties.

## Contributions

Our research question can be stated as follows: *Can we guarantee the integrity of dynamic aggregation systems if some data contributors or aggregators are in the hands of untrusted entities?* We will explore this subject in the remainder of this dissertation, seeking solutions which give strong integrity guarantees. In light of efficiency being one of the primary motivations for adoption of in-network aggregation, we seek solutions which impose minimal overhead in terms of messaging and processing.

Our proposed set of solutions is based on the principles of trusted computing (TCSEC, 1985). In essence, we propose to extend the trusted computing base (TCB) (Lampson, 1974) of the otherwise corruptible nodes in an aggregation network by hosting trusted modules. Trusted data sources, e.g. trusted sensors (Jónsson & Vigfússon, 2012a), provide unmodifiable outputs, while trusted aggregators (Jónsson & Vigfússon, 2011) ensure correct function evaluation. Finally, robust authentication protocols (Jónsson & Vigfússon, 2012b) and cryptographic primitives, enable us to construct a trusted aggregation overlay, providing transitive aggregate trust relations from sources to querier over an arbitrary dynamic system. We support this work by a proof-of-concept prototype (Rúnarsson, Kristinsson, & Jónsson, 2010) of a secure centralized monitoring system of trusted sensors, as well as a design for a trusted hierarchical aggregation system (Jónsson & Vigfússon, 2011).

The trusted overlay provides strong guarantees in terms of overall aggregate *correctness*, that is, the property that the set of partial aggregates delivered to a trusted querier are consistent with truthful local inputs of contributing nodes. The complementary objective of *completeness*, that is, that the entire set of contributed local inputs are delivered to the querier, is a more elusive goal and one which is unlikely to be achieved, even assuming strictly benign faults. To address completeness, we consider light-weight means of decreasing the potential adversarial influence. To this end, we present a protocol which exploits the trusted overlay and local per-edge misbehavior, enabling trusted modules to choose the forwarding paths least likely to be faulty. As a case-study, the protocol is applied to construct a secure version on the Generic Aggregation Protocol (GAP) (Dam & Stadler, 2005). We validate the protocol by means of simulation. Our results indicate that the protocol significantly limits the influence an adversary can have over the aggregation process in terms of completeness.

This dissertation makes the following contributions:

(i) To tackle the problem of integrity preserving sensing, we propose, design and implement a *trusted sensor* (Jónsson & Vigfússon, 2012a), that provides data source integrity guarantees for the sub-goal of correctness. The concept is supported by a full system design and a proof-of-concept prototype (Rúnarsson, Kristinsson, & Jónsson, 2010).

(ii) We propose a fully distributed aggregation system that provides integrity guarantees. A trusted aggregation overlay is proposed, composed of trusted modules and secure communications protocols that provide correctness guarantees (Jónsson & Vigfússon, 2011). We consider the problem of simulation attacks, which can be mounted against a trusted aggregation system in the event of cryptographic keys being discovered. As a countermeasure, we propose an enhanced authentication protocol, incorporating physically unclonable functions (Jónsson & Vigfússon, 2012b).

(iii) Having addressed the integrity sub-goal of correctness, we turn our attention to the more challenging goal of completeness. We show (informally) that under our adversarial model, no solution is likely to exist that provides completeness on all problem instances. We instead design and evaluate a secure aggregation protocol based on the Generic Aggregation Protocol (GAP) (Dam & Stadler, 2005), integrating the trusted devices approach to correctness with a protocol which enhances completeness by monitoring faults and disseminating fault information.

## Overview

The dissertation is structured as follows:

- We provide an overview of issues pertaining to in-network aggregation and distributed systems security in Chapter 2.
- The problem of data source integrity in distributed sensing systems and the trusted sensors concept and prototype is discussed in Chapter 3.
- The trusted sensors concept is extended to a general solution for integrity-preserving in-network aggregation in Chapter 4.
- In Chapter 5, we consider the issue of completeness in in-network aggregation and propose a practical protocol that achieves reduction of the potential influence an adversary can wield.
- We conclude in Chapter 6 and suggest directions for future work.

# Chapter 2

# Background

## 2.1 Aggregation in Networked Systems

*Aggregation* is the act of composing an aggregate – to assemble a whole from a collection of smaller pieces. In terms of networked systems, which are the focus of this dissertation, we refer to aggregation as the act of collecting and processing information, often from a large set of participants. The processed data set provides us with a concise and manageable digest, representing a set of locally observable phenomena in the system. Aggregation in distributed systems is a broad field, encompassing many different paradigms. We address the subject abstractly in the following chapters, with the goal being the construction of a general, broadly applicable and efficient solution to the problem of secure aggregation in large networked systems. Let us now consider some examples highlighting the applications of aggregation as motivation for the material in the remainder of this dissertation.

#### Smart Grids

The smart grid (Amin & Wollenberg, 2005) is a vision for the next-generation power delivery systems, integrating the traditional power grid and a communications network. Smart grid systems enable new and more efficient power delivery and billing models by means of integrated data processing and decision-making capabilities. One example of innovative power delivery models is the concept known as vehicle-to-grid (V2G) (Kempton & Tomić, 2005), in which electric plug-in vehicles can be used as a resource for the grid at large, using their batteries as a power source during peak load periods. In our opinion, aggregation will prove to be a key component of such systems, enabling a fine-grained but dynamic overview of large power delivery systems.

### Network Monitoring and Management

Network monitoring is essential for effective network management, enabling informed decision making based on a complex set of inputs in enterprise-scale networks and clusters. An aggregate view of networked systems is commonly used to provide a dynamic view of the overall system state. A wide range of variables are commonly monitored in networked systems. For instance, operators may be interested in the average and maximum VoIP traffic on various links as an aid to more efficient resource usage and allocation, the percentage of peer-to-peer traffic and the number of SYN packets passing through a domain gateway (Dam & Stadler, 2005; Wuhib et al., 2008; Gonzalez Prieto & Stadler, 2009). Information of this sort can be used to identify network provisioning problems or potentially malicious events, such as distributed denial-of-service (DDoS) attacks.

*Flow monitors*, such as *NetFlow* (Claise, 2004) and *SNORT* (Sourcefire, 2012), are commonly used in network monitoring. In flow monitoring, one or more *sensors* are strategically placed on major backbones, intercepting the traffic and directing filtered aggregates to a set of management stations. Flow monitoring by definition provides an aggregate view by intercepting all messages in the *aggregate flow*. However, the on-line processing necessary to provide usable digests from such flows is non-trivial, demanding considerable processing and storage capabilities from the sensors and management stations. Further, the changing nature of network traffic decreases the utility of in-network monitoring as a significant fraction of typical network traffic is tunneled and encrypted for security reasons, leaving flow monitors blind to much of the information passing through (Cooke et al., 2006).

In contrast to the aggregate flow, one may be interested in monitoring a set of properties that involve querying and processing information from individual nodes, for instance workstations, servers and routers in an enterprise-scale computer network. The *Simple Network Management Protocol* (SNMP) (Case et al., 1990) is one example, in which a *management station* queries the *Management Information Base* (MIB) of managed devices in a round-robin based fashion. SNMP is primarily a polling (pull) mechanism, while push-type monitoring of local states has been proposed, for instance by Mortier et al. (2005) and Cooke et al. (2006). In the push paradigm, individual monitored systems publish their local states to the management station, following some pre-defined schedule or in response to local events. Monitoring protocols, such as SNMP, allow flexible queries to be executed over the system, providing a relatively efficient steady-state monitoring, while more extensive "drill-down" queries can be executed on demand, for instance to analyze potential problems.

## Environmental Monitoring via Sensor Networks

Wireless sensor networks and their potential applications (Akyildiz et al., 2002; Roush, 2003; Yick et al., 2008) have been extensively researched for over a decade. The classical vision for such networks is a self-organizing (ad-hoc) network of simple and cheap battery powered wireless nodes, commonly called *sensor motes*. The archetypal application example of sensor networks is the smart dust concept (Dickinson, 2010) in which cheap disposable motes are randomly placed in some area to be monitored, for instance watching for forest fires (Yu et al., 2005; Doolin & Sitar, 2005) or enemy troop movement (Arora et al., 2004; He et al., 2006). Random scattering of motes from airplanes has been suggested (Arora et al., 2004). Motes deployed in this fashion must be autonomous, forming self-organized communications meshes. The vision of large-scale deployment of sensors, perhaps over large geographic areas, implies large volumes of data, thereby highlighting the importance of aggregation for this class of systems.

The vision of small and cheap motes implies limited battery power. For this reason, mote radios are generally low power and low rate, limiting their communications capabilities. Hence, wireless sensor motes typically form high diameter networks in which multiple hops may be required in order to forward a report to the *sink* (recipient). For this reason, care must be taken when designing routing and aggregation mechanisms in sensor networks to ensure even utilization of communications and processing resources.

While the vision of smart dust has yet to be fulfilled, the fundamental concepts behind sensor networking have been applied in current systems. For instance, systems of battery powered sensor nodes are currently proposed for applications such as radiation monitoring in nuclear power plants (Barbarán et al., 2007) and building integrity monitoring (Gomez et al., 2009).

## Cooperative Sharing and Processing of Measurements

Sophisticated distributed aggregation networks can be composed of devices owned and operated by volunteers. One example is the Weather Underground[1], a collaborative system that collects and aggregates information provided by networked weather stations operated by volunteers around the world. Systems of this sort enable fine-grained weather reporting and analysis (Geller, 2007). Pachube[2] is a generalization of this concept, implementing a virtual patch-bay into which volunteers can connect sensors and share data. IrisNet (Gibbons et al., 2003) is another example of a proposed globally integrated sensing framework. The NETI@home project (Simpson, Jr. & Riley, 2004) collects measurements contributed by end-users to be aggregated by a central system, providing an unique and detailed aggregate view on data flows in networked systems. Collaborative aggregation systems can in some respect be considered an extension of the collaborative processing paradigm, in which a distributed system of volunteers collaborates on solving a computationally intensive task. Examples include the SETI@home[3] and Folding@home[4] projects, as well as distributed password cracking (Crumpacker, 2009).

Current smart phones are sophisticated computing platforms, capable of hosting and communicating with a multitude of sensors. Shared, or participatory, sensing (Trossen & Pavel, 2005; Burke et al., 2006; Kansal et al., 2007; Kansal & Zhao, 2007) is a paradigm that exploits the proliferation of such devices to construct large-scale cooperative sensor systems – a virtual version of participatory urbanism (Paulos et al., 2007). SenseWeb[5] (Kansal et al., 2007) is an example of such a system, whose goal is to enable applications based on participatory sensing via mobile devices. Usage examples include pollution particle count (Burke et al., 2006), air quality monitoring (Paulos et al., 2007), environmental impact (Mun et al., 2009), environmental temperature contour mapping (Kansal et al., 2007), detection of radioactive material (Venere & Gardner, 2008) and even grocery bargain hunting (Deng & Cox, 2009).

[1] `www.wunderground.com`
[2] `https://pachube.com`
[3] `http://setiathome.ssl.berkeley.edu`
[4] `http://folding.stanford.edu`
[5] `http://research.microsoft.com/en-us/projects/senseweb`

### The Internet of Things

The Internet of Things (IoT) (Gershenfeld et al., 2004; Ashton, 2009) is a concept, rather than a paradigm at this point in time, referring to the expected metamorphosis of collections of diverse artifacts into a networked whole – a pervasive "ambient" network of devices. In the vision of IoT, commonplace artifacts and devices, and even persons, will have a networked presence – a virtual representation (Stajano & Anderson, 1999).

This move towards a pervasively connected world is currently in its infancy. However, we may surely expect numerous new applications to be enabled by such technology, many of which are bound to be based on the production, sharing and processing of data, and hence, enabled to a large extent by scalable aggregation. For instance, we can consider a the concept of a smart home in which networked utilities metering devices which interact with environmental sensors and even the inhabitants (Gershenfeld et al., 2004; Sundmaeker et al., 2010; Fleisch, 2010).

## 2.2 Scalability

Current networked systems are growing in size and scope. In case of the Internet, the growth is such that IPv4 address space is in practice depleted, even with the stop-gap measure of Network Address Translation (NAT) (Lagerholm, 2012; Huston, 2012). New trends and paradigms, such as sensor networking, shared and cooperative sensing, industrial automation and control systems and the concept of the Internet of Things serve as a preview into a new world in which large and heterogeneous networked systems combine to provide us with unprecedented riches in terms of information, as well as enabling new and unforeseen applications. However, the sheer scale of current and future networked systems raises the issue of scalability – how a system is able to cope with size expansion in terms of resource usage bottlenecks.

In distributed systems theory, we generally define scalability with respect to the asymptotic relationship of several performance metrics and system size (Peleg, 2000, Ch. 2.2):

***Time complexity*** is the time required to execute a distributed algorithm to completion. Processing and communications delays contribute to the time complexity.

***Space complexity*** is the maximum number of bits required by the execution of a distributed algorithm by any node.

***Message complexity*** is the overall number of messages generated for one execution of a distributed algorithm.

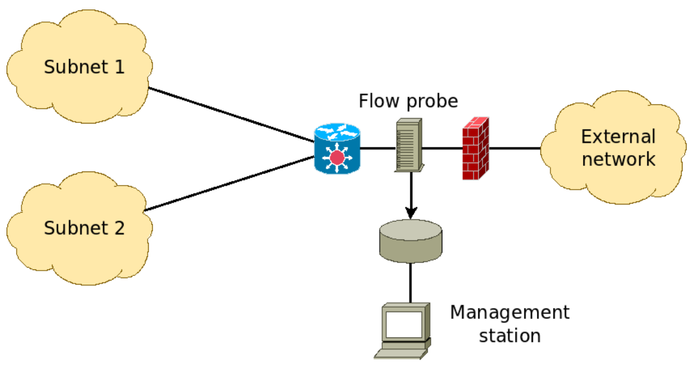


**Figure 2.1:** *A flow monitoring system. The aggregate data stream is monitored by a probe that in turn stores processed information in a database. Network management stations can query the database to obtain an approximate view of the network state.*

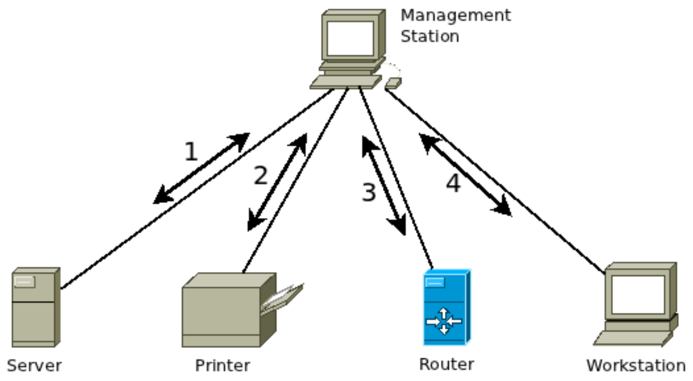


**Figure 2.2:** *SNMP (Case et al., 1990) polling operation. A central management station queries a number of managed devices in a round-robin fashion. Intermediary nodes, for instance routers, are not shown.*

In classical distributed systems theory, researchers commonly rate algorithms in terms of their overall complexity. A more practical systems-oriented view is to consider worst and average case complexity per-node and per-link. For instance, the node and link congestion metric (H. Chan et al., 2006) is directly derived from the definition of message complexity but relates to the maximum load which can be experienced by any node in a system. The complexity of two types of commonly used aggregation systems is discussed in Examples 2.1 and 2.2.

**Example 2.1 (Scalability of flow monitoring).**
Flow monitoring systems view the aggregate data flow in a subnet, as shown in Figure 2.1. Hence, data collection is straight-forward and time complexity is low. However, the processing demands on the collected data are considerable. Flow monitoring depends on relatively few heavily loaded monitors. Hence, the memory (space complexity) and processing power required to monitor a flow directly impacts the system scalability (Cooke et al., 2006). Message complexity is proportional to the granularity and frequency of updates from flow monitors to the management infrastructure.

**Example 2.2 (Scalability of centralized monitoring systems).**
Consider a monitoring system in which one querier polls a population of managed nodes in a round-robin fashion, as shown in a simplified view in Figure 2.2. The polling operation implies end-to-end communications between the management station and each managed node. Hence, the time required to collect one complete aggregate result scales linearly with the system size. The end-to-end polling operation causes an imbalance in link utilization, as the expected number of messages related to the management protocol increases exponentially with decreased distance to the querier. This imbalance in link utilization translates directly to congestion potential in the vicinity of the querier.

## 2.3 In-Network Aggregation

Current aggregation practices, for instance those outlined in Examples 2.1 and 2.2, have potentially serious scalability implications, as noted for instance by Keshav (2006). Flow probes are a bottleneck in terms of processing power and memory usage (time and space complexity), while centralized polling introduces an imbalance in network link loading (congestion potential) as well as linear scaling of system polling time (time complexity). Alternative solutions must be considered for efficient aggregation in very large networked systems.

In resource constrained systems, such as wireless sensor networks, the imbalance of link utilization has potentially serious consequences for the system lifetime and usability. Consider a system of wireless battery powered nodes that implement a scheme similar to SNMP polling. The implications are that the more heavily loaded nodes closest to the sink (querier) can be expected to run out of power the quickest. Hence, the most critical nodes may be expected to fail long before the usable battery power of the majority of the nodes is exhausted. The problem is exacerbated by the expected sparsity of alternative communications path due to the limited radio range of such devices. To counter the problems of uneven resource expenditure, sensor network research has from the very beginning incorporated in-network computing – a paradigm in which the computational load involved in performing the aggregation is spread across the monitored system, leading to more even resource expenditure and messaging loads (Intanagonwiwat et al., 2000; Madden et al., 2002a; Rajagopalan & Varshney, 2006).

While most prominent in the field of sensor networks, distributed approaches have also been considered for monitoring in more traditional systems. An extensive body of work focuses on the utilization of in-network aggregation for monitoring of computer networks,

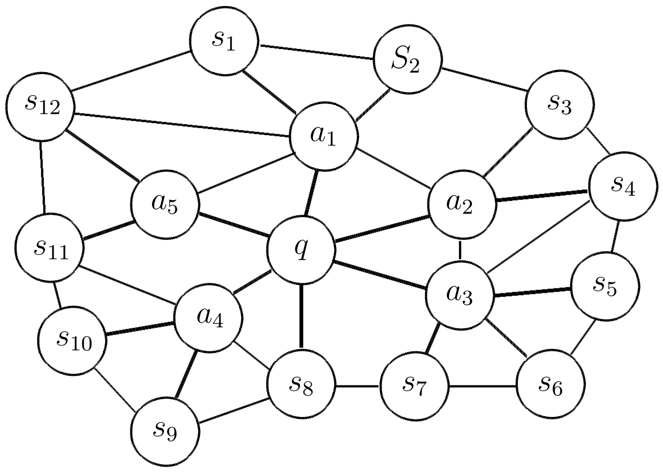


**Figure 2.3:** *Sample system diagram. Nodes $s_i$ provide updates based on local inputs and processed by intermediary aggregators $a_j$. A spanning-tree overlay is shown (bold lines) on top of the underlying communications graph G. A heterogeneous network is shown in which data providers, e.g. sensors, and aggregators are distinct node types.*

that is, as an alternative to SNMP and flow monitoring (Keshav, 2006; Stadler et al., 2008).

### 2.3.1 An Example System Model

Let us begin by outlining a system model for a distributed aggregation system, similar to the ones used for the remainder of this dissertation. We consider a dynamic networked system, which can be visualized as a dynamically evolving communications graph $G(t) = (\boldsymbol{V}(t), \boldsymbol{E}(t))$. $\boldsymbol{V}(t)$ is the set of nodes (vertices) at time $t$, while $\boldsymbol{E}(t)$ represents links (edges) in the graph $G(t)$. We often leave out the $t$ parameter for brevity; dynamism is implicit in the work presented in this dissertation, unless otherwise noted. Dynamism in this context refers to networks whose membership and topology may change over time.

An aggregation algorithm is executed over the networked system, in which a small group of *queriers* $\boldsymbol{Q}$ learns the aggregate state of some group $\boldsymbol{S}$ of *data providers* with respect to a set of local inputs. In some instances, a set $\boldsymbol{A}$ of aggregators receives inputs from one or more nodes and provides (partial) aggregate outputs. We will call aggregation of this type *in-network aggregation*. An example of a tree-based in-network aggregation system is shown in Figure 2.3. An *aggregation overlay* $\boldsymbol{V}' = \boldsymbol{Q} \cup \boldsymbol{S} \cup \boldsymbol{A}$ is composed of the sets of queriers $\boldsymbol{Q}$, data providers $\boldsymbol{S}$ and aggregators $\boldsymbol{A}$. The set of queriers is usually a minority of the nodes in $\boldsymbol{V}'$ (we assume $|\boldsymbol{Q}| = 1$ unless otherwise noted). The aggregation overlay spans some subset of nodes in a connected component of the graph $G$ that includes the set of queriers.

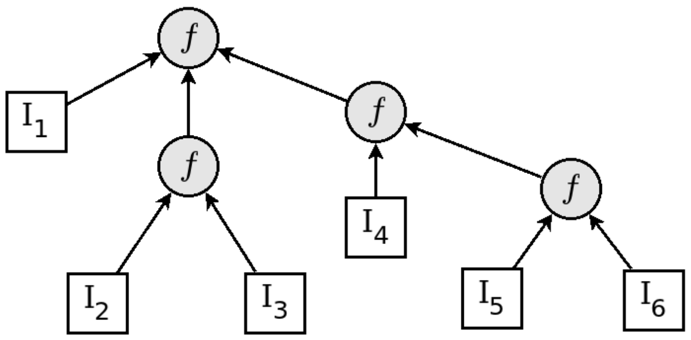


**Figure 2.4:** *Execution of a distributed aggregation function. Inputs $I_i$ are aggregated in-network by the repeated application of a function $f$. The intermediary results in each stage are called partial aggregates.*

An example is shown in Figure 2.3. The network shown is a *heterogeneous* aggregation network, in which special nodes, perhaps more powerful ones, function as aggregators. The specialization of participating nodes imposes limits on the possible configurations of aggregation overlays. *Homogeneous* networks are a common modeling assumption, for instance in wireless sensor networks, in which participating nodes simultaneously function as data providers and aggregators. Since nodes in homogeneous systems have equivalent capabilities, they can self-organize more freely, implying a greater range of possible network configurations.

## 2.3.2 The Aggregation Function

An *aggregation function* $y = f(x_1, \ldots, x_k)$ takes a vector or multi-set of inputs $\boldsymbol{x}$ and produces an output $\boldsymbol{y}$, s.t. $|\boldsymbol{y}| < |\boldsymbol{x}|$. Distributed aggregation algorithms use an aggregation function as the local component of a distributed function. In the most general case, we cannot make assumption regarding the network structure or the execution order. Hence, we require aggregation functions employed in such networks to be both commutative and associative. An example of aggregate computation in a distributed aggregation network is

$$y = f(I_1, f(I_2, I_3), f(I_4, f(I_5, I_6)))$$

where $I_i$ is a set of *local inputs* of a node $i$. This particular execution corresponds to the example shown in Figure 2.4. The intermediary results, for instance $f(I_5, I_6)$, are called *partial aggregates*. Each node in a distributed aggregation network operates on its own inputs (if any) and those contributed by peers, and produces a partial aggregate output to be incorporated in future computations. For simplicity, the aggregate over such a system may be written as $f(\boldsymbol{I})$, where $\boldsymbol{I}$ is the set of all local inputs in a distributed system.

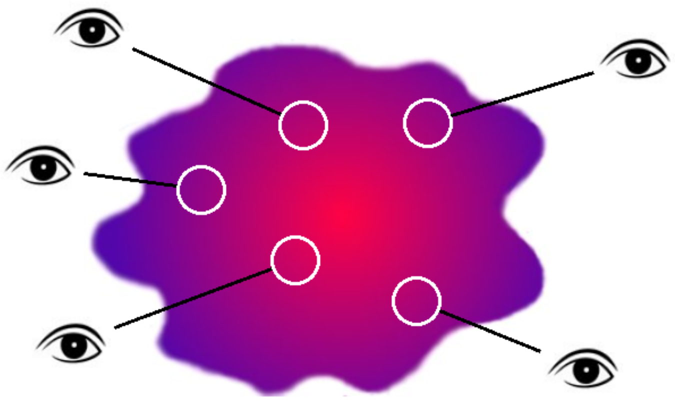


**Figure 2.5:** *Observations of an environment. A number of observers view a potentially unique aspect of a common environment.*

A local input is an observation potentially unique to a participant in a distributed system. We can say that the goal of a distributed aggregation algorithm is to observe the state of some system, the *environment*. Each local input represents an unique *view* on this environment, as illustrated in Figure 2.5. We define the concept of an environment broadly as any system that a contributing node can observe. Examples include local state of managed nodes in network monitoring applications (Stadler et al., 2008), environmental variables (Yao & Gehrke, 2002) and pollution particle count (Paulos et al., 2007).

Examples of distributed aggregation functions include scalar functions, such as SUM (see Example 2.3), COUNT, MAX, MIN, AVERAGE and MEDIAN. More complex functions, such as merging of video streams (Y. Liu & Das, 2006), histograms (Jurca & Stadler, 2009) and top-$k$ aggregates (Chen & Min, 2010) can also be computed. The PACK aggregation function is presented in Example 2.4. PACK is a practical aggregation function which can be used to compress information in networked systems, with the property that individual aggregate inputs are *recoverable* (expandable) in the sense that the aggregate can be expanded into a representation of the original contributions. In contrast, aggregation functions such as SUM (Example 2.3) and COUNT are *non-recoverable* (non-expandable), as the the original contributions are lost except for their aggregate representation.

**Example 2.3 (The SUM function).**
The function $\text{SUM}(\boldsymbol{m})$ takes a set of update messages

$$\boldsymbol{m} = \{m_1 = \langle x_1 \rangle, \ldots, m_k = \langle x_k \rangle\}$$

where $x_i$ is a scalar quantity, and computes $\sum_{i=1}^{k} x_i$. In the in-network aggregation case, we can execute the function iteratively over the local inputs. For example, let us compute SUM over the graph shown in Figure 2.4:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = \text{SUM}(x_1, \text{SUM}(x_2, x_3), \text{SUM}(x_4, \text{SUM}(x_5, x_6)))$$

SUM is an example of a non-recoverable (non-expandable) aggregation function, as individual inputs cannot be retrieved from the computed aggregate.

**Example 2.4 (The PACK function).**
The function $\text{PACK}(\boldsymbol{m})$ takes a set of update messages, each consisting of one or more samples

$$\boldsymbol{M} = \{m_1 = \langle \boldsymbol{x}_1 \rangle, \ldots, m_k = \langle \boldsymbol{x}_k \rangle\}$$

and produces a single update message

$$m = \langle x_{11}, \ldots, x_{kj} \rangle$$

as output, where received contributions are concatenated to produce the new partial aggregate. PACK can for instance be applied to use network packets more efficiently. For instance, a system which produces updates that are considerably smaller than the minimum transmitted packet size benefits from packing as many such fragments into a single network packet, rather than performing simple routing. This is an example of recoverable (expandable) aggregation, as individual inputs can be retrieved from the aggregate. However, PACK is not scalable, as message expansion is inevitable.

### 2.3.3 Aggregation Based on Spanning-tree Overlays

We will focus on the class of tree-based aggregation protocols in this dissertation. Alternative aggregation approaches include those based on gossiping (Kempe et al., 2003; Jelasity et al., 2005) and consensus propagation (Moallemi & Van Roy, 2006; Aurell & Pfitzner, 2009), which is based on the principles of belief propagation (Yedidia et al., 2002).

#### 2.3.3.1 Broadcast/Convergecast Protocols

Let us begin by considering a synchronized aggregation protocol, along the lines of the broadcast/convergecast protocol (Peleg, 2000, Ch. 3); a more concrete example in the context of network management is the ECHO protocol (Lim & Stadler, 2001; Adam et al., 2005). We can view the function of this class of algorithms as a sort of distributed polling – a one-shot query operation. Madden et al. (2002a, 2002b) and Yao and Gehrke (2003) consider SQL-like queries in the more application-specific domain of sensor networking.

We use the AVERAGE function (composed of SUM and COUNT) as our working example. A broadcast/convergecast protocol is executed under a synchronous lossless model, akin to the $\mathcal{LOCAL}$ model (Peleg, 2000, pp. 27):

**Broadcast:** The querier $q$ initiates the execution by broadcasting a query message, for instance $\langle \text{QUERY}, \text{AVERAGE}\{\mathit{var}, *\}\rangle$ (average over variable *var* over the entire system), on all incident edges. Receiving nodes forward the message to their neighbors until the network edge is reached, at which point a *convergecast* phase is initiated. The initial broadcast may propagate over an already established spanning-tree overlay, for instance one established with the Spanning Tree Protocol (STP) (Perlman, 1985). Alternatively, flooding (Peleg, 2000, pp. 33) may be used to build node relations simultaneously with the query distribution.

**Convergecast:** Partial aggregate updates are forwarded progressively towards the querier, as shown in Figure 2.6. A leaf node $s_i$ generates an update message $m_{s_i}^t = \langle I_{s_i}^t \rangle$ at time $t$ and sends to its parent in the tree. An interior aggregator node $a_j$, that receives one or more update messages $\{m_1^t, \ldots, m_k^t\}$, computes a *partial aggregate* $y_j^{t+1} = f(I^{t+1}; m_1^t, \ldots, m_k^t)$ over its local input $I$ (if any) at time $t+1$ and update messages received from contributing peers. An update message $m_a^{t+1} = \langle y_a^{t+1} \rangle$ is then sent upstream to the parent. This process continues iteratively until the querier produces an aggregate result in the final round by combining the received partial aggregates.

**Efficiency.** The message complexity of flooding under the synchronous model is $\mathcal{O}(|\boldsymbol{E}|)$. For the convergecast phase, exactly one message traverses each edge in the spanning tree, giving a message complexity of $|\boldsymbol{E}|$. Hence, the convergecast algorithm is highly efficient, assuming an established spanning tree overlay. Further, the per-edge messaging load is of constant order across the system, implying that the *congestion potential* of any node is low. Contrast the expected $\Delta_r$ messages received by the root node in a convergecast

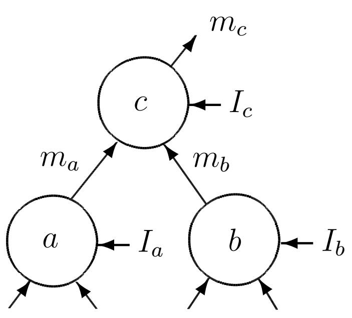


**Figure 2.6:** *A fragment of an aggregation tree. Aggregation node c receives update messages $m_a = \langle y_a \rangle$ and $m_b = \langle y_b \rangle$ from children (downstream nodes) a and b. Local inputs are $I_a$, $I_b$ and $I_c$. Node c computes $y_c = f(I_c, y_a, y_b)$ and forwards message $m_c = \langle y_c \rangle$ to the upstream node, its parent in the tree.*

algorithm (where $\Delta_r$ is the number of incident edges) to the $|\boldsymbol{V}|$ messages received per round in centralized polling. Time complexity of the broadcast/flooding and convergecast phases is $\mathcal{O}(D)$, the diameter of $G$, logarithmic in the system size for practical networks. Contrast this with the $\mathcal{O}(|\boldsymbol{V}|)$ time complexity of centralized polling schemes.

#### 2.3.3.2 Continuous Monitoring Protocols

A query/response protocol is efficient in case we want to obtain a snapshot of the state of a system. However, the rate of updates is determined by the polling frequency of the querier. Let us now consider the case in which we have nodes on independent and asynchronous update schedules, for instance in the case where local updates are triggered by relatively infrequent events.

The Generic Aggregation Protocol (GAP) (Dam & Stadler, 2005) is designed for continuous monitoring in a dynamic network. Building upon a self-stabilizing algorithm for distributed spanning-tree construction (S. Dolev et al., 1993), GAP nodes use a dynamically maintained overlay to periodically send partial aggregates based on local state observations to the current parent. The parent in turn performs in-network aggregation, incorporating the most recent cached partial aggregates, received from its children, into its current state using a local aggregation function. A state update is submitted to the parent in the tree after a rate limitation delay. This process continues until the updates reach the root of the tree, producing a fast aggregate approximation of the global system state. The protocol is efficient in terms of time and message complexity when compared to centralized monitoring alternatives. It also dynamically adapts to changes in topology at the rate at which neighbor discovery and failure detection is handled in the underlying networking layers.

GAP is an event-driven asynchronous protocol. Hence, the concept of rounds does not apply as in the synchronous case considered earlier. However, the protocol guarantees that any update produced by a node $u$ at time $t$ will be delivered to the querier no later than time $t + d_u \cdot \delta$, where $d_u$ is the current distance of the node $u$ from $q$ in terms of the number of hops and $\delta$ is a rate limitation delay. The length of the rate limiting delay determines the efficiency of the protocol: a relatively small $\delta$ causes the protocol performance to degenerate to that of simple routing, that is, an update is in all probability triggered by each received one, whereas a longer delay increases the probability of a partial aggregate being constructed by combining several contributions. A balance must be struck between the timeliness (accuracy) of the aggregate and the edge congestion potential. A-GAP (Gonzalez Prieto & Stadler, 2007) extends GAP by adding accuracy objectives, where heuristics are employed to control the rate of updates for an acceptable estimation error. TCA-GAP (Wuhib et al., 2008) is a further refinement of GAP that investigates efficient aggregation based on threshold crossing alerts.

We discuss the GAP protocol further in Chapter 5 and suggest modifications to increase its resilience to malicious events.

### 2.3.4 Multi-path Aggregation

Consider a tree-based aggregation network, as described by Madden et al. (2002b) and Adam et al. (2005). The accuracy of the aggregate produced depends on the reliability of the system – events such as routing faults (Karlof & Wagner, 2003) and node churn (Stutzbach & Rejaie, 2006) increase the uncertainty of the aggregate as a valid estimate of the real global state of the system (Madden et al., 2002b; Zhao et al., 2003; Considine et al., 2004; Bawa et al., 2007). Hence, the low messaging overhead associated with tree-based aggregation protocols may also be considered a liability in terms of accuracy. Reliable transport protocols (Wan et al., 2004; Stann & Heidemann, 2003; Kosanović & Stojčev, 2008) may help to mitigate the problem, as may protocols designed for dynamic systems, such as GAP (Dam & Stadler, 2005).

Taking advantage of redundant paths available in a network is a one approach to increasing the robustness of a networked system. If we forward a copy of a message on multiple paths, then the probability of delivery to the intended recipient is increased. However, the resiliency gained comes at the expense of increased message complexity, as several copies of each message are transmitted. The construction of robust forwarding paths also complicates such protocols. Forwarding paths should be as ideally be independent to ensure that a fault in a single node affects the minimum number of nodes. A practical solution for robust multi-path forwarding is the braided paths concept suggested by Ganesan et al. (2001).

Multi-path aggregation proposals include the Rings (Roy, 2008) and Wildfire (Bawa et al., 2007) protocols. Such protocols, by definition, send multiple copies of messages. Hence, we are faced with the problem of multiple inclusion, or double counting (Keshav, 2006): naively aggregating over all received messages inevitably causes inflation in the overall aggregate. A trivial (but unscalable in the general case) approach is to carry sufficient identifying information in each packet to enable filtering of duplicates. A form of *order and duplicate insensitive* updates can be constructed, which in principle eliminates the double counting problem. The probabilistic counting scheme of Flajolet and Martin (1985) has been used to overcome the double counting problem in aggregation. (Garofalakis et al., 2007; Nath et al., 2008). However, probabilistic counting schemes are limited to scalars or other simple data types and the form of updates must be tailored to the aggregation function in use. Further, the estimation error associated with the probabilistic counting method may be as large as 33% (Keshav, 2006). Kempe's push synopses (2003) can also be applied to overcome the double counting problem but becomes problematic when we allow for node failures and message corruption in our system model.

## 2.4 Security of Distributed Systems

Security incidents have been known since the advent of computing systems. The earliest examples were often mere pranks and attempts to gain access to exclusive computing resources (Sterling, 1993; Levy, 2001). However, attacks have become ever more serious as more vital resources move on-line. Examples of cybercrime for financial gain, as well as attacks motivated by industrial or military concerns, are well documented and seem to be on the rise (Fisher, 2010; McAfee Labs, 2012).

Security is a vital property for a networked computing systems. Yet, the current state of the field is far from acceptable. In practical terms, most if not all computing platforms in a distributed system are currently vulnerable to a plethora of threats (Sophos, 2012; OWASP, 2012). The challenges stem largely from the conflicting goals of cost, features and usability versus security and robustness.

Systems designers often choose to ignore the difficult issue of node compromise, sometimes seeking justification in the controversial method of security-through-obscurity. For instance, industrial control systems have been considered immune to corruption owing to their relatively obscure systems, compared to the Internet at large, and specialized, proprietary networks (Synergist SCADA, 2012). However, such assumptions are challenged by the recent Stuxnet worm (IEEE Comp., 2010; Matrosov et al., 2010) that targeted industrial control systems, as well as the trend towards integrating industrial and business networks (Panetto & Molina, 2008). Historically, *motes* in wireless sensor networks have been considered inherently vulnerable due to their low cost (Shi & Perrig, 2004). Designers of such systems commonly assume motes, and the environment they are placed in, to be benign, an assumption that must be challenged as sensor networks become more widely deployed and serve more critical purposes.

Let us now briefly review the field of distributed systems security, touching upon some of the more important topics.

### 2.4.1 Vectors of Attack for Node Corruption

Several vectors of node corruption can be considered, for instance

- An adversary may *corrupt individual nodes* by hacking them remotely, inserting viruses, corrupt via worms or by gaining physical access. Many of these are well-known from hacker exploits over the past decades, but opportunities for such attacks will remain for the foreseeable future. Physical access is a real problem in many cases, such as in distributed measurement networks with accessible nodes, for instance sensor networks due to the general assumption of inherently vulnerable low-cost nodes (Shi & Perrig, 2004).
- An adversary may *insert new nodes* under his control. For, example the adversary may join a cooperative measurement network several times with different identities. In the case of sensor networks, the adversary may buy or steal nodes, modify their code and insert into a network (Parno et al., 2005).
- An adversary may attempt to escalate his influence by *simulating* a relatively large number of nodes on a few higher capacity platforms. This is known as the *Sybil attack* (Douceur, 2002). The simulated identities are then directed to attack the system in a coordinated fashion, thereby potentially increasing the influence of the attacker.
- An adversary can *attack the sensing mechanisms* itself, e.g. physically heat a thermometer or bend the arm of a float in a water level meter. Attacks of this class have been called *process-of-measurement* attacks (Trappe et al., 2005; Zug et al., 2007).

In this dissertation, we consider security issues stemming from node compromise of any sort, apart from external influences, such as process-of-measurement attacks.

### 2.4.2 Security Objectives

Systems security is often analyzed in terms of a set of objectives, including the following:

*Confidentiality* refers to the property of data secrecy. Many applications require inputs and data to be kept confidential, that is, hidden from unauthorized users. This is a fundamental problem in computer security, but one which is readily solved by symmetric encryption and decryption primitives under the assumption of securely shared keys.

*Integrity* refers to the wholeness of data, that is, the trust we can place in its authenticity. A classical problem is ensuring that data in transit from node $a$ to node $b$ arrives unmodified. The primary mechanisms used to ensure integrity are symmetric Message Authentication Codes (MACs), cryptographic hash functions, and asymmetric digital signature algorithms. We elaborate on the property of integrity as it applies to aggregation in Chapter 3.

*Availability* refers to the ability of the system to carry out its task without interruption at all times. For instance, denial-of-service attacks, which cause undue expenditure of communications, processing and memory resources, reduce the availability of a system. Jamming attacks against wireless networks also interfere with communications (Wood et al., 2003; Perrig et al., 2004). Wireless sensor networks, whose function depends on limited power resources, can be devastated by attacks which cause motes to expend more communications or processing resources than needed by the proper protocol (Roosta et al., 2006). Resource expenditure in turn depletes the limited battery power of resource constrained nodes.

*Privacy* is a concept related to confidentiality. However, the crucial difference is that data private to a party $a$ is never disclosed to any other party. Privacy is the fundamental objective of multi-party computation (MPC) protocols (Maurer, 2006).

We limit our discussion in this dissertation to the integrity and confidentiality objectives in the context of in-network aggregation.

### 2.4.3 The Adversary and Adversarial modeling

We will use the term *adversary*, common in the cryptographic literature, for our opponents, rather than attacker, hacker or cracker more commonly used in network security research. Unless otherwise noted, we will talk about a single adversary corrupting a number of nodes in a system, which are henceforth colluding to further the adversarial goals. A threshold $t$ for the number of "traitors" is commonly specified as the amount of corrupt nodes that can be tolerated while maintaining the specified security goals.

Adversarial models, that is concrete assumptions about capabilities and intentions of adversaries, are fundamental in order to be able to reason about security properties of a system. Properties, topologies and security requirements of systems differ so that no single model can fit all purposes. Let us nevertheless briefly discuss some commonalities.

The broadest categorization of adversaries is *internal* and *external*. External adversaries, *outsiders*, participate in the system in some capacity, for instance as routers, but are not part of the system. Hence, outsiders do not have access to system secrets, such as cryptographic keys. Their goals include key discovery for eavesdropping on communications, as well as crippling the network by jamming and similar means. The outsider can be assumed to be neutralized with respect to his ability to eavesdrop (confidentiality) on or modify (integrity) communications by the assumption secure channels, i.e. ones in which the confidentiality and integrity are cryptographically ensured by shared keys. Examples include TLS (Dierks & Rescorla, 2008) for traditional networked systems and the alternatives proposed by Perrig et al. (2002) and Karlof et al. (2004) for sensor network motes. Secure channels prevent an outsider from modifying messages (within computational security bounds). However, such mechanisms give no guarantees of delivery, nor do they protect against the insider adversary, which is our primary focus.

Internal adversaries, *insiders*, are more powerful than the outsider. They are members of the network and therefore possess a subset of its secrets and are able to disturb and modify communications more stealthily than the outsider. Once a node participating in the system is corrupted, it is considered under the control of the insider adversary. Note that secure channels, as discussed above, give no security against the insider adversary.

We focus on the insider adversary in this dissertation. In particular, we work from the assumption of the existence of a *stealthy insider* (Perrig et al., 2007), that is, one that deviates from the protocol in an attempt to bias the aggregate computation in his favor, while at the same time remaining undetected for an extended period of time. Contrast the behavior of the stealthy insider to "noisy" availability attacks, such as denial-of-service, which are relatively easy to detect, although not straight-forward to stop or prevent.

#### 2.4.3.1 Capabilities

The *non-adaptive*, or static, adversary corrupts a number of nodes at the start of the computation, while the more powerful *adaptive* one can corrupt new nodes during execution of the protocol. A threshold is frequently specified as the tolerance of a protocol against adversarial corruption, that is, the number of corrupt nodes which the protocol in question tolerates without reduction in security (Maurer, 2006).

A *passive adversary* can corrupt some number of nodes and learn their internal secrets, but not alter their protocol. Hence, the passive adversary, also called *semi-honest* or *honest-but-curious* (Goldreich, 2004, pp. 603), may eavesdrop on communications and collect local inputs, but not actively modify messages. In contrast, the *active adversary* can behave arbitrarily, modifying messages as well as eavesdropping.

A system model may have several classes of corrupt nodes in terms of their capabilities. For instance, a distinction of *mote-class* and *laptop-class* adversaries is common in the sensor network literature (Roosta et al., 2006). The former has little resources available, essentially the same as any other member of the network. On the other hand, the laptop-class adversary has considerably more resources available in terms of battery power, computing power and memory. A more capable adversary may be able to break simple encryption and launch Sybil attacks. A node with a more powerful radio than the general population can use its superior capabilities to compromise the network, for instance modify the network topology by creating virtual channels and jam large segments (Karlof & Wagner, 2003; Wood et al., 2003).

#### 2.4.3.2 Fault Models

Adversarial actions are often analyzed with the tools of systems reliability theory. The simplest *fault model* is the *crash failure* one, under which nodes are assumed to fail without side effects and stay inactive for the remainder of the protocol execution. This failure model can for instance be applied to sensor networks, in which nodes fail eventually due to battery power exhaustion.

The *Byzantine model* (Castro & Liskov, 1999) is a more realistic model, commonly assumed in systems analysis. In terms of security, a Byzantine adversary is one which may perform arbitrarily, including sending conflicting messages to its individual neighbors – also known as *equivocation* (Levin et al., 2009). Byzantine nodes can also fail arbitrarily and provide arbitrary outputs. In contrast to the crash failure model, Byzantine nodes can appear to crash or malfunction temporarily with regards to some of their neighbors and be resurrected at any time.

The *rational adversary* deviates from the protocol at will, but does so to maximize his own utility (Shneidman & Parkes, 2004; Nielson et al., 2005). In contrast, *altruistic* nodes always follow the protocol, even if the logical (and selfish) option is to deviate (Aiyer et al., 2005).

## 2.5 Trusted Systems

Trusted systems theory was formalized in the 1970s in an effort to build high assurance computing systems with definable and provable security goals (Tasker, 1981; TCSEC, 1985), based on the principles put forth in the seminal paper of Lampson (1974).

Central to the concept of trusted systems is the *Trusted Computing Base* (TCB) that defines the "security perimeter" of a trusted system (TCSEC, 1985, pp. 66). Specifically, the *TCB of a trusted system is the collection of all hardware, firmware and software critical to its integrity*. If any one component of the TCB is compromised then the security of the entire system must be considered broken.

Clearly defining the concept of a TCB has proven to be a hard task, one which is still not completed today (Gebhardt et al., 2010). The essential task of verifying the correctness of a TCB is even more challenging. Yet, the concepts and lessons learned from the pioneering early work remain cornerstones of modern systems security. In particular, the modularity concept, minimal scope, and clearly defined interfaces are the only known method to make verification of TCB functionality tractable to some degree.

One of the fundamental constructs of trusted systems theory is the *Reference Monitor (RM)*. A reference monitor is a trusted always-on device that mediates requests into the TCB. The properties of the RM are given in Definition 2.5.

The reference monitor concept, introduced by J. P. Anderson (1972), was the culmination of work on systems integrity by, amongst others, Lampson (1969), Graham and Denning (1971) and Wilkes and Needham (1979). The hardware assisted implementations of reference monitors have been called security kernel (Ames, 1981), often a construct combining memory virtualization and a restricted set of system calls, designed to protect the integrity of its TCB.

---

**Definition 2.5 (Reference monitor (TCSEC, 1985, pp. 58)).**

(i) *Always-on mediation:* The RM must always be invoked and mediate all request to the TCB to enforce the security policy.

(ii) *Physical security:* The physical integrity of a RM must be ensured, for instance by tamper-proofing.

(iii) *Verification of correctness:* The correctness of a RM must be ensured. In practice, this means that the set of operations and overall code size must be small enough so that comprehensive verification of its compliance with stated security goals can be carried out.

---

**Trusted Computing Group.** *Trusted computing* (Challener et al., 2008) is a concept promoted by the *Trusted Computing Group* (TCG)[6]. Central to that concept is a trusted device, the *Trusted Platform Module* (TPM) (Trusted Computing Group, 2011), used to build a TCB. The TPM is implemented as a tamper-resistant chip integrated into the hosting platform. A software stack running in unprotected user space utilizes the secure chip to perform the platform security functions. Broadly, the TPM provides mechanisms to ensure *machine security* (Dell Inc., 2004), that is, the integrity and authenticity of a computing platform. Mechanisms are provided for tasks such as storage and generation of cryptographic keys, hashing, attestation of machine state and random number generation. Three trusted computing primitives are provided by the TPM (Challener et al., 2008; Saroiu & Wolman, 2010):

*Remote attestation* enables a remote entity with the ability to verify that a certain hardware configuration of a machine and that a certain combination of software and firmware was instantiated.

*Sealed storage* enables storage of sensitive data in unprotected memory, binding the data to a particular TPM and software configuration.

*Secure boot* ensures that a machine can only boot a certain configuration of software for a given hardware configuration.

The TPM concept is powerful but is not without criticism, for instance potentially infringing on personal liberties (Stajano, 2003; R. Anderson, 2004; Stallman, 2007). The integrity of the TPM device itself has also been called into question. For instance, Kursawe et al. (2005) take advantage of an implementation weakness to execute a passive attack

[6] `http://www.trustedcomputinggroup.org/`

against a TPM module. Further, current implementations of TPMs are anything but minimal, which undermines the verifiability property that trusted devices must have. A further concern is the origin of the chips and the trust one can place in the manufacturer (Simha et al., 2006; Kursawe, 2011; Schneier, 2012).

# Chapter 3

# Data Source Integrity

The integrity of source data is one of the fundamental properties of distributed measurement systems, such as the aggregation systems we consider in this dissertation. For instance, in the case of networked sensors, *how can we ascertain that a measurement reported by a sensor is a truthful representation of its view on the environment?* Focusing on a single (honest) aggregator model, we investigate whether the aggregation process can be secured in the *general case* of arbitrary data types, aggregation functions and in dynamic networks, while at the same time giving sufficiently strong integrity guarantees. Further, we require the security guarantees to be achievable without placing undue overhead on the underlying aggregation process. In our assessment, such a level of security is infeasible, unless one assumes some means of establishing a basis of trust at the sensor itself: we need to establish the integrity of the sensor data as soon as possible in the processing chain in order to ensure the integrity of the overall aggregation process.

Several integrity preservation mechanisms that apply to the single trusted aggregator model have been proposed and a selection is reviewed in this chapter. However, none fulfill our objectives of simultaneously achieving strong security guarantees and low overhead. Hence, we propose an alternative approach based on the principles of trusted systems (TCSEC, 1985) to construct a *trusted sensor* – a verifiably correct, tamper-proof smart sensor (Breckenridge, 1980) that streams verifiably correct updates to authorized recipients. The security guarantees are based on the concept of embedding the security mechanisms at the earliest possible point in the sensor chain: at the sensor "head" itself. This allows us to construct a secure measurement network of general-purpose networked observation platforms, for instance, wireless sensor nodes, routers or commodity PCs in a collaborative sensing system. We present a comprehensive client/server-based secure measurement system, *TSense*, based on the concept of trusted sensors in conjunction with

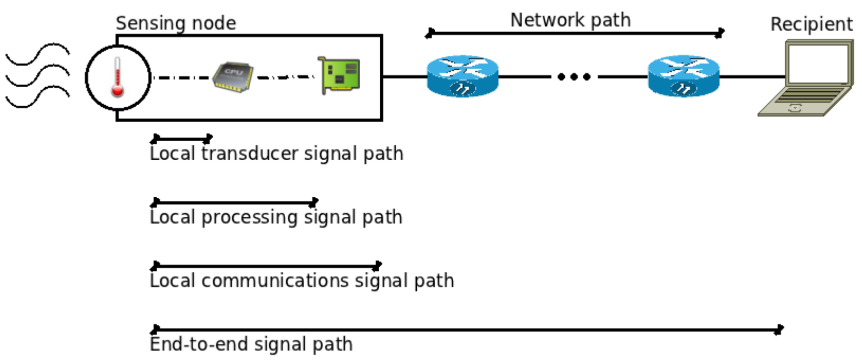

Sensing node
Network path
Recipient
CPU
Local transducer signal path
Local processing signal path
Local communications signal path
End-to-end signal path

**Figure 3.1:** *Overview of a networked sensing system. A sensing node views its environment via a transducer, whose output is made available in the form of discrete updates to a remote recipient. The sensing node is composed of sub-systems, such as analog to digital conversion, signal conditioning and networking, any of which may be under adversarial control.*

an implicitly trusted support architecture. Our system design is supported by a proof-of-concept prototype.

The work is described here has been published in part in Rúnarsson, Kristinsson, and Jónsson (2010), Jónsson and Vigfússon (2011), Jónsson and Vigfússon (2012a) and Jónsson, Palmskog, and Vigfússon (2012). The sensor prototyping project was supported by a Student Innovations Fund grant from Rannís, the Icelandic Centre for Research, and carried out in the summer 2010.

## 3.1 Defining Integrity

Let us begin by delving into the subject of integrity. Beginning with the most generic of definitions, *integrity* is

(i) *The state of being unimpaired; soundness.*
(ii) *The quality or condition of being whole or undivided; completeness.*

The American Heritage © Dictionary of the English Language.

Searching further, we have that "applied to information, integrity is the representational faithfulness of the information to the condition or subject matter being represented by the information" (Boritz, 2005), which brings us closer to the goal of suitably defining integrity as applied to distributed aggregation. Boritz goes on to define information integrity in terms of accuracy/correctness, completeness, currency/timeliness and validity/authorization. We focus on the *correctness* aspect of information integrity in this chapter, extending the discussion to the property of *completeness* in Chapter 5.

**Integrity of sensing.** In this chapter, we are concerned with the concrete problem of being able to trust measurements provided by an array of networked sensors. We require some form of assurance that the received information is a faithful and unmodified representation of the *view* of sensors (perhaps in aggregate) on some environment under observation, as discussed in Section 2.3.2. From the perspective of the recipient, we must consider the following vectors of attack on the data produced by a networked sensor system as shown in Figure 3.1:

(i) A corrupt agent in the environment can execute process-of-measurement attacks, modifying the view of the sensor node on its environment (Trappe et al., 2005; Zug et al., 2007).

(ii) A corrupt sensor node can modify the representation of the observed view at any point in the local processing chain and choose at will if and at which times update messages are generated. Attacks of this sort that are carried out by members of the system are classified as *insider attacks*.

(iii) Intermediary parties, for instance routers, on the network path from sensor to recipient can drop, corrupt, reorder or delay messages. Attacks of this sort are classified as *outsider attacks*.

We explicitly exclude process-of-measurement attacks in this dissertation and focus on the latter two, limiting our attention to the problem of establishing and preserving message integrity in the communications chain from sensor to recipient, as stated in Definitions 3.1.

---

**Definition 3.1 (Integrity of sensing).** We define a system of networked sensors, in which each device $s$ has an unique *view* $\omega_s^{\mathcal{E}}$ of some observable phenomena in an environment $\mathcal{E}$. An honest representation $\bar{\omega}_s^{\mathcal{E}}$ of a sensor view is the most accurate representation of $\omega_s^{\mathcal{E}}$ accounting for non-malicious effects, such as transducer inaccuracy and quantization errors. A *representation* of the observation of a sensor is made available to recipients as an *update message* $m_{s,t} = \langle \bar{\omega}_{s,t}^{\mathcal{E}} \rangle$ at time $t$.

The problem of maintaining integrity of sensing is to ensure that the reported observations are consistent with the local representation of the sensor view: for a reported observation $m_{s,t} = \langle \bar{\omega}_{s,t} \rangle$ received as $m'_{s,t} = \langle x \rangle$ by a node $v$, the integrity of the observation can be considered intact if $x$ can be proven to be identical to $\bar{\omega}_{s,t}^{\mathcal{E}}$.

---

Outsider attacks can be countered in terms of integrity of data by standard cryptographic primitives, as outlined in Example 3.2. However, the corrupt insider problem is difficult, as the corrupt party is a legitimate data producer in the system. In terms of Example 3.2, the insider must be assumed to hold copies of a subset of cryptographic keys, and hence, to be able to alter, manufacture, drop or delay updates at will. Message authentication codes or digital signatures do not solve this problem. Furthermore, by definition of sensing, such malfeasance cannot be positively identified, since each sensor has an unique view on the environment.

**Example 3.2 (Protecting message integrity with MAC).**
Node $a$ produces an update $y$ to be sent to a communicating partner $b$. $a$ computes a digest, or *tag*, over the contents of the message using a MAC[1] function: $t = \mathcal{T}_K(y)$, where $K$ is a symmetric cryptographic key shared by $a$ and $b$. A message $m = \langle y \parallel t \rangle$ is produced and sent to $b$, composed of the original (plaintext) message $y$ concatenated with the tag. A message $m' = \langle y' \parallel t' \rangle$ received by $b$ can be considered an authentic representation of the original one if $\mathcal{T}_K(y') = t'$, contingent upon the computational infeasibility of an adversary forging the tag $t'$ for altered message contents $y'$.

In this chapter, we will focus on the following aspects of integrity from the perspective of systems security:

**Correctness** – the property that the messages received are truthful representations of messages sent.

**Completeness** – the property that the messages sent are delivered to the intended recipient.

**Authenticity** – the property that the messages received can be attributed to a particular contributor.

We neglect several important aspects of integrity, as defined by Boritz (2005). For, instance, we do not address timeliness, which is a challenge in any networked system due to unpredictable delays. Further, node churn, a fact of life in all dynamic networked systems, causes considerable uncertainty in message delivery. Bawa et al. (2007) discuss the *query semantics* of sensor networks in the presence of faults. Likewise, we do not consider the granularity and dynamic range of the sensor signal, assuming that an honest sensor always provides the most accurate representation of a sensor view. Update messages provided by networked sensors are the *local inputs* to the aggregate computation, as

[1] Message Authentication Code

discussed in Section 2.3.2. An honest sensing node provides an update message consistent with the actual observation, while a corrupt one can deviate arbitrarily.

### 3.1.1 Correctness and completeness

The integrity sub-goals of correctness and completeness are defined by Narasimha and Tsudik (2006) in the context of outsourced databases. We restate the definitions in terms of aggregation in networked systems as Definitions 3.3 and 3.4. Both definitions assume a synchronous convergecast model in an arbitrary network graph. A subset of honest data providers simultaneously release update messages at $T_0$ (the first round of the protocol) that are eventually received by a single recipient, the querier $q$, who computes an aggregate $y_q$. The adversarial model allows Byzantine failures, that is arbitrary deviations from the protocol, by some corrupt subset of data providers and all intermediary nodes (e.g. routers), while the querier is considered implicitly honest.

---

**Definition 3.3 (Correct aggregation (synchronous)).**
We define a communications graph $G = (\boldsymbol{V}, \boldsymbol{E})$. A subset of data producers $\boldsymbol{S} \subset \boldsymbol{V}$ hold local inputs $I$ and release (synchronously) a set of update messages $\boldsymbol{M} = \{m_1, \ldots, m_k\}$ at time zero, destined for a querier $q$. An arbitrary number of intermediary nodes forward the messages to $q$. Messages $m \in \boldsymbol{M}$ are tuples $(v, I_v^0)$.

Let $\boldsymbol{I}'$ be the view that $q$ has on the system at time $T$ when some $\boldsymbol{M}'$ update messages have been received. The aggregate $y_q = f(\boldsymbol{I}')$ can be considered *correct* at time $T$ if $\forall\, m' = (v', I') \in \boldsymbol{M}' \;\exists\, m = (v, I_v^0) \in \boldsymbol{M}$ s.t. $v = v' \wedge I_v^0 = I'$.

---

---

**Definition 3.4 (Complete aggregation (synchronous)).**
We define a communications graph $G = (\boldsymbol{V}, \boldsymbol{E})$. A subset of data producers $\boldsymbol{S} \subset \boldsymbol{V}$ hold local inputs $I$ and release (synchronously) a set of update messages $\boldsymbol{M} = \{m_1, \ldots, m_k\}$ at time zero, destined for a querier $q$. An arbitrary number of intermediary nodes forward the messages to $q$. Messages $m \in \boldsymbol{M}$ are tuples $(v, I_v^0)$.

Let $\boldsymbol{I}'$ be the view that $q$ has on the system at time $T$ when some $\boldsymbol{M}'$ update messages have been received. The aggregate $y_q = f(\boldsymbol{I}')$ can be considered *complete* at time $T$ if $\forall\, m \in \boldsymbol{M} \;\exists\, m' \in \boldsymbol{M}'$ s.t. $v = v'$.

---

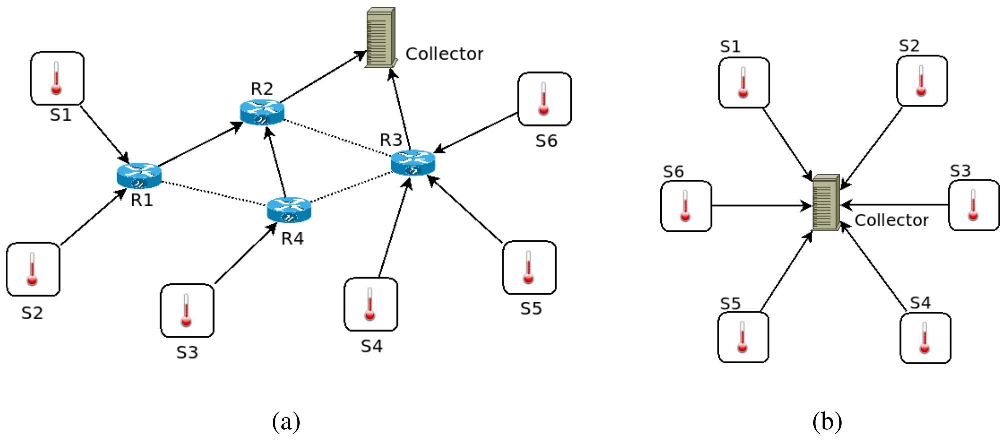


**Figure 3.2:** *A client/server measurement system in which data providers (sensors) provide observations to a collector (querier) over some arbitrary communications graph. Figure (a) shows a small network consisting of a querier (collector), six sensing nodes ($S_1 \ldots S_6$) and four routers (outsiders). Figure (b) shows the aggregation overlay formed over the network of insiders.*

The correctness objective states that all data eventually included in an aggregate computed by the querier must be unaltered aggregate representations of authentic local inputs. However, arbitrarily many update messages may be dropped in the network. On the other hand, the completeness objective states that all updates based on local observations released by data providers must be delivered to the querier, but the correctness of individual members of the set delivered to the querier is not addressed. We can quantify completeness, for instance, as the ratio of transmitted to received packages. However, this ratio is only observable in an ideal system, where an omniscient observer can count both messages sent and received.

Ideally, we want the aggregate to be both correct and complete. However, this ideal is a difficult. Data source correctness is a difficult goal in itself, as we discuss further in this chapter. Completeness is impossible to guarantee in practice, as we discuss further in Chapter 5.

## 3.2 Aggregation in the Single Aggregator Model

Let us now progress towards a solution for trustworthy sensing. We focus on a simple single aggregator model in this chapter – a classical client/server architecture, as shown in Figure 3.2 and described in Definition 3.5. A similar aggregation model is presented by Wagner (2004). The single aggregator model is conceptually simple, yet widely applicable, for instance in centralized network monitoring (e.g. SNMP polling), cloud-based sensing services (e.g. Pachube[2]) and the relatively new paradigm of shared sensing via mobile devices (Kansal et al., 2007).

We assume the existence of an adversary that can corrupt a subset of the data providers and direct them to modify the data on which the aggregate computation is based. One may ask to what end such attacks may be carried out. Indeed, the integrity requirements of an aggregation process must be considered in the context of the consuming application and its operators. Our opinion is that the only applications which can tolerate arbitrary inputs, and thus arbitrary aggregates, are trivial ones suitable only for the most basic of tasks. Consider an important shared or cooperative sensing application, for instance the cooperative radiation monitoring application described by Venere and Gardner (2008). Trustworthy data with the minimum of false positives is required for such an application. We also want to have access to as much data as possible, implying a large user base. However, the price we pay is in terms of security, as establishing personal trust with individual users becomes increasingly more difficult as the user base grows.

Note that the terms "sensor" or "sensing node" do not imply that the nodes in question are wireless sensor network motes. In fact, any type of platform that provides data is a sensing node for the purposes of the following discussion. Henceforth, we will use the generic term *data provider* for such a node.

---

**Definition 3.5 (Single honest aggregator model).** For an arbitrary communications graph $G = (\mathbf{V}, \mathbf{E})$, we define an *aggregation overlay* $G' = (\mathbf{V}', \mathbf{E}')$, a single connected component s.t. $\mathbf{V}' \subseteq \mathbf{V}$ and $\mathbf{E}' \subseteq \mathbf{E}$. The aggregation overlay is composed of a set of *data providers* $\mathbf{S} \subset \mathbf{V}$ and a single querier $q$. Each data provider has a path $(s_i, q)$ to the querier, perhaps a virtual one spanning several nodes in the underlying network graph.

Nodes $v' \in \mathbf{V}'$ are defined as insiders, that is, members of the aggregation network, while $\mathbf{V} \setminus \mathbf{V}'$ are outsiders. Any data provider may be corrupted by the adversary, while the querier is considered inherently trusted and incorruptible.

---

### 3.2.1 Aggregation model

The querier receives a set of update messages $\mathbf{M}^{t,\tau}$ from some subset of sensing nodes $\mathbf{S}' \subseteq \mathbf{S}$ within some window of time $[t, t + \tau)$. Each update message is composed of one or more local inputs, $m_i = \langle I_{i1}, \ldots, I_{iN} \rangle$, produced by a single node in $\mathbf{S}'$. An aggregate $y^t = f(\mathbf{M}^{\mathbf{t},\tau})$ is computed over the contributions by the recipient $q$. A simple example

[2] `http://www.pachube.com`

is the SUM aggregation function: for contributions $[m_1, \ldots, m_k]$ received in $[t, t+\tau)$, the aggregate is

$$y_q^t = \sum_{i=1}^{k} m_k = \sum_{i=1}^{k} \sum_{j=1}^{|m_k|} I_{ij}$$

Note that the work described in this chapter is not restricted to scalar data types or aggregation functions. Rather, we consider the general case of any source data types and any computable function.

The model supports asynchronous querying (one-shot) pull and push models of data delivery, as discussed in Section 2.3. In a push-based data delivery scheme, each node operates on an independent schedule (perhaps loosely synchronized) or delivers updates in response to local events.

### 3.2.2 Adversarial Model

The adversarial goal is to *stealthily* (Przydatek et al., 2003) induce the querier to accept biased aggregate results. The adversary can direct corrupt data providers to produce faulty reports – to omit or modify true readings, as well as manufacture arbitrary ones. The sensing nodes may be operated by entities whose goals run contrary to those of the collector operator. Hence, we assume the sensing nodes can be literally in the hands of an adversary, who can corrupt any aspect of their hardware, firmware or software in order to carry out stealthy attack against the aggregate computation.

The querier is considered implicitly honest and incorruptible, while other nodes can be corrupted by the adversary. The trust in the querier is based on the fact that it is the originator of queries and consumer of aggregate data. Hence, any malicious manipulation of the aggregate by the collector must be considered contrary to its objectives.

We restrict the set of insiders to the set of sensors and the single querier, which can be accomplished in a straight-forward manner by assuming the use of standard cryptographic primitives and a set of pre-shared keys amongst insiders. All other nodes, for instance the routers in Figure 3.2(a), are classified as outsiders and may be disregarded in the analysis.

We restrict our work to the security objective of *data integrity* and disregard other important objectives such privacy. Data confidentiality is not an explicit goal of this work but may nevertheless be included at a small added cost in terms of processing and message size. We do not consider availability attacks, such as routing and DoS attacks, because they run counter to the goals of the stealthy adversary as defined above. We further ignore process-of-measurement attacks in which the sensing process itself is attacked, restricting our attention to network or host-based attacks on the measurement process in the communications path from sensor transducers to the querier, as shown in Figure 3.1.

### 3.2.3 Vulnerabilities

#### 3.2.3.1 Correctness

The correctness of the aggregate under the adversarial model stated, depends primarily on the robustness, or resiliency, of the aggregation process against malicious manufacture or modification of the local inputs. The resiliency of aggregation functions is analyzed by Wagner (2004). Briefly, resilient aggregation refers to the problem of aggregation in a system where the adversary can modify the local inputs of individual nodes. In a single aggregator model the integrity of messages in transit can be ensured by encryption and authentication. On the other hand, local readings can be arbitrarily modified by corrupt insiders. Wagner provides a mathematical framework for formally evaluating the security of several common aggregation functions in a single aggregator model, analogous to Definition 3.5. The conclusions of his work are that many common aggregation functions, such as SUM, AVERAGE (see Example 3.6), MAX and MIN, are inherently insecure. Hence, arbitrary bias can be introduced by even a single corrupt data provider. Wagner's results are extended to TOP-K aggregation in Example 3.7.

**Example 3.6 (Resiliency of AVERAGE (Wagner, 2004)).**
An average is computed as

$$y = \textsf{AVERAGE}(x_1, \ldots, x_n) = (x_1 + \cdots + x_n)/n$$

where $n$ is the number of observations provided. Assume a single corrupt node under adversarial control, $s_i \in \mathcal{S}$, reports a falsified value $x_i^*$. An aggregate

$$\begin{aligned} y^* &= \textsf{AVERAGE}(x_1, \ldots, x_i^*, \ldots, x_n) \\ &= (x_1 + \cdots + x_i^* + \cdots + x_n)/n \\ &= y + (x_i^* - x_i)/n \end{aligned}$$

is computed. If the adversary can choose $x_i^* = x_i + \sigma$ freely, then an overall bias $y^* = y + \delta$ can be introduced by choosing $\sigma = \delta n$.

**Example 3.7 (Resiliency of TOP–K–WEIGHTED).**
Consider the TOP–K–WEIGHTED aggregation function (Jónsson, Palmskog, & Vigfússon, 2012), which can be decomposed as follows:

$$\mathbf{a}' = \text{TRUNC}_k(\text{SORT}(\text{MERGE}(\mathbf{a}_1, \ldots, \mathbf{a}_x)))$$

A number of weighted observations $(v, w)$ are merged, creating a proper set over identifiers $v$. The resulting set is then sorted and the first $k$ tuples returned. Let us consider an application in which contributions are merged by summation: all weights $w$ for tuples tagged with the same identifier $v$ are summed, creating a single tuple $(v, w')$.

By extension of Wagner's (2004) result for SUM, we conclude that a merge function based on summation is inherently insecure. The same applies for other inherently vulnerable functions, such as MIN and MAX that may be used to compute the weight for some applications. The outcome of the outer two functions depends on the computed weights. Hence, their results are trivially influenced via an insecure merge function. We conclude that TOP–K–WEIGHTED must be considered as insecure in Wagner's model as the MERGE function w.r.t. the ordering, and hence top-$k$ ranked data.

#### 3.2.3.2 Completeness

Completeness, as specified in Definition 3.4, is hard to guarantee in general, as discussed further in Chapter 5. Let us briefly consider the problem in terms of the single aggregator model.

Each observation node has initially a path to the querier and is able to communicate. However, link faults, corrupt insiders and outsiders can impact message delivery. We can counter intermittent faults by reliable transfer protocols. Let us simplify the problem by considering only message drops by corrupt insiders in a synchronous network model. At some point in time, the adversary controls some $t$ data providers. Let us assume the corrupt nodes are prevented from modifying messages. However, we must assume the adversary controls the communications channels via his control of the corrupt nodes and can influence message delivery. A reliability layer does not help in this case, since the corrupt host can drop messages before being submitted to the networking protocol. The maximum impact the adversary can have in terms of Definition 3.4 is to direct each of the $t$ corrupt nodes to drop their update messages in each round. Hence, in a single aggregator model, as considered in this chapter, the maximum impact an adversary can have in terms of completeness is to drop $t$ messages out of expected $|\mathbf{V}'|$ in each round.

### 3.2.4 Establishing Integrity

Accurate and trustworthy information is the primary product of a networked measurement system. Hence, we believe that addressing the subject of data integrity is an important task and one to which we devote the remainder of this chapter. Our second goal is efficiency, as networked measurement systems are capable of generating high volumes of traffic. Scalability issues preclude us from adding arbitrarily expensive security protocols to the operational demands. Hence, our goal is to increase the security level of a networked measurement system, while imposing minimum overhead on top of the underlying measurement and aggregation protocol stack. The trivial solution is to unconditionally trust each data provider. However, as discussed previously, this is not an option for systems that are expected to deliver trustworthy data. Let us begin by considering means of guaranteeing integrity preserving aggregation.

#### 3.2.4.1 Increasing Aggregate Resiliency

The severity of the bias introduced depends on the resiliency of the aggregation function, as discussed by Wagner (2004). For instance, the MEDIAN is inherently more resilient than the MEAN. However, the two are not interchangeable but for some subset of applications. Secure median aggregation has been considered, for instance by Roy et al. (2008). Wagner suggests a number of measures to increase the resiliency of insecure aggregation functions. The most generally applicable one (which Wagner considers naive) is truncation of local inputs, which limits the possible bias which an adversary can introduce. However, the drawback is a loss of dynamic range in the local input, more so as we restrict the bounds on the adversary. Truncation is used by e.g. H. Chan et al. (2006) to upper bound the bias an adversary can introduce. Applying domain knowledge, for instance eliminating outliers based on the past history of inputs, is a more flexible method of input sanitation (Buttyán et al., 2006). However, methods of this sort are only applicable to measurements of the "normal" system state and automatically exclude the outliers that are of interest for many applications.

#### 3.2.4.2 Hardening Nodes

Let us now assume we can harden each member to be invulnerable to node compromise and simultaneously ensure that the protocol is unconditionally observed. As before, we assume outsiders are excluded by secure channels connecting each pair of nodes. Such a system allows us to claim *correct overall aggregation*: each data provider delivers correct updates and the updates received by the querier are correctly processed by virtue of *a-priori* trust relations with each participant. Secure channels allow us to assume correct delivery of update messages between trusted parties.

The act of securing an entire distributed system is easier said than done. In practical terms, most if not all computing platforms in a distributed system are currently vulnerable to a plethora of threats, as discussed in Section 2.4. Recall that secure channels are of little additional use once a node has been corrupted, considering the objective of our stealthy adversary. Hence, the brute-force approach of securing each participating node in an aggregation network is bound not only to be expensive in terms of resources, but also ultimately futile, given the current state of computing technology.

#### 3.2.4.3 Hardening the Essential Functionality

The field of trusted systems theory (J. P. Anderson, 1972; TCSEC, 1985) provides us with a potential solution. Let us decompose the functionality of a sensing node, as depicted in Figure 3.1 into that essential and non-essential to the goal of guaranteeing aggregate integrity. All sensing and processing functionality that can influence the correctness of the observations must be considered essential if we want to prevent an adversary from introducing bias. However the task of transferring data can be considered non-essential with regards to the objective of correctness, if we can assume integrity preserving transport mechanisms, such as discussed in Example 3.2, between the components that provide essential functionality. Given this, the majority of node services can be untrusted, which significantly simplifies the task of constructing a trusted sensing node.

Given that we succeed in proving the invulnerability and correctness of the essential functionality of sensing and provide integrity guarantees over the signal path, we can claim to have solved the integrity of sensing problem, as stated in Definition 3.1, in terms of aggregate correctness.

## 3.3 Trusted Sensor

Let us now progress towards the goal stated in the previous section of securing the essential functionality of sensing. To this end, we define a *trusted sensor* – a tamper-resistant smart sensor (Breckenridge, 1980), further described in Definitions 3.8 and 3.9, and shown schematically in Figure 3.3(a). The vision is for the trusted sensor to be a permanently sealed "black box", whose functionality can be unconditionally trusted. However,

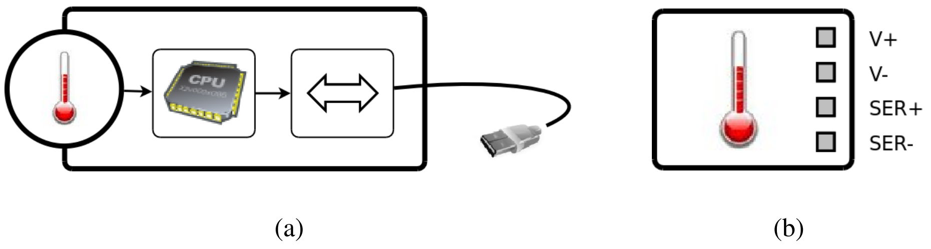


**Figure 3.3:** *Trusted sensor: (a) Schematic of a trusted sensor with a USB-connector, (b) Sample pinout diagram. Bold outlines indicate tamper-proof enclosures.*

in the event of breaches, the device is rendered inoperable and in such a condition that its internal secrets cannot be extracted. The requirements for a trusted sensor closely resemble those of the reference monitor construct (see Definition 2.5) from the trusted systems literature (TCSEC, 1985). Physical integrity is ensured by tamper-resistant packaging, while the correctness of functionality and interface is ensured by rigorous specification and verification procedures. Integrity of communications (and optionally secrecy) is ensured via an integral cryptographic processor and the establishment of secure channels between mutually trusted parties.

The trusted sensor solution enables us to guarantee data integrity at the earliest possible stage in the measurement and aggregation process – at the sensor "head" itself – providing strong data source integrity guarantees in terms of correctness over the end-to-end signal path, as shown in Figure 3.1.

Our vision is for a trusted sensor to be a minimal device in terms of computational and communications resources. In particular, we envision such devices to have a rudimentary communications capabilities, limited to the IPC[3] mechanisms provided by the hosting nodes. The sensor interface exposes a limited set of functions and must be well secured, as stated in Definition 3.8. Hence, a trusted sensor operates in a close symbiotic relationship with its host and the inherently untrusted hosting node still plays an important role in the routing protocol, including handling routing and communication. Our prototype, described in Section 3.5, uses a rudimentary serial interface and can hence have a pin-out as simple as the one shown in Figure 3.3(b).

Given the fact that the trusted sensor is inspired by the reference monitor construct, it is logical to view its integration into a host platform as an extension of the latter's *trusted computing base* (TCB, see Section 2.5), encompassing the essential functionality of trusted sensing.

[3] Inter-Process Communications

**Definition 3.8 (Trusted module).** A **trusted module** is a dedicated hardware device, whose internal operations and secrets, such as private cryptographic keys, are protected within a tamper-resistant enclosure. A trusted module implements a set of operations and exposes an interface, both of which must be well defined in terms of security, functionality and semantics, verified as correct and attested to by a trusted agency. Further, the trusted module provides always-on mediation of all incoming and outgoing communications. Each trusted module is uniquely identified by a globally unique identity tuple consisting of public and private identities. Attestation of a trusted agency on the identity tuple serves as a proof of membership in a particular group of trusted devices.

**Definition 3.9.** A **trusted sensor** is a trusted module (Definition 3.8) that views an environment and produces a corresponding cryptographically unmodifiable stream of update messages to peers in the same or overlapping group of trusted devices.

The family of trusted devices, which the trusted sensor belongs to, includes devices such as some SmartCards (Rankl & Effing, 2001) and RFID tags (Thornton et al., 2006). Trusted devices have been applied in a range of situations, for instance, to solve the fair exchange problem (Avoine & Vaudenay, 2003; Dashti, 2009), to provide secure storage primitives (Levin et al., 2009) and in secure multi-party computation (Fort et al., 2006). Trusted sensors have been considered recently by several authors (Dua et al., 2009; Saroiu & Wolman, 2010; Winkler & Rinner, 2011). The projects cited all implement trusted sensing in a similar manner to our proposed solution, but with the exception that trust is based on a Trusted Platform Module (TPM). In contrast, we propose to integrate dedicated minimal line-speed security logic in the sensors, rather than using a TPM, since the current generation of TPMs is bloated in terms of functionality and provides slow cryptographic operations (Kursawe, 2011). We believe our approach of a dedicated and minimal sensor device is more prudent and in the spirit of the original concepts of trusted systems.

### 3.3.1 Requirements

Let us now consider the requirements for a trusted sensor device. The primary requirement relates to the integrity guarantees of the trusted sensor: a recipient that receives updates produced by an untrusted sensing node hosting a trusted sensor can verify the authenticity of the updates. Hence, given that we can unconditionally trust the trusted sensor device, we can unconditionally trust that the updates produced are consistent with a correct representation of the sensor view on its environment.

Achieving this goal requires a number of intermediary steps, which can be derived from Definitions 3.8, 3.9 and 2.5: A trusted sensor:

(i) is uniquely identifiable and can be positively authenticated by a communicating party as part of the same or overlapping groups of trust,

(ii) is well-defined in terms of functionality and security by a clear and concise specification,

(iii) is verifiable as complying with functionality and security specifications,

(iv) provides physical protection by means of tamper-resistant packaging,

(v) provides always-on mediation of all data passing through its communications interface:

   a) only accepts inputs allowed by the security specification,

   b) only provides outputs which leak no secrets, as defined by the security specification,

   c) only provides outputs (data updates) verifiable as correct and destined for trusted peers.

Let us now consider means of achieving the goals stated.

#### 3.3.1.1 Identity and Device Authentication

Each trusted sensor is uniquely identified by an identity tuple $(id_{pub}, id_{priv})$ assigned at time of construction by the manufacturer. The public identity $id_{pub}$ can be freely disclosed, while the private one must never be revealed. A globally unique and verifiable identity is a prerequisite for preventing Sybil (Douceur, 2002) and cloning attacks, while a private identity (cryptographic key) is required to establish trust relations with peer nodes. We assume the secret identity is symmetric cryptographic key in our prototype, as discussed later on, but an asymmetric key may also be used.

#### 3.3.1.2 Specification, Verification and Certification

Providing a complete and concise specification is the first step towards constructing a trusted device. A rigorous specification covers all aspects of the functionality of the trusted device, as well as its security requirements. The specification can be of any form that unambiguously describes the functionality of the device in terms of its operational semantics. Some variant of structural operational semantics (Aceto et al., 2001; Plotkin, 2004) may for instance be used to construct a rigorous specification.

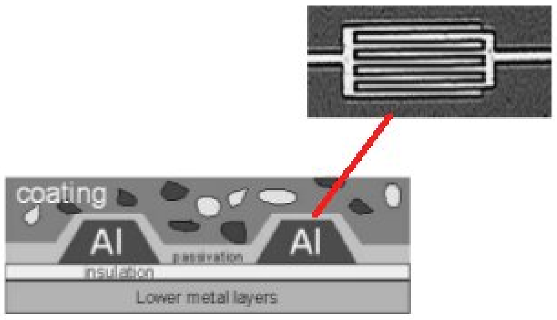
Implementing a device in accordance with the stated requirements is a straight-forward exercise in software and hardware engineering. However, upon completion of the task, we need to verify the correctness of the implementation. Strong trust requires a formal verification procedure (Aceto et al., 2007), carried out by some trusted party, which proves that all aspects of the implementation are in agreement with the specification. The manufacturer and verification/certification party are assumed to be trusted for the purposes of this work; we do not consider the untrusted manufacturer problem (Simha et al., 2006). In practice, the integrity of the hardware on which the software and firmware is placed must be rigorously assessed, as demonstrated by a recent account of a "back-door" in FPGA chips (Schneier, 2012).

Verifiability implies minimality in terms of the scale of code and circuitry, as well as implemented functionality. While formal verification methods are progressing rapidly, the ultimate goal of verifying arbitrarily large systems is still out of reach. However, the task of verifying a specialized device with a small set of functionality may be considered practical.

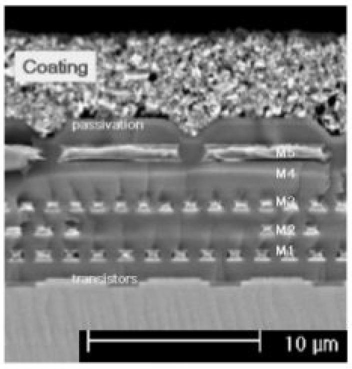
If a device is found to comply with the specifications, the verifying party affixes a verifiable attestation of authenticity. A digital signature over the identity tuple may for instance be considered a verifiable attestation to the fact that the unit complies with the published operational and security specifications. Any party that communicates with a trusted sensor can query and receive its public identity and certification information and submit to the signer (root of trust) for verification. In our prototype, we use the possession of a symmetric cryptographic key as implicit proof of authenticity, that is, the act of generating a symmetric key and installation in a trusted device is considered sufficient proof of authenticity. The identification and certification can be carried out by writing the identity tuple to a special memory location in the sensor device and then physically detaching the programming interface, e.g. sawing it off or blowing a set of fuses (Rankl & Effing, 2001) prior to the application of tamper-resistant packaging.

#### 3.3.1.3 Tamper-resistance.

A verified device can be considered as having been constructed in accordance to a rigorous specification and verified as semantically compliant. However, the security guarantees



(a) A schematic representation



(b) Cross-section of a coating PUF

**Figure 3.4:** *A Physically Unclonable Function (PUF) implemented by randomly doping a coating layer (Tuyls et al., 2006).*

are void if the adversary is able to pry open captured devices, alter their functionality or extract secrets. Transient secrets may be extracted by means such as the cold-boot attacks against RAM chips (Halderman et al., 2008), while permanent secrets can be extracted by carefully examining hardware in a powered-off state. Exposed pins or buses on otherwise trusted devices can reveal internal device secrets (Kursawe et al., 2005).

To counter the possibility of leakage of secrets, we must consider the issue of *tamper-resistance*. A trusted device should be physically protected so that a breach takes considerable expenditure in terms of time and resources and (ideally) renders the device inoperable. Protecting the sensor device by a strong sealed enclosure is a reasonable first step. However, a determined adversary could attempt to drill holes through the enclosure and insert micro probes to extract secrets. Such attacks can be countered by embedding of screening layers, that short out the chip in the event of a physical breach.

A further level of protection can be provided by so called Physically Unclonable Functions (PUF) (Gassend et al., 2002; Tuyls & Batina, 2006). A PUF is a hardware feature sufficiently unique to be used as a physical one-way function. Combined with challenge-response authentication protocols, PUFs provide strong guarantees of device integrity. Several classes of PUFs have been proposed. One class is coating-based PUFs, which are based on random doping of a dedicated layer on top of the sensor processor with dielectric particles (Tuyls et al., 2006), as shown in Figure 3.4. Excitation of this layer ideally produces uncorrelated measurable values, thereby serving as a physical one-way function that can be integrated into challenge-response authentication protocols.

Tamper-resistance is not a trivial issue, as shown by various attacks against supposedly tamper-proof devices (R. Anderson & Kuhn, 1996, 1997), but one that is vital to the goal of constructing a trusted sensor device. However, we will not delve further into the nuances of tamper-resistance in this work.

#### 3.3.1.4 Complete Mediation and Communications Security.

The trusted sensor must provide complete mediation of all instructions and data that crosses its interface. Hence, the trusted sensor is analogous to the reference monitor construct discussed previously. The verification procedure should ensure that the instruction set of the device is well-defined and the adversary cannot exploit it to extract secrets from the device. Further, the sensor interface must not allow any instructions that could leave it in an unsafe state.

The integrity of submitted updates must be protected. To this end, trusted sensors must only communicate with trusted peers with whom they are able to establish secure channels. In the single aggregator case, the common trusted peer is the querier. Communications over secure channels must be protected against modification, for instance by application of a MAC tag. While integrity is the primary security objective of the trusted sensor, confidentiality can be ensured at a small cost by encrypting communications with a shared symmetric key, independent of the one used to generate the tag.

The trustworthiness guarantees provided by a trusted sensor can only be considered meaningful when providing measurements to a trusted peer. Hence, we assume the trusted sensor will only provide output once it has established relations with a trusted recipient. This rule can conceivably be relaxed, allowing trusted sensors to operate as generic sensors providing data to untrusted recipients. However, no guarantees can be stated with regards to the correctness of the output in this mode of operation.

### 3.3.2 Security

Let us assume a specification has been written that describes the semantics of the trusted sensor device precisely. Further, we assume a rigorous security specification defines all allowable inputs and under which condition corresponding outputs are produced. Given such a specifications, absolute verification of functionality against security requirements is conceivable, although difficult given the current state of formal verification methods.

Let us now assume a verification result is available, stating that a device design is compliant with both operational and security specifications. Given that the manufacturer is trusted, the verified design can be translated into hardware and embedded firmware. Further, given that the design of hardware and firmware is a perfect representation of the specification, the resulting device is as secure as the accuracy of the specifications. Hence, we can state that (ideally) the circuitry and code is compliant with the security specifications for all conceivable inputs.

We assume perfect tamper-resistance that leaves the sensor device inoperable in the event of any sort of breach. Hence, the adversary does not gain control over a compromised device. However, the adversary can conceivably extract secrets or residues of secrets from the sensor hardware and/or firmware after a breach. We propose to add a layer of difficulty by incorporating a Physically Unclonable Function (PUF) into the tamper-resistant packaging, guaranteeing that breaches will be revealed. Authentication with PUFs is discussed further in Section 3.4.3.

A tamper-proof sensor, whose functionality is guaranteed to be correct, allows us to fulfill the primary security goal stated earlier in this chapter, that only observations based on faithful representations of views on the environment are produced. Further, each contributed observation can be attributed to a particular sensor and its authenticity can be verified.

## 3.4 *TSense*: A Secure Measurement System

We will now describe a particular design for a trusted sensing system as a test-case for integration of trusted sensor devices into a cohesive distributed aggregation system. A proof-of-concept prototype implementation is described in Section 3.5.

### 3.4.1 System Overview

The system design presented here is a centralized (client/server) architecture, as shown in Figure 3.5. We consider a distributed version of TSense in Chapter 4. *Observation nodes* $s_i$ indicated as thermometer symbols, provide verifiably correct measurements to a network of *collectors* ($C_1 \ldots C_6$ in the example) by virtue of hosting trusted sensors. This system model deviates from the single aggregator model, discussed in Section 3.2, in that the querier can be a distributed structure of trusted collectors. Each sensing node $s_i$ is a generic untrusted hosting platform, hosting one or more trusted sensor modules $\sigma_{ij}$,

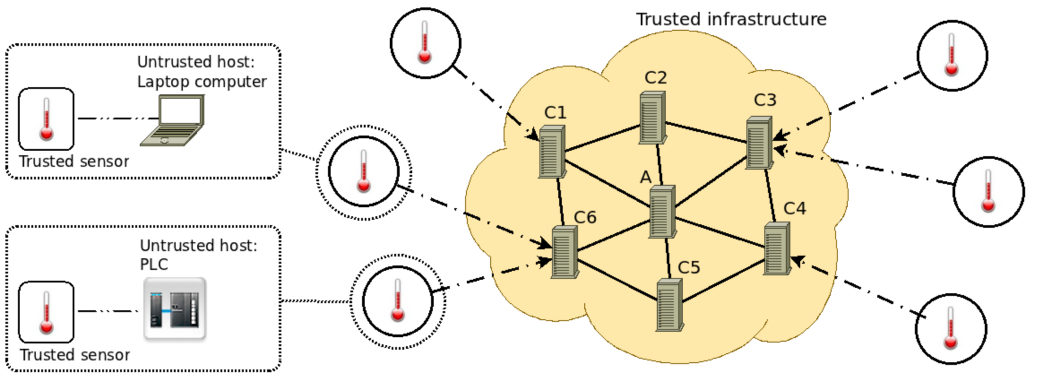


**Figure 3.5:** *TSense system overview. A number of observation nodes equipped with trusted sensors communicate with a trusted infrastructure (secure subnet), consisting of a number of collectors and an authenticator. Secure channels are indicated by bold lines, while channels potentially under adversarial control are shown as dash-dot lines. The secure subnet, indicated by a cloud symbol, is considered implicitly trusted.*

as shown in the blow-ups in Figure 3.5. Any type of hosting platform with networking capabilities can be considered. For example, we show a laptop and a programmable logic controller (PLC) functioning as observation nodes in the figure.

The channels between trusted sensors, untrusted hosts and collector must be considered under adversarial control. However, protocols (described in Section 3.4.3) establish pairwise secure channels $\sigma_{ij} \Leftrightarrow C_k$ between individual trusted sensors and a collector. An *authentication service* $A$ provides authentication services and mediates the establishment of the shared cryptographic keys that facilitate the establishment of trusted channels. The key used for the initial key establishment is the permanent secret device key, which is shared pairwise between each individual sensor and the authenticator.

We call the network bounded by the cloud in Figure 3.5 the *trusted infrastructure*. This network consists of one or more servers in the roles of queriers (collectors) and a single authenticator. Each such server must be assumed to be specially hardened and well protected. In particular, the authenticator, which holds the cryptographic keys for the entire system, should be well protected. The trusted infrastructure is assumed to be interconnected by trusted channels, using mechanisms such as SSL/TLS (Dierks & Rescorla, 2008). Sensing nodes can establish communications with any collector to which they have a path. However, the system must ensure that no sensor identity takes part in multiple pairings simultaneously. This precaution is necessary in order to limit the impact of a cloned sensor in case of a breach. A *Distributed Hash Table (DHT)* (Stoica et al., 2003) implemented over the subnet of queriers (collectors) can for instance provide the necessary lookup capabilities.

### 3.4.2 Security objectives and Adversarial Model

The primary security objectives for TSense are as follows:

(i) Provide guarantees of verifiably correct sensor observations.

(ii) Provide the property of graceful degradation of security in the event of key capture.

The first objective is met by the combination of a trusted sensor with secure protocols, as described in Sections 3.3 and 3.4.3. The second property of *graceful degradation* of security is important for distributed systems. In essence, we can claim that a system has the property of graceful degradation if the compromise of each individual device reduces the overall security level of the system by some quantifiable amount, rather than failing catastrophically. For example, a sensor network in which communications security depends on a single globally shared key is bound to fail catastrophically if that key is ever revealed.

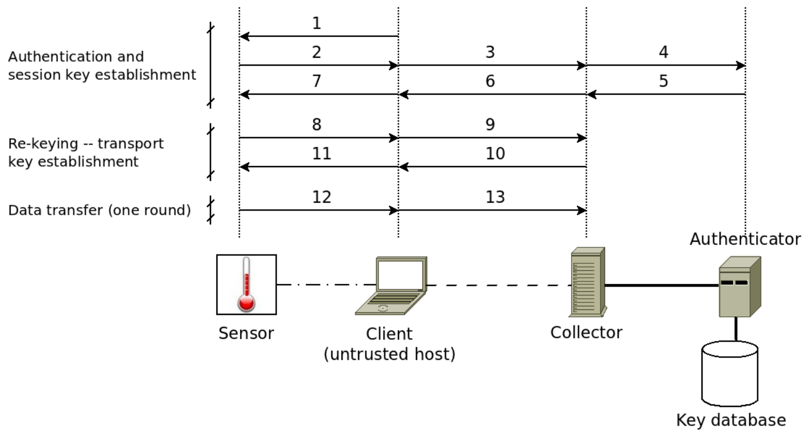
Given the assumption of a layer of trusted devices connected by secure channels, we can amend the adversarial model of Section 3.2.2. The participants in the measurement network are classified as trusted and untrusted. The trusted sensors and the collector/authenticator infrastructure are considered trusted elements, while all other participants are untrusted. We abstract the untrusted elements of the measurement network into the pool of untrusted outsiders, i.e. routers and the like, which are not part of the network. Since the untrusted clients carry the channels, we can classify all attacks against protocols as attacks on the channels. Hence, the revised adversarial model is comparable to the Dolev-Yao model (D. Dolev & Yao, 1983), confining adversarial actions to the channels.

### 3.4.3 TSense Protocols

We have so far described two classes of trusted entities, sensors and collectors. Now, we need to establish a trusted relationship between the two to form a cohesive trusted measurement system. This is accomplished by means of cryptographic protocols, outlined in Figure 3.6 and described in the following sections.

#### 3.4.3.1 Entity Authentication and Session Key Establishment

We begin by describing the authentication and key negotiation protocols that facilitate the establishment of secure channels between trusted sensors and collectors. Each trusted



Authentication and session key establishment
Re-keying -- transport key establishment
Data transfer (one round)
1
2
3
4
7
6
5
8
9
11
10
12
13
Authenticator
Sensor
Client (untrusted host)
Collector
Key database

**Figure 3.6:** *Overview of trusted sensor protocols. Messages are sequentially numbered. Untrusted intra-node channels are indicated by dot-dash lines, untrusted external channels as dashed and trusted channels as solid lines.*

sensor is uniquely identified by an identity tuple, as described in Definition 3.8, assigned at time of construction by the manufacturer. Our prototype design uses a symmetric secret key $K_{A\sigma}$, shared pairwise between sensor $\sigma$ and a trusted third party (TTP), the authentication server $A$ in Figure 3.5, as the secret device identity. The key $K_{A\sigma}$ serves as an implicit proof of device correctness, as well as being used in the initial authentication step upon insertion of a sensor into a trusted aggregation network. The key is generated by the configuring entity and permanently embedded into the device, as described in Section 3.3.1.

The authentication protocol uses the shared secret $K_{A\sigma}$ to securely establish a symmetric *session key* $K_{C\sigma}$, shared pairwise between a sensor $\sigma$ and collector. The session key is in turn used to establish and periodically refresh a *transport key* $K_T$, again shared pairwise between sensor and collector.

The protocol is a variant of the Needham-Schroeder symmetric trusted third party protocol, adapted to suit the application (Needham & Schroeder, 1978, 1987), (Schneier, 1996, pp. 58), Menezes et al. (1996, pp. 502). A symmetric TTP protocol of this sort is a logical choice for the proposed centralized system. The protocol assumes the existence of an untrusted application agent $s$, running on the hosting node, that bootstraps the protocol. We split the protocol into two phases, the first being a mutual authentication and session key establishment protocol.

*Message 1:*

$$s \rightarrow \sigma : \langle \text{IDENTIFY} \rangle$$

- An untrusted aggregation agent running on the observation node, represented by entity $s$, requests an identity packet from its hosted trusted sensor in order to bootstrap the authentication process.

*Messages 2,3 and 4:*

$$\sigma \rightarrow s \rightarrow C \Rightarrow A : \langle \text{AUTH}_1, \sigma, \mathcal{E}_{A\sigma}(\sigma, N_\sigma) \rangle$$

- $\sigma$ responds with a message containing its public identity and an encryption over the id and a nonce $N_\sigma$. The permanent private device key $K_{A\sigma}$ is used. The message is routed from $\sigma$ to $A$, treating both $s$ and $C$, as well as any intermediary nodes, as outsiders, as none hold $K_{A\sigma}$.
- $A$ looks up the encryption key $K$ for the identity $\sigma$ in its key database and decrypts $\sigma'$ and $N'_\sigma$. $A$ then verifies $\sigma = \sigma'$. A match of decrypted information against unencrypted information and stored state is considered a sufficient proof that $\sigma$ holds the key $K_{A\sigma}$, and hence as an implicit proof of the identity of the sensor device and its membership in the group of trusted devices.

*Message 5:*

$$A \Rightarrow C : \langle \text{AUTH}_2, \sigma, R_{C\sigma}, \mathcal{E}_{A\sigma}(\sigma, C, N_\sigma, R_{C\sigma}) \rangle$$

- $A$ generates a random number $R_{C\sigma}$ which is the key material to be used by $\sigma$ and $C$ later on in the protocol to establish session keys. The key material should preferably be of length equal to that of a cryptographic key in the system and generated by a cryptographically secure (high entropy) random number generator.
- $A$ prepares an authentication packet $\mathcal{E}_{A\sigma}(\sigma, C, N_\sigma, R_{C\sigma})$. This packet is encrypted with the permanent device key $K_{A\sigma}$, and hence, assumed to be readable only by $A$ and $\sigma$ by the assumption of the exclusivity of the key.
- $A$ sends a message to $C$, containing the key material, the public identity of the sensor and the encrypted authentication packet. Communications security is implicit in this step by the assumption of a trusted $A \Leftrightarrow C$ channel.
- $C$ derives a session key set $K_{C\sigma} = KDF(R_{C\sigma})$ for the identity $\sigma$.

*Messages 6 and 7:*

$$C \rightarrow S \rightarrow \sigma : \langle \text{AUTH}_3, \sigma, \mathcal{E}_{A\sigma}(\sigma, C, N_\sigma, R_{C\sigma}) \rangle$$

- $C$ forwards the authentication packet to $\sigma$. Note that $C$ does not hold the key $K_{A\sigma}$ and is hence unable to modify the packet.
- $\sigma$ decrypts $(\sigma', C, N'_\sigma, R_{C\sigma})$ and verifies $\sigma = \sigma'$, $N_\sigma = N'_\sigma$. The proper encryption proves implicitly to $\sigma$ that the provider of the key material is a trusted authority in the system by the assumption of exclusivity of the key $K_{A\sigma}$.
- $\sigma$ derives a session key $K_{C\sigma} = KDF(R_{C\sigma})$ and stores for the identity $C$.

#### 3.4.3.2 Session Key Confirmation and Transport Key Establishment

Upon establishment of the shared session key, the sensor and collector execute phase two of the authentication protocol, based on the key confirmation phase of the Needham-Schroeder protocol, to establish an initial transport key. This phase also serves to confirm the newly established session key.

*Messages 8 and 9:*

$$\sigma \rightarrow s \rightarrow C : \langle \text{REKEY}, \sigma, \mathcal{E}_{C\sigma}(\sigma, C, N'_\sigma) \rangle$$

- $\sigma$ initiates the protocol by sending $C$ an encryption, using the newly established session key $K_{C\sigma}$, over the identity of the two trusted peers and a fresh nonce $N'_\sigma$.
- $C$ looks up a session key $K$ for the identity $\sigma$ and decrypts $(\sigma', C', N_\sigma)$ and verifies $\sigma = \sigma'$, $C = C'$. $C$ stores $N_\sigma$ and compares against stored values. The proper decryption of the message payload proves to $C$ that $\sigma$ holds the correct session key. Hence, $C$ can implicitly accept that $\sigma$ is a member of the group of trusted devices.
- $C$ picks a random number $R_T$ and derives transport keys $K_T = KDF(R_T)$. The key material $R_T$ should be of sufficient length for the security level required and generated by a cryptographically secure random source.

*Messages 10 and 11:*

$$C \rightarrow S \rightarrow \sigma : \langle \text{NEWKEY}, \mathcal{E}_{C\sigma}(C, \sigma, N'_\sigma - 1, R_T) \rangle$$

- $C$ sends the key material $R_T$ to $\sigma$, along with a predictably related nonce.
- $\sigma$ decrypts $(C', \sigma', N''_\sigma, R)$ using the current session key $K_{C\sigma}$ and verifies $C' = C$ (against the current state), $\sigma = \sigma'$, $N''_\sigma = N'_\sigma - 1$. Proper verification confirms to $\sigma$ that $C$ holds the same session key.

- $\sigma$ derives transport keys $K_T = KDF(R_T)$.

This protocol can be executed as often as needed to maintain the freshness of the transport keys, as the proper practice for cryptographic protocols is to limit the lifetime of keys to minimize the threat of compromise.

#### 3.4.3.3 Data Transport Protocol

The third protocol is a transport protocol, leveraging fast symmetric authenticating encryption with the transport key pair $K_T$ to secure the integrity and secrecy of the transmitted data against active and passive attackers. Note that in all cases, we assume $K$ to be a key pair, consisting of a key for encryption and an independent one for authentication, in keeping with the key separation principle (Gligoroski et al., 2008),(Barker et al., 2007, sec. 5.2).

**Data format.** The format of the data is not of primary importance in the transport protocol, but two example formats can be given as follows:

1. $\sigma$ creates a data vector of the form:

   $$\boldsymbol{D} = [t_\sigma \parallel l_D \parallel m_1 \parallel \cdots \parallel m_k]$$

   where $t_\sigma$ is a timestamp of the first observation, $m_i$ are observations and $l_D = k$.

2. Alternatively, each observation can be timestamped as

   $$\boldsymbol{D} = [l_D \parallel \{t_1, m_1\} \parallel \{t_2, m_2\} \parallel \cdots \parallel \{t_k, m_k\}]$$

**Data transfer.** Data transfer is a straight-forward application of authenticating encryption over a data vector $\boldsymbol{D}$, given a shared transport key $K_T$. The data to be contributed by a sensor $\sigma$ is packed into a transport message and routed to $C$ through $s$:

$$\sigma \to s \to C : \langle \text{DATA}, \sigma, c_\sigma, l_\mathcal{E}, \bar{\mathcal{E}}_T(\sigma, c_\sigma, \boldsymbol{D}) \rangle$$

- $\sigma$ encrypts a message consisting of its identity $\sigma$, a monotonically increasing counter $c_\sigma$ and the data vector $\boldsymbol{D}$. An *authenticating encryption function* $\bar{\mathcal{E}}$ is used with a transport key set $K_T$. The public node identity, counter and the length of the ciphertext $l_\mathcal{E}$ is sent in plaintext.

- $C$ looks up the current transport key $K_T$ for the pairing $(C, \sigma)$, decrypts and verifies the message. $C$ stores the data vector as a contribution of $\sigma$ to the aggregate computation if the message is verified as authentic. Otherwise, the message is discarded.

**Authenticating encryption.** The authentication and key exchange protocols do not require explicit authentication of messages as proper decryption of well specified message fields was assumed to be a sufficient proof of authenticity. Such assumptions no longer hold in the case of data transfer. Hence, we need to add explicit authentication of messages to the protocol. Several implicitly authenticating modes of operation exist for block ciphers, including the Offset Counter Mode (OCB) (Rogaway et al., 2003) and Galois Counter Mode (GCM) (Dworkin, 2007). We leave the choice of method of authentication open and assume the more generic compositions for symmetric block ciphers and MACs discussed by Bellare and Namprempre (2007). The strongest of the compositions analyzed is *Encrypt-then-MAC (EtM)*

$$\bar{\mathcal{E}}_K(M) = C \parallel \mathcal{T}_m(C),\ C = \mathcal{E}_e(M)$$

where $K = (K_e, K_m)$ is a pair of independently derived keys for encryption and MAC, $M$ is the plaintext message and $C$ is the ciphertext. Note that the tag can be truncated to any number of bits which gives an acceptable level of security.

Small update messages whose data vectors are of a size much smaller than the block length (in case of block ciphers) may be optimized by including the tag in the encryption, that is, use an *MAC-then-Encrypt (MtE)* composition of the form

$$\langle \text{DATA}, \sigma, c_\sigma, l_\mathcal{E}, \mathcal{E}_T(\sigma, c_\sigma, \boldsymbol{D}, \mathcal{T}_T(\sigma, c_\sigma, \boldsymbol{D})) \rangle$$

**Mode of operation.** We assume a block cipher for the following discussion. Our prototype is based on the AES block cipher, but other secure ciphers can be used as well. Block ciphers are operated in several modes of operation. The simplest is the *Electronic Code Book (ECB)* mode, under which each block of plaintext is independently encrypted by repeated application of the cipher algorithm under a single static key. ECB mode is the simplest but also most insecure mode of operation. One of the disadvantages of ECB is that a repeating plaintext encrypts to predictably repeating ciphtertext. This immediately leaks information to the adversary – that the source data is repeating. A more secure mode of operation should be chosen to ensure unpredictability in the generated ciphertext. For instance, the *Cipher Block Chaining (CBC)* mode of operation (Dworkin, 2001) ensures variability by "feeding" the previous ciphertext block into the encryption of the next one:

$$C_i = \mathcal{E}_K(P_i \oplus C_{i-1})$$

where $P_i$ are plaintext blocks and $C_i$ are the corresponding ciphertext blocks. The first ciphertext block $C_0$ is an *initialization vector (IV)*, which for CBC must be unpredictable to ensure that a sequence of ciphertext blocks is always unique, even for static plaintext. An acceptable method of generating an unpredictable IV is to encrypt or MAC a non-secret nonce, which may be a monotonically increasing counter (Dworkin, 2001, Appendix C). Hence, we can use the plaintext counter $c_\sigma$ as a nonce source, encrypting under the transport key to generate the IV.

**Authentication vs. authenticating encryption.** We assume authenticating encryption in the TSense system design, even though authentication suffices for our primary objective of ensuring data integrity. Encryption comes at a small cost in terms of memory and processing costs, as well as some message expansion in the case of block ciphers. However, encryption buys us the property of confidentiality, which may be important for some applications. Further, as we explore in Chapter 5, encryption reduces the adversary's ability to make intelligent dropping decisions, thereby reducing his effectiveness in attacking the completeness of the aggregation process.

#### 3.4.3.4 Authentication Incorporating Physically Unclonable Functions

The correctness guarantees provided by the TSense system depend on the assumption of sensor invulnerability. However, we must consider the eventuality of a resourceful adversary gaining access, despite any level of physical protection. Even assuming a trusted sensor device is rendered inoperable by tampering, we cannot rule out the possibility of extraction of secret device data, as discussed in Section 3.3.1. In case of such a breach, the adversary has sufficient information to simulate the corresponding device, as algorithms are not assumed to be secret. We can argue for the expenditure of time and resources to be prohibitive in terms of the adversary ever gaining the upper hand in terms of the number of devices compromised. Nevertheless, as discussed in Section 3.2.3, an adversary controlling even a single device is in a position to arbitrarily influence the aggregate computation. Hence, simulation of a single device represents a failure in terms of our primary objective of aggregate correctness.

Clearly, we must limit the possibility of sensor simulation in the eventuality of secret key extraction. Physically unclonable functions (PUF) present a potential solution to the problem, unequivocally linking the physical protection and cryptographic properties of the sensor device, as discussed further in Section 3.3.1. PUFs can be viewed as a cryptographically secure keyed hash function, unique to the device. The configuring party (the trusted manufacturer) for device $v$ issues a set of $i$ challenges and harvests the responses, $r_{v,i} = \text{PUF}(c_{v,i})$. This series represents (with high probability) a unique fingerprint for the device $v$. The probability that an adversary can guess a response to a particular challenge is ideally $1/2^{l_R}$, where $l_R$ is the bit length of the response expected, given that challenges and responses are uncorrelated.

Let us assume we have a PUF that is integral to the physical protection of the trusted device, for instance a coating PUF (Tuyls et al., 2006) and the function is destroyed by any tampering that can result in the extraction of secret device keys. Further, we assume the PUF is ideal in the sense that it always provides an output consistent with its input, but without any correlation between input and output bits. In other words, we assume the PUF is a *random oracle* (Bellare & Rogaway, 1993).

Let us now amend the authentication protocol described previously to incorporate a PUF. The first steps are identical to the previous protocol but a challenge/response step is inserted between messages 4 and 5.

*Message 4a:*

$$A \Rightarrow C \rightarrow S \rightarrow \sigma : \langle \mathcal{E}_{A\sigma}(c_{A\sigma}, N_A) \rangle$$

$A$ routes a challenge to $\sigma$ end-to-end encrypted under the permanent device key $K_{a\sigma}$. A nonce is included to ensure ciphertext unpredictability in case of challenge reuse.

*Message 4b:*

$$\sigma \rightarrow S \rightarrow C \Rightarrow A : \langle \mathcal{E}_{A\sigma}(r_{A\sigma}, N_A - 1) \rangle$$

$\sigma$ computes $r_\sigma = PUF(c_A)$ and sends to $A$, which compares $r_\sigma$ to its stored $C/R$ pairs for $\sigma$. $A$ breaks off contact if verification fails, preventing the key material (Message 5) from being delivered.

Given a secure PUF integral with the tamper-resistance measures applied to the trusted sensor device, we can claim that the probability of successfully simulating a sensor, in the event the adversary is able to extract the secret device key, is negligible. Hence, the primary security goal of correctness can be guaranteed if we can assume the existence of a perfect PUF integrated into the tamper-resistant packaging of the trusted sensor.

#### 3.4.3.5 Protocol Verification

The correctness of cryptographic protocols is of great importance but at the same time difficult to prove. Traditionally, protocols have been considered sufficiently strong after having been cryptanalyzed extensively for a number of years. That is, if no efficient attacks are discovered, we can consider a protocol adequate for the time being. The protocols presented in this work are both relatively simple and based on existing protocols that are believed to be sufficiently strong. However, the slightest modifications of cryptographic protocols can have severe implications in terms of their strength against possible attacks. Hence, verification is required, even for seemingly simple protocols, such as ours.

We turn to formal verification tools to prove the correctness of the protocols. Both the tool-sets Avispa (*Avispa Project*, 2012) and ProVerif (Blanchet, 2009, 2012) were considered.

The full authentication protocol has been modeled in ProVerif (Kristinsson, 2013). Queries show that no secrets are leaked, given the secrecy of the key $K_{A\sigma}$. The version of the protocol with PUF challenges has also been modeled. In the analysis, we assume the key $K_{A\sigma}$ has been revealed. The PUF is modeled in ProVerif as a perfect keyed hash function. This key corresponds to the physical features in our proposed protocol. As would be expected, ProVerif queries indicate that the adversary is not able to respond correctly to a challenge, unless this key (the physical feature) is intact. The results for the main authentication protocol were independently validated by Avispa modeling and analysis (Tryggvason, 2012).

### 3.4.4 Security of TSense

The security goals of the TSense system are met by the combination of trusted modules and secure protocols. A trusted module is secure in isolation, as discussed in Section 3.3.2, by virtue of (assumed) verified compliance with a stated specification and physical hardening.

The security of the protocols depends on their design and implementation, as well as the security of the underlying cryptographic primitives. The security guarantees of ciphers and MAC functions are generally computational, implying that the overall integrity of the system depends on the key length and the implied difficulty of an exhaustive search. This of course assumes a secure primitive, i.e. one for which no efficient attacks are known.

#### 3.4.4.1 Authentication Protocol

We base our authentication protocol on a known design which has been extensively cryptanalyzed for a number of years (Needham & Schroeder, 1978; Denning & Sacco, 1981; Needham & Schroeder, 1987) and is used with modifications as the basis for the Kerberos protocol (Neuman & Ts'o, 1994). Unlike Needham-Schroeder, our authentication protocol is bootstrapped by an outsider, the untrusted application agent $s$ executing on the sensing node. However, its involvement is limited to sending the first bootstrap message and thereafter acting as a router for encrypted communications. After the first message, its role is comparable to any intermediary node in a Needham-Schroeder or Kerberos exchange. Instead of communicating directly with the authentication server, the sensor goes through a proxy, the collector $C$, which also happens to be its intended communicating party. However, $C$ is treated like an outsider for the initial phases of the protocol; the initial exchange is end-to-end secure between $\sigma$ and $A$.

**Replay attacks.** The original Needham-Schroeder protocol was vulnerable to replay attacks (Denning & Sacco, 1981), which can be fixed by using nonces and/or timestamps (Needham & Schroeder, 1987). We use nonces in our protocols to counter replay attacks. Using timers to guarantee freshness is problematic in our case, as the sensor relies on the untrusted client to set its time. Instead, we use randomly initialized monotonically increasing counters in our `tsensor` prototype as client-side nonce sources. The nonces also provide variability in the encrypted payload, thereby increasing the difficulty of cryptanalyzing the protocols.

**Authenticity of messages.** Messages in the authentication protocol are solely encrypted, not authenticated. We assume the implicit authentication based on verifiably correct decryption of protocol packets is sufficient to ascertain the authenticity of the exchange.

**Discovery of keys.** The permanent secret key $K_{A\sigma}$ of the sensor identification tuple is never released in any form by the sensor, but solely used to encrypt the $(\sigma, N_\sigma)$ tuple in the initial sensor message and the delivered session key $K_{S\sigma}$ at the end of the first authentication phase. Extracting $K_{A\sigma}$ by observing messages must be considered computationally infeasible, given that the nonce causes payloads to appear random, assuming a strong encryption algorithm. Note that even $C$, the trusted recipient, is treated as an outsider for the first phase of the authentication protocol. Given the infeasibility of extracting $K_{A\sigma}$, the delivery of the session key $K_{S\sigma}$ at the end of the first authentication phase must be considered secure. Likewise, the delivery of the transport key material at the end of the second phase is secure, given the secrecy of $K_{S\sigma}$. In the event the session key is revealed, it can be used for the current session but cannot be re-used for future sessions, since both sensors and collectors require fresh keys per session.

The *Man-in-the-Middle (MitM)* attack a classical one in distributed systems: the adversary places a corrupt node between two legitimate entities, intercepting and even modifying their communications. By definition, the TSense system contains a number of untrusted entities in the messaging chain, the observation nodes. However, assuming discovery of keys $K_{A\sigma}$ and $K_{C\sigma}$ is infeasible, precludes the untrusted agent $s$ from mounting a MitM attack. The work involved is no easier for nodes that are not proper members of the system (outsiders).

#### 3.4.4.2 Transport Protocol

The transport protocol is a straight-forward application of authenticating encryption. We assume the generic encrypt-then-MAC composition as described by Bellare and Namprempre (2007). The strength of the protocol depends on the cipher and MAC function used and the composition of the two.

**Discovery of $K_T$.** A considerable bulk of data transport messages can be expected to be delivered for each instance of the key $K_T$. Hence, the adversary may attempt to eavesdrop on this exchange and attempt to uncover the key by analyzing captured ciphertext. The probability of such an attack succeeding depends on the strength of the underlying cryptographic primitives and protocols. In our simple data transfer protocol, the adversary can conceivably mount a known plaintext attack, since the public identity of the sensor is the first field in the encrypted message. However, the difficulty of this attack is comparable to that of an exhaustive key search, given a secure encryption primitive and mode of operation. The security of the basic transport protocol can be increased by adding a random number of random bits to the front and back ends of the message prior to encryption

$$\bar{\mathcal{E}}_T(r_1 \parallel \# \parallel (\sigma, c_\sigma, \boldsymbol{D}) \parallel \# \parallel r_2)$$

where # is a marker, enabling the random bits to be stripped off once the plaintext is known. The last block of the message is further padded by random bits $r_2$.

In the event a transport key is revealed, the adversary can read and modify messages. However, the transport key has a limited lifetime, requiring periodic re-keying operations.

Hence, the adversarial advantage gained in the unlikely event of transport key compromise is limited.

The adversary is precluded (within computational bounds) from forging messages, as long as the authentication key of the transport key set remains secret. The authenticity guarantees achievable depend on the strength of the algorithms, as well as their composition, used to encrypt and authenticate the message.

Transport message payload is encrypted and extraction of information must be considered infeasible, given the assumption of secrecy of $K_T$. Assuming an Encrypt-then-MAC (EtM) composition of strong encryption and tagging primitives, the forging of messages is considered computationally infeasible (Bellare & Namprempre, 2007).

#### 3.4.4.3 PUF Authentication

Let us assume the PUF function is a random oracle (Bellare & Rogaway, 1993). For every query with a new input value, the random oracle generates a fresh cryptographically secure random number, stores it in a table and outputs the corresponding bits. For subsequent inputs that have been encountered before, the random oracle responds by returning the stored value. Hence, a random oracle maps every conceivable input to a particular cryptographically secure random response in its output domain.

The difficulty of simulating a sensor increases in proportion to the probability of forging a response without having access to the PUF, which we assume is destroyed as part of the tampering operation. Given a PUF that behaves like a random oracle, the probability of forging a response to a query is $1/2^{l_R}$. For a moderate set of challenges and a large enough response (in bits), the probability of guessing the response must be considered small. However, the adversary can try phases multiple times in the protocol given. The difficulty of guessing or brute-forcing responses to challenges can be increased by by introducing a significant delay in case of failed challenge/response exchanges, as is the current practice in protecting computer systems against password guessing. A further protection step can be added in which the authentication server caps the number of tries $k$, thereby upper-bounding the probability of guessing a challenge, $k/2^{l_R}$. The adversary may attempt to observe a number of authentication exchanges in an effort to uncover the set of challenge/response pairs. However, in the protocol given, both challenges and responses are encrypted by the secret key $K_{A\sigma}$. We base the security of the PUF-augmented authentication protocol on the assumption that the PUF is disrupted in the event this key is revealed. Hence, the adversary can never observe responses, although challenges are readable if the adversary is in possession of $K_{A\sigma}$.

#### 3.4.4.4 Protocol Interruptions.

The reliance on untrusted services for discovery and trusted connection mediation exposes a potential vector of attack – that of choosing to ignore trusted devices or disrupt communications between them. However, ignoring trusted devices accomplishes little except to exclude the node in question from the overlay, which can be interpreted as an availability attack against compromised nodes. Likewise, disruption of the authentication and/or initial re-keying protocols does not increase adversarial influence over the aggregate, since sensors remain inert until the initialization is complete. No effort is made towards securing the system against disruption of this sort or indeed other availability attacks. However, this is a reasonable choice in light of our stated security goal of ensuring correct aggregation. If corrupt nodes contribute data, then it must be correct. Corrupt nodes that choose to deviate from the protocol do so in isolation and do not accomplish anything beyond excluding their own contributions.

## 3.5 Prototype Implementation

We support the TSense system design by a proof-of-concept prototype (Rúnarsson et al., 2010). The code is released under an open-source license and available at `http://code.google.com/p/tsense`.

### 3.5.1 Prototype Components

#### 3.5.1.1 `tsensor` – Trusted Sensor

The trusted sensor prototype, `tsensor`, is a USB dongle with temperature and luminosity sensors, powered off the USB bus. A picture of the prototype is shown in Figure 3.7. A simple sensing array of a thermistor and photo-resistor was constructed for the prototype but more sophisticated transducer arrays can be envisaged for future work. We base our implementation on a commercially available embedded systems processor, the Atmel ATmega328 (Atmel, 2010). For ease of prototyping, we used the Arduino Duemilanovae (Arduino, 2009) experimentation board as our development platform.

The ATmega328 is a RISC-based microprocessor, intended for embedded systems and capable of running at up to 20 MHz. It has on-board 32K of Flash program memory, 2K of RAM and a 1K EEPROM. Several analog and digital inputs and outputs are provided. The Duemilanove board comes with an ATmega328 in a 28-pin DIL package and includes all

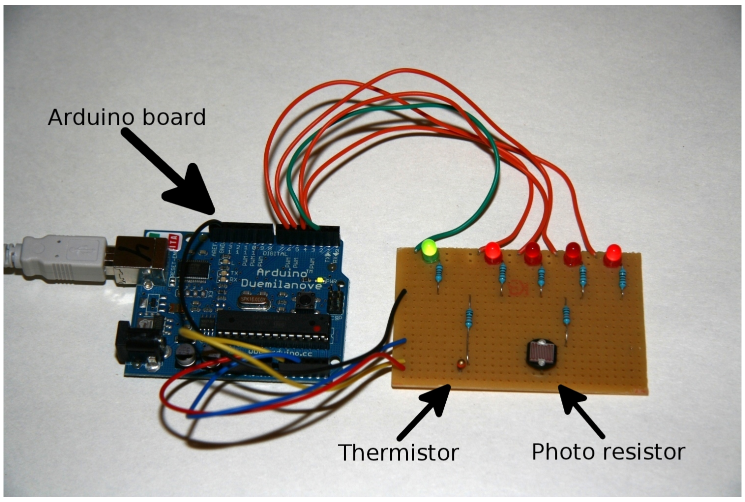


**Figure 3.7:** *The trusted sensor prototype. A sensor interface board is connected to the Arduino experimentation board. USB cable connects the Arduino board to a laptop computer (not shown) acting as a host platform.*

the peripherals necessary for running the CPU. Additionally, a USB-to-serial converter is provided, allowing the board to be connected directly to a host computer for programming and data delivery.

The security properties of the TSense system depend on the strength of the implemented cryptographic primitives and protocols. We use the industry-standard symmetric AES block cipher (Daemen & Rijmen, 2000; FIPS, 2001) with 128-bit keys (shared pairwise between individual sensors and the authentication service) in CBC-mode (Dworkin, 2001) for encryption. The CMAC (Dworkin, 2005) algorithm, based on the AES block cipher, is used for authentication. Hence, only a single cryptographic base primitive needed to be implemented, conserving program memory space on the ATmega processor.

We designed the cryptographic library to be re-usable on the various system components. Concretely, the same implementation of AES in CBC mode and CMAC compiles for the ATmega328 on the `tsensor` and for 32- and 64-bit Intel/AMD platforms for the sink- and authentication servers under Linux and OS-X/BSD. A version of the cryptographic library for the Arduino platform is available as an independent open-source project[4].

The `tsensor` prototype is written in about 1400 lines of code (SLOC) in a C++ variant developed for the Arduino platform. The cryptographic library, which is shared between the `tsensor` and server platforms, accounts for estimated 880 SLOC thereof. The compiled `tsensor` binary is 14.5KB and the executing code uses approximately 900 bytes of RAM, including measurement and processing buffers.

[4] `https://github.com/kristjanvj/ACrypto`

#### 3.5.1.2 `tsclient` – Untrusted Client-side Agent

The *observation node* that hosts the `tsensor`, is the untrusted and corruptible entity in our system model. In our prototype, the observation node is a networked laptop computer, hosting an USB-connected `tsensor`. A small script, `tsclient`, is required to provide the untrusted client with the ability to interface with a hosted `tsensor` and communicate with the sink server.

`tsclient` is written in Python (v. 2.6), using the serial and socket libraries. No attempt is made to secure the script, which is readable and modifiable by any process and user having access to the hosting node. This is consistent with the system security goals: no tampering by the adversary, including modifying or replacing `tsclient`, should reduce the overall security of the system.

#### 3.5.1.3 Trusted Infrastructure

The trusted infrastructure (see Figure 3.5) in our prototype consists of two Unix daemon services, a singe collector (sink) server (`tssinkd`) and an authentication server (`tsauthd`). The collector daemon acts as an authentication gateway for the population of clients in addition to collecting and recording submitted measurements. The authenticator authenticates clients (or rather their hosted `tsensors`) and generates key material for session keys. Our prototype system consists of two virtual machines running Ubuntu Linux 10.04.1 LTS (2.6.32-24). The trusted servers communicate securely over a per-request TLS tunnel, implemented using OpenSSL[5], with mutual authentication via x.509 certificates.

Both daemons are written in C++. Forking is used to service incoming requests on a per client basis. A persistence layer, based on MySQL, is used to store per-sensor parameters, such as current session and encryption keys. The cryptographic library used in the server code is the same as used in the `tsensor` prototype.

No attempt was made to secure the trusted infrastructure components in the prototype. Future implementations should address security issues, such as buffer overflow vulnerabilities. For the time being, the two servers considered implicitly trusted and invulnerable in accordance with the adversarial model.

[5] http://openssl.org

### 3.5.2 Performance of cryptographic primitives

#### 3.5.2.1 AES Encryption and Decryption in CBC-Mode on Arduino

The precise timing instruction `micros`[6] was used to time successive encryption and decryption operations on randomly generated data of one to several blocks in length, as shown in Table 3.1. An Arduino Duemilanovae board with an ATmega328 running at 16MHz was used for the test. Briefly, the results were that our implementation achieves an average throughput of 1213 blocks/sec for encryption and 592 blocks/sec for decryption on the Arduino. The performance difference between encryption and decryption operations is most likely due to the more complex AES `mixColumns` transformations required for decryption and can be addressed by future optimizations. Given the expected small demands in terms of messaging frequency and size on individual sensors, we conclude that the performance of the AES cipher on the Arduino platform is sufficient to deliver a reasonable rate of updates securely. Future optimizations of the protocol, specifically for the target platform, may yield higher throughput. AES may also be efficiently implemented in hardware if further performance enhancements are required.

**Comparison with the XTEA block cipher.** We implemented the XTEA (Needham & Wheeler, 1997) block cipher on the Arduino in order to compare its performance with the more complex AES-128. XTEA is designed to be an efficient and secure block cipher on resource constrained systems. The XTEA implementation on the Arduino is done in a total of 16 lines of C++ code. The test was performed in the same manner as that for our AES implementation. The results were a throughput of 1316 blocks/sec for encryption and 1574 blocks/sec for decryption. The recommended 32 rounds were used for both operations. We conclude that our AES implementation compares favorably with the XTEA one, given that the XTEA test operated in the simpler ECB mode and has a shorter block length of 64 bits. Accounting for the differing block length shows acceptable throughput for our AES implementation, as shown in Table 3.1.

| | **Encrypt** | | **Decrypt** | |
|---|---|---|---|---|
| | *(blocks/sec)* | *(Kbytes/sec)* | *(blocks/sec)* | *(Kbytes/sec)* |
| **AES** | 1213 | 19.0 | 592 | 9.3 |
| **XTEA** | 1316 | 10.3 | 1574 | 12.3 |

**Table 3.1:** *Comparison of AES and XTEA performance on the Arduino platform (ATmega328 at 16MHz). Note that AES has a block length of 16 bytes compared to the shorter 8 byte block length of XTEA.*

[6] `http://arduino.cc/en/Reference/Micros`

#### 3.5.2.2 AES Encryption and Decryption in CBC-Mode on Intel-based Platforms

The same cryptographic library code as timed on the Arduino platform was compiled in a test framework on a 32-bit Intel laptop with a Centrino Dual CPU, each core running at 1GHz, with 4MB L2 cache; only one core of the CPU was used in the test. The test machine ran Ubuntu Linux 9.10 (kernel 2.6.31-22). Chunks of random data, from 1 to 256 blocks at 16 bytes each, were generated, encrypted and decrypted. The encryption and decryption operations were timed using the C `gettimeofday` function, each measurement averaged over several repetitions. A separate test using the Unix `time`[7] utility was also conducted. An average throughput of 2.7 Mbytes/sec was observed in the test framework.

Our test servers achieved an average throughput of 4.3 MBytes/sec running the same test. The test servers are 32-bit virtual machines running Ubuntu Linux 10.04.1 LTS (2.6.32-24). The single CPU runs at 1995 MHz and has a 4MB L2 cache. The performance of the collectors versus the potential throughput of sensors indicates that a collector in the current implementation can service at least a hundred continuously transmitting sensors. Sensors are more likely to produce updates in intermittent short messages, implying that the practical ratio of sensors to a collector are several hundred.

The AES implementation used in this project is byte oriented and close to the original published pseudo code (FIPS, 2001). Using a more efficient 32-bit table-based algorithm (Daemen & Rijmen, 1999) would be expected to perform considerably better. A comparison of byte-oriented versus table-based AES implementations indicates that the performance increase can be at least fourfold.

### 3.5.3 *tsensor* Memory Footprint

Code size is quite important on a small device, such as the ATmega processor. Our compiled `tsensor` code binary is currently 14.5 KB of the total available program memory (flash) of 32KB. The AES test code binary, used for the timing tests in Section 3.5.2.1, is approximately 11 KB, while the XTEA test code binary is approximately 7 KB. Most of the difference in binary size between the two tests is due to the more complex AES algorithm. However, a size difference of less than factor two is quite acceptable, in our opinion, for the stronger AES block cipher. Note also that our full `tsensor` implementation with the AES block cipher currently occupies less than 50% of the available program memory of the ATmega328.

[7] `http://unixhelp.ed.ac.uk/CGI/man-cgi?time`

RAM memory usage is critical on a resource constrained device. Unfortunately, there are currently no methods to accurately measure the RAM use in the current Arduino library. However, we can estimate the amount of free RAM by the difference between the heap and stack pointers. A serial API command for reporting the free RAM is included in the current `tsensor` prototype version. At start-up, the free RAM is reported as 1638 bytes, some of which is used by the controller itself and the Arduino boot loader. After full initialization, key expansion and allocation of a 20 byte measurement buffer, the available RAM is reported as 734 bytes. RAM use can be improved, for example by expanding keys only as needed – AES-128 uses a key schedule of ten 16-byte keys, derived from the original 16 byte one. The current prototype caches all keys in an expanded form, which requires 320 bytes of RAM. Expanding the seldom used session key on demand would therefore free up considerable RAM.

### 3.5.4 Cost of Materials and Production Sensor Size

We base our sensor prototype on a typical general purpose microprocessor, the Atmel ATmega series, commonly used in embedded applications. Our prototype is essentially a software implementation on a ready-made experimentation board, the Arduino Duemilanovae. The cost of the Duemilanovae board was $29.95 (+ shipping and import costs) from `http://www.sparkfun.com` in September 2010. An Atmel ATmega328 was available at that time for $4.40 in quantities of 100+ from Sparkfun. The cost of sensors varies according to their sophistications, but the NTC thermistor and photo-resistor used in our project cost less than $2 a piece in individual units.

Production sensors would use a much smaller custom PCB and a surface mount version of the processor. Estimated size for a production unit is less than 2x2 cm, using a surface mount ATmega328 package. Even smaller sizes can be achieved using custom ICs, as demonstrated by the tiny but sophisticated processors found on current SmartCards (Rankl & Effing, 2001; Finkenzeller, 1999). The size of the final package depends on the type of sensor and the hardening to be applied, but we can conclude that it is certainly practical to manufacture a reasonably small fully enclosed trusted sensor. Further, the production cost would be expected to be dominated by the sensing element and packaging. Tamper-resistant packaging will add to the cost of a trusted sensor device but can be expected to be on the order of the cost of manufacture of environmentally sealed sensors.

### 3.5.5 Security

#### 3.5.5.1 Physical Security

We do not address physical security in the prototype, as can readily be seen from Figure 3.3. The non-trivial issues regarding tamper-resistance (R. Anderson & Kuhn, 1996, 1997) as well as active and passive probing of the device interface (Kursawe et al., 2005) are reserved for future work.

#### 3.5.5.2 Cryptographic Primitives.

We picked strong and currently industry-standard cryptographic primitives for the construction of our protocols, which are based on known protocols whose properties have been analyzed, both analytically and cryptanalytically, for a number of years. However, the strength of the cryptographic protocols as such was not the main objective of this prototype; rather, our intent was to construct a proof-of-concept – a complete working system. A production version of the proposed system would require more thorough development of protocols, implementation and cryptanalysis.

AES-128 is at first glance the weakest of the three official AES algorithms as it has the shortest key length (128 bits) and the fewest number of rounds (ten). However, there are currently no known attacks against AES-128 that achieve significantly better performance than an exhaustive key search (Biryukov et al., 2009; Schneier, 2009). All variants of AES are listed as unbroken on Cryptolounge[8]. Attacks against reduced round variants of AES exist, such as the super SBOX attack (Gilbert & Peyrin, 2009) against 8-round AES-128. Biryukov et al. (2009) present a number of practical (in terms of complexity) attacks against 9- and 10-round AES-256, the variant with the longest key. However, these attacks are nowhere close to breaking full strength AES. Ferguson et al. (2000) also present cryptanalysis and attacks against reduced rounds AES variants and raise concerns regarding weaknesses in the key schedule of the supposedly strongest variant, AES-256.

The CMAC authentication primitive is apparently still quite strong and its status is unbroken on Cryptolounge, although the analysis is not quite up to date[9]. The strength of CMAC depends largely on the underlying block cipher, in our case AES-128. It is reasonable to assume that CMAC remains strong as long as AES-128 does so.

[8] `http://www.cryptolounge.org/wiki/AES`
[9] `http://www.cryptolounge.org/wiki/CMAC`

We have also designed our protocols for cipher independence: even if one of the primitives would be found to be insecure, another block cipher or MAC primitive can be substituted. Increasing the security margin may be as simple as increasing the number of rounds for the block cipher.

#### 3.5.5.3 Side-channel Attacks

Hardware and implementation related weaknesses, also known under the umbrella term of *side-channel attacks*, are often exploited as a back door for attacking a secure cryptographic algorithm. We did not consider side-band attacks of any kind in our prototype implementation. Nevertheless, this is an important topic for any secure device, such as our tsensor, and will be discussed briefly.

Cache timing attacks (Bonneau & Mironov, 2006) are an example of side-channel attacks, in which timing of the execution of the cryptographic operation is used to deduce parts of the key. Such attacks have e.g. been proven effective against some AES implementations. Cache timing attacks are defeated by implementations which have guaranteed constant running time for all operations. A naive method of achieving this is to add delays or idle loops in the implementations of cryptographic algorithms, guaranteeing nearly constant execution time on any input and key. A better approach is to use bit-slicing, that is, implement cryptographic operations in a bit-parallel manner, which produces nearly constant running time implementations of cryptographic algorithms (Scheibelhofer, 2007).

Differential power analysis – passively monitoring and analyzing the power consumption of hardware during cryptographic operations – is another effective side-channel attack (Kocher et al., 1999; McEvoy et al., 2007). This attack is particularly relevant to tamper-proof devices, since it may reveal bits of information to a passive attacker. An effective defense against simple power analysis attacks is to produce code or hardware that guarantees nearly constant power consumption. The more powerful differential power analysis methods require more sophisticated means, such as introducing noise or non-linearity into the cryptographic operations (Kocher et al., 1999).

An attack can be mounted by exploiting known properties of the padding applied to block ciphers, that is, the bytes that are added to fill the last block. A *Padding Oracle Attack* against block ciphers in CBC-mode is described by Vaudenay (2002) and developed further for attacks against cryptographic hardware by Bardou et al. (2012). This is a potential side-channel attack that must be investigated further for production versions of trusted devices. It is based on the predictability of padding bits, e.g. as standardized in RFC 5652 (Housley, 2009), and harvesting of error messages caused by deliberately in-correct padding. An efficient attack can be mounted, given access to a *padding oracle* that returns true only if the padding of a message is valid. In particular, the attack applies to devices, such as smart cards, that are literally in the hands of the adversary.

## 3.6 Concluding Remarks

We discussed the problem of trusted sensing in a single aggregator setting, in which the sensing nodes can be corrupt. We proposed a solution based on the principles of trusted systems, consisting of tamper-proof trusted sensor modules, a set of protocols and trusted infrastructure components. Together, these components form a trusted client/server-based measurement network that in turn guarantee (within computational bounds) safe data delivery from the tamper-proof sensor to a trusted collector.

We described an open-source proof-of-concept prototype implementation of our system, based on symmetric cryptographic primitives and a trusted third party authentication service. Our cryptographic code is cross-platform, compiling on both Arduino and Linux platforms and has been released into the public domain as an Arduino library. Although our prototype is limited in several ways, we find it to be a reasonable stepping-stone towards the implementation of a trusted sensor. Possible future work includes construction of a tamper-resistant prototype. Miniaturization may be achieved by using a custom security chip akin to SmartCard or RFID processors, which are currently up to the task in terms of size, cost and processing power. Further developments also include development of a version based on asymmetric cryptographic primitives in an effort to decrease the reliance on trusted infrastructure services.

# Chapter 4

# Trusted Aggregation

Aggregation systems that perform function evaluation *in-network* (see Section 2.3) are an important topic of research because of their inherent scalability. Hence, we need to extend the trusted centralized model from Chapter 3 to a fully distributed one, which is the topic of the present chapter. Guaranteeing integrity in a hierarchical aggregation model where any node can be corrupted is a hard but essential task, as the integrity of aggregate data is of primary importance in such a system. We show that this problem can be solved, provided that the data source integrity guarantees presented in the previous chapter hold. A relatively straight-forward, yet low impact, approach to this goal is to extend the trusted sensor concept to a trusted aggregation module, in which the essential functionality of correct function evaluation is protected. We consider this solution in this chapter and present an extension of the system design from Chapter 3.

The material presented in this chapter has been previously published in part by Rúnarsson, Kristinsson, and Jónsson (2010), Jónsson and Vigfússon (2011), Jónsson, Palmskog, and Vigfússon (2012), Jónsson and Vigfússon (2012a) and Jónsson and Vigfússon (2012b).

## 4.1 Implications of Corrupt Aggregators

Let us begin by extending our analysis from the previous chapter, allowing for the possibility of aggregator node corruption. We begin by considering the impact of aggregator node corruption in a single aggregator model before extending the discussion to a hierarchical one.

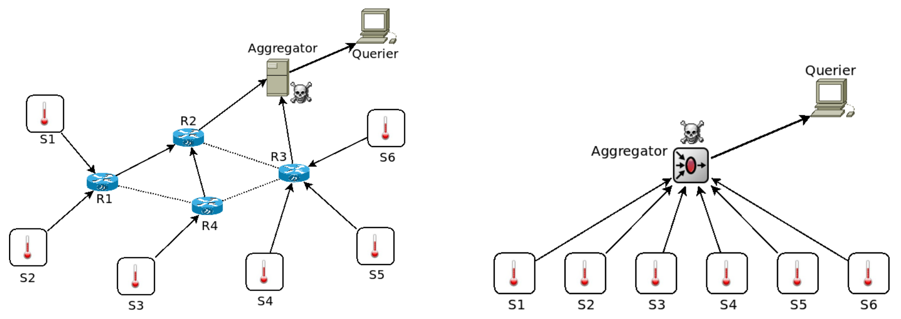


**Figure 4.1:** *Single corruptible aggregator system. Aggregator corruption is indicated by scull-and-crossbones. A single corruptible aggregator node collects and processes contributions from a system of sensing nodes before submitting the result to an inherently trusted querier. Figure (b) shows the aggregation overlay, excluding outsiders.*

### 4.1.1 Single Corruptible Aggregator Model

We previously formulated the single (honest) aggregator model in Definition 3.5. In this model, the corruptible parties are the data providers, while the aggregator was considered inherently trusted by virtue of being the consumer of the produced aggregate data. Let us begin by extending this model to a *single corruptible aggregator* model in Definition 4.1. As shown in Figure 4.1, the network consists of several sensing nodes (data providers) that feed update messages to a singe aggregator, which in turn executes an aggregation function and sends the resulting updates to a querier. A similar model is commonly encountered in the sensor networks literature, e.g. by Perrig et al. (2007), who define a *base station* in the sensor network as the vulnerable aggregator and a remote invulnerable *home server* as the trusted party (querier in our model).

---

**Definition 4.1 (Single corruptible aggregator model).**
For a communications graph $G = (\boldsymbol{V}, \boldsymbol{E})$, we define an *aggregation overlay* $G' = (\boldsymbol{V}', \boldsymbol{E}')$ spanning a single connected component of $G$. The aggregation overlay is composed of the nodes $\{q, a\} \cup \boldsymbol{S}$, which we collectively call insiders, that is, formal members of the system. The members are a set of *data providers* $\boldsymbol{S}$, a single *aggregator* $a$ and a single *querier* $q$. The querier $q$ is considered inherently trusted by virtue of being the consumer of the information produced. However, the aggregator $a$ is considered corruptible and can deviate arbitrarily from the protocol.

For each $s \in \mathbf{S}$, we have paths $(s, a)$ and $(a, q)$, perhaps spanning several nodes in the set of outsiders, $\mathbf{V} \setminus \mathbf{V}'$. Outsiders are not expected to participate in the protocol in any role other than routing.

---

#### 4.1.1.1 Implications of Aggregator Corruption

We previously discussed the impact of untrusted data sources on the aggregate integrity in terms of the resiliency of aggregation functions and found the bias that even a single corrupt data source can introduce to be unacceptable (see Section 3.2.3). In the case of aggregator corruption under the model of Definition 4.1, the adversary has an advantage due to the fact that a single node compromise yields control over the entire aggregation network. Let us consider the relative power that the adversary can wield under the two models in the case of the AVERAGE function by means of Examples 3.6 and 4.2.

**Example 4.2 (Resiliency of AVERAGE – corrupt aggregator).**
An average is computed as $y = f(x_1, \dots, x_n) = (x_1 + \cdots + x_n)/n$, where $n$ is the number of observations provided, under a single corrupt aggregator model. We assume all data providers honestly contribute their local inputs to the aggregator. However, the aggregator biases some some $k$ contributions by $\sigma$ as $x_i^* = x_i + \sigma$:

$$y^* = (x_1 + \cdots + \{x_i^* + \dots, x_{i+k}^*\} + \cdots + x_n)/n = y + k\sigma/n$$

is computed. Hence the implication of allowing the adversary to choose $\sigma$ freely is a bias of $\delta = k\sigma/n$ introduced in the aggregate computation.

Intuitively, the adversary that is able to corrupt aggregators appears more powerful. However, that is not necessarily the case. Comparing Examples 3.6 and 4.2, we note that the two adversaries are of comparable strength when considering inherently insecure aggregation functions, such as AVERAGE. The adversary that is only capable of corrupting data providers can given some expenditure of work corrupt $k$ sensors. The adversary that is able to corrupt an aggregator can match the effects of $k$ colluding corrupt sensors by just one compromise. However, the latter is no stronger in the case of inherently insecure functions, since the former is capable of introducing arbitrary bias via even a single corrupt node. Considering more robust functions, such as COUNT, we see that the adversary that can corrupt aggregators is more powerful. In the honest aggregator case, COUNT can be secured by enforcing an input of zero or one from each contributor. However, once the aggregator has been corrupted, we can make no such assumptions. Hence, we must conclude that the adversary that can corrupt aggregators is more powerful, both in terms of the amount of work required as well as the achievable bias.

#### 4.1.1.2 Securing the Single Aggregator Model

Let us now consider means of securing the aggregate computation or at least bounding the impact an adversary can have in the single corruptible aggregator model.

**Message authentication.** The assumption of data provided by a trusted sensor gives the querier in Chapter 3 the ability to assess the authenticity of individual contributions prior to aggregate computation. However, source verifiability of this sort under the single corrupt aggregator model is only achievable by simple means if we limit the functionality of the aggregator to routing or computing the PACK function (see Example 2.4). In the case of PACK, the querier can employ a MAC or digital signature scheme with pre-shared keys to verify the correctness of inputs, as shown in Example 4.3. However, the functionality of the aggregator is severely restricted in this case. Enabling similar verifiability over a more complex aggregate computed by an untrusted aggregator may be achieved by some form of MAC or signature aggregation (Boneh et al., 2003; Mykletun et al., 2004; A.-F. Chan & Castelluccia, 2008; Katz & Lindell, 2008). However, to the best of our knowledge, such schemes involve considerable message expansion and are not applicable to the general aggregation case considered in this dissertation.

**Example 4.3 (Aggregation over authenticated inputs).**
A set of honest data providers $s_1, \ldots, s_k$ release update messages $m_1, \ldots, m_k$ synchronously in round 1. Messages are of the form $m_i = \langle i, I_i, \mathcal{T}_i(I_i) \rangle$, where $i$ is an unique data provider identifier, $I_i$ is the contributed local input and $\tau_i = \mathcal{T}_K(I_i)$ is a MAC computed over $I_i$ using a key $K$ shared between data provider $s_i$ and querier $q$. The aggregator submits $m_a = \text{PACK}(m_1, \ldots, m_k)$ to the querier $q$ in round 2. Each contribution in the aggregate received by $q$ can be attributed to a particular contributor. Hence, the recipient can verify the overall correctness of the aggregate.

**Range limitation and input validation.** The set of solutions suggested by Wagner (2004) in the single (honest) aggregator model apply to some extent in the corrupt aggregator case. Consider for instance truncation. We can impose bounds on the allowable range of values that local inputs can take and, hence, apply equivalent bounds on the output. However, truncation only succeeds in bounding the adversarial influence and is contingent on $|\boldsymbol{S}|$ being known, as an adversary that is able to lie about the system size can introduce arbitrary bias, despite input range limitations. In practical terms, acceptable limitations of adversarial influence, even in the known $|\boldsymbol{S}|$ case, will affect the dynamic range of the local inputs, which may be unacceptable for many applications.

**Eliminating equivocation.** The ability to provide an output inconsistent with the corresponding inputs lies at the heart of the corrupt aggregator problem. This behavior is also known as *equivocation* (Levin et al., 2009) – the act of providing conflicting information to peers, as shown in Example 4.4. Eliminating the adversaries' ability to equivocate, that is, force aggregators to produce an output consistent with a set of inputs and a particular aggregation function, is not a trivial task.

Let us first consider how the ability to equivocate can be eliminated or at least reduced from the perspective of the querier $q$. In order to remove the potential for equivocation completely, the querier must be able to view all inputs and outputs, which trivially negates the purpose of an intermediary corruptible aggregator. This is analogous to the case considered in Example 4.3. Probabilistic guarantees can be achieved using protocols that check the consistency of an output for a subset of inputs. For instance, Perrig et al. (2007) propose an interactive proof protocol which allows the querier to probabilistically verify the actions of the aggregator with adjustable efficiency versus accuracy trade-off. However, this protocol still requires knowledge of the contributing population, which may not be an option in dynamic distributed systems. The problem becomes even more difficult if both data source and aggregation nodes can be corrupt.

We can also consider non-equivocation from the perspective of data providers. Empowering individual data contributors under the single corruptible aggregator model with the ability to check the aggregator output against their contribution eliminates the adversaries ability to stealthily equivocate. Schemes based on exploiting opportunistic overhearing can achieve this goal to some degree (Bekara et al., 2007). However, such schemes depend on being able to overhear messages, an assumption that we try to avoid, and unavoidably incur considerable per-node work in terms of storage, computing and communications resources. Furthermore, an absolute guarantee of non-equivocation requires knowledge of all messages in the aggregators neighborhood to be available to all contributors, which is an impractical proposition.

**Example 4.4 (Equivocation).**
A single corrupt aggregator $\hat{a}$ receives inputs $\boldsymbol{I} = \{I_1, I_2, I_3\}$ from nodes $\boldsymbol{S} = \{s_1, s_2, s_3\}$, all assumed to be honest. A synchronous protocol is assumed in which all inputs are released simultaneously in round 1. The protocol dictates that a function $y = f(I_1, I_2, I_3)$ should be computed at the end of round 1 and an update message $m = \langle y \rangle$ sent to the querier in round 2. Instead, the corrupt aggregator provides an output $m^* = \langle y^* \rangle$, which is inconsistent with the set of inputs, for instance $y = f(\emptyset)$ or $y = f(c \cdot I_1)$, where $c$ is some scalar constant.

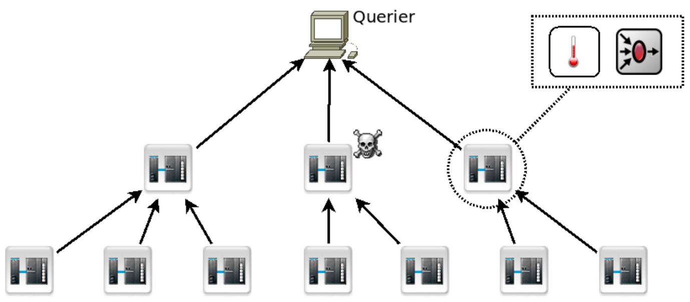


**Figure 4.2:** *Hierarchical aggregation over a spanning tree overlay. A distributed function is computed by a convergecast originated by data providers and culminating with an aggregate being computed by the querier (root of the spanning tree). Intermediary nodes that serve as aggregators can provide local inputs, as indicated in the figure by the symbolic representation of sensing and aggregation functionalities.*

Let us assume nodes $s \in \boldsymbol{S}$ have knowledge of each other's inputs but not of $m^*$ produced by $\hat{a}$. We amend the protocol and require $\hat{a}$ to broadcast the update $m^*$ in an effort to enable $s \in \boldsymbol{S}$ to verify their contributions. The corrupt $\hat{a}$ responds by sending $m = \langle y \rangle$ to $s \in S$, while still sending $m^* = \langle y^* \rangle$ to the querier.

## 4.1.2 Hierarchical Aggregation Model with Corruptible Aggregators

Let us now extend the discussion to a distributed aggregation model in which a number of aggregators cooperatively compute the aggregate *in-network* by the repeated application of an aggregation function. We limit the present discussion to a hierarchical model in which interior nodes in a spanning-tree aggregation overlay compute the aggregation function, as described in Section 2.3.3. An aggregation network of this type is shown in Figure 4.2.

We define a tree-based hierarchical model more precisely in Definition 4.5, in essence applying Definition 4.1 repeatedly over a spanning-tree graph in which data providers are leafs and interior nodes are aggregators. Note that the model allows for a homogeneous population of nodes in which each node can function both as a data provider and aggregator, in which case we define the local inputs of nodes as virtual leafs, while the nodes themselves are defined as interior vertices in the tree. A similar tree-based model is defined by H. Chan et al. (2006) in the context of static network graphs over wireless sensor networks. This model can be easily adapted to heterogeneous models, such as the one depicted in Figure 2.3.

---

**Definition 4.5 (Hierarchical corruptible aggregator model).**
Let us extend Definition 4.1 and define an overlay $T' = (\boldsymbol{V}', \boldsymbol{E}')$ on the graph $G$. The graph $T'$ is a spanning tree over insiders $\boldsymbol{V}' = \{q\} \cup \boldsymbol{A} \cup \boldsymbol{S}$ that belong to a single connected component of $G$. Let us call $T'$ the *aggregation tree*. The local inputs in $\boldsymbol{S}$ are the leafs of $T'$, while interior vertices are aggregators in $\boldsymbol{A}$. All nodes in $\boldsymbol{V}'$, except $q$, are considered vulnerable to node compromise.

---

#### 4.1.2.1 Adversarial Model

As before, the primary adversarial goal is to induce the querier to accept biased aggregate results. The adversary gains influence by corrupting one or more insiders. We initially assume the corruption to be limited to a subset of aggregators. The querier is considered implicitly honest as the originator of queries and the sole consumer of aggregate results.

Recall our earlier discussion on the power of the insider adversary: briefly, a corrupt member of the system is in complete control of all node services and can manipulate the protocol to further the adversarial goals. A Byzantine failure model is assumed, enabling corrupt nodes to arbitrarily deviate from the protocol. Pairwise cryptographic channels offer no protection in the case of insider corruption. We assume the adversary is stealthy (Perrig et al., 2007), that is, manipulates the protocol to produce maximum bias, while at the same time evading detection. Hence, “noisy” attacks, such as denial-of-service and other availability attacks, are disregarded.

#### 4.1.2.2 Implications of Aggregator Corruption

Let us now extend our earlier discussion on the power of the adversary in the single corrupt aggregator model to the hierarchical one. Of the two, the adversary in the single aggregator model is more powerful, in the sense that it achieves the position to introduce arbitrary bias by just a single node compromise. In contrast, the adversary in the hierarchical model has to corrupt several strategically placed nodes in order to achieve the same power. However, both adversaries are in a position to arbitrarily influence inherently insecure (Wagner, 2004) aggregation functions with just a single node compromise.

The adversary in the hierarchical case controls a subset the aggregator population with a single compromise. The extent of his influence is the entire subtree of the corrupt

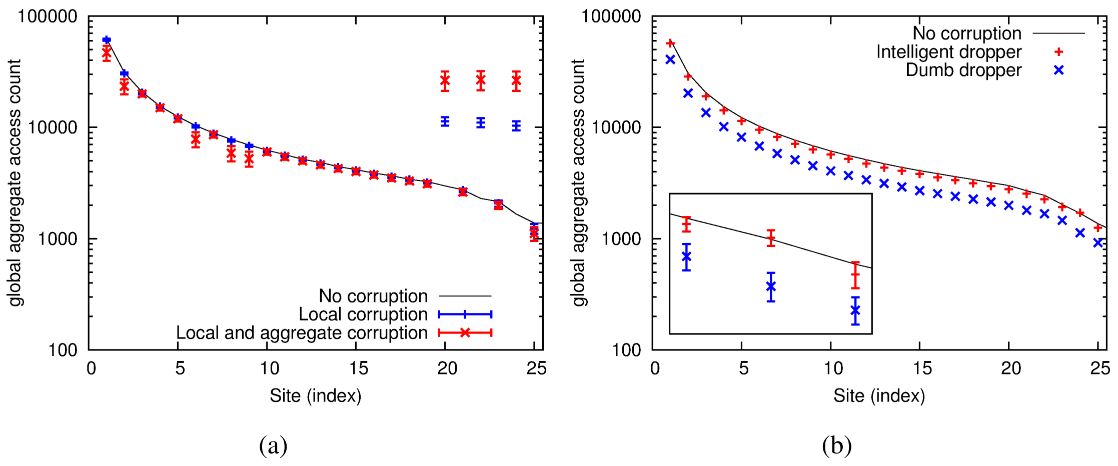


**Figure 4.3:** *Web site popularity assessed by distributed in-network aggregation over an $n$-ary tree with $\Delta = 3$, $h = 3$. Expected 15% of nodes are corrupted at the beginning of the simulation. (a) Unrestricted adversary, error bars show 95% confidence intervals. (b) Adversary restricted to drop attacks (error bars omitted for clarity). Insert shows expanded view centered around the measurement for site 24.*

aggregator. We can say that the adversary wields power inversely proportional to the level of the corrupt aggregator in the aggregation tree, as shown in Example 4.6. Note that the power of the adversary in the example is equivalent for all three cases if $\sigma$ can be chosen freely. However, range limitation and input validation can limit the power of the hierarchical adversary in proportion to its position. Corruption of more aggregators increases the influence of the adversary to a larger subgraph, which we may view as a forest of corrupt aggregation trees.

**Example 4.6 (Resiliency of AVERAGE – hierarchical corrupt aggregator).** Consider an aggregation tree which is a perfect binary tree of height $h = 5$. An aggregator at $l = 1$ is corrupted, resulting in the adversary controlling a binary subtree of height $h = 4$, 31 nodes. Matching the bias introduced in the case of a single corrupt aggregator (see Example 4.2) requires $\sigma' = 62/31\sigma = 2\sigma$. Now, consider the case of a single compromised aggregator at $l = 4$ that controls a binary subtree of height $h = 1$, 3 nodes. In this case, the adversary needs $\sigma' = 62/3\sigma$ to match the influence wielded in the single aggregator case.

**The effects of the unrestricted adversary.** Quantifying the effects of the unrestricted adversary is by definition impossible, but let us consider a small case study that demonstrates the hazards of ignoring this threat (Jónsson, Palmskog, & Vigfússon, 2012). The TOP-K aggregation function (see Example 3.7) is trivially vulnerable to re-ordering and removal of contributions by corrupt aggregators, as demonstrated in the following experiment.

Let us consider a hypothetical distributed system that assesses the global popularity of Web sites by aggregating over user domains (counting outgoing requests). Each contributing domain outputs the current weighted ranking of the local aggregate of outgoing requests. This local aggregate is the local input into a distributed aggregation function, whose local component is the TOP–K–WEIGHTED function. This method can be considered as an alternative to the currently predominant privacy-invasive practice of using browser-based trackers and tracking services (Krishnamurthy & Wills, 2006). A similar distributed query operation is discussed by Palmskog et al. (2010).

In the simulation scenario, $N$ nodes, representing Web portals, are initially created and assigned randomly generated web site usage statistics over some $m = \mathcal{N}(\mu, \sigma)$ visits. We assume that hits to Web sites follow a power-law distribution, ranking sites in expected descending order according to the probability density function $p(i, \alpha) \propto i^{-\alpha}$, where $i \in \mathbb{N}$ is both the website ID and its expected rank, and $\alpha$ is a parameter of the distribution. Web site popularity rankings, as well as a plethora of other network phenomena, have been shown to follow a power-law distribution (Adamic & Huberman, 2002).

We begin by considering an unrestricted adversary that corrupts a number of nodes in a static a small $n$-ary tree ($\Delta = 3$, $h = 3$). A single one-shot query with $k = 10$ is issued by the root with the objective of obtaining the global aggregate access counts for the top ten Web sites. The local state of each node is initialized at startup of each run using $\lceil \mathcal{N}(10000, 2500) \rceil$ web requests, the target of each being randomly selected according to a power-law distribution with $\alpha = 1.0$. We average 50 independent simulation runs to produce the result.

The objective of the adversary in this experiment is to falsely promote the low ranking sites 22, 24 and 26. An expected $15\%$ of nodes, excluding the root, are corrupted at the start of the simulation and execute a local corruption function, in this case

$$w_i = f_C(\mathbf{a}) = \mathcal{N}(\gamma \cdot w_1, \varrho \cdot w_1)$$

where the weight of a falsely promoted site $i$ is drawn from a normal distribution centered around the weight of the currently highest ranked site. Figure 4.3(a) shows the results for $\gamma = 1$ and $\varrho = 1/3$. In the first experiment, we let the corrupt nodes modify only local observations, while also modifying the aggregate computation in the second one. The unmodified aggregate is shown as a baseline reference and follows a power-law over several orders of magnitude. Observe that the attack on the aggregate computation is the more effective of the two for the chosen parameters, as would be expected. However, arbitrary bias can be introduced by either method due to the inherent insecurity of TOP-K. Note that the querier has no way of disputing the aggregate result. Even statistical comparisons with previous runs must be considered inconclusive since transient effects, such as flash crowds (Barford et al., 2002), might well introduce legitimate bias in current or past results.

We now restrict the capability of corrupt to dropping packets, simulating the introduction of trusted devices, which is the subject of this chapter. Two such cases are shown in Figure 4.3(b). In the first case considered, the adversary directs the corrupt nodes to discard all aggregate updates. In the second case, we assume that the corrupt nodes can view, but not modify, partial aggregates. This enables the corrupt nodes to mount a more intelligent attack. We consider an attack in which the adversary drops all packets other than those containing a measurement for site 24. Neither attack is particularly effective in influencing the ranking, although the intelligent dropper manages to introduce a small bias, as can be seen in the enlarged view. This small experiment serves to demonstrate the effectiveness of restricting the capabilities of the adversary, in this case, forcing corrupt nodes to contribute either none or correct updates.

#### 4.1.2.3 Securing In-Network Aggregation

We have already reviewed a selection of approaches to limit the power of the adversary in the single aggregator case. Let us now continue our discussion of means to secure the aggregator functionality, this time focusing on the distributed model. The task of providing integrity guarantees is no easier in the in-network aggregation case than in the single aggregator one. For instance, in a dynamic network, in which associations can change continuously, an honest aggregator has an even vaguer notion of the number of contributors or the actual configuration of the network at any given time.

The in-network aggregation case can be viewed as a repeated application of a single aggregator. Hence, solutions which work in the single aggregator case do in principle work in the distributed as well, although coordination becomes more difficult. We can consider end-to-end authentication of individual updates. However, the implied limitations on the functionality of the aggregators is contrary to our overall goal of efficient aggregation. Input range limitation and validation can only bound the bias that the adversary can achieve, but not eliminate it. As discussed previously, limiting the range of inputs is bound to negatively impact the dynamic range of the system. Further, measures of this sort are of limited use against an adversary that can falsify the number of contributions.

A more sophisticated form of validation is based on modeling of perceived "normal" inputs, for instance, obtained by observing the system and compiling models of expected input distributions. Models can be applied to enable honest aggregators to eliminate contributions of dishonest data providers by filtering out perceived outliers (Buttyán et al., 2006). However, it is less obvious if this technique can be applied in the case of corrupt aggregators. Furthermore, methods based on removing outliers are counter-productive in the case where we are specifically monitoring for abnormalities, such as forest fires (Yu et al., 2005; Doolin & Sitar, 2005) or radiation spikes (Barbarán et al., 2007; Venere & Gardner, 2008).

The key to solving the problem of the corrupt in-network aggregator is to eliminate its ability to equivocate, which is no less difficult than under the single corrupt aggregator model. To positively identify equivocation by a neighbor, a honest peer needs access to all its inputs and outputs and the ability to independently verify its actions. Even if possible, this scheme would be contrary to the efficiency goals that make distributed aggregation attractive in the first place from an efficiency standpoint. A second problem is the distribution of information in an efficient manner. Overhearing may be exploited in some instances, but more robust protocols are required in the general case. Intuitively, such protocols can easily negate the efficiency gains of distributed aggregation schemes.

H. Chan et al. (2006) describe a protocol, that can guarantee upper bounds on adversarial modifications using only hashes. This protocol is also known as the hash tree (HT) protocol (H. Chan, 2009). Building on a aggregate-commit-prove approach (Przydatek et al., 2003), they describe a scheme in which a distributed Merkle tree is iteratively constructed by transmission of verifiable commitments between nodes. The verifiable commitments are similar to verification objects, as described by Narasimha and Tsudik (2006). They are composed of compact hashes, typically logarithmic in the size of data contributors. The hashes enable nodes to verify the actions of their upstream aggregators. The distribution of verification objects comes at a cost, as expected, but with optimizations incurs a sub-linear $O(\Delta\ log^2 n)$ per-node overhead, later improved to $O(\Delta\ log\ n)$ by Frikken and Dougherty (2008). The security guarantees bound the total adversarial actions of all corrupt aggregators to $S_L \leq S \leq (S_L + \mu r)$, where $S_L$ is the sum of legitimate inputs, $\mu$ is the total number of corrupt aggregators and $r$ is an upper bound on the input range of data providers. Note that the input range is bounded, which affects the dynamic range of the system (Wagner, 2004). The HT-protocol is a promising solution to the problem of bounding adversarial bias in a system of untrusted nodes. However, it is limited in several aspects with regards to our previously stated goals of secure aggregation in the general case. They consider specific scalar aggregation functions, namely ones which can be derived from SUM and their protocol is limited to tree-based aggregation protocols. In contrast, the solutions that we discuss later on are in principle capable of securing arbitrary aggregation protocols and data types. Another drawback of the HT-protocol is that the security guarantees are an all-or-nothing proposition: a trivial stealthy availability attack against the protocol can be mounted by a corrupt node that performs its actions correctly in all aspects apart from flipping a single bit in the verification phase, thereby forcing the querier to discard an entire aggregate. However, this vulnerability can be mitigated by protocols which search out and eliminate misbehaving nodes (Haghani et al., 2007).

*Eventual* detection of equivocation can be guaranteed, for instance by requiring nodes to keep unmodifiable logs (Chun et al., 2007; Levin et al., 2009) of all received inputs and contributed outputs. Given such logs, a global algorithm can be executed by a trusted entity, identifying inconsistencies. However, the work required to guarantee fast detection converges to that of aggregation under the single trusted aggregator model. We briefly explored means of reducing this work by means of auditing a random sample of the node population per protocol round (Jónsson & Dam, 2010). However, the trade-off between speed of detection and the amount of work required by trusted entities is problematic: for fast detection, the work required converges to that of a central trusted aggregation model. On a similar note, accountability systems (Haeberlen et al., 2007; Ho et al., 2008) can be used to increase the reliability of distributed systems. For instance, PeerReview (Haeberlen et al., 2007) guarantees that Byzantine faults are eventually traced to the culprit. However, protocols of this type are prohibitively expensive for the special case of detecting misbehavior in an arbitrary aggregation system.

## 4.2 Hierarchical TSense

In contrast to the previous work, some of which is outlined in the previous sections, we search for strong integrity guarantees on the correctness of the computed aggregate. We limit our work in this chapter to the objective of correctness and postpone discussion of the complimentary objective of completeness until Chapter 5.

The proposed solution is based on the principle of enforcing non-equivocation by means of a system of trusted data sources and trusted aggregators, interconnected via pairwise secure channels. However, in contrast to the approaches outlined in the previous section, the proposed solution imposes low and constant messaging overhead and requires no central coordination. We base our concept on the observation that an aggregator can be viewed as an extension of a trusted data source: a device that receives trusted inputs, performs secure function evaluation and provides trusted output. In fact, the task of enforcing input/output consistency turns out to be easier in some respects, since an aggregator does not have to contend with the problems of process-of-measurement attacks (Trappe et al., 2005).

The population of participating nodes can be *homogeneous* with regards to their participation in the aggregation protocol, meaning that each node is able to contribute local inputs as well as provide partial aggregates based on contributions received from peers. We call the general population *observation nodes* but assume the existence of a trusted infrastructure, consisting of at least a single trusted querier. Observation nodes form a single tree overlay with the querier $q$ as root. The tree spans the connected component of the underlying graph that contains $q$.

We assume the same network and adversarial model as for the preceding discussion, in which the observation nodes are corruptible insiders. We will not delve into availability attacks, such as jamming or denial-of-service, because they contradict the objectives of our stealthy adversary. In the same vein, we disregard attacks against node discovery and routing: we assume that the stealthy adversary executes the protocol correctly in all aspects other than attempting to manipulate the aggregate computation. We restrict this work to network or host-based attacks on the measurement process in the communications path (see Figure 3.1). Hence, process-of-measurement attacks are excluded.

### 4.2.1 Design Objectives

Our goal is to support arbitrary dynamic systems, hence ruling out (in practical terms) corroboration schemes, such as proposed by H. Chan et al. (2006). This also rules out opportunistic schemes, such as those based on overheard radio transmissions. Further, we require our solution to be efficient, ideally imposing constant overhead on top of the underlying aggregation protocol stack. Hence, we are mindful of the complexity imposed by algorithmic approaches which may easily negate the performance gains which make distributed aggregation protocols attractive in the first place.

The assumption of scalar inputs and a limited set of aggregation functions is a common thread in a large body of the previous in-network aggregation work. We find this limiting, as more complex inputs and aggregation functions may be expected in future applications, e.g. involving multi-sensor fusion (Hall & Llinas, 1997). Hence, our goal is to support arbitrary inputs as well as arbitrary aggregation functions.

We focus on tree-based aggregation protocols. However, the ability to choose aggregation protocols, for instance ones based on gossiping (Kempe et al., 2003; Jelasity et al., 2005; Wuhib et al., 2007) or consensus propagation (Moallemi & Van Roy, 2006; Aurell & Pfitzner, 2009) provides the systems designer with greater flexibility. Hence, our design objectives include the ability to support secure aggregation in general, regardless of the underlying aggregation protocol.

#### 4.2.1.1 Security Objectives

*The primary security objective of the system is to guarantee correct aggregation in the sense of Definition 3.3.* Ideally, any system should be invulnerable to adversarial corruption and influence. This goal is difficult to achieve in practice. In the approach described in this chapter, we assume any observation node can be corrupted but rely on trusted systems principles to deny the adversary the ability to forge or modify (i) local inputs and (ii) aggregator inputs and outputs.

As in Chapter 3, the overall security of the system depends on the correctness and physical resiliency of the individual trusted components, as well as the security of communications protocols. However, perfect tamper-resistance is difficult to achieve in practice, so it is prudent to assume the adversary can breach trusted modules, given sufficient time and resources. Hence, we must insist on the property of *graceful degradation*: in the event of a trusted module breach, the level of security provided should be decreased, rather than the entire system failing catastrophically.

### 4.2.2 Essential Functionality

As discussed in Chapter 3, a trivial but generally impractical solution is to harden the entire observation node. Instead, we propose to secure the bare minimum of functionality – the essential functionality of aggregation. This includes the following:

(i) *The provision of correct inputs.* Trusted sensors, as described in Chapter 3, fulfill this objective.

(ii) *Correct function evaluation.* The ability to trust that the inputs are properly represented in partial aggregates is critical to the overall integrity guarantees. Hence, the local part of the distributed aggregation function should be protected. This is the functionality assigned to trusted aggregators, as further described in this chapter.

(iii) *Correct communications.* This includes the ability to verify inputs received from peers as well as to ensure that contributions reach the intended recipients. Hence, the essential functionality must include verifiability of both sender identity and contributed data (local inputs and partial aggregate contributions).

### 4.2.3 Trusted Aggregator

A *trusted sensor* was introduced in Chapter 3 – a construct that solves the integrity problem with regards to correctness in the single honest aggregator model (Definition 3.5). We apply the same principles to the task of securing the action of aggregation, i.e. ensuring correct function evaluation. To do so, we extend Definition 3.8 to a dedicated trusted aggregator module.

---

**Definition 4.7.** A **trusted aggregator** is a trusted module that accepts update messages from trusted peers, i.e. trusted sensors or other trusted aggregators. The aggregator applies an aggregation function over the inputs received during some window of time, producing a new cryptographically unmodifiable output in the form of update messages to trusted peers. The aggregator only accepts inputs from and provides output to peers in the same or overlapping group of trusted devices. Furthermore, only inputs verified as correct are accepted.

---

Implementing the trusted aggregator in a manner analogous to that of the trusted sensor is the simplest and most secure option. In this case, the trusted aggregator is a dedicated hardware device that performs the essential functionality of correct function evaluation (Jónsson & Vigfússon, 2011). Inputs are accepted from any number of trusted contributors (trusted sensors or aggregator peers) whose contributions can be verified. A hard-coded aggregation function is executed over the inputs and a consistent output provided. As in the case of the trusted sensor, the trusted aggregator operates in a symbiotic relationship with the untrusted hosting node (observation node), which is assumed to provide a number of services such as IPC communications and networking services.

A more flexible solution can be envisaged in which a number of aggregation functions are programmed into the aggregator, allowing more flexible queries to be executed. This concept can be extended to a device which can be reprogrammed on-the-fly by trusted parties. Given a mechanism for verifying and distributing trusted code, the aggregation functionality could even be entrusted to a secure execution environment running on the observation node, for instance a trusted hypervisor as described by Gummadi et al. (2009)

and Gehrmann et al. (2011). However, we will confine our discussion in this dissertation to a hardware implementation with a single hard-coded aggregation function.

#### 4.2.3.1 Requirements

The requirements for a trusted aggregator are in essence the same as for a trusted sensor. Refer to Section 3.3 for further details.

**Identity and device authentication.** A trusted aggregator needs to be able to enter into secure relations with peers, both trusted sensors and aggregators. Hence, a unique identity is essential along with the ability to identify peers as members of the same (or overlapping) groups of trust.

**Specification, verification and certification of functionality.** The evaluation of the aggregation function is the core functionality of the trusted aggregator and must be included in the specification and verification procedures. Certification is analogous to that of the trusted sensor.

**Tamper-resistance.** Tamper-resistance is essential in protecting the integrity of the trusted aggregator. Tamper-resistance is a non-trivial but conceptually simple procedure in the case of a dedicated hardware device, analogous to the protection assumed for a trusted sensor. However, protecting the integrity of a virtual construct such as a trusted hypervisor is a more difficult task, but one that is outside the scope of this dissertation.

**Complete mediation and communications security.** The communications facilities of a trusted aggregator module can be as simple as that of a trusted sensor and a rudimentary serial interface is sufficient. As in the trusted sensing case, no inputs should be received from parties other than those which have been verified as belonging to the same (or overlapping) group of trusted devices.

#### 4.2.3.2 Transitive Trust Establishment.

The property of *transitive trust* is central to the overall security guarantees that the trusted sensor adds to a distributed aggregation system. Let us begin by considering an aggregator $a$ that interfaces with a single sensor $s$, both hosted by the same untrusted observation node. The two nodes enter into a trust relationship, which we can indicate as $s \Leftrightarrow a$. The aggregator $a$ in turn enters into a trust relationship with an aggregator $a'$, hosted by a peer observation node, s.t. $a \Leftrightarrow a'$. The secure channels, denoted as $\Leftrightarrow$, indicate that the nodes in question have verified each others membership in a mutual trust group and established the shared keys necessary to communicate securely.

The trusted sensor produces verifiable update messages of the form $m_{s,t} = \langle y_{s,t} \parallel \tau_{s,t} \rangle$, where $\tau_{s,t}$ is an authenticity tag, such as a MAC or digital signature, and $t$ is a timestamp. Assuming a MAC, $\tau_{s,t} = \mathcal{T}_{sa}(y_{s,t})$, using a symmetric cryptographic key $K_{sa}$ shared pairwise between $s$ and $a$. Given a shared key $K_{sa}$, $a$ can accept the received $m'_{s,t} = \langle y'_{s,t} \parallel \tau'_{s,t} \rangle$ as *correct* if $\tau'_{s,t} = \mathcal{T}_{sa}(y'_{s,t})$.

In turn, $a$ produces an output $m_{a,t+1} = \langle y_{a,t+1} \parallel \tau_{a,t+1} \rangle$ (assuming a synchronous protocol), verifiable by a trusted upstream peer $a'$ assuming a shared key. The original authenticity verification tags do not need to be forwarded to $a'$, as it is sufficient for an $a'$ that trusts $a$ to receive its authenticated partial aggregate in order to trust the entire aggregation process up to that point.

We can say that if $a$ trusts $s$ and receives its updates and $a'$ trusts $a$ and receives its updates, then $a'$ trusts $s$ by *transitive extension* of the $(a', a)$ trust relations. This property extends transitively over the trusted aggregation overlay from leafs to root.

More formally, we define a relation $u \mapsto v$ as $v$ trusts $u$ (or $v$ is willing to accept inputs from $u$) and $\boldsymbol{U} \overset{*}{\mapsto} v$ as $v$ trusts all members of a set $\boldsymbol{U}$. The trust relation is transitive by our earlier (informal) argument in terms of correctness: verifiability of source data implies correct inputs and trusted function evaluation implies a correct output. Hence, in terms of aggregate correctness, we can state the transitivity property

$$\text{if } u \mapsto v \text{ and } v \mapsto w \text{ then } u \mapsto w$$

$$\text{if } \boldsymbol{U} \overset{*}{\mapsto} v \text{ and } v \mapsto w \text{ then } \boldsymbol{U} \overset{*}{\mapsto} w$$

In our system model, we can have a one-to-one or many-to-one relation between co-located sensor and aggregator, $s_{i,j} \overset{*}{\mapsto} a_i$ for an observation node $i$ hosting $j$ sensors. Aggregators provide updates to exactly one upstream peer. However, aggregators (by definition) receive inputs from one or more aggregator peers, giving $\boldsymbol{A}' \overset{*}{\mapsto} a$, where $\boldsymbol{A}' \subset \boldsymbol{A}$. The entire aggregation tree can be represented by such relations for all pairwise trust relations. Hence, the trusted aggregation overlay provides transitive trust from leafs (data sources) to the root (querier).

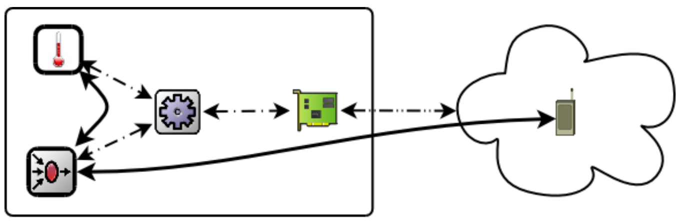


**Figure 4.4:** *Observation node schematic. Trusted modules and channels are shown in bold. Untrusted services utilized by the aggregation process are indicated by the cogwheel and network interface card symbols.*

### 4.2.4 Trusted Observation Node

As stated before, observation nodes are inherently untrustworthy. Examples include commodity PCs or mobile phones operated by individuals in a shared sensing application, unhardened motes in a wireless sensor network and general-purpose PLCs in an industrial control application. We assume the adversary has access to the devices (local or remote) and can influence a number of services critical to the task of secure aggregation. Instead of attempting the difficult and likely futile task of hardening the observation node in its entirety, we propose an approach in which the essential functionality is delegated to trusted modules, trusted sensors (Section 3.3) and trusted aggregators (Section 4.2.3).

A simplified schematic of an observation node is shown in Figure 4.4. A single trusted sensor and an aggregator (indicated by bold outlines) are hosted by an otherwise untrusted observation node. Untrusted platform services used by the aggregation process are indicated by a cogwheel. Such services include peer discovery, routing and storage. As in the TSense design presented in Chapter 3, we assume the existence of an untrusted software agent executing on the observation node that handles overall protocol management, including node discovery and bootstrapping of trust relations. Networking facilities are also considered untrusted, as the adversary must be assumed to have extensive opportunities to compromise the device and even add external redirection and rewriting proxies.

Secure protocols, described in Section 4.2.5, establish secure channels between trusted modules. In Figure 4.4, the trusted sensor and aggregator hosted by the observation node have established a secure *intra-node* channel. Further, the trusted aggregator has established a secure *inter-node* channel to a peer aggregator. Hence, we can view the aggregators as trusted proxies for the observation node in a trusted aggregation overlay, as shown in Figure 4.5, in which trusted sensors are the leafs and trusted aggregators are the interior vertices of a spanning tree. Given that the assumptions of sensor and aggregator integrity hold, we can trust the aggregate results by the property of transitive trust, even though the hosting observation node is fully compromised.

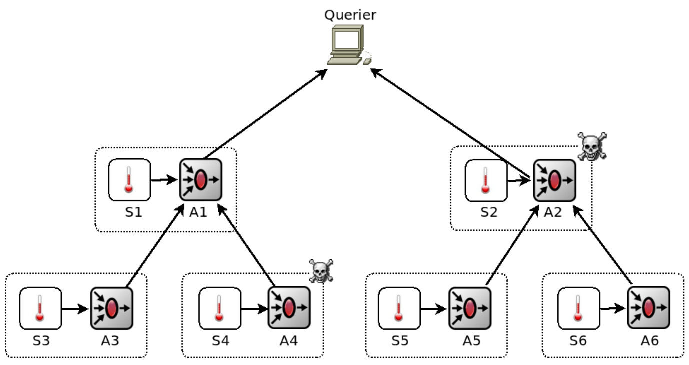


**Figure 4.5:** *Secure aggregation network. Trusted devices, hosted by vulnerable platforms (observation nodes), form a trusted aggregation overlay. Pairwise secure channels between trusted devices form a spanning tree over trusted devices, embedded in the underlying communications graph, in which trusted sensors are leafs and trusted aggregators interior vertices.*

#### 4.2.4.1 Revised Adversarial Model

As in Chapter 3, we can apply the classical Dolev-Yao model, considering the trusted entities (sensors and aggregators) to be trusted, while the adversary is in control of the channels. The trusted overlay in essence relegates the untrusted observation node to the role of an outsider with regards to the sensing and aggregation functionality. The observation nodes are not members of the trusted aggregation overlay, despite hosting trusted modules and initiating their insertion into the system. Hence, we can amend the adversarial model for ease of analysis. Under the revised model, insiders are strictly trusted modules and trust relations are pairwise. The communications services of the hosting platform, the observation node, give the adversary control over the channel, including delivery, delay and reordering of messages. However, pairwise trusted relations prevent the adversary (at least within computational security bounds) from being able to manipulate the message content, that is, the correctness of delivered messages. If encryption is applied in data transfer, the adversary loses the ability to read messages.

### 4.2.5 Protocols

Protocols are the glue that assembles a collection of disjoint trusted modules into a cohesive trusted aggregation system. In this section, we focus on the task of mutual authentication and establishment of the shared keys necessary to guarantee secure communications. Establishment of shared keys is one of the classical problems in secure distributed systems, as discussed in the context of sensor network security by Perrig et al. (2004). A naive key distribution solution, yet one which has been widely employed, is to create one master symmetric secret key $K$ and installing on all members of a network. All nodes which hold $K$ are considered insiders and can communicate securely, that is, until an adversary breaks $K$ or corrupts one of the nodes, at which point the security of the entire system fails catastrophically. A more promising approach is probabilistic key pre-distribution schemes (Eschenauer & Gligor, 2002; H. Chan et al., 2003; D. Liu & Ning, 2003; Du et al., 2005) in which each node holds a sub-set of the total pool of secret symmetric keys, hence, with some probability being able to communicate with any given peer. In such schemes, compromise of a single node reveals only a subset of the total pool of secret keys.

We prefer to avoid distribution of keys beyond the bare minimum. Hence, protocols for entity authentication and key establishment are needed. We consider both a trusted-third party solution, that limits sharing of secrets to exactly two parties (a trusted node and an authentication service), and an asymmetric protocol that requires no sharing of secret keys.

#### 4.2.5.1 Symmetric Authentication Protocol

Symmetric trusted-third-party (TTP) protocols, such as Needham-Schroeder (1978, 1987) use a mutually trusted service to mediate the establishment of trust relations. This is straight-forward in the single aggregator case, as considered in Chapter 3, where the authenticator serving as a TTP was in essence co-located with the single querier. The weak point of TTP systems is the distribution of secret keys, which creates problems in terms of secrecy in transfer, key renewal and key revocation. Further, a TTP service is a single point of failure, which is problematic from a systems reliability standpoint. However, symmetric keys are relatively small and the cryptographic primitives involved in such protocols fast, both of which are positive properties, especially for resource constrained systems. Hence, we begin by considering a symmetric TTP protocol, based on the protocol discussed in Section 3.4.3. As before, we assume each trusted device has a permanent identity tuple $(v, K_{Av})$, consisting of a public identity $v$ and a private symmetric key $K_{Av}$ which is shared only with the authentication server $A$ (the TTP in the protocol). A fresh secret key is generated for each trusted device and can be viewed as a verifiable attestation of authenticity and membership in the group of trusted devices.

A schematic diagram representing the protocol message exchange in the inter-node case (between two aggregators) is shown in Figures 4.6 and 4.7. Two trusted aggregators $a_1$ and $a_2$ are hosted by untrusted observation nodes $S_1$ and $S_2$, respectively. A trusted authentication server $A$ serves as a trusted third party. We assume the existence of an au-

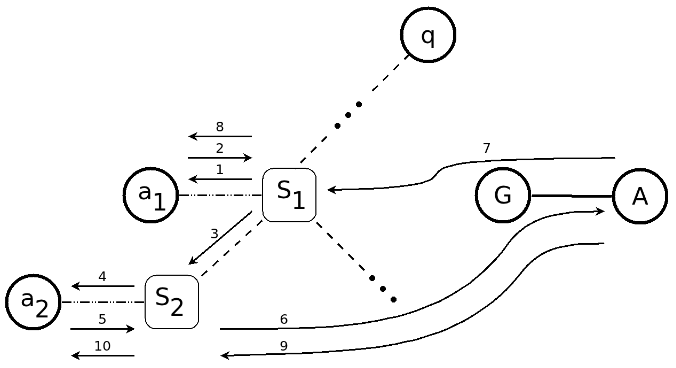


**Figure 4.6:** *A partial aggregation tree, showing querier q and two observation nodes $S_1$ and $S_2$, hosting (respectively) trusted aggregators $a_1$ and $a_2$. Untrusted channels in the aggregation tree are indicated by dotted lines. Authentication infrastructure composed of an gateway G and authenticator A is shown. Reachability of the authentication infrastructure is assumed (by the reachability of q) and channels are omitted for clarity. Messages in the entity authentication and session key establishment phase are shown.*

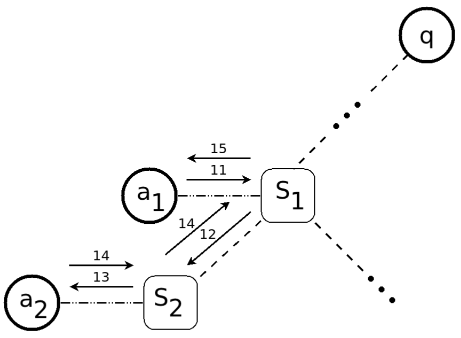


**Figure 4.7:** *A partial aggregation tree. Messages in the key confirmation and transport key exchange protocol are shown.*

thentication gateway $G$ to shield $A$ against direct access. In practice, $G$ can be co-located with $q$ as was the case for the TSense prototype system, described in Chapter 3.

**4.2.5.1.1 Mutual authentication and session key establishment.** The protocol is initiated by an untrusted agent (in our case $S_1$) whose trusted aggregator has already joined the trusted aggregation overlay.

*Initiating event:* $S_1$ discovers $S_2$. If $S_1$ hosts an aggregator which is already part of the trusted overlay, then it initiates the protocol. Otherwise, it caches $S_2$ and waits.

*Message 1:*

$$S_1 \rightarrow a_1 : \langle S_2 \rangle$$

- $S_1$ bootstraps the protocol by notifying its trusted aggregator of the existence of $S_2$. The identity $S_2$ is a public identity, such as a MAC or IP address, of the observation node, not its trusted modules.

*Message 2:*

$$a_1 \rightarrow S_1 : \langle \mathcal{A}_{a_1a_2} \rangle$$

- $a_1$ sets up provisional state for $S_2$, storing the nonce for later reference. The provisional state times out if the protocol remains incomplete after a period of time.
- $a_1$ constructs an authentication token $\mathcal{A}_{a_1a_2} = \mathcal{E}_{Aa_1}(S_2, N_{a_1a_2})$ and sends to its host $S_1$. The key used is the permanent device secret key $K_{Aa_1}$, held only by $a_1$ and $A$, rendering $\mathcal{A}_{a_1a_2}$ unreadable by any party except $a_1$ and $A$.

*Messages 3 and 4:*

$$S_1 \rightarrow S_2 \rightarrow a_2 : \langle \text{INVITE}, S_1, a_1, \mathcal{A}_{a_1a_2} \rangle$$

- $S_1$ sends a connection invite message to $S_2$ which forwards it to its trusted aggregator $a_2$. The newly generated authentication token $\mathcal{A}_{a_1a_2}$ is included.
- $a_2$ sets up a provisional state for $(S_1, a_1)$, storing $\mathcal{A}_{a_1a_2}$ and a fresh nonce $N_{a_2}$. The provisional state times out if the protocol remains incomplete after a period of time.

*Messages 5 and 6:*

$$a_2 \rightarrow S_2 \rightarrow \cdots \rightarrow G \Rightarrow A : \langle \text{AUTH}, a_2, \mathcal{E}_{Aa_2}(a_2, a_1, S_2, S_1, N_{a_2}, \mathcal{A}_{a_1a_2}) \rangle$$

- $a_2$ sends an authentication message to $A$. The authentication message includes the identities of the trusted modules and their hosts, the nonce $N_{a_2}$ and the authentication token from $a_1$. The message must be routed through its host $S_2$. Otherwise, the routing path can be arbitrary, with the role of untrusted nodes limited to routing.
- $A$ verifies the mapping $(S_1, a_1)$ and stores $(S_2, a_2)$. Implicit entity authentication is accomplished by the verification of encrypted information against unencrypted information and stored state. $A$ also decrypts the authentication packet $\mathcal{A}_{a_1a_2}$ and checks its nonce for freshness against its stored state. It also verifies that $\mathcal{A}_{a_1a_2}$ is valid for the sender $S_2$. Finally, $A$ generates a fresh random number $R$ that will be used by $a_1$ and $a_2$ (independently) to derive a new session key set for the $a_1a_2$ association; the key material is not kept by $A$ after being sent off.

*Messages 7, 8, 9 and 10:*

$$A \Rightarrow G \rightarrow \cdots \rightarrow S_1 \rightarrow a_1 : \langle \text{AUTH}, \mathcal{E}_{Aa_1}(a_1, a_2, N_{a_1}, R) \rangle$$

$$A \Rightarrow G \rightarrow \cdots \rightarrow S_2 \rightarrow a_2 : \langle \text{AUTH}, \mathcal{E}_{Aa_2}(a_2, a_1, N_{a_2}, R) \rangle$$

- $A$ routes a pair of messages independently to the trusted peers $a_1$ and $a_2$. The routing path can be arbitrary, with any nodes on the path between $A$ and $S_*$ acting as routers. The messages contain the identities of the pair, their respective nonces and fresh key material $R$. The payload of both messages is encrypted by the secret device keys $K_{Aa_*}$ and hence cannot be read by any party other than the intended recipient $a_*$.

- The untrusted intermediaries $S_*$ forward the messages to their respective trusted aggregators $a_*$.
- $a_1, a_2$ decrypt the messages and verify the enclosed peer identities and nonces against their internal state.

The first stage of the protocol is complete at this point and a provisional session state set up for the $a_1a_2$ association by both nodes, storing $K_{a_1a_2} = \text{KDF}(R)$ as a *session key*, where KDF() is a key derivation function.

Alternatively, the authentication server can route the key material for $a_2$ through $a_1$. In this case, we can view $a_1$ as fulfilling the role of the collector in the protocol described in Section 3.4.3, that is, one of an authentication proxy whose trust is limited.

**Key confirmation and transport key establishment.** The trusted aggregators $a_1$ and $a_2$ initiate a key confirmation and transport key establishment phase upon establishment of the session key. Either node can initiate the exchange, but logical choice is for the inviter $a_1$ to begin as described below.

*Messages 11, 12 and 13:*

$$a_1 \rightarrow S_1 \rightarrow S_2 \rightarrow a_2 : \langle \text{REKEY}, a_1, \mathcal{E}_{a_1a_2}(a_1, a_2, N'_{a_1}) \rangle$$

- $a_1$ sends a re-key message to $a_2$ (routed through untrusted hosts $S_1$ and $S_2$), containing the identities of the trusted peers and a fresh nonce. The message is encrypted by the new session key $K_{a_1a_2}$, and hence, only readable by $a_1$ and $a_2$.
- $a_2$ looks up the session key for $a_1$, decrypts and verifies the message. $a_2$ generates a fresh random number $R$ and derives transport key set $K_T = \text{KDF}(R)$ for the $a_1, a_2$ association. $K_T$ is stored in the state for $a_1$, along with an expiration timestamp.

*Messages 14, 15 and 16:*

$$a_2 \rightarrow S_2 \rightarrow S_1 \rightarrow a_1 : \langle \text{NEWKEY}, a_2, \mathcal{E}_{a_1a_2}(a_2, a_1, N'_{a_1} - 1, R) \rangle$$

- $a_2$ sends an encryption under the session key of the identities of the nodes (reversed), a related nonce (in this case, the original nonce decremented by one) and the random number $R$ used as key material.

- $a_1$ looks up the session key for the $(a_1, a_2)$ pairing and decrypts the message. The identities of the trusted nodes and the nonce are checked against the internal state for the pairing and a transport key set $K_T = \text{KDF}(R)$ derived.

It is a good practice in cryptographic protocols to replace keys well before the adversary can collect enough information to mount analysis attacks. Hence, the re-keying protocol can be executed as often as needed.

One of the fundamental problems in practical applications of cryptographic algorithms is generation of high entropy random values. This can be accomplished on more capable computing platforms using dedicated high entropy sources, such as `/dev/random` on Unix platforms. However, good random sources may be scarce and difficult to construct on resource constrained devices, such as trusted sensors and aggregators (Kristinsson, 2011). This presents a problem in the protocols described above that depend on key material being generated and exchanged between two such nodes. In contrast, the much more capable aggregator generated and delivered key material in the system described in Chapter 3.

**Pre-association of co-located devices.** The authentication protocol described above can be used with minor modifications in the association of co-located sensors and aggregators, that is, for intra-node authentication. The protocol will in fact be similar to the symmetric TTP protocol of Section 3.4.3 with an authenticated aggregator taking the place of the collector. The process of authenticating co-located sensors and aggregators on every bootstrap of the observation node is wasteful in terms of messaging if the local configuration can be assumed to be static. However, persistent session keys may be used to reduce the overhead. In this case, the trusted devices maintain the last session key and require only a local handshake process to establish new transport keys. The impact of a persistent session key on security must be weighted against the economy of reduced messaging. However, if the session key is of comparable strength to the permanent device key, then we may assume that security of the exchange of pre-authenticated devices is similar to that of establishing a new session key on each observation node bootstrap. If necessary, the session key can be replaced at any time by any of the co-located trusted nodes, perhaps after a certain number of bootstraps.

**Authentication in query distribution phase.** A hierarchical aggregation protocol may include a query setup phase based on flooding of parameters, as described in Section 2.3.3. We can easily integrate the authentication phase into the query distribution phase by bootstrapping pairwise authentication at the same time the spanning tree overlay is established.

**Using physically uncloneable functions.** An authentication protocol that uses a physically unclonable function (PUF) is described in Section 3.4.3. A PUF adds resilience against node simulation in case the permanent secret key has been recovered by the adversary. Let us now consider how the protocol described above can be augmented by PUF features for added security.

A straight-forward solution is to insert a challenge-response exchange between $A$ and each aggregator individually before messages 7 and 9 in the protocol. However, the number of messages can be reduced by delegating the challenge/response tasks to the aggregator nodes themselves by including the necessary information in messages 7 and 9. This is accomplished by the aggregator appending encrypted challenge/response pairs into these messages as follows: $A$ includes a challenge/response pair $\mathcal{CR}_2 = (\mathcal{E}_{Aa_2}(C_2, N_2), R_2)$ into message 7 (destined for $a_1$) and $\mathcal{CR}_1 = (\mathcal{E}_{Aa_1}(C_1, N_1), R_1)$ pair into message 9 (destined for $a_2$). The challenges are encrypted to prevent the recipient from learning a challenge/response pair for a peer. Nonces add variability in the encrypted message and reduce the opportunity to mount replay attacks against the challenge/response exchange. The key confirmation protocol is amended as follows:

*Messages 11, 12 and 13:*

$$a_1 \rightarrow S_1 \rightarrow S_2 \rightarrow a_2 : \langle \text{REKEY}, a_1, \mathcal{E}_{a_1a_2}(a_1, a_2, N'_{a_1}, \mathcal{CH}_2)\rangle$$

- $a_1$ initiates the protocol. The initiating message is identical to the previous version of the protocol, except for the challenge $\mathcal{CH}_2 = \mathcal{E}_{Aa_2}(C_2, N_2)$.
- $a_2$ receives the message, decrypts using the shared session key $K_{a_1a_2}$ and verifies packet data. The challenge $C_2$ is extracted from $\mathcal{CH}_2$ and processed by computing $R_2 = \text{PUF}(C_2)$.

*Messages 14, 15 and 16:*

$$a_2 \rightarrow S_2 \rightarrow S_1 \rightarrow a_1 : \langle \text{CHALLENGE}, a_1, \mathcal{E}_{a_1a_2}(a_2, a_1, N'_{a_1} - 1, \mathcal{CH}_1, R_2)\rangle$$

- $a_2$ returns the response $R_2$ and issues the challenge $\mathcal{CH}_1 = \mathcal{E}_{Aa_1}(C_1, N_1)$ for $a_1$.
- $a_1$ begins by verifying the received response $R'_2$ against the $R_2$ supplied by $A$. $a_1$ then extracts the challenge $C_1$ from $\mathcal{CH}_1$ and computes $R_1 = \text{PUF}(C_1)$.

*Messages 17, 18 and 19:*

$$a_1 \rightarrow S_1 \rightarrow S_2 \rightarrow a_2 : \langle \text{CHALLENGE}, a_2, \mathcal{E}_{a_1 a_2}(a_1, a_2, N'_{a_1} - 2, R_1) \rangle$$

- $a_1$ submits the response $R_1$ to $a_2$.
- $a_2$ verifies the received $R'_1$ against the $R_1$ supplied by $A$.

*Messages 20, 21 and 22:*

$$a_2 \rightarrow S_2 \rightarrow S_1 \rightarrow a_1 : \langle \text{NEWKEY}, a_2, \mathcal{E}_{a_1 a_2}(a_2, a_1, N'_{a_1} - 3, R) \rangle$$

- $a_2$ generates the key material $R$ and submits to $a_1$.
- Both nodes derive transport keys $K_T = \text{KDF}(R)$.

**Verification.** As discussed in Chapter 3, proper verification of cryptographic protocols, such as the ones presented in this section, is essential in order to give overall guarantees of system integrity. The symmetric protocol described above has been modeled in ProVerif and results indicate that it is secure in terms of the adversary being unable to modify contents or extract secret information (Kristinsson, 2013). Methodology and assumptions for the ProVerif modeling is analogous to that discussed in Chapter 3.

#### 4.2.5.2 Asymmetric Authentication Protocol

The symmetric TTP protocol achieves the objectives of mutual entity authentication and shared key establishment but at the expense of some messaging overhead. In addition, the need for an TTP to be available at all times may be unreasonable for some networks. This can be avoided by employing public key (asymmetric) cryptography.

Public key cryptographic primitives solve the key distribution problem, as originally shown in the seminal work of Diffie and Hellman (1976) and Merkle (1978). Each participating node must have a set of $(K_{pub}, K_{pri})$ keys, whose public part $K_{pub}$ can be freely distributed. Encryption of a plaintext $P$ by a party $a$ destined for a party $b$ is accomplished by executing an encryption function $C = \mathcal{E}(K_{pub,b}, P)$, provided $a$ holds $b$'s public key. Conversely, decryption of the ciphertext $C$ by $b$ is $P = \mathcal{D}(K_{pri,b}, C)$. Public key primitives can also be used for authenticating messages. A message $M$ can be digitally signed by a party $a$ using a signature algorithm: $s = Sign(K_{pri,a}, M)$. A recipient of $M$ can verify the signature as $Verify(s, K_{pub,a}) = \{true, false\}$. The cost of asymmetric cryptography is the relatively large keys required for acceptable level of security and generally computationally expensive algorithms.

The sensor network literature, for instance Perrig et al. (2004), commonly assumes sensor motes to be incapable of complex cryptographic operations. Hence, it is easy to dismiss such approaches to securing the resource constrained trusted devices we are considering. However, implementations of asymmetric primitives exist for resource constrained devices, whose scale is on the order of the devices we are considering, such as the more powerful SmartCard processors (Rankl & Effing, 2001). Watro et al. (2004); Batina et al. (2006); D. J. Malan et al. (2008); A. Liu and Ning (2008) discuss the feasibility of public key implementations on resource constrained systems, while Finnigin et al. (2007) provide a cryptanalysis of Malan's original EccM (2004) design.

For the remainder of this section, we assume each trusted device has a permanently installed identity consisting of a public ID and a single digital certificate, including public and private keys signed by a root of trust. The identity tuple is of the form

$$ID = (v, K_{pub}, K_{pri}, \mathcal{S}\{v, K_{pub}\}_{RT}, RT_{pub})$$

where $K_{RT} = (K_{pri}, K_{pub})$ is the signing key pair for a root of trust $RT$.

$$\textit{Cert} = (v, K_{pub}, \mathcal{S}\{v, K_{pub}\}_{RT})$$

is the public key certificate submitted to peers in the protocol. The asymmetric keys in the identity tuple will be used exclusively for signing in the protocol that follows.

We require the chosen protocol to have the properties of mutual entity authentication and explicit key authentication. A number of suitable protocols exist, including the station-to-station (STS) protocol (Diffie et al., 1992), (Menezes et al., 1996, pp. 519), and an asymmetric version of Needham-Schroeder (Needham & Schroeder, 1978), (Menezes et al., 1996, pp. 508).

Let us now outline the application of an elliptic curve version of the STS protocol, EC-STS (Hankerson et al., 2004, pp. 193), in hierarchical TSense. Public key cryptography based on elliptic curves (Koblitz, 1987) requires considerably smaller keys for equivalent level of security, compared to RSA and similar methods based on the hardness of factorization (Barker et al., 2007, sec. 5.6.1). The intra-node authentication case is shown in which a trusted sensor $\sigma_{i,0}$ and trusted aggregator $a_i$, both hosted by an observation node $S_i$, establish trust relations. $D = (q, FR, S, a, b, P, n, h)$ are the elliptic curve parameters, as further described by Hankerson et al. (2004, Chapter 3).

*Message 1:*

$$S_i \rightarrow \sigma_{i,0} : \langle \textsf{invite}, a_i, \mathit{Cert}_{a_i} \rangle$$

- The protocol is initiated by an untrusted agent $S_i$ (representing observation node $i$) upon discovery of sensor $\sigma_{i,0}$ (sensor zero of node $i$). We assume the trusted aggregator $a_i$ is already known to $S_i$ and its public key certificate cached.
- $S_i$ sends an invite message on behalf of $a_i$.
- $\sigma_{i,0}$ caches certificate $\mathrm{Cert}_{a_i}$.

*Messages 2 and 3:*

$$\sigma_{i,0} \rightarrow S_i \rightarrow a_i : \langle \textsf{authenticate}, \sigma_{i,0}, a_i, R_\sigma, \textsf{Cert}_{\sigma_{i,0}} \rangle$$

- $\sigma_{i,0}$ verifies and caches the public key certificate of $a_i$. See verification procedure below.
- $\sigma_{i,0}$ picks $k_\sigma \in_R [1, n-1]$ and submits the ephemeral public key $R_\sigma = k_\sigma P$ and its public identity to $a_i$, routed via the untrusted $S_i$, where $P$ is a point on the elliptic curve.
- $a_i$ validates the certificate for $\sigma_{i,0}$ as discussed below. $a_i$ validates $R_\sigma$, selects $k_a \in_R [1, n-1]$ and computes the ephemeral public key $R_a = k_a P$. $a_i$ then computes $Z = hk_a R_\sigma$ s.t. $Z \neq \infty$. A symmetric key pair $(K_1, K_2) =$ KDF($x_Z$) is then derived from the x-coordinate of $Z$.

*Messages 4 and 5:*

$$a_i \rightarrow S_i \rightarrow \sigma_{i,0} : \langle \textsf{authenticate}, \sigma_{i,0}, a_i, R_a, s_a, \tau_a \rangle$$

- $a_i$ submits a message to $\sigma_{i,0}$ containing its public identity, $R_a$, a signature $s_a = \mathcal{S}_{a_1}\{R_a, R_\sigma, \sigma_{i,0}\}$ and a tag $\tau_a = \mathcal{T}_{K_1}(R_a, R_\sigma, \sigma_{i,0})$ computed using the derived key $K_1$.
- $\sigma_{i,0}$ validates $R_a P$ and computes $Z = hk_\sigma R_a$ s.t. $Z \neq \infty$. $\sigma_{i,0}$ then derives the symmetric key pair $(K_1, K_2) =$ KDF($x_Z$) from the x-coordinate of $Z$. $\sigma_{i,0}$ then verifies the signature $s_a$ using $\mathrm{Cert}_a$ and that $t_a = \mathcal{T}_{K_1}(R_a, R_\sigma, \sigma_{i,0})$. The signature convinces $\sigma_{i,0}$ of the identity of $a_i$ and that the message has not been tampered with, while the tag proves to $\sigma_{i,0}$ that $a_i$ has derived the shared secret $Z$.
- The session key for the $(\sigma_{i,0}, a_0)$ association is $K_2$.

*Messages 6 and 7:*

$$\sigma_{i,0} \rightarrow S_i \rightarrow a_i : \langle \textsf{authenticate}, a_i, \sigma_{i,0}, s_\sigma, \tau_\sigma \rangle$$

- $\sigma_{i,0}$ submits a message containing $s_\sigma = \mathcal{S}_\sigma\{R_\sigma, R_a, a_i\}$ and a tag $\tau_a = \mathcal{T}_{K_1}(R_\sigma, R_a, a_i)$ computed using the derived key $K_1$.
- $a_i$ verifies the signature $s_\sigma$ and tag $t_\sigma$. The signature convinces $a_i$ of the identity of $\sigma_{i,0}$ and that the message has not been tampered with, while the tag proves to $a_i$ that $\sigma_{i,0}$ has derived the shared secret $Z$. The session key for the $(a_0, \sigma_{i,0})$ association is $K_2$

This protocol achieves the establishment of a shared secret $Z = hk_\sigma k_a P$, derived from the ephemeral public keys $R_\sigma$ and $R_a$. The shared secret is then used to establish a session key $K_{a\sigma} = K_1 = \text{KDF}(x_Z)$ using the x-coordinate of $Z$. The re-keying phase described for the symmetric protocol can then be executed to establish transport keys. In comparison with the symmetric TTP protocol, the messages required to establish the session key are fewer and routed over a shorter distances (exactly two hops) compared to the arbitrarily long paths in the TTP case.

**Verification of certificates** Trusted nodes must establish that the communicating peer is indeed a member of an accepted group of trust. This was easy in the symmetric protocol due to the use of a single TTP that held (or had access to) all the keys of devices certified as trusted. In the public key protocol, the identity tuple of the devices hold a certification of membership in the form of the signature of a root-of-trust ($RT$).

The case in which both devices are certified by the same $RT$ is easy: nodes run a verification procedure on received certificates $\textit{Verify}_{RT}(\text{Cert})$ using the embedded public signing key $RT_{pub}$ of their root-of-trust.

The case in which devices are signed by separate $RT$s cannot be solved using local knowledge, as neither has the information to verify the $RT$ signature on their prospective peers identity. Further, they have no knowledge of whether their own RT considers the peer RT trusted. In this case, the untrusted node must (prior to the execution of the protocol) retrieve the public key certificates for both RT's signed by the respective RT. As an example, we assume observation node $S_i$ hosts $a_i$ and $\sigma_i$. $a_i$ is certified and signed by RT $A$ and $\sigma_i$ certified and signed by RT $B$. The protocol to be executed by the untrusted node is as follows:

1. $S_i$ submits Cert$_{a_i}$ to $B$.
2. $S_i$ submits Cert$_{\sigma_i}$ to $A$.
3. if $A$ trusts $B$ (have a common ancestor in the root-of-trust hierarchy) then it signs Cert$_{\sigma_i}$ and returns to $S_i$.
4. if $B$ trusts $A$ (have a common ancestor in the root-of-trust hierarchy) then it signs Cert$_{a_i}$ and returns to $S_i$.

Now, both entities $a_i$ and $\sigma_i$ can verify the certificates brokered by $S_i$ as representing a trusted device using the embedded public key of their respective RTs. This procedure requires $A$ and $B$ (or an representative thereof) to be reachable by all $S$ that require their services. This is not an unreasonable requirement if $A$, $B$ are reachable by $q$, the querier that is by definition reachable by all $S$ able to participate in the aggregation protocol. In the event that $A$ and $B$ are not reachable, the system configuration must include off-line processing and caching of certificates on each $S$ prior to deployment.

## 4.3 Properties of TSense

### 4.3.1 Security

We refer to the properties discussed previously in Chapter 3 and expand in this section as applies to the hierarchical model.

The hierarchical TSense solution provides end-to-end integrity guarantees, comparable to those of the simpler client/server model in Chapter 3. As before, the trusted sensor provides the foundation of trust, enabling next hop aggregators to verify their contributions. In turn, upstream aggregators can verify the partial aggregate contributions of their downstream peers. Since trusted aggregators by definition perform their computations correctly, we can assume transitive trust relations from leaves (trusted sensors) to the querier, guaranteeing overall aggregate correctness. As in the system considered in Chapter 3, the security guarantees achievable in a hierarchical aggregation system depend on the assumed resiliency of the trusted devices, the strength of the protocols and underlying cryptographic primitives.

We have assumed the cryptographic primitives and protocols to be secure, which in turn is the basis for the claimed overall system security. We may expect advances in cryptanalysis and, to some extent, computing hardware to degrade or break the security of cryptographic protocols and primitives over time. However, the system is designed for cipher-independence: any secure cryptographic primitive which can be implemented on trusted devices may be substituted for those suggested in this or the preceding chapter. Likewise, the asymmetric authentication is not bound to the EC-STS algorithm.

Successful man-in-the-middle attacks would allow the adversary to gain the capability of influencing the aggregate. We refer to Hankerson et al. (2004) on a full analysis of the resiliency of EC-STS against MitM. Drop attacks during the authentication and key establishment phase are availability attacks but do not further the adversaries stated goals of gaining the capability of stealthily influencing the aggregate computation. Rather, drops in the initialization phases cause the affected trusted devices to stay inert and not participate in the protocol. Hence, the correctness properties of the aggregate computation are intact in the case of misbehavior during initialization.

### 4.3.2 Scalability

One of our original design goals was securing distributed aggregation, while preserving as far as possible the scalability properties of the underlying aggregation algorithm protocol stack. Let us now consider the impact of the proposed solution on the overall performance. The secure transport protocol adds some cryptographic overhead in terms of message expansion and authentication codes. However, the overhead per message is constant and the number of update messages remains the same as for the underlying aggregation protocol. The relatively expensive authentication protocols add considerable overhead. However, we may assume authentication is carried out relatively infrequently. This allows us to amortize the cost of authentication over a prolonged period of data transfer. Assuming the overhead imposed by authentication is negligible in comparison with data transfer, we arrive at the conclusion that the messaging overhead (number and size) imposed by the trusted devices is of constant order.

### 4.3.3 Generality

Our original design goals were for the system to support arbitrary data types and aggregation functions. Any data type that can be securely encrypted and authenticated for transport can be supported by a trusted sensor. Any data type and associated aggregation function that can be implemented in a trusted device can be supported by a trusted aggregator. We conclude that the TSense trusted overlay approach is independent of data types and aggregation functions.

Another design goal was for the approach to be applicable to a range of aggregation protocols. We have so far discussed protocols based on spanning-trees. However, the trusted systems approach is applicable to any aggregation protocol, whose local function can be implemented in a trusted aggregator. Trustworthy routing, independent of aggregation protocol, is provided by the trust relationships established between trusted peers. Hence, we can claim that the trusted devices approach can be used to support arbitrary aggregation protocols.

## 4.4 Concluding Remarks

We extend the trusted systems approach presented in Chapter 3 to hierarchical aggregation in dynamic networked systems. Specifically, we define a trusted aggregator (Definition 4.7) – a dedicated hardware device that reliably executes the local aggregation function. Secure channels, established pairwise between trusted devices, constitute a trusted aggregation overlay. The trusted system as a whole effectively provides a secure distributed execution environment for arbitrary aggregation protocols. Given such an environment, we can give quite strong guarantees in terms of the aggregate integrity. In particular, we can state that if a trusted sensor provides a reading, then it will be correctly represented in the computed aggregate, or not included at all. Hence, we prevent the most damaging attacks under our adversarial model – the stealthy data modification attack.

We have already constructed a proof-of-concept system for trusted sensing in a single trusted aggregator model. Future agenda includes extending this prototype to a fully distributed system, such as the one outlined in this chapter. A conceivable approach is to implement a dynamically configurable secure aggregation system, based on executing verified and attested software modules in trusted environments, such as hypervisors, on the aggregation nodes. Issues, such as tamper-resistance and device manufacture, remain to be resolved, along with evaluation of the impact of trusted devices on hosting nodes in terms of power, computational and communications resources. Further development of the PUF aspects of the protocol is also an interesting direction for future research.

# Chapter 5

# Complete Aggregation

In this chapter, we address the objective of *completeness* – that all updates contributed by observation nodes are included in the computed aggregate. This objective compliments the discussion in the previous chapters, where the focus was on delivering correct updates. That is, *if* a partial aggregate was delivered to the querier then we could trust its integrity. Now, we turn our attention to ensuring that correct updates generated by trusted nodes are actually delivered in the form of partial aggregate contributions to the querier. Our goal is to to construct mechanisms that are low impact in terms of communications and processing costs, while approaching this ultimate goal. We analyze the problem of guaranteeing completeness and survey a selection of existing solutions. Our findings are that in the general case, no solution can guarantee completeness. Rather than attempting what is in our opinion a futile task, we propose a simple protocol that builds on the correctness layer proposed in Chapter 4, adding mechanisms that reduce the influence of potentially malicious nodes.

Part of the material presented in this chapter has been published by Jónsson, Palmskog, and Vigfússon (2012) and Jónsson, Vigfússon, and Helgason (2012).

## 5.1 The Importance of Completeness

*Completeness* was previously given as Definition 3.4. Recapping, the completeness objective states that all updates based on local observations released by (honest) data providers must be delivered to the querier. The complimentary objective with regards to integrity is *correctness* – that all observations delivered to the querier are correct (Definition 3.3).

Let us begin by motivating the discussion in this chapter by considering the importance of completeness in terms of the trustworthiness of aggregate observations. We assume a corrupt insider is able to observe, but not modify, the contents of messages, for instance by the assumption of a trusted layer, as described in Chapter 4, which we may also call a *correctness layer*. However, the correctness layer described in the previous chapters does not prevent the adversary from dropping "inconvenient" observations. Hence, the adversary still wields some control over the aggregation system in terms of his ability to prune the sample set. Let us express this control as the *potential adversarial influence*: the adversary *may* at any time remove samples, thereby intuitively reducing the fidelity of the aggregate. The adversarial influence can be described by the ratio of nodes under adversarial control (whose updates the adversary can disrupt) to the total number of nodes. Let us call this the *adversarial influence factor*.

The bias an adversary can introduce by removing updates is strongly dependent on the underlying aggregation network topology and positioning of corrupt nodes. Consider a centralized system, such as the one discussed in Chapter 3. A single corrupt node can at most influence its own reading (that of its hosted trusted sensor), implying that the potential adversarial influence factor is exactly $t/n$, where $t$ is the total number of corrupt nodes and $n$ is the total number of nodes. In a tree-based aggregation network, such as the one presented in Chapter 4, the adversary effectively controls a subnetwork of a size equal to the subtrees of all corrupt nodes. Limiting the influence of the adversary to the set of corrupt nodes is the ultimate goal.

The bias achievable by affecting message delivery is dependent upon the aggregation function being computed. Consider for instance the MAX and TOP_QUANTILE functions that estimate the upper range of some measured quantity in a observed system. Given some population of corrupt nodes, the adversary is in a position to induce the querier to accept a false maximum or top quantile reading by suppressing legitimate readings. Aggregation functions such as AVERAGE are more resilient against malicious drops. Intuitively, removing samples from a Gaussian distribution does not affect the expected value. However, a reduced sample size due to malicious drops reduces the fidelity of the sample, thereby likely increasing the sampling error.

## 5.2 Feasibility of Complete Aggregation

We have previously discussed how the objective of correctness can be met, given a trusted aggregation layer, which we will call a *correctness layer* in this chapter. Let us begin by considering whether similarly efficient means of guaranteeing completeness exist.

Let us consider an aggregation network, such as the one described in Chapter 4. The system is efficient in terms of message and time complexity, and guarantees correctness within the bounds placed by the resiliency of trusted modules. However, the system is best-effort with regards to the delivery of messages, meaning that no guarantees are given regarding completeness. The weakest form of a completeness guarantee is that all messages released by data contributors are eventually delivered to the querier, either directly or as contributions to partial aggregates. Guaranteeing timely delivery, which would be necessary for a practical system, is a harder goal. Intuitively, even eventual delivery is impossible to guarantee in an aggregation system, such as the ones discussed in Chapters 3 and 4, given the non-negligible probability of node and message corruptions and drops in real-world networked systems. For instance, nodes can malfunction, messages can be corrupted in transit due to environmental conditions or link re-configurations in dynamically evolving networks. *Benign faults* of this sort may be reduced by the addition of reliability mechanisms, for instance TCP or a protocol tailored to distributed systems (Stann & Heidemann, 2003). However, even reliability layers cannot guarantee completeness, demonstrating the inherent difficulty of countering an adversary that can simulate link faults.

Nodes that maliciously drop updates are by definition Byzantine in that they break the protocol in communications with some of their peers. We previously reviewed some techniques to counter Byzantine behavior in Chapter 4. However, differentiating between legitimate and malicious absence of information is intuitively an even harder problem than that of verifying the consistency of delivered information.

### 5.2.1 Guaranteeing Completeness

Let us assume all drops are malicious, that is, channels deliver all data, except if corrupt entities deliberately drop messages. Completeness of contributions by honest nodes can be guaranteed under this assumption, but only if the following conditions hold:

(i) At least one uncompromised path exists between any given honest data provider and the querier. An uncompromised path is one in which all nodes are honest and follow the protocol with regards to routing.

(ii) Contributing nodes send their updates on all available paths.

Under these conditions, at least one copy of each message is bound to reach the querier, hence guaranteeing completeness. By the same argument, sending less than all available channels forces us to settle for probabilistic completeness guarantees. The stipulation of a good path is rather strict for practical applications, but the probability of such a path existing can be increased by careful construction of redundant paths. An example is the *braided paths* proposed by Ganesan et al. (2001). Exploiting path redundancy to any degree requires by definition a quite significant increase in messaging, which is counterproductive in terms of our goal of efficient aggregation. Further, multi-path solutions are problematic, as previously discussed in Section 2.3.4, due to the multiple inclusion problem (Keshav, 2006). Multi-path techniques require duplicate insensitivity, either by employing solely functions which are inherently duplicate insensitive, such as MIN or MAX, or solutions such as order-and-duplicate insensitive (ODI) messages (Keshav, 2006; Nath et al., 2008). Both options limit the applicability of the aggregation systems in terms of the data types that can be processed and the aggregation functions that can be computed. Further, multi-path approaches require considerable communications and processing overhead.

We can reduce the effectiveness of the adversary in terms of making intelligent dropping decisions by denying him access to information. That is, we can encrypt message traffic to reduce its utility to malicious nodes. This is further discussed in Section 4.1.2.

### 5.2.2 Assessing Completeness

If we accept the futility of guaranteeing complete aggregation, the next logical step is considering whether we can quantify the completeness of an aggregate. This is straight forward in the case of a static synchronous network with a known node population. The centralized case is easy: we simply count the messages received at the end of execution of the algorithm and give a reliability rating proportional to the messages received vs. the total number of known contributors. Similarly, we can assess the completeness in a hierarchical aggregation network by comparing an aggregate count with the expected number of contributors. If the protocol is executed over a correctness layer, we can accept this count as accurate. A proportional reliability rating can be computed as a ratio of the known node population, given a reliable contributor count. Allowing for asynchronous operation complicates the issue, since delays may impact ordering and delivery times of messages. The problem is compounded in systems in which contributors may release updates sporadically, for example in response to local events such as threshold crossings (Wuhib et al., 2005). Considering systems in which nodes can fail or leave the system without warning makes quantification of completeness even harder to do in a reliable manner.

## 5.3 Building Blocks for Reducing Adversarial Influence

Efficiency is one of our primary criteria in the search for security mechanisms. Hence, we dismiss multi-path approaches to the problem of complete aggregation due to the fact that they are bound to impact message complexity, while still not providing absolute completeness guarantees. Similarly, protocols that attempt to achieve completeness by introducing elaborate sub-protocols are bound to sacrifice efficiency without achieving guaranteed completeness.

In this chapter, we explore simple means of increasing the resilience of aggregation protocols to malicious drops. Increased resiliency translates to decreased adversarial influence, and thereby higher probability of a complete sample set being aggregated. The methods described are efficient in terms of messaging overhead, which is our primary performance criteria. In light of our earlier discussion, we make no claims towards guaranteeing completeness in any sense, even probabilistically. Rather, we settle for a reduction in the influence an adversary can wield over completeness compared to an unmodified best-effort aggregation protocol. We describe the general mechanisms in this section and move on to applying the principles to a real protocol in Section 5.4.

The methods described in this section are applicable to any best-effort aggregation protocol, including gossip-based aggregation protocols (Wuhib et al., 2007). However, we confine our discussion to a tree-based best-effort aggregation system as described in Chapter 4 and later in this chapter. The aggregation overlay is composed of a single implicitly trusted querier and a number of observation nodes that provide observations (contribute local inputs) as well as functioning as aggregators. Observation nodes are generic corruptible platforms but trusted sensors and aggregators form a secure aggregation overlay, guaranteeing aggregate correctness. For simplicity, we assume channels are reliable and all loss events are caused by adversarial actions.

The fact that observation nodes (hosts of trusted modules) can be corrupted forces us to regard communications channels between trusted modules as under adversarial control

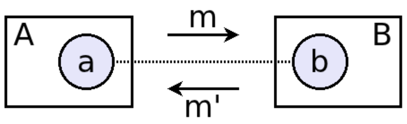


**Figure 5.1:** *Transmission of message $m_{ab}$ from trusted module $a$ to peer $b$. The two trusted modules $a$ and $b$ are hosted (respectively) by untrusted observation nodes $A$ and $B$.*

in terms of message drops, delays and reordering. Hence, aggregate completeness is not addressed in the system of Chapter 4. Let us now discuss the basic components – building blocks – which can be applied to reduce the adversarial influence in this and other aggregation protocols.

1. *Fault detection* is necessary in order to formulate a response.
2. *Local response*, for instance re-routing, is the first step towards responding to events which can be considered potentially malicious.
3. *Dissemination of fault information* empowers peers in the system with the ability to cooperatively respond to potential threats.

We assume the protocol enhancements considered in the latter part of this chapter are executed on the correctness layer, that is, by trusted aggregators. Hence, we can assume that all messages which belong to the protocol are correct, even though their delivery cannot be guaranteed.

### 5.3.1 Fault Detection

The ability to detect faults is a prerequisite for effective response. Let us consider a single update message $m_{ab}$ sent from trusted module $a$ to a peer $b$, hosted (respectively) by untrusted observation nodes $A$ and $B$, as shown in Figure 5.1. The channel $(a, b)$ is under adversarial control in terms of eventual delivery, delay and reordering of messages. Either one of the untrusted hosts $A$ or $B$ (or both) can be malicious, implying that they can affect message delivery at will. Our primary objective is to convey information of such actions to the trusted modules $a$ and $b$, enabling them to respond effectively. As discussed in the previous chapters, the trust relationship between $a$ and $b$ (the correctness layer) guarantees that *if* a message $m_{ab}$ sent from $a$ reaches $b$ then it is correct. We exploit the existence of a correctness layer in the mechanisms proposed in this chapter.

#### 5.3.1.1 Receiver-side Fault Detection

Detecting a missing message hinges on having knowledge of a message having been sent but not delivered. This is difficult from the perspective of the receiver, unless we assume some knowledge of the schedule of the sender, which may be a reasonable assumption in some instances. Let us assume node $a$ is on a known update schedule, enabling $b$ to detect the absence of messages: if a message $m_{ab}$ is not received within some $\delta$ of the expected delivery time, $b$ knows that the message was dropped by either of the untrusted hosts $A$ or $B$ under the stated adversarial model. However, node $b$ does not know which of the two hosts (or both) is corrupt and node $a$ has no knowledge of the fault whatsoever at that point in time.

We can enable receiver-side detection of missing messages, regardless of update schedule, by adding sequence numbering to messages sent over the $(ab)$ channel. Gaps in the sequence of messages indicates to the receiver that loss events have occurred. The trust relations between $a$ and $b$ (correctness layer) guarantee that the adversary in control of the $(a, b)$ channel cannot modify the sequence number (or any other part of the message) without being detected. Note however, that detection in this case hinges on a message eventually being delivered from $a$ to $b$ – a complete block of messages on the $(a, b)$ channel gives $b$ no error information, except in the case of a pre-determined update schedule. Compulsory periodic pings can be added to any protocol to ensure that a channel is "alive". The performance impact is minimal if such messages are infrequent. In fact, most link layer protocols implement periodic handshake messages on to which our protocol messages can be piggybacked for increased efficiency. However, a slow rate of link layer handshakes slows the detection of faults. Hence, we advocate the integration of fault detection into the aggregation protocol itself.

#### 5.3.1.2 Sender-side Fault Detection

Enabling senders to detect faults requires some form of a feedback mechanism. For instance, adding an acknowledgement message $m'$ (see Figure 5.1) to a basic best-effort aggregation protocol gives the sender $a$ the necessary information. Node $b$ responds to an update message $m$ by generating an acknowledgement message transmitted over the $(b, a)$ channel to $a$. Let us assume host $B$ is malicious and drops either $m$ or $m'$. The host $a$ can determine that a fault occurred if $m'$ is not received within some time $\delta$ of the transmission. However, $a$ cannot determine whether $A$ or $B$ was faulty, without corroborating information. Further, $b$ has no information of the fault at this point in time, unless update schedules are pre-determined to some degree. Note that we assume acknowledgements are protected by the correctness layer, as discussed earlier.

#### 5.3.1.3 Rating Channel Reliability.

The detecting node, whether it is sender or receiver, is limited to determining that a fault occurred on the $(a, b)$ channel. As shown in Figure 5.1, both of the untrusted hosts $A$ and $B$ participate in relaying messages on the $(a, b)$ channel. Hence, we rate the reliability of channels, rather than hosts, in the subsequent discussion.

A trusted module $a$ maintains a local rating $\alpha_{ab} \in [0, 1]$ for a communicating peer $b$. Two update functions $f_F$ and $f_R$ (fault and redemption) are defined to update the $\alpha$ rating. Each detected fault, acknowledgement timeout or missing serial number, invokes the fault function $f_F$ and decreases the relevant channel rating. Any reduction function can be used, but we use reduction by a constant in the examples that follow. Optionally, misbehaving nodes can redeem themselves and increase the channel rating upon correct behavior. That is, a correctly delivered message invokes the redemption function $f_R$ and increases the channel rating.

### 5.3.2 Local Response

Local response to faults is possible, once edge ratings are available. We must differentiate between response to benign faults that are caused by unintentional errors, and malicious faults caused by the adversary.

Intermittent benign faults can be countered by retransmissions. However, persistent benign faults require re-routing, for instance to bypass failed intermediary nodes. Ideally, malicious faults should be countered by permanently routing “around” corrupt nodes. In practice, this is a difficult objective due to the possibility of benign faults. In the case of a trusted aggregation system, the problem is compounded by the fact that the host of the detecting node may be corrupt. Consider for instance the example of Figure 5.1. Let us say $A$ is corrupt and drops one or more update messages from $a$ destined for $b$ and that $a$ detects this by the absence of acknowledgement messages. $a$ detects the problem and reduces the $\alpha$ rating of the $(a, b)$ edge accordingly. However, $a$ is unable to respond effectively by any means, since its own host is malicious. In contrast, $a$ has an opportunity to respond to $(a, b)$ faults by re-routing in case $B$ is malicious and drops messages. Note that statically configured networks are unable to respond at all.

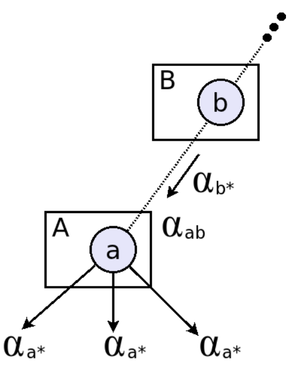


**Figure 5.2:** *Excerpt of a binary tree, showing two nodes. $\alpha_{ab}$ is the local rating of node a for the $(a, b)$ parent edge and $\alpha_{x*}$ is the reported (back propagated) parent edge rating.*

### 5.3.3 Disseminating Fault Information

Informing peers in the aggregation network of perceived faults enables pro-active response. Given fault detection capability as outlined above, $a$ and/or $b$ can inform surrounding nodes of a history of faults on the $(a, b)$ edge. Recipients of such alerts can in turn take measures to avoid one or both of the potentially malicious hosts $A$ and $B$, thereby reducing the chance of being affected by future faults.

Flooding a special alert message is one option of disseminating fault alerts. However, flooding is potentially hazardous in terms of messaging and storage demands. Further, alert messages are not likely to work in case the host of the alerting node is malicious. Including an indication of past performance in regular protocol updates is more promising. Let us consider a tree-based aggregation network, as described in Chapter 4. For simplicity, we confine our discussion to the two observation nodes $A$ and $B$ shown in Figure 5.2. The trusted aggregator $a$ is the child of $b$ in the trusted aggregation overlay (tree). $a$ has available the local rating of the edge $(a, b)$ as $\alpha_{ab}$. A fault on this edge indicates that either $A$ or $B$ (or both) are faulty. In this event, the safest recourse for the children of $a$ and other surrounding peers is to avoid the $(a, b)$ edge.

In the case of tree-based aggregation networks, we propose a simple mechanism that "feeds back" the rating of parent edges to lower level nodes, enabling avoidance of potentially faulty edges. Figure 5.2 demonstrates this graphically. Aggregator $b$ submits a rating of its parent path (not shown) as $\alpha_{b*}$ to its downstream child $a$. In turn, $a$ submits $\alpha_{a*} = g(\alpha_{ab}, \alpha_{b*})$ to its downstream children, where $g$ is a function that rates the overall reliability of the forwarding path by utilizing reported forwarding path ratings. One example is $\alpha_{a*} = \alpha_{ab} \cdot \alpha_{b*}$.

An example is shown in Figure 5.3. The host $B$ of a trusted module $b$ is malicious and has dropped a single message on the $(b, a)$ channel, detected by $b$ by a missing acknowledge-

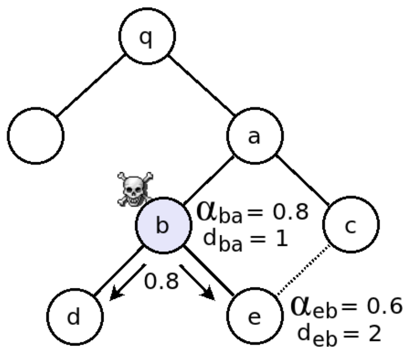


**Figure 5.3:** *Example of $\alpha$ rating in a small binary tree. Vertices represent trusted aggregators. Malicious actions are confined to edges controlled by the adversary.*

ment. $b$ executes an $\alpha$ update function that subtracts $0.2$ for each perceived fault. Hence, the $(b, a)$ channel has a rating of $\alpha_{ba} = 0.8$ from the perspective of $b$. Node $a$ has not detected any faults at this point in time, so its rating of the forwarding path is $\alpha_{a*} = 1$. Hence, $b$ computes a reliability rating for its forwarding path $(b, a)$ as $\alpha_{b*} = \alpha_{a*} \cdot \alpha_{ba} = 0.8$ and back-propagates to its peers. Node $e$ has detected two drops on the $(e, b)$ channel. Hence, its local rating for the channel is $\alpha_{eb} = 0.6$. Taking the back propagated report from $b$ into account gives a rating of $\alpha_{e*} = \alpha_{b*} \cdot \alpha_{eb} = 0.48$ for the current forwarding path $(e, b)$. Given this rating, node $e$ would be better served by selecting the alternate $(e, c)$ channel, which currently has a rating of one. Node $d$ has not detected any drops on the $(d, b)$ channel but nevertheless sets its rating of the forwarding path $\alpha_{d*} = 0.8$ in response to the report received from $b$.

**Blocking $\alpha$ updates.** An adversary may attempt to block the dissemination of "inconvenient" update messages, for instance the back-propagated $\alpha$ reports in the previous example. However, such behavior will not prove to be productive. As reports are built into regular protocol messages (aggregate updates and acknowledgements) any blocking of messages will only result in lowering of the local rating of a malicious node. The host of node $b$ may for instance try to prevent node $d$ in Figure 5.3 from learning of the $\alpha_{b*}$ rating. Again, we assume the back-propagation of $\alpha$ ratings is built into regular acknowledgements from $b$ to $d$. Blocking of even a single acknowledgement results in an immediate drop in the local $\alpha_{db}$ rating on node $d$.

**The multi-armed bandit.** Experiments in which outcomes of past trials affect future decisions can be formulated in terms of the Multi-armed Bandit (MAB) problem (Vermorel & Mohri, 2005): a gambler pulls the levers of $k$ slot machines (one-armed bandits) one at a time, keeps track of the rewards and uses the results to maximize his rewards, that is, to play the machine that has the highest probability of producing a win. In opaque bandit formulations, the gambler only observes the reward for the lever pulled. Hence, in order to establish the probable returns per lever, all have to be played a certain number of time. MAB algorithms use some strategy which maximizes gains by playing the best lever as often as possible (exploitation), while playing the other levers sufficiently often to learn their probable returns (exploration). For instance, in $\epsilon$-greedy algorithm, a random lever is chosen with some frequency $\epsilon$, while choosing the lever with the current best rewards with probability $(1 - \epsilon)$.

We can liken transmissions of messages in an aggregation protocol to "pulling the lever" for the corresponding channel: each trial gives us information about the reliability of a channel. Hence, MAB algorithms may be of interest in future work, enabling more efficient exploration/exploitation of channel reliability to minimize the adversarial influence.

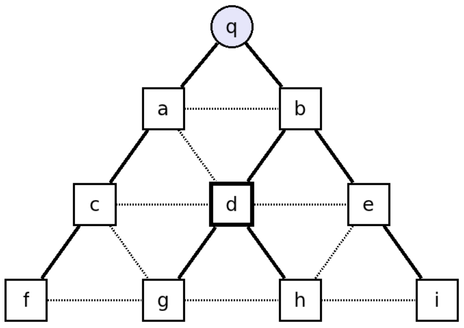
## 5.4 Secure GAP

Let us now apply the building blocks discussed in Section 5.3 to an actual aggregation protocol, the *Generic Aggregation Protocol (GAP)* (Dam & Stadler, 2005). We present a version of GAP, which we will call S-GAP, that is secure against modification of data (correctness attacks) and resilient against adversarial drops (completeness attacks). The correctness of the aggregate computation is ensured by the inclusion of trusted devices, as discussed in Chapter 4, while completeness is addressed by the application of the protocol building-blocks outlined in Section 5.3.

### 5.4.1 The GAP Protocol

Our case study involves securing GAP, an efficient tree-based aggregation protocol, designed for continuous monitoring of a distributed system. While efficiency was one of the primary design goals for GAP, security was not addressed in the original protocol, nor its subsequent enhancements (Gonzalez Prieto & Stadler, 2007; Wuhib et al., 2007, 2008; Stadler et al., 2008).

Nodes that execute the GAP protocol self-organize into a spanning tree (aggregation overlay) by continually choosing the most advantageous peers, in terms of minimizing total path length, as parents. GAP is resilient against benign crash faults, that is, non-reachable nodes, due to peer monitoring and updating of the overlay topology. However, the role



(a) A schematic representation of the aggregation overlay. Bold lines indicate edges in the spanning tree. Dotted lines indicate reachable neighbors.

| id | **status** | **level** | **weight** |
|---|---|---|---|
| $d$ | *self* | 2 | 56 |
| $a$ | - | 1 | 128 |
| $b$ | *parent* | 1 | 17 |
| $c$ | peer | 2 | 97 |
| $e$ | peer | 2 | 112 |
| $g$ | *child* | 3 | 73 |
| $h$ | *child* | 3 | 23 |

(b) The neighborhood table of node $d$ for the network instance shown in (a).

**Figure 5.4:** *Small example of a GAP aggregation network.*

of fault detection is relegated to the underlying network layer in the original protocol, implying that considerable delay is to be expected until a peer is deemed unreachable. Intermittent faults are not handled by the protocol. Hence, attacks against the completeness of the protocol are quite plausible. A further problem is that the fault detection is independent of the aggregation protocol itself.

The GAP protocol maintains the dynamic spanning tree overlay by means of a neighborhood table maintained locally by each participating node. An example is shown in Figure 5.4. The neighborhood table maintains the current weight (the partial aggregate) and the status of all reachable one hop peers (child, peer, parent) along with their reported distance (in hops) to the root of the tree (querier in our terminology). Each node elects a single node as parent, as shown in Algorithm 5.1, with the objective of minimizing the distance to the root of the tree. Briefly, nodes $(v)$ monitor their immediate neighborhood (one hop) $\Gamma$ for nodes $(u)$ that report a shorter distance to the root than the current parent. If such a node $u$ exists, then $v$ selects $u$ as its parent, updates its own level accordingly and broadcasts a state update message to all peers. State update messages are also broadcast whenever the local weight (local state) is updated and when a new partial aggregate (weight) is computed in response to received partial aggregates from designated children (downstream nodes). Hence, the dissemination of aggregate updates is in essence embedded into the tree maintenance protocol. We will exploit the state broadcast inherent in the GAP protocol in our S-GAP protocol.

The local aggregation function is computed over the weights delivered by current *child* and *self* nodes. In the example shown in Figure 5.4, node $d$ delivers a partial aggregate update with *weight* $= 56 + 73 + 23 = 152$ to node $b$. In our implementation of GAP, described in Section 5.5, we use a rate limiting delay $\delta$ from the instance the state table is modified and until an update message is produced. This delay increases the messaging efficiency of the protocol by increasing the probability of combining more than one state modification in each update message, but at the cost of delaying dissemination of changes in the monitored environment state to the querier.

**Algorithm 5.1** GAP parent selection

```
1: On neighborhood modified for node $v$ do:
2: if $\exists\, u \in \Gamma_v$ $s.t.$ $level(u) < parent\_level(v)$ then
3:     $parent_v \leftarrow u$
4:     $level_v \leftarrow level(u) + 1$
5:     $update\_neighbors()$
6: end if
```

Let us now describe the proposed modifications of the original GAP protocol to achieve the two objectives of correctness and completeness.

### 5.4.2 Correctness in S-GAP

A GAP overlay is composed of generic observation nodes, that is, computing platforms that can be compromised by an adversary. The goal of the S-GAP protocol is to ensure the integrity of the computed aggregate, both in terms of its correctness and completeness, when executed in such an environment.

The correctness objective is met by employing trusted sensors and aggregators, as described in Chapters 3 and 4. This requires a decomposition of the original GAP protocol into trusted and untrusted components. Trusted components of the S-GAP protocol execute exclusively on trusted modules (sensors and aggregators) and handle all functionality that must be considered essential to the primary mission of the system. Services, such as peer discovery, link-layer failure detection and routing, may remain under the control of an untrusted agent, comparable to `tsclient` in the prototype discussed in Chapter 3.

#### 5.4.2.1 Essential Protocol Information.

Let us now consider the protocol state information that must be considered part of the essential functionality.

| **Property** | **Access** |
|---|---|
| Public node identity | READ |
| Private node information, incl. cryptographic keys | NONE |
| Level | READ |
| Status | READ |
| Weight | READ (optional) |

**Table 5.1:** *Protocol properties and allowed access under the S-GAP protocol.*

| | | | | |
|---|---|---|---|---|
| id | status | level | weight | $\mathbf{K}$ |

(a) Trusted neighborhood table

| | | |
|---|---|---|
| id | network address | level |

(b) Untrusted neighborhood table

**Figure 5.5:** *S-GAP neighborhood table. The original neighborhood table of GAP is augmented and divided into two trusted and untrusted components.*

*Public node identity* is part of the secure data exchanged between trusted modules. Hence, node identity should always be protected against modifications. While an untrusted host may read public node identities, private identities, including cryptographic keys must be kept secret. Hence, session and transport keys, as described in Chapter 4, must remain secret.

*Level* is required to compute node status and essential to the proper function of the protocol. Hence, the integrity of level report received from peers must be ensured. However, level does not have to be kept secret, since the untrusted routing services need (and presumably have) the same information, independent of the S-GAP protocol.

*Status* of peer nodes is essential to correct functionality of the protocol. However, status does not have to be kept secure as the protocol can function properly by computing peer status prior to each update. However, in the interest of efficiency, the untrusted hosting node should be prevented from arbitrarily modifying the state assigned to peers by the S-GAP protocol.

*Weight* is the essential commodity of the protocol and must at minimum be protected against modifications to meet the objective of correctness. Maintaining the secrecy of the weight may be a requirement for some applications.

In the terminology of access control, each trusted module that implements the S-GAP protocol may grant the untrusted protocol service rights as specified in Table 5.1.

**Neighborhood table.** Given this analysis, we can divide the neighborhood table of GAP into two overlapping parts, as shown in Figure 5.5. One part is maintained by the trusted module and the other by the untrusted host. The untrusted agent plays an important role in the protocol. Its primary role is to manage the neighborhood, that is, discover new nodes, detect failures and to bootstrap an observation node into the aggregation protocol. This was previously discussed in Chapter 4. In particular, the untrusted component of the S-GAP protocol must maintain a routing table that will in essence be a duplicate of the level and state information maintained by the protected part. The primary goal of the S-GAP protocol is to prevent any untrusted services executing on the observation node from influencing the aggregate computation. Note that the S-GAP protocol enables verification of routing by including distances in the secure communications, allowing receiver to verify routing decisions made by the untrusted agents.

The secure neighborhood table may impose undue demands on the storage capabilities of a resource constrained device, especially in a dynamic environment that requires caching of a large number of nodes. Solutions exist which allow secure storage of information in untrusted memory (Chun et al., 2007; Levin et al., 2009), allowing the resource constrained trusted module to offload much of this burden to the untrusted observation node services. Solutions similar to the secure storage primitives of a TPM (Challener et al., 2008) may also be used. For simplicity, we assume the trusted table will be maintained within the trusted device itself for the remainder of this discussion.

**Protocol messages.** Two messages in the original GAP protocol, *update* and *weight*, need to be protected. Weight is assumed to be provided to the trusted aggregator by a trusted sensor, and hence, already protected, as described in Chapter 3. The protected version of the update message is a composition of the original GAP *update* and the data transport message from Chapter 4:

$$\langle \mathsf{update}, a, C_{ab}, l_{\mathcal{E}}, \bar{\mathcal{E}}_{ab}(a, b, C_{ab}, w, l, p) \rangle$$

where $a$ and $b$ are the trusted sender and receiver node identities (respectively), $C_{ab}$ is a monotonically increasing counter for messages sent on the $(a, b)$ channel, $l_{\mathcal{E}}$ is the length of the encrypted payload and $\bar{\mathcal{E}}_{ab}$ denotes authenticating encryption using key set $K_{ab}$. Note that proper cryptographic practices dictate use of a key set, that is, independently derived keys for the functionalities of tagging and encryption (Barker et al., 2007, sec. 5.2), (Gligoroski et al., 2008).

### 5.4.3 Increasing Completeness

The trusted systems principles applied to secure the S-GAP protocol achieve guarantees against adversarial modifications of data, that is, *correctness attacks*. However, message delivery, including re-ordering and artificially introduced delays, remains under adversarial control. Our goal in the remainder of this chapter is to minimize the impact an adversary can have by manipulating message delivery. We exploit the existence of the correctness layer, that is, the correctness protocols which we describe in this section are ensured by the same trust relationships used to ensure the wholeness of the aggregate data.

#### 5.4.3.1 Detecting Misbehavior

Let us now apply the building blocks described earlier to increase the resiliency of the S-GAP protocol against malicious drops. First of all, a counter is needed to enable receiver side detection of missing messages. The non-secret nonce (counter) already included in the *update* message is sufficient for this purpose. However, we need to add an acknowledgement message to enable sender-side misbehavior detection. The complete transport protocol is shown in Protocol 5.1. Note that we include a parent edge $\alpha$ rating in both messages.

$$\begin{aligned} a \Rightarrow b: &\quad \langle \text{UPDATE}, a, C_{ab}, l_{\mathcal{E}}, \bar{\mathcal{E}}_T(a, b, C_{ab}, \alpha_{a*}, w, l, p\rangle \\ a \Leftarrow b: &\quad \langle \text{ACK}, b, \bar{\mathcal{E}}_T(b, a, C_{ab}, \alpha_{b*}, h(w, l, p)\rangle \end{aligned}$$

**Protocol 5.1:** *Transport protocol for the* `update` *exchange with acknowledgements. The $a \Rightarrow b$ relation denotes communications between trusted peers $a$ and $b$ on a pairwise trusted $(a, b)$ channel. $\bar{\mathcal{E}}_K()$ denotes authenticating encryption using the transport key set $K_T$ for the $a \Leftrightarrow b$ association. $C_{a,b}$ is a counter for messages sent on the $(a, b)$ channel. $\alpha_{x*}$ denotes a rating $\in [0, 1]$ of the current forwarding path of node $x$. $h(w, l, p)$ is a hash computed over the (weight, level, parent) protocol data update.*

The trusted sender $a$ increments the counter $C_{a,b}$ and sets a timer $T_{a,b}$ when transmitting a message to the trusted peer $b$. The counter and timer fields need to be added to the neighborhood table, as shown in Figure 5.6. We limit the number of active exchanges between $a$ and $b$ to one in this protocol. Upon reception, $b$ constructs an acknowledgement message and sends to $a$. A counter $C_{ab}$ for the $(a, b)$ edge prevents replays and enables receiver-side misbehavior detection. The counter $C_{ab}$ keeps track of messages sent by $a$ on the $(a, b)$ edge and functions both as a non-secret nonce and means for $b$ to detect drops. The acknowledgement mechanism serves as *application layer fault detection*: The results w.r.t. $a$ are *success* if the acknowledgement from $b$ is received before the timer elapses and

| id | status | level | weight | $\mathbf{K}$ | $\mathbf{C}$ | $\mathcal{A}$ |
|---|---|---|---|---|---|---|

**Figure 5.6:** *S-GAP secure neighborhood table modified to include the fields required to enhance completeness. Sets of keys $\mathbf{K}$, counters $\mathbf{C}$ and edge ratings $\mathcal{A}$ are added.*

*fail* otherwise. The $\alpha$ rating for the $(a, b)$ edge is updated upon both success and failure. A set $\mathcal{A}$ of $\alpha$ ratings is maintained per neighbor in the neighborhood table. The set includes $\alpha_{ab}$, the local rating of node $a$ for the $(a, b)$ channel as well as $\alpha_{b*}$ reported by the peer $b$ for its current forwarding edge.

#### 5.4.3.2 Modified Parent Selection

In contrast to the *opaque bandit problem* mentioned earlier, the GAP protocol uses status message broadcasts as its mechanism for aggregation as well as tree maintenance – in essence, pulls all levers simultaneously. Hence, detailed and up-to-date information is available in order to select the best available parent in each round.

We modify the GAP parent selection criteria, described in Algorithm 5.1, to pick the best parent, based on available $\alpha$ ratings. In Algorithm 5.2, we present a minimal modification of the GAP parent selection algorithm: a node $v$ selects a parent from the set of nodes with smaller distance to the querier, picking the one with the current best $\alpha$ rating. This peer selection process does not affect the dynamic tree maintenance features of GAP, while favoring high-$\alpha$ nodes.

**Algorithm 5.2** GAP-SC parent selection (policy II)

```
1: ON neighborhood modified for node $v$ do:
2: $d_{min} \leftarrow \min_{u \in \Gamma_v} d_u$
3: $\boldsymbol{P} \leftarrow \{u \in \Gamma_v \mid s.t.\ d_u = d_{min}\}$
4: $\alpha_{max} \leftarrow \max_{u \in \boldsymbol{P}} \alpha_u$
5: if $d_p \neq d_{min} \vee \alpha_p < \alpha_{max}$ then
6:     $p_v \leftarrow p \in \boldsymbol{P}\ s.t.\ \alpha_p = \alpha_{max}$
7:     $d_v \leftarrow d_p + 1$
8:     $update\_neighbors()$
9: end if
```

This protocol achieves no reduction in adversarial influence if all neighbors having a shorter distance to the querier are corrupt. This can be addressed by allowing nodes to select parents from the set of designated as *peers* in GAP, that is, nodes with the same distance to the querier. The modified algorithm is shown in Algorithm 5.3.

The algorithm selects as parent the nodes with the current best $\alpha$ rating nodes either from nodes with less ($u \in \boldsymbol{L}_0$) or equal ($u \in \boldsymbol{L}_1$) distance to the querier, allowing greater flex-

**Algorithm 5.3** GAP-SC parent selection (policy III)

```
1: On neighborhood modified for node $v$ do:
2: $d_{min} \leftarrow \min_{u \in \Gamma_v} d_u$
3: $\boldsymbol{L}_0 \leftarrow \{u \in \Gamma_v \mid d_u = d_{min}\}$, $\boldsymbol{L}_1 \leftarrow \{u \in \Gamma_v \mid d_u = d_{min} + 1\}$
4: $\alpha_0 \leftarrow \max_{u \in \boldsymbol{L}_0} \alpha_u$, $\alpha_1 \leftarrow \max_{u \in \boldsymbol{L}_1} \alpha_u$
5: if $\alpha_0 \geq \alpha_1$ then
6:     $p' \leftarrow u \in \boldsymbol{L}_0 \; s.t. \; \alpha_u = \alpha_0$
7:     $\alpha' \leftarrow \alpha_0$
8:     $d' \leftarrow d_{min}$
9: else
10:    $p' \leftarrow u \in \boldsymbol{L}_1 \; s.t. \; \alpha_u = \alpha_1$
11:    $\alpha' \leftarrow \alpha_1$
12:    $d' \leftarrow d_{min} + 1$
13: end if
14: if $d_p \neq d_{p'} \vee \alpha_p < \alpha'$ then
15:    $p_v \leftarrow p'$
16:    $d_v \leftarrow d' + 1$
17:    $update\_neighbors()$
18: end if
```

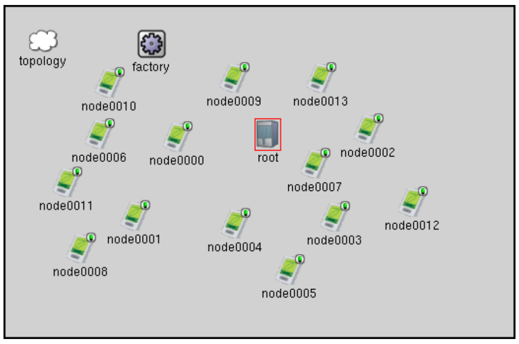
ibility in selecting parents that have not exhibited faulty behavior. However, the aggregation tree may degenerate from the best case of a BFS spanning tree to a taller one, since distances advertised do not necessarily correspond to the minimal ones. Nevertheless, the graph will remain a connected valid tree by the invariants enforced in the protocol.

## 5.5 Protocol Validation

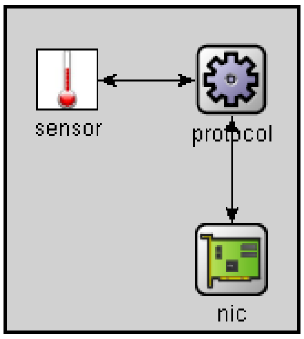
We implement protocols and carry out experiments using the `OMNeT++` discrete event simulation system (Varga, 2001). A screenshot of a S-GAP scenario in OMNeT++ running in GUI mode is shown in Figure 5.7(a), and the compound object representing an observation node in Figure 5.7(b). Communications and graph generation are handled by the *Dynamic Random Graph Simulator Library*, DRGSimLib[1] (Jónsson, Vigfússon, & Helgason, 2012), which consists of a set of components that enable construction of dynamic simulation scenarios.

A `factory` component dynamically instantiates and manages lifetime of simulated node objects in a simulation scenario. The factory component of DRGSimLib is an extension of earlier work by Helgason and Jónsson (2008).

[1] The components are available at `https://github.com/kristjanvj/DRGSimLib` and released under a open source license.



(a)



(b)

**Figure 5.7:** *The OMNeT++ simulation environment. Figure (a) shows a screenshot of a simulation scenario running in GUI mode in the OMNeT++ development environment, whereas Figure (b) shows a the sub-components of a sensing and aggregation node in the scenario.*

A `topology` component (shown in Figure 5.7(a)) assembles communications graphs dynamically upon registration of factory-generated modules. Conversely, the `topology` component removes nodes from the communications graph at the end of their life-time. `topology` uses a set of *generators*, derived from the `BasicGenerator` class and implementing the `IBasicGenerator` interface, to construct graphs. Examples described by Jónsson, Vigfússon, and Helgason include generators for Erdös-Rényi (Erdös & Rényi, 1959) and scale-free Barabási-Albert (Barabási & Albert, 1999; Barabási & Bonabeau, 2003) random graphs.

### 5.5.1 S-GAP Implementation

We base our implementation of S-GAP on a `OMNeT++` implementation of GAP described by Jónsson, Vigfússon, and Helgason (2012). The S-GAP protocol is implemented as a `OMNeT++` *simple module*. The protocol module implements a generic `IProtocol` interface, which plugs into an *observation node module*, as shown in Figure 5.7(b). Periodic updates of local inputs are generated by a *sensor module*.

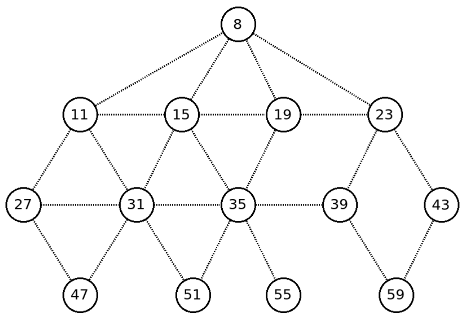
A simulation scenario (an example is shown in Figure 5.7) contains three fixed nodes, `factory` and `topology`, as described earlier, and a single `querier` node. A number of observation nodes are then dynamically instantiated by the `factory` and assembled into a connected communications graph by the `topology` component.

The S-GAP protocol is initiated by the querier after the initial topology has been constructed. A query flooding phase distributes query parameters to the network and builds the initial spanning tree at the same time, establishing the querier as root. Nodes generated after protocol initiation receive a cached copy of the query from their neighbors as part of the initial post-discovery handshake process.

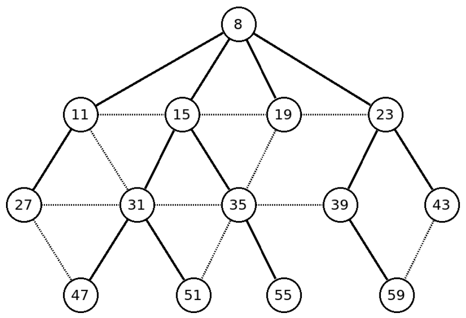
## 5.5.2 Completeness in a Small Fixed-topology Graph

Let us now turn our attention to the property of completeness and how the S-GAP protocol performs with respect to reducing the overall adversarial influence factor.

The trials described in this section are executed over a graph of fourteen nodes, as shown in Figure 5.8(a), using the `OMNeT++` implementation of S-GAP and `DRGSimLib`. The node population is pre-determined and static in this trial for ease of analysis. A network tree is constructed as shown in Figure 5.8(b). The construction of the tree depends on the initial message distribution, whose sending times and delays are drawn from a random distribution. However, a constant random seed is used in the following trials, resulting in a consistent initial spanning tree.



(a) Network graph $G$ over fourteen nodes. Node 8 is the implicitly honest querier. Nodes 15 and 19 are corrupt.



(b) Initial configuration of the tree.

**Figure 5.8**

**The tree state query protocol.** GAP is a continuous aggregation protocol, designed to propagate local state updates as quickly as possible to the querier. However, it is not suited to producing a concise state snapshot over the tree. The *ECHO protocol* (Adam et al., 2005) is a better candidate for the task. In the following trials, we run GAP to aggregate continuous updates and maintain the aggregation tree, but query the tree state periodically using a separate broadcast/convergecast protocol similar to ECHO: the querier broadcasts a query message which is broadcast to all children. Each recipient in turn forwards a copy of the query message to all its children in the current GAP tree. The convergecast phase is initiated once the query reaches a leaf node, i.e. one with no designated children. During the convergecast phase, nodes marked as corrupt increment (honestly) the current partial aggregate count to include the size of their current sub-tree – their current sphere of influence. The aggregate result received by the querier is a honest representation of the number of nodes controlled directly or indirectly by the adversary in terms of message delivery.

#### 5.5.2.1 No Adversarial Avoidance

Let us now explore the issue of completeness abstractly, without considering the effects of an incomplete set of results on the computed aggregate. In this scenario, we have $t = 2$ (two corrupt nodes) out of $n = 13$ (excluding the querier) nodes. Node 15 is corrupted at T=600s and node 19 at T=800s. The adversarial influence factor is plotted versus time in Figure 5.9. Recall, the adversarial influence factor is a measure of the potential impact an adversary *can* have on the modeled graph, regardless of past or present actions.

The adversarial influence is explored, given the static tree shown in Figure 5.8(b). When node 15 is corrupted at time $T = 600s$, the adversary controls a subtree of size six, which we can quantify as a ratio of nodes potentially under his influence to the total number of nodes, $6/13 = 0.46$. At time $T = 800s$, the influence factor is increased to $7/13 = 0.54$ with the inclusion of node 19 in the set of corrupt nodes. Analysis reveals $7/13$ to be the worst case adversarial influence for the graph instance shown in Figure 5.8(a). The average adversarial influence factor over the simulated run (from the first corruption) is $53.5\%$, close to the worst-case of $54\%$.

#### 5.5.2.2 Random Parent Switching

The initial spanning-tree, shown in Figure 5.8(b), represents a worst-case scenario in terms of the adversarial influence due to corruption of both nodes 15 and 19. Several tree configurations are possible that reduce the influence factor to the lowest achievable $4/13 = 0.31$ for the particular communications graph examined. Hence, one may explore the effects of selecting parent nodes at random, in an effort to utilize better configurations. The results of a single simulation run of such an experiment is shown in Figure 5.10. In the time interval $T = [600s, 800s]$, several configurations give an influence factor $\in [1/13, 6/13]$. After $T = 800s$, the adversarial influence factor is $[4/13, 7/13]$. This particular run gives an average adversarial influence over the simulation time of $36.7\%$, which is a reduction from the worst case represented in Figure 5.9. Examining the mean

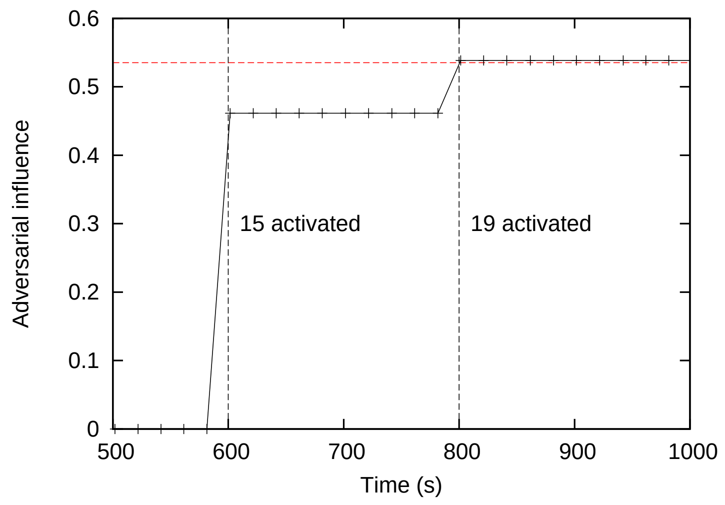


**Figure 5.9:** *Adversarial influence factor vs. simulation time for the spanning tree configuration shown in Figure 5.8(b). Node 15 is corrupted at $T = 600s$ and node 19 at $T = 800s$. No adversarial avoidance algorithm is employed (static graph configuration), resulting in a mean adversarial influence of 53.5% as shown by the horizontal line.*

adversarial effect over 30 simulation runs with different random seeds, as shown in Figure 5.10, confirms this ballpark figure, with slightly tighter confidence intervals as the frequency of parent reassignment is increased. However, the cost in this particular protocol is increased messaging, as shown in Figure 5.11(b), exponential with increased parent reassignment probability. The messaging cost can be reduced by protocol optimizations, but the random avoidance solution remains unsatisfactory in terms of guarantees of completeness.

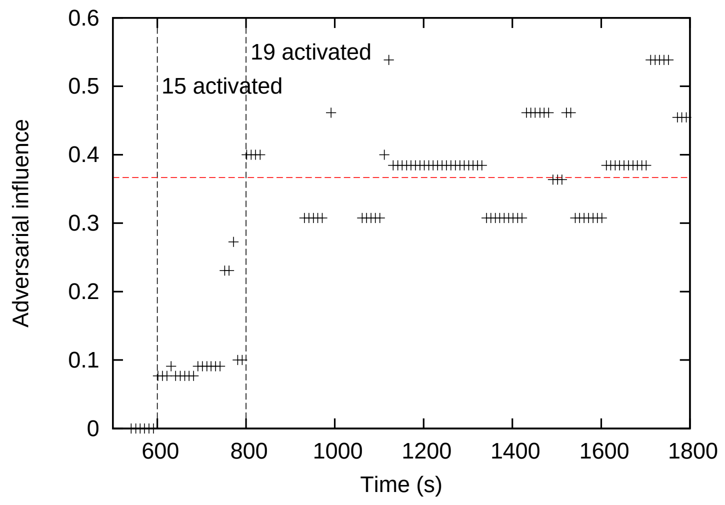
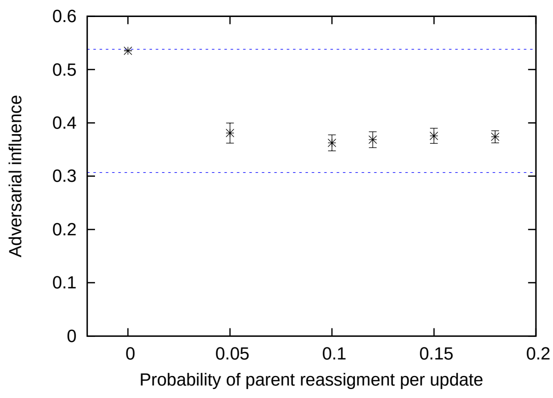
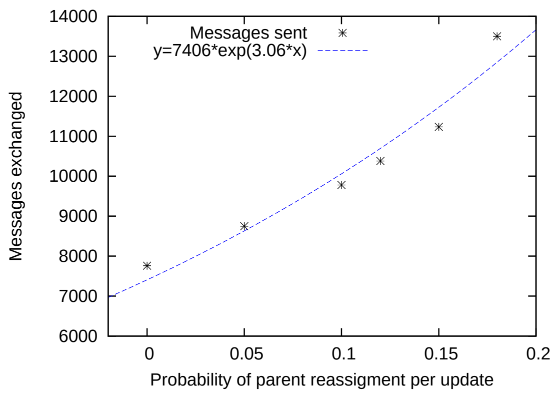


**Figure 5.10:** *Adversarial influence factor vs. simulation time for random reconfigurations (parent elections) in the communications graph of Figure 5.8(a). The results of a single simulation run are shown. Node 15 is corrupted at $T = 600s$ and node 19 at $T = 800s$. Parents are reassigned per state table update with a probability $p = 0.1$.*



(a) Average potential adversarial influence.



(b) Messaging cost for the GAP protocol vs. parent reassignment probability.

**Figure 5.11:** *Adversarial influence factor for random reconfigurations (parent elections) in the communications graph of Figure 5.8(a). Results are averaged over 30 statistically independent runs. Node 15 is corrupted at $T = 600s$ and node 19 at $T = 800s$. Parents are reassigned per state table update with a probability $p = 0.1$.*

#### 5.5.2.3 Avoiding Corrupt Nodes – First Approach (Switching Policy II)

We now employ Protocol 5.2 to select alternative parents based on peers fault history. As before, node 15 is corrupted at T=600s and node 19 at T=800s. Corrupt nodes drop update messages with a probability $p_{drop} = 0.1$. The results of a single run are shown in Figure 5.12.

The edge rating $\alpha$ is updated using the functions

$$\alpha = f_F(\alpha) = \text{MIN}(1, \alpha - \gamma)$$

$$\alpha = f_R(\alpha) = \text{MIN}(1, \alpha \cdot \varrho)$$

where $0 < \gamma \leq 1$ and $\varrho > 1$ are constants allowing tailoring of the penalization upon errors, as well as the “grudge” held against a faulty edge. A fault on the channel (from the perspective of $a$) will reduce the rating, while a successful transmission can increase it up to a maximum of one. The constants used to update $\alpha$ for this run are $\gamma = 0.25$ and $\varrho = 1.15$.

At $T = 600s$, the adversary effectively controls a subtree of size six, and hence, has a potential influence factor of $6/13$. A fault is detected at $T = 734s$ when node 15 drops a single message, as shown in the log excerpt given in Figure 5.13. The single fault causes node 31 to select node 11 as its alternative parent, thereby immediately reducing the potential adversarial influence by three. A second switch at $T = 787s$ reduces the

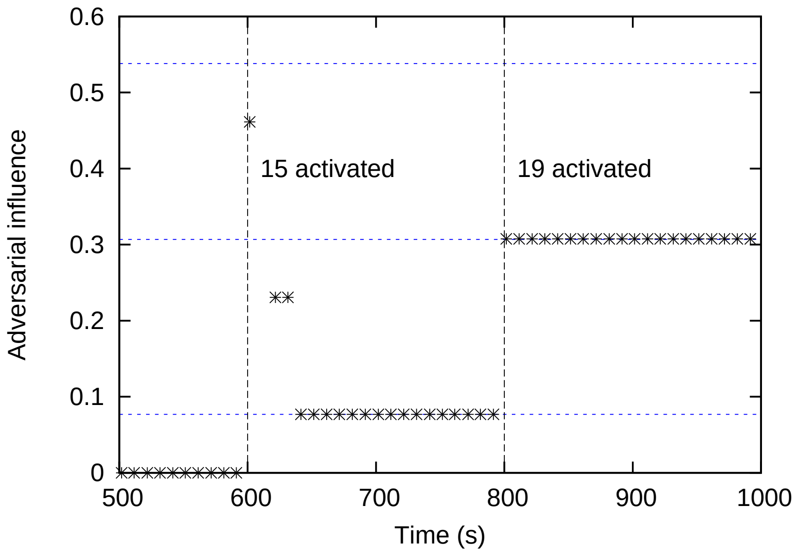


**Figure 5.12:** *Adversarial influence factor. Faults are monitored locally by each participating node (trusted module). Nodes use Protocol 5.2 to select the current best $\alpha$ peer as parent from the set of nodes that advertise shorter distance to the querier (proper parents in the spanning tree).*

```
 734.950547 SWITCH      31 15 11 PARENTPEER 0.800000 1.000000
 787.139119 SWITCH      35 15 19 PARENTPEER 0.800000 1.000000
 990.479567 SWITCH      35 19 15 PARENTPEER 0.800000 1.000000
 992.088208 SWITCH      35 15 19 PARENTPEER 0.800000 1.000000
1496.947260 SWITCH      35 19 15 PARENTPEER 0.600000 0.800000
1592.660109 SWITCH      35 15 19 PARENTPEER 0.600000 0.780000
```

**Figure 5.13:** *Excerpt from fault detection log. Algorithm 5.2 (switching policy II) used to select best $\alpha$ parent. Drop probability is 10%. First column is simulation time, while third, fourth and fifth columns represent node, previous parent and new parent, respectively. Finally, the current $\alpha$ for the previous and newly selected parent are shown in the last two fields.*

adversarial influence to the minimum of one for the $t = 1$ nodes corrupted at that time. Node 19 is corrupted at $T = 800s$ and begins to drop messages with 10% probability. Now, the adversary controls a sub-graph of at least size four, the corrupt nodes 15 and 19 as well as nodes 35 and 55 which have no recourse under this protocol but to select either nodes 15 or 19 as parents. Hence, the lowest achievable potential adversarial influence factor is $4/13$. However, the protocol attempts to maximize completeness by selecting the best possible current parent: as can be seen in Figure 5.13, node 35 continues to switch between potential parents 15 and 19 during the run, based on the current best $\alpha$ rating.

Figure 5.14 shows an excerpt from another run with $p = 0.25$ drop probability. The parameters for the $\alpha$ computation are $\gamma = 0.25$ and $\varrho = 1.15$. As before, node 15 is corrupted at $T = 600s$ and 19 at $T = 800s$. $\alpha_{(35,15)}$ and $\alpha_{(35,19)}$ are plotted, as observed by node 35. Switch-over points for potential parents 15 and 19, w.r.t. node 35, are shown.

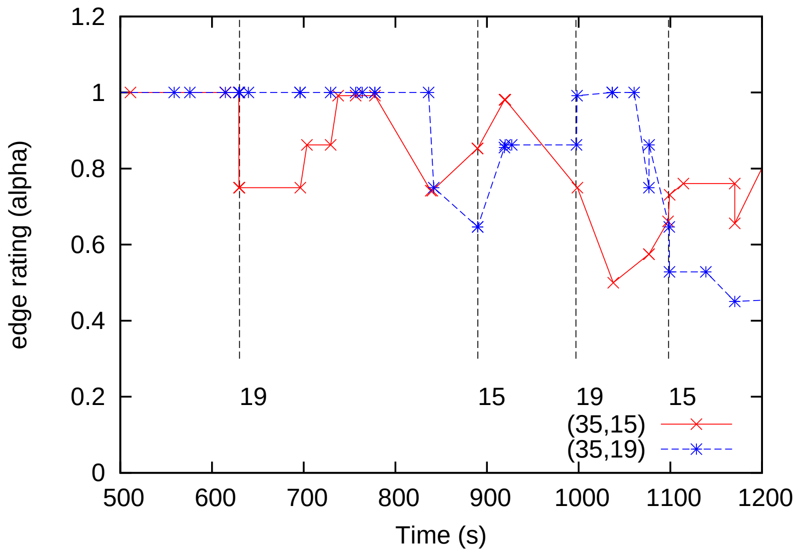


**Figure 5.14:** *Progression of* $\alpha_{(35,15)}$ *and* $\alpha_{(35,19)}$ *as observed by node 35. Switch-over points for potential parents 15 and 19 are shown.*

Observe how node 35 selects the best current $\alpha$ parent during the run at the time updates are available.

The $\alpha$ rating of an edge is extended to a path reliability rating by each node feeding back its current parent reliability rating to its peers, as discussed in Section 5.4.3.1. The $\alpha$ feedback mechanism is demonstrated in Figure 5.15. A series of three send faults are generated on the $(15, 8)$ edge at $T = 1200s$, while all other edges are fault-free in this experiment. The $\alpha_{(15,8)}$ rating is immediately reduced by the trusted module of node 15. The fault is reported back to node 35 on the first $15 \to 35$ update, which reflects at that time in the $\alpha_{(35,15)}$ rating. In turn, this affects the $\alpha_{(55,35)}$ on the next $35 \to 55$ update. The alpha rating is gradually increased on subsequent successful transfers. The feedback delay is beneficial in the protocol, promoting switching to better parents nearest to the fault. In this case, node 15 would have the first opportunity to select a new parent, and if successful, report a better $\alpha$ to downstream peers.

#### 5.5.2.4 Avoiding Corrupt Nodes – Second Approach (Switching Policy III)

Protocol 5.2 represents a minor modification of the original GAP protocol and retains its properties in terms of BFS spanning tree construction. In contrast to randomized parent selection, this protocol does achieve a clear reduction in potential adversarial influence. We can do even better by relaxing the parent selection rules to select from a larger set of potential parents, for instance by employing Protocol 5.3. In the example considered here, we can bypass corrupt nodes entirely, given that they do reveal themselves and

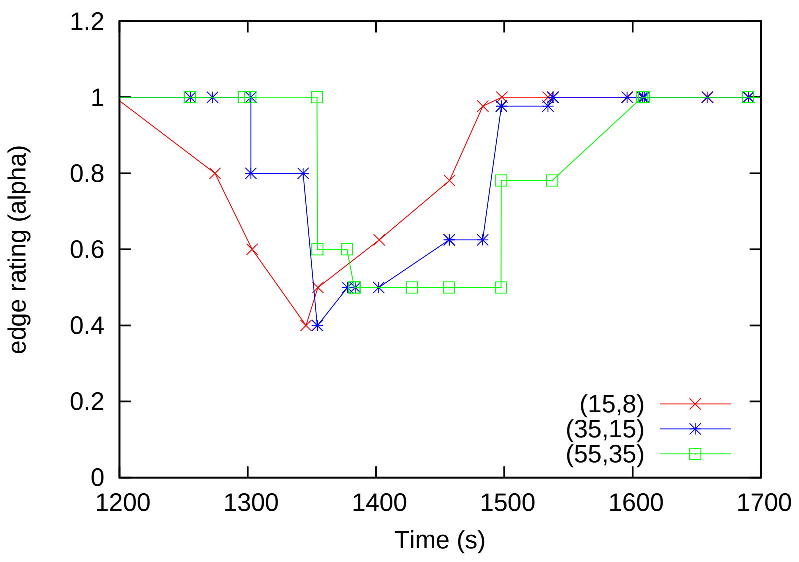


**Figure 5.15:** *Demonstration of the $\alpha$ feedback mechanism. A series of three send faults are triggered at $T = 1200s$ on the $(15, 8)$ edge. All other edges are fault-free.*

act maliciously. Node 35, occluded by the adversary in the previous section, can use the parent selection rules of Protocol 5.3 to elect either of its honest peers 31 or 39 as parent.

The results of two simulation runs using Protocol 5.3 are shown in Figure 5.16 for drop probabilities $p = 0.1$ and $p = 0.2$. An excerpt from the fault detection log for $p = 0.1$ is shown in Figure 5.17. In the case of $p = 0.1$ drop probability, the adversarial influence is reduced to the current minimum of $1/13$ shortly after corruption of node 15 at $T = 600s$. The adversarial influence is increased briefly once node 19 is corrupted at $T = 800s$. However, node 35's selection of alternative parent 31 lowers the influence to the current minimum at that time of $2/13$. Note that node 15 is temporarily *redeemed* at T=1227s and the potential adversarial influence factor temporarily increased accordingly. However, upon detecting fault on the $(15, 35)$ edge, node 35 once again selects node 35 as alternative parent. In the case of the heavier 20% drop probability, the faulty behavior of node 15 causes node 35 to stick with its alternative parent node 31.

**Using positional information.** The protocols described above can be optimized by taking the positioning of the nodes in the network graph into account. In GAP, a fist level trusted node that experiences loss events can determine with full certainty that it is its own host that is malicious, as the counterpart, the querier, is by definition honest. However, this assumption is only valid for the particular adversarial model in which the querier is honest. If we assume this fact, then first level nodes should not waste effort on trying to

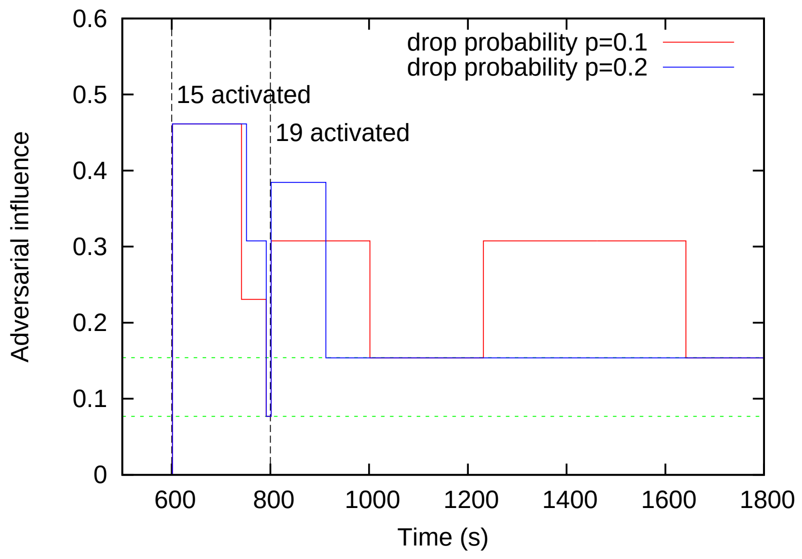


**Figure 5.16:** *Adversarial influence factor. Faults are monitored locally by each participating node (trusted module). Nodes use Protocol 5.3 to select the current best $\alpha$ peer as parent from the set of nodes that advertise shorter or equal distance to the querier.*

```
  734.950547 SWITCH      31 15 11 PARENTPEER 0.750000 1.000000 1 1
  787.139119 SWITCH      35 15 19 PARENTPEER 0.750000 1.000000 1 1
  990.479567 SWITCH      35 19 31 PEER       0.750000 1.000000 1 2
 1227.732792 SWITCH      35 31 15 UNASSIGNED 1.000000 1.000000 2 1
 1627.843370 SWITCH      35 15 31 PEER       0.750000 1.000000 1 2
```

**Figure 5.17:** *Excerpt from fault detection log. Protocol 5.3 (switching policy III). Nodes use Protocol 5.3 to select the current best $\alpha$ peer as parent from the set of nodes that advertise shorter or equal distance to the querier. Drop probability is $p = 0.1$.*

obtain reliability by choosing alternative peers other than the querier. This knowledge is used in the previous runs to reduce the switching noise in the simulation.

Path length can be weighted into routing decisions, which helps the protocol in terms of efficiency. In terms of security, the path length does not have an impact. However, accuracy (timeliness of information) is increased as path lengths are shortened.

### 5.5.3 Completeness in a Kleinberg Random Graph

Let us now consider how the protocols perform in a random graph. In the simulations that follow, the `factory` component selects a number of nodes uniformly at random and marks as corrupt. Corrupt nodes drop update messages with some probability $p$. A random communications graph is then constructed, using `DRGSimLib`.

#### 5.5.3.1 Kleinberg Graph Construction

We have previously described two types of random graph generators implemented in the `DRGSimLib` set of components. For our subsequent trials, we use a third type of generator, `KleinbergGenerator`, that constructs Kleinberg topologies (Kleinberg, 2000). The Kleinberg algorithm takes a number of nodes and constructs a square $n \times n$ grid by adding edges, *local contacts*, to all neighbors within lattice distance $p \geq 1$, as shown in Figure 5.18. We restrict the initial grid topology to odd $n$ and $p = 1$ in our implementation. After construction of the basic grid, the Kleinberg algorithm adds random "shortcuts" as follows: For each node $v$ we add $q$ long-range contacts with the probability of adding an edge $(u, v)$ being proportional to their distance over the grid $d(u, v)^{-r}$, where $r$ is the clustering exponent.

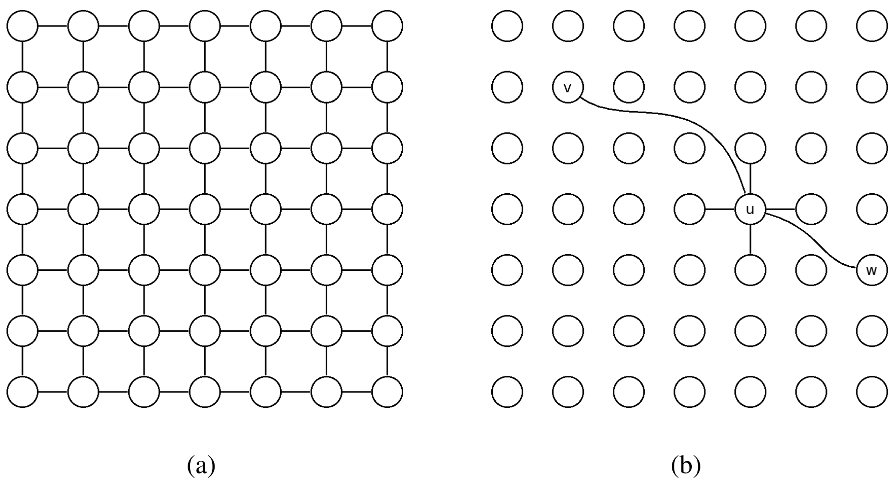
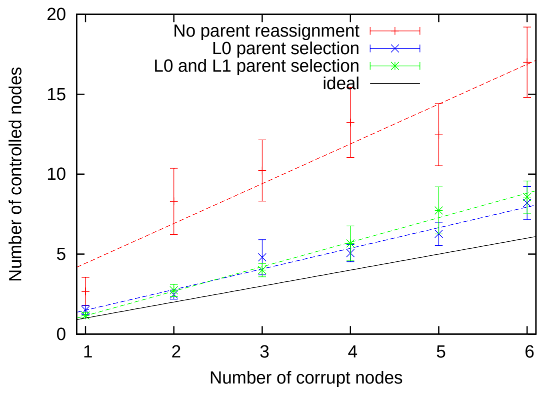


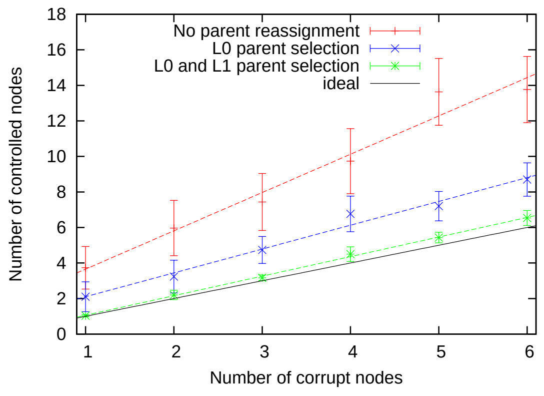
**Figure 5.18:** *Construction of a Kleinberg random graph. Figure (a) shows the initial lattice, in this example a $7 \times 7$ grid for $p = 1$. Edges in the graph are the initially generated local contacts. Figure (b) shows examples of local and long-range contacts of a node $u$ for $q = 2$.*

#### 5.5.3.2 Comparing Parent Selection Policies II and III

Figure 5.19 shows a sample run for a $7 \times 7$ Kleinberg graph. In the experiment shown in Figure 5.19(a), we construct a series of pure $7 \times 7$ grids, that is, lattice graphs with no random shortcuts. The random factor in this case is solely the distribution of corrupt nodes. The experiment represented in Figure 5.19(b) is conducted over randomly generated Kleinberg graphs with $q = 1$ and $r = -4.0$, producing a relatively sparse graph with a high clustering coefficient. The results of 30 statistically independent runs were averaged to produce the results shown.

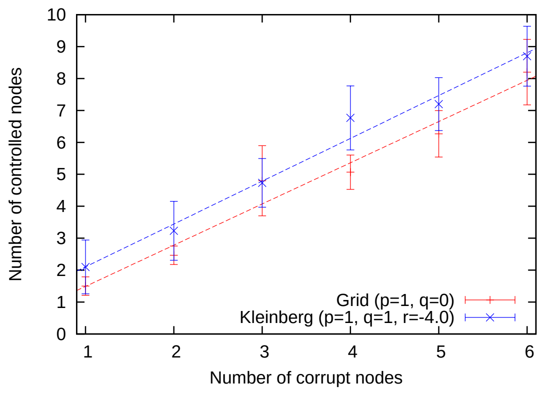


(a) Pure grid ($p = 1$, $q = 0$)

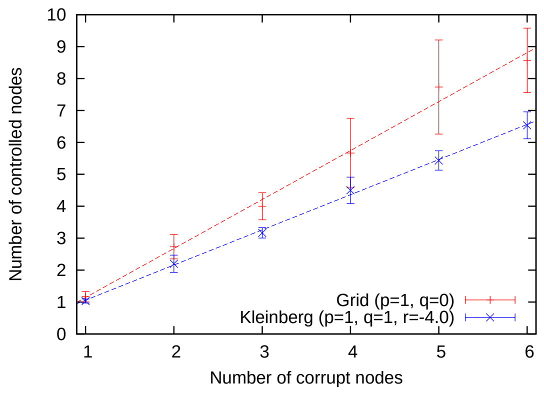


(b) Clustered Kleinberg graph ($p = 1$, $q = 1$, $r = -4.0$)

**Figure 5.19:** *Adversarial influence over a $7 \times 7$ Kleinberg random graph. The figures show results for three algorithms, (i) the original GAP algorithm (no reassignment on faults), (ii) Protocol 5.2 (selection from the set of best $\alpha$ proper parents) and (iii) Protocol 5.3 (selection from same or lower level peers). Adversarial influence is given as the average of the total number of controlled nodes over 30 statistically independent runs. Error bars are 95% confidence intervals.*



(a) Algorithm 5.2



(b) Algorithm 5.3

**Figure 5.20:** *Adversarial influence over a $7 \times 7$ Kleinberg random graphs with $p = 1$, $q = 0$ (pure grid) and $p = 1$, $q = 1$, $r = -4.0$ (clustered random graph). Comparing the of performance of (a) Algorithm 5.2 and (b) Algorithm 5.3. Adversarial influence is given as the average of the total number of controlled nodes over 30 statistically independent runs. Error bars are 95% confidence intervals.*

In the pure grid graph experiment, we can see that the potential adversarial influence increases with the number of corrupt nodes, as would be expected, in the case of the original algorithm (no parent reassignment). Algorithms 5.2 and 5.3 produce a significant improvement. However, the test is inconclusive[2] with regards to the relative performance

[2] Statistical significance was determined with a paired t-test, computed with the R statistical package (R Development Core Team, 2011). 99% confidence level was used. We also used the visual approximate test described by Jain (1991, pp. 211).

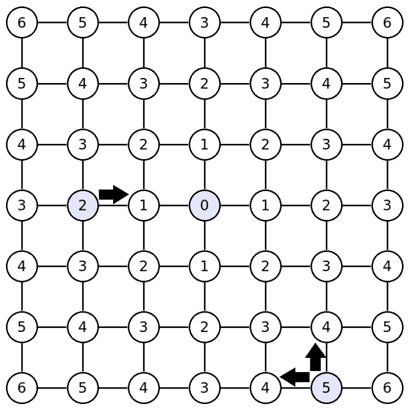


**Figure 5.21:** *Basic $p = 1$ Kleinberg grid with annotated distances (in lattice steps) to the root. The node at level two has a single lower level neighbor and the node at level five has two. In no case do nodes have same level peers.*

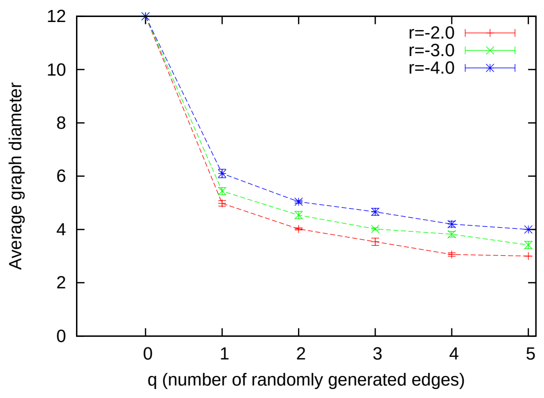
of Algorithms 5.2) and 5.3. From Figure 5.21, we can see that any given node has either one or two alternative parents, but no same-level peers. Hence, Algorithm5.3 may be expected to be ineffective in this case, as no same level peers are available.

In the second experiment, represented in Figure 5.19(b), we add random links, the probability of a node $v$ linking to a node $u$ being proportional to $d(u, v)^{-4.0}$. Hence, we expect relatively short random links for a high clustering coefficient. This added randomness introduces the possibility of selecting same level neighbors as parents, which reflects in the statistically significant improvements ($p = 0.0006095$ for a paired t-test with 99% confidence level) of Algorithm 5.3 over 5.2 in this case, as can be seen from Figure 5.20(b). Further, note that Algorithm 5.3 achieves close to the ideal minimum adversarial influence of $t/|\boldsymbol{V}|$, where $t$ is the number of corrupt nodes.

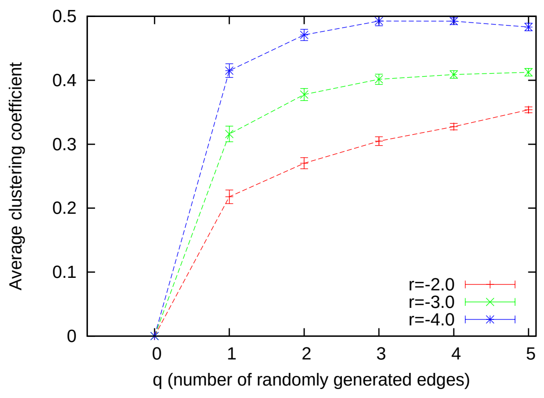
#### 5.5.3.3 Effects of Topology on Adversarial Influence.

As expected, the $q$ and $r$ parameters have a significant influence over the topology of the Kleinberg random graph, as shown for a range of parameters in Figure 5.22. The average diameter (Figure 5.22(a)) can be seen to be strongly dependent on the clustering coefficient $r$, since lower $r$ causes long links to be more probable. The average clustering is also dependent on the distribution of random link lengths.

We would expect the graph diameter to have an impact on the adversarial influence. Consider a binary tree of size $n = |\boldsymbol{V}|$. In the worst case, a single corrupt node can control half of the tree. If we modify the graph to a $k$-ary tree, the potential worst and average case adversarial influence is expected to decrease, since each corrupt node controls a shallower sub-tree. At the extreme of a star graph ($k = n - 1$), each corrupt node controls



(a) Average diameter



(b) Average clustering

**Figure 5.22:** *Topology analysis of a* $7 \times 7$ *Kleinberg random graph* ($p = 1$)*. Results are averages over 30 statistically independent runs. Error bars are 95% confidence intervals.*

only itself, that is, adversarial influence is reduced to the minimum of exactly $t/n$, the ratio of corrupt nodes to the total node population.

Figure 5.23 demonstrates the effect of running the basic GAP algorithm (no avoidance) over two Kleinberg graph topologies, a pure grid ($p = 1$, $q = 0$) and a clustered graph with $p = 1$, $q = 1$ and $r = -4.0$. The clustered graph has a considerably lower diameter (≈6 compared to 12) due to the random links, and hence, a lower adversarial influence would be expected. Examining Figure 5.23 gives an indication that this may be the case, but the results are not statistically significant. We examine this further in an experiment depicted in Figure 5.24, processing a dataset of simulation runs for $t = \{1, \ldots, 6\}$, $q = \{1, 2, 3\}$, $r = \{-2.0, -3.0, -4.0\}$ and 30 statistically independent runs executed for each combination. Again, vague trends towards decreasing adversarial influence with decreasing diameter can be observed, but the predominant effect is $t$, the number of corrupt entities. We conclude that depending on a small average graph diameter as the sole means of decreasing the adversarial influence is not a promising approach.

## 5.6 Concluding Remarks

We consider the problem of guaranteeing completeness in distributed aggregation networks, an important property in the case of in-network aggregation, since malicious nodes can affect the completeness of the set of contributions of honest ones. We begin by arguing for the impossibility of guaranteeing completeness by any proactive means, including the expensive measure of forwarding on all available paths.

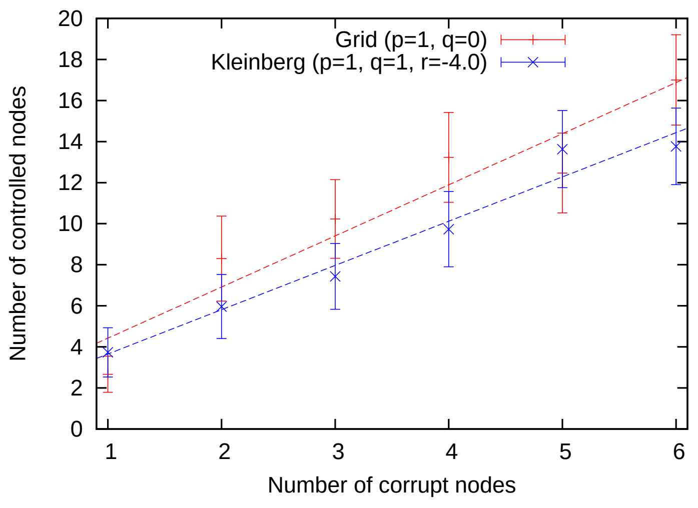


**Figure 5.23:** *Adversarial influence over a* $7 \times 7$ *Kleinberg random graph with* $p = 1$, $q = 0$ *(pure grid) and* $p = 1$, $q = 1$, $r = -4.0$ *(clustered random graph). Comparison of the effects of topology on the adversarial influence when executing the basic GAP algorithm (no adversarial avoidance). Adversarial influence is given as the average of the total number of controlled nodes over 30 statistically independent runs. Error bars are 95% confidence intervals.*

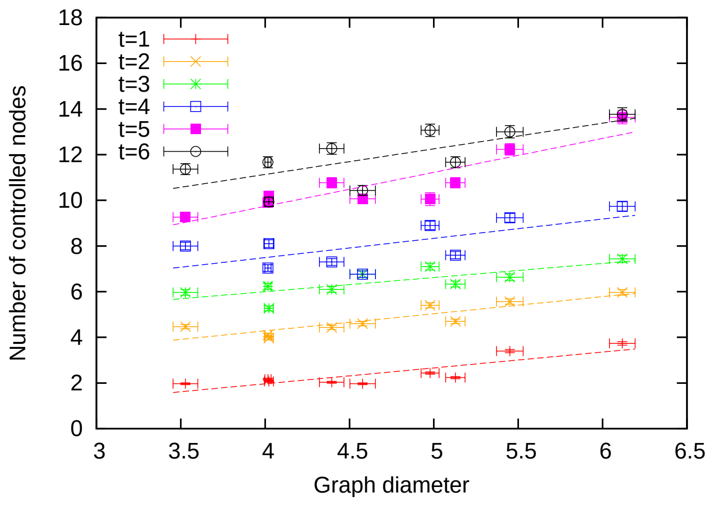


**Figure 5.24:** *Adversarial influence over a series of* $7x7$ *Kleinberg graphs, for configurations with* $q = \{1, 2, 3\}$ *and* $r = \{-2.0, -3.0, -4.0\}$*. The number of corrupt nodes is* $t = \{1, \ldots, 6\}$ *for each graph configuration. Adversarial influence is given as the average of the total number of controlled nodes over 30 statistically independent runs. Error bars are 95% confidence intervals.*

We proceed by considering the mechanisms necessary for a failure detection and response protocol. A case study of securing the GAP protocol is presented, demonstrating the application of the trusted modules concept from the previous sections combined with light-weight means of reducing the influence of potentially corrupt nodes in terms of completeness. The adversarial influence is reduced by avoiding nodes that have dropped messages. To this end, a failure detection protocol is executed over the trusted layer, whose correctness is guaranteed by the same principles as discussed in the preceding chapters. The protocol is inexpensive, based solely on the straight-forward mechanisms of acknowledgements and sequence numbering of messages. We support the protocol design by simulation trials, whose results indicate that a significant reduction in the potential adversarial influence can be achieved.

# Chapter 6

# Conclusions

We have discussed the various aspects of secure in-network aggregation in distributed systems in an attempt to answer our original research question: *Can a dynamic aggregation system composed of several data contributors and aggregators in the hands of untrusted entities be secured to give sufficiently strong integrity guarantees?* Having presented the material in this dissertation, the answer is a qualified yes. We can indeed give strong security guarantees in terms of the integrity sub-goal of *correctness*. However, the *completeness* sub-goal is in many respects a more difficult one. In fact, no practical approach appears likely that can guarantee aggregate completeness. However, the correctness attack is definitely the more powerful of the two, and, hence, more important to prevent.

Our primary goal was to present means of securing in-network aggregation in the *general case*, in particular to ensure end-to-end integrity of aggregate data in the event of corruption of one or more participants in such a system. Hence, we searched for efficient solutions, broadly applicable to a wide range of networked systems and aggregation applications. In particular, we searched for solutions applicable to a wide range of input data types and aggregation functions, as well as arbitrary network transports, topologies and aggregation protocols. Furthermore, our goal was to support arbitrary dynamic networked systems. This set of requirements placed restrictions on the range of possible solutions.

We analyzed the problem of aggregation integrity, beginning from a centralized single honest aggregator model and progressing to a hierarchical tree-based model, widely used in distributed aggregation applications. Two separate problems were examined: those of *data source* and *aggregate* integrity. In the case of networked measurement systems, the data source integrity problem relates to the trustworthiness of original observations, which is hard to establish, even with corroborating evidence. Hence, we turned to the established principles of trusted systems theory to provide an anchor of trust. Specifically, we proposed a *trusted sensor* – a minimal self-contained hardware device that, given some integrity assumptions, can guarantee correctness and verifiability of source data. The trusted sensors concept was supported by a proof-of-concept prototype of a client/server measurement system, composed of general-purpose untrusted nodes augmented with trusted sensors. The trusted sensors concept was then extended to the problem of trusted in-network aggregation in arbitrary dynamic networked systems. A trusted aggregator is a generalization of a trusted sensor – a dedicated device that extends the trusted functionality to correct evaluation of aggregation functions. Given trusted sensors and aggregators, we can construct a trusted overlay over a system composed of arbitrary untrusted observation nodes. The trusted overlay delivers correct results by virtue of transitive trust relations, from the trusted data sources (trusted sensors) to the implicitly trusted querier (recipient of aggregate information). The trusted aggregation concept was supported by a design for a tree-based aggregation system.

Given a trusted overlay of sensors and aggregators, we can claim aggregate correctness. The complementary goal of completeness is more difficult, if only for the fact that losses due to link faults and churn are commonplace in dynamic networked systems. Rather than attempting the likely futile task of guaranteeing aggregate completeness, we investigated light-weight means of increasing completeness, while maintaining the efficiency of the underlying aggregation protocol. To this end, we outlined a simple protocol that can increase completeness by monitoring anomalies and avoiding suspicions nodes. A case study of S-GAP, a secure version of the Generic Aggregation Protocol (GAP), was presented, which applies trusted systems principles to ensure correctness and the proposed adversarial avoidance measures to increase completeness. The protocol was shown by means of simulations to produce a significant reduction in the potential influence an adversary can wield in terms of aggregate completeness, while imposing a small overhead on the underlying aggregation protocol.

## Future directions

As this project draws to a close, the ways in which this work can be extended appear limitless. Let us briefly touch on a few directions for future work. The TSense system can be developed more fully and a prototype of a full distributed system produced. The trusted sensor design requires fuller consideration of tamper-proofing and packaging of the trusted sensor, including measures to secure against side-channel attacks. A full prototype of a tamper-proof trusted sensor is a challenging but necessary next step in the development of a trusted sensing system. Formal specification and verification of the functionality and security of trusted modules is a crucial task for which techniques and procedures should be developed, including the trusted manufacturers certification procedure. The cryptographic protocols used in the TSense system should be analyzed more rigorously with regards to their security and efficiency. Formal verification of a subset of the presented protocols was carried out as student projects, using both the ProVerif and Avispa tools. The results indicate that the presented versions of the protocols are secure against key discovery and data alterations. Efficiency of protocols and cryptographic primitives should be analyzed and optimized, as both are crucial issues in resource constrained systems.

We proposed the use of Physically Unclonable Functions as a means of preventing simulation attacks against trusted aggregation systems in the event of key discovery. Our proposed protocols and their analysis assumes the PUF used to behave like a random oracle. Further work is required to use a PUF securely in a real implementation. For instance, one must consider the entropy provided by the PUF used, as well as the repeatability of the measurements on which the output is based. Further, the feasibility of the PUF solution in terms of the storage requirements imposed by the required challenge/response pairs must be addressed.

The trusted aggregation work presented assumed trusted hardware devices. This is a robust but yet limiting choice, as the set of functions that can be supported in such a system is by definition pre-determined. A more flexible solution based on security-oriented hypervisors was discussed briefly as an alternative to the approach of using dedicated hardware devices. This should be explored further in future work. Methods for constructing and attesting trusted software, and the distribution of trusted software modules, need to be addressed as part of this work.

# Bibliography

Aceto, L., Fokkink, W., & Verhoef, C. (2001, February). Structural operational semantics. In J. Bergstra, A. Ponse, & S. Smolka (Eds.), *Handbook of process algebra* (pp. 197–292). Elsevier.

Aceto, L., Ingólfsdóttir, A., Larsen, K. G., & Srba, J. (2007). *Reactive systems: Modelling, specification and verification*. Cambridge University Press.

Adam, C., Lim, K., & Stadler, R. (2005, December). *Decentralizing network management* (Tech. Rep.). KTH.

Adamic, L. A., & Huberman, B. A. (2002). Zipf's law and the internet. *Glottometrics*, *3*, 143–150.

Aiyer, A. S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.-P., & Porth, C. (2005, October). BAR fault tolerance for cooperative services. In *SOSP'05 20th ACM Symposium on Operating Systems Principles* (p. 45-58). Brighton, United Kingdom.

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002, March). Wireless sensor networks: A survey. *Computer Networks*, *38*(4).

Ames, S. R., Jr. (1981). Security kernels: A solution or a problem? *IEEE Symposium on Security and Privacy*, *0*, 141. doi: http://doi.ieeecomputersociety.org/10.1109/SP.1981.10016

Amin, S. M., & Wollenberg, B. F. (2005, sept.-oct.). Toward a smart grid: power delivery for the 21st century. *IEEE Power and Energy Magazine*, *3*(5), 34 - 41. doi: 10.1109/MPAE.2005.1507024

Anderson, J. P. (1972, October). *Computer security technology planning study* (Tech. Rep. No. ESD-TR-73-51). U.S. Airforce Electronic Systems Division, Deputy for Command and Management Systems, HQ Electronic Systems Division.

Anderson, R. (2004). Cryptography and competition policy - issues with 'trusted computing'. In L. Camp & S. Lewis (Eds.), *Economics of information security* (Vol. 12, p. 35-52). Springer US. Retrieved from `http://dx.doi.org/10.1007/1-4020-8090-5_3`

Anderson, R., & Kuhn, M. (1996). Tamper resistance – a cautionary note. In *USENIX* *workshop on electronic commerce* (pp. 1 – 11).

Anderson, R., & Kuhn, M. (1997). Low cost attacks on tamper resistant devices. In *Proceedings of the 5th international workshop on security protocols* (pp. 125–136). London, UK: Springer-Verlag. Retrieved from `http://portal.acm.org/citation.cfm?id=647215.720528`

Arduino. (2009). *Arduino Duemilanove.* Retrieved from `http://arduino.cc/en/Main/ArduinoBoardDuemilanove` (Retrieved June 2012)

Arora, A., Dutta, P., Bapàt, S., Kulathumani, V., Zhang, H., Naik, V., ... Miyashita, M. (2004). A line in the sand: a wireless sensor network for target detection, classification, and tracking. *Computer Networks*, *46*, 605–634.

Ashton, K. (2009, June). That 'internet of things' thing. *RFID Journal.* Retrieved from `http://www.rfidjournal.com/article/view/4986`

Atmel. (2010). *Data sheet: Atmega48a/48pa/88a/88pa/168a/168pa/328/328p.* (Retrieved Sept. 2010)

Aurell, E., & Pfitzner, R. (2009, May). Gaussian belief with dynamic data and in dynamic network. *Europhysics letters*, *87*(6). Retrieved from `http://arxiv.org/abs/0905.0266`

*Avispa project.* (2012). Retrieved from `http://www.avispa-project.org/`

Avoine, G., & Vaudenay, S. (2003). Fair Exchange with Guardian Angels. In *The 4th International Workshop on Information Security Applications - WISA* (pp. 188–202). Retrieved from `http://www.springer.com/`

Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, *286*(5439), 509 – 512.

Barabási, A.-L., & Bonabeau, E. (2003). Scale-free networks. *Scientific American.*

Barbarán, J., Díaz, M., Esteve, I., & Rubio, B. (2007). RadMote: A mobile framework for radiation monitoring in nuclear power plants. *International Journal of Electrical and Computer Engineering*, *2*(10).

Bardou, R., Focardi, R., Kawamoto, Y., Simionato, L., Steel, G., & Tsay, J.-K. (2012, April). *Efficient padding oracle attacks on cryptographic hardware* (Tech. Rep. No. 7944). INRIA.

Barford, P., Kline, J., Plonka, D., & Ron, A. (2002). A signal analysis of network traffic anomalies. In *ACM SIGCOMM Internet measurement workshop.*

Barker, E., Barker, W., Burr, W., Polk, W., & Smid, M. (2007, March). *NIST special publication 800-57. recommendation for key management – part 1: General (revised).*

Batina, L., Mentens, N., Sakiyama, K., Preneel, B., & Verbauwhede, I. (2006). Low-cost elliptic curve cryptography for wireless sensor networks. In *Proceedings of the third european conference on security and privacy in ad-hoc and sensor networks* (pp.

6–17). Berlin, Heidelberg: Springer-Verlag. Retrieved from http://dx.doi.org/10.1007/11964254_3 doi: 10.1007/11964254_3

Bawa, M., Gionis, A., Garcia-Molina, H., & Motwani, R. (2007). The price of validity in dynamic networks. *J. Comput. Syst. Sci.*, *73*(3), 245–264. doi: http://dx.doi.org/10.1016/j.jcss.2006.10.007

Bekara, C., Laurent-Maknavicius, M., & Bekara, K. (2007). SAPC: A secure aggregation protocol for cluster-based wireless sensor networks. In *Mobile ad-hoc and sensor networks.* Springer Berlin / Heidelberg.

Bellare, M., & Namprempre, C. (2007). *Authenticated encryption: Relations among notions and analysis of the generic composition paradigm.* London, UK: Springer-Verlag.

Bellare, M., & Rogaway, P. (1993). Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on computer and communications security* (pp. 62–73). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/168588.168596 doi: 10.1145/168588.168596

Birman, K. P., & van Renesse, R. (2002). Scalable data fusion using Astrolabe. In *Proceedings of the Fifth International Conference on Information Fusion (IF).*

Biryukov, A., Dunkelman, O., Keller, N., Khovratovich, D., & Shamir, A. (2009). *Key recovery attacks of practical complexity on AES variants with up to 10 rounds.* Cryptology ePrint Archive, Report 2009/374. (http://eprint.iacr.org/)

Blanchet, B. (2009, July). Automatic verification of correspondences for security protocols. *Journal of Computer Security*, *17*(4), 363-434.

Blanchet, B. (2012). *ProVerif: Cryptographic protocol verifier in the formal model.* Retrieved from http://prosecco.gforge.inria.fr/personal/bblanche/proverif/ (Retrieved Nov. 2012)

Boneh, D., Gentry, C., Lynn, B., & Shacham, H. (2003). A survey of two signature aggregation techniques. *CryptoBytes*, *6*(2), 2003.

Bonneau, J., & Mironov, I. (2006). Cache-collision timing attacks against AES. In *Cryptographic hardware and embedded systems – CHES 2006* (pp. 201–215).

Boritz, J. E. (2005). IS practitioners' views on core concepts of information integrity. *International Journal of Accounting Information Systems*, *6*(4), 260 - 279. Retrieved from http://www.sciencedirect.com/science/article/pii/S1467089505000473 doi: 10.1016/j.accinf.2005.07.001

Boyd, S., Ghosh, A., Prabhakar, B., & Shah, D. (2006). Randomized gossip algorithms. *IEEE/ACM Trans. Netw.*, *14*(SI), 2508–2530. doi: http://dx.doi.org/10.1109/TIT.2006.874516

Breckenridge, S. J., R. A.; Katzberg. (1980). Smart sensors for the 80's - the status of smart sensors. In *Sensor systems for the 80's conference.*

Burke, J., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., & Srivastava, M. B. (2006). Participatory sensing. In *Workshop on world-sensor-web (WSW'06): Mobile device centric sensor networks and applications* (pp. 117–134).

Buttyán, L., Schaffer, P., & Vajda, I. (2006). Resilient aggregation with attack detection in sensor networks. In *PERCOMW '06: Proceedings of the 4th annual IEEE international conference on pervasive computing and communications workshops* (p. 332). Washington, DC, USA: IEEE Computer Society. doi: http://dx.doi.org/10.1109/PERCOMW.2006.119

Case, J., Fedor, M., Schoffstall, M., & Davin, J. (1990, May). *RFC 1157: Simple Network Management Protocol (SNMP).* Retrieved from `http://www.faqs.org/rfcs/rfc1157.html`

Castro, M., & Liskov, B. (1999, February). Practical byzantine fault tolerance. In *OSDI: Third symposium on operating systems design and implementation.* New Orleans, USA.

Challener, D., Yoder, K., Catherman, R., Safford, D., & van Doorn, L. (2008). *A practical guide to trusted computing*. IBM Press.

Chan, A.-F., & Castelluccia, C. (2008, July). On the (im)possibility of aggregate message authentication codes. In *ISIT 2008. IEEE international symposium on information theory* (p. 235-239). doi: 10.1109/ISIT.2008.4594983

Chan, H. (2009). *Authenticated communication and computation in known-topology networks with a trusted authority*. Doctoral dissertation, School of Computer Science, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA.

Chan, H., Perrig, A., & Song, D. (2003). Random key predistribution schemes for sensor networks. In *IEEE symposium on security and privacy.*

Chan, H., Perrig, A., & Song, D. (2006). Secure hierarchical in-network aggregation in sensor networks. In *13th ACM conference on computer and communications security (CCS)* (pp. 278–287). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1180405.1180440

Chen, B., & Min, G. (2010, July). Robust top-k query evaluation in wireless sensor networks. In *IEEE CIT* (pp. 660–667). doi: 10.1109/CIT.2010.131

Chun, B.-G., Maniatis, P., Shenker, S., & Kubiatowicz, J. (2007). Attested append-only memory: making adversaries stick to their word. *SIGOPS Oper. Syst. Rev.*, *41*(6), 189–204. doi: http://doi.acm.org/10.1145/1323293.1294280

Claise, B. (2004). *RFC3954: Cisco Systems NetFlow Services Export Version 9.* http://www.faqs.org/rfcs/rfc3954.html. Retrieved from `http://www.faqs.org/rfcs/rfc3954.html`

Considine, J., Li, F., Kollios, G., & Byers, J. (2004). Approximate aggregation techniques for sensor databases. In *ICDE* (pp. 449–460).

Cooke, E., Mortier, R., Donnelly, A., Barham, P., & Isaacs, R. (2006). Reclaiming Network-wide Visibility Using Ubiquitous Endsystem Monitors. In *USENIX.*

Costa, P., Donnelly, A., Rowstron, A., & O'Shea, G. (2012, April). Camdoop: Exploiting in-network aggregation for big data applications. In *NSDI.* USENIX.

Crumpacker, J. R. (2009). *Distributed password cracking*. MSc. dissertation, Naval Postgraduate School, Monterey, California.

Daemen, J., & Rijmen, V. (1999, March). *AES proposal: Rijndael.*

Daemen, J., & Rijmen, V. (2000). The block cipher Rijndael. In *CARDIS '98: Proceedings of the the international conference on SmartCard research and applications* (pp. 277–284). London, UK: Springer-Verlag.

Dam, M., & Stadler, R. (2005, June). A generic protocol for network state aggregation. In *RVK 05.* Linköping, Sweden.

Dashti, M. T. (2009). Optimistic fair exchange using trusted devices. In *Proceedings of the 11th international symposium on stabilization, safety, and security of distributed systems* (pp. 711–725). Berlin, Heidelberg: Springer-Verlag. Retrieved from `http://dx.doi.org/10.1007/978-3-642-05118-0_49` doi: http://dx.doi.org/10.1007/978-3-642-05118-0_49

Dell Inc. (2004, December). *Securing network-based client computing: User and machine security.* Retrieved from `http://www.dell.com/downloads/global/solutions/Dell_TPM_Whitepaper.pdf`

Deng, L., & Cox, L. P. (2009, February). LiveCompare: Grocery bargain hunting through participatory sensing. In *HotMobile.* Santa Cruz, CA, USA.

Denning, D. E., & Sacco, G. M. (1981). Timestamps in key distribution protocols. *Commun. ACM*, *24*(8), 533–536. doi: http://doi.acm.org/10.1145/358722.358740

Deshpande, A., Nath, S., Gibbons, P. B., & Seshan, S. (2003). Cache-and-query for wide area sensor databases. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on management of data* (pp. 503–514). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/872757.872818

Dickinson, B. (2010, May). *With 'smart dust' a trillion sensors scattered around the globe.* SmartPlanet. Retrieved from `http://www.smartplanet.com/technology/blog/science-scope/building-a-real-world-web-with-smart-dust/1673/`

Dierks, T., & Rescorla, E. (2008). *RFC 5246: The transport layer security (TLS) protocol. version 1.2.* Retrieved from `http://tools.ietf.org/html/rfc5246`

Diffie, W., & Hellman, M. E. (1976, November). New directions in cryptography. *IEEE Trans. on Information Theory*, *IT-22*, 644-654.

Diffie, W., Oorschot, P. C. V., & Wiener, M. J. (1992). *Authentication and authenticated key exchanges.*

Dolev, D., & Yao, A. (1983, March). On the security of public key protocols. *IEEE Transactions on Information Theory*, *29*(2), 198 - 208. doi: 10.1109/TIT.1983.1056650

Dolev, S., Israeli, A., & Moran, S. (1993). Self-stabilization of dynamic systems assuming only read/write atomicity. *Distrib. Comput.*, *7*(1), 3–16. doi: http://dx.doi.org/10.1007/BF02278851

Doolin, D. M., & Sitar, N. (2005). Wireless sensors for wildfire monitoring. In *SPIE symposium on smart structures and materials: Sensors and smart structures technologies for civil, mechanical, and aerospace systems.*

Douceur, J. R. (2002). The Sybil Attack. In *IPTPS '01: Revised papers from the first international workshop on peer-to-peer systems* (pp. 251–260). London, UK: Springer-Verlag.

Du, W., Deng, J., Han, Y. S., Varshney, P. K., Katz, J., & Khalili, A. (2005). A pairwise key predistribution scheme for wireless sensor networks. *ACM Trans. Inf. Syst. Secur.*, *8*(2), 228–258. doi: http://doi.acm.org/10.1145/1065545.1065548

Dua, A., Bulusu, N., Feng, W.-C., & Hu, W. (2009). Towards trustworthy participatory sensing. In *Proceedings of the 4th USENIX conference on hot topics in security* (pp. 8–8). Berkeley, CA, USA: USENIX Association. Retrieved from `http://portal.acm.org/citation.cfm?id=1855628.1855636`

Dworkin, M. (2001, December). *NIST special publication 800-38a: Recommendation for block cipher modes of operation: Methods and techniques.*

Dworkin, M. (2005, May). *NIST special publication 800-38b: Cipher modes of operation: The CMAC mode for authentication.*

Dworkin, M. (2007, November). *NIST special publication 800-38d: Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC.*

Erdös, P., & Rényi, A. (1959). On random graphs. *Publ. Math. Debrecen*, *6*, 290-297.

Eschenauer, L., & Gligor, V. D. (2002). A key-management scheme for distributed sensor networks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security* (pp. 41–47). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/586110.586117

Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., & Whiting, D. (2000). *Improved cryptanalysis of Rijndael.*

Finkenzeller, K. (1999). *RFID handbook: Radio-frequency identification fundamentals and applications*. Wiley.

Finnigin, K. M., Mullins, B. E., Raines, R. A., & Potoczny, H. B. (2007, April). Cryptanalysis of an elliptic curve cryptosystem for wireless sensor networks. *Int. J. Secur. Netw.*, *2*, 260–271. Retrieved from `http://portal.acm.org/citation.cfm?id=1359025.1359033` doi: 10.1504/IJSN.2007.013179

FIPS. (2001, November). *Federal Information Processing Standards Publication 107. Announcing the Advanced Encryption Standard (AES).* Retrieved from `http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

Fisher, D. (2010, February). *Google attack was tip of the iceberg.* Retrieved from `http://threatpost.com/en_us/blogs/google-attack-was-tip-iceberg-020510` (Retrieved May 2012)

Flajolet, P., & Martin, G. N. (1985). Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, *31*(2), 182–209. doi: http://dx.doi.org/10.1016/0022-0000(85)90041-8

Fleisch, E. (2010, January). *What is the internet of things? an economic perspective.* Auto-ID Labs White Paper WP-BIZAPP-053.

Fort, M., Freiling, F., Penso, L. D., Benenson, Z., & Kesdogan, D. (2006). TrustedPals: Secure multiparty computation implemented with smart cards. In *ESORICS.*

Frikken, K. B., & Dougherty, J. A., IV. (2008). An efficient integrity-preserving scheme for hierarchical sensor aggregation. In *WiSec '08: Proceedings of the first ACM conference on wireless network security* (pp. 68–76). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1352533.1352546

Ganesan, D., Govindan, R., Shenker, S., & Estrin, D. (2001). Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, *5*(4), 11–25. doi: http://doi.acm.org/10.1145/509506.509514

Garofalakis, M., Hellerstein, J., & Maniatis, P. (2007, April). Proof sketches: Verifiable in-network aggregation. *IEEE 23rd International Conference on Data Engineering ICDE 2007*, 996-1005. doi: 10.1109/ICDE.2007.368958

Gassend, B., Clarke, D., van Dijk, M., & Devadas, S. (2002). Silicon physical random functions. In *Proceedings of the 9th ACM conference on computer and communications security* (pp. 148–160). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/586110.586132` doi: 10.1145/586110.586132

Gebhardt, C., Dalton, C. I., & Tomlinson, A. (2010). Separating hypervisor trusted computing base supported by hardware. In *Proceedings of the fifth ACM workshop on scalable trusted computing* (pp. 79–84). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1867635.1867648` doi: 10.1145/1867635.1867648

Gehrmann, C., Douglas, H., & Nilsson, D. (2011, jan.). Are there good reasons for protecting mobile phones with hypervisors? In *IEEE consumer communications and networking conference (CCNC)* (p. 906 -911). doi: 10.1109/CCNC.2011.5766638

Geller, T. (2007). Envisioning the wind: Meteorology graphics at weather underground. *IEEE Computer Graphics and Applications*, *27*, 92-97. doi: http://doi.ieeecomputersociety.org/10.1109/MCG.2007.124

Gershenfeld, N., Krikorian, R., & Cohen, D. (2004, October). The Internet-of-Things. *Scientific American*.

Gibbons, P. B., Karp, B., Ke, Y., Nath, S., & Seshan, S. (2003). IrisNet: An architecture for a worldwide sensor web. *IEEE Pervasive Computing*, *2*(4), 22–33. doi: http://dx.doi.org/10.1109/MPRV.2003.1251166

Gilbert, H., & Peyrin, T. (2009). *Super-sbox cryptanalysis: Improved attacks for AES-like permutations.* Cryptology ePrint Archive, Report 2009/531. Retrieved from `http://eprint.iacr.org/2009/531`

Gligoroski, D., Andova, S., & Knapskog, S. (2008). On the importance of the key separation principle for different modes of operation. In L. Chen, Y. Mu, & W. Susilo (Eds.), *Information security practice and experience* (Vol. 4991, p. 404-418). Springer Berlin / Heidelberg. Retrieved from `http://dx.doi.org/10.1007/978-3-540-79104-1_29`

Goldreich, O. (2004). *Foundations of cryptography: Volume II, basic applications*. New York, NY, USA: Cambridge.

Gomez, L., Laube, A., & Ulmer, C. (2009). Secure sensor networks for public safety command and control system. In *IEEE conference on technologies for homeland security.*

Gonzalez Prieto, A., & Stadler, R. (2007). A-GAP: An adaptive protocol for continuous network monitoring with accuracy objectives. *IEEE Trans. on Network and Service Management*.

Gonzalez Prieto, A., & Stadler, R. (2009). *Controlling performance trade-offs in adaptive network monitoring* (Tech. Rep. No. IR-EE-LCN-2009-001). Stockholm, Sweden: Royal Institute of Technology (KTH).

Graham, G. S., & Denning, P. J. (1971). Protection – principles and practice. In *AFIPS '71 (Fall): Proceedings of the November 16-18, 1971, fall joint computer confer-*

*ence* (pp. 417–429). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1478873.1478928

Gummadi, R., Balakrishnan, H., Maniatis, P., & Ratnasamy, S. (2009). Not-a-Bot (NAB): Improving service availability in the face of botnet attacks. In *NSDI.*

Haeberlen, A., Kouznetsov, P., & Druschel, P. (2007). PeerReview: Practical accountability for distributed systems. *SIGOPS Oper. Syst. Rev.*, *41*(6), 175–188. doi: http://doi.acm.org/10.1145/1323293.1294279

Haghani, P., Papadimitratos, P., Poturalski, M., Aberer, K., & Hubaux, J.-P. (2007). Efficient and robust secure aggregation for sensor networks. In *NPSEC '07: Proceedings of the 2007 3rd IEEE workshop on secure network protocols* (pp. 1–6). Washington, DC, USA: IEEE Computer Society. doi: http://dx.doi.org/10.1109/NPSEC.2007.4371623

Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J. A., ... Felten, E. W. (2008). Lest we remember: Cold boot attacks on encryption keys. In *USENIX security symposium.*

Hall, D. L., & Llinas, J. (1997). An introduction to multisensor data fusion. *Proc. of the IEEE*, *85*(1), 6–23.

Hankerson, D., Menezes, A., & Vanslone, S. (2004). *Guide to elliptic curve cryptography.* Springer.

He, T., Vicaire, P., Yan, T., Luo, L., Gu, L., Zhou, G., ... Abdelzaher, T. (2006). Achieving real-time target tracking using wireless sensor network. In *IEEE real time technology and applications symosium* (pp. 37–48).

Helgason, Ó. R., & Jónsson, K. V. (2008). Opportunistic networking in OMNeT++. In *First international conference on simulation tools and techniques for communications, networks and systems (SIMUTOOLS 2008), OMNeT++ workshop.* Marseille, France.

Ho, C., van Renesse, R., Bickford, M., & Dolev, D. (2008). Nysiad: practical protocol transformation to tolerate byzantine failures. In *NSDI'08: Proceedings of the 5th USENIX symposium on networked systems design and implementation* (pp. 175–188). Berkeley, CA, USA: USENIX Association.

Horowitz, M. (2008, June). Visualizing big data: Bar charts for words. *Wired Magazine.* Retrieved from `http://www.wired.com/science/discoveries/magazine/16-07/pb_visualizing##ixzz0llT2DN5j`

Housley, R. (2009, September). *RFC 5652: Cryptographic message syntax (CMS).* Retrieved from `http://tools.ietf.org/html/rfc5652`

Huston, G. (2012). *IPv4 address report.* Retrieved from `http://www.potaroo.net/tools/ipv4/index.html` (Accessed May 2012)

Intanagonwiwat, C., Govindan, R., & Estrin, D. (2000). Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCom* (pp. 56–67).

Jain, R. (1991). *The art of computer systems performance analysis*. John Wiley & Sons.

Jelasity, M., Montresor, A., & Babaoglu, O. (2005). Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, *23*(1), 219-252.

Jónsson, K. V., & Dam, M. F. (2010, June). Towards flexible and secure distributed aggregation. In *AIMS 2010 - PhD Workshop.* Zurich, Switzerland.

Jónsson, K. V., Palmskog, K., & Vigfússon, Ý. (2012, June). Secure distributed top-$k$ aggregation. In *IEEE International Conference on Communications (ICC).* Ottawa, Canada.

Jónsson, K. V., & Vigfússon, Ý. (2011, Oct.). Securing distributed aggregation with trusted devices. In *NordSec.* Tallinn, Estonia.

Jónsson, K. V., & Vigfússon, Ý. (2012a, Feb.). Bootstrapping trust in networked measurement systems with secure sensors. In *Sensor Applications Symposium (SAS).* Brescia, Italy. (Top 3 student paper award)

Jónsson, K. V., & Vigfússon, Ý. (2012b, Oct.). Robust authentication in trusted sensing networks with physically unclonable functions. In *NordSec.* Karlskrona, Sweden.

Jónsson, K. V., Vigfússon, Ý., & Helgason, Ó. R. (2012, March). Simulating large-scale dynamic random graphs in OMNeT++. In *SIMUTools, OMNeT++ Workshop.* Desenzano, Italy.

Jurca, D., & Stadler, R. (2009, August). *Computing histograms of local variables for real-time monitoring using aggregation trees* (Tech. Rep.). Royal Institute of Technology (KTH).

Kansal, A., Nath, S., Liu, J., & Zhao, F. (2007). SenseWeb: An infrastructure for shared sensing. *IEEE Multimedia*, *14*(4), 8–13. doi: 10.1109/MMUL.2007.82

Kansal, A., & Zhao, F. (2007). Location and mobility in a sensor network of mobile phones. In *ACM SIGMM 17th international workshop on network and operating systems support for digital audio & video (NOSSDAV).*

Karlof, C., Sastry, N., & Wagner, D. (2004). TinySec: a link layer security architecture for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on embedded networked sensor systems* (pp. 162–175). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1031495.1031515

Karlof, C., & Wagner, D. (2003). Secure routing in wireless sensor networks: Attacks and countermeasures. In *First IEEE International Workshop on Sensor Network Protocols and Applications* (pp. 113–127).

Katz, J., & Lindell, A. Y. (2008). Aggregate message authentication codes. In *Topics in cryptology – CT-RSA'08 (lecture notes in computer science).* Springer Berlin / Heidelberg.

Kempe, D., Dobra, A., & Gehrke, J. (2003, October). Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foudations of Computer Science (FOCS'03).* Cambridge, MA, USA.

Kempton, W., & Tomić, J. (2005). Vehicle-to-grid power implementation: From stabilizing the grid to supporting large-scale renewable energy. *Journal of Power Sources*.

Keshav, S. (2006). Efficient and decentralized computation of approximate global state. *SIGCOMM Comput. Commun. Rev.*, *36*(1), 69–74. doi: http://doi.acm.org/10.1145/1111322.1111338

Kleinberg, J. (2000). The small-world phenomenon: An algorithmic perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on theory of computing* (pp. 163–170). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/335305.335325

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of Computation*, *48*(177), 203-209.

Kocher, P. C., Jaffe, J., & Jun, B. (1999). Differential power analysis. In *Proceedings of the 19th annual international cryptology conference on advances in cryptology* (pp. 388–397). London, UK, UK: Springer-Verlag. Retrieved from `http://dl.acm.org/citation.cfm?id=646764.703989`

Kosanović, M., & Stojčev, M. (2008). Reliable transport of data in wireless sensor network. In *6th international conference on microelectronics (MIEL 2008).*

Krishnamachari, B., Estrin, D., & Wicker, S. B. (2002). The impact of data aggregation in wireless sensor networks. In *ICDCSW '02: Proceedings of the 22nd international conference on distributed computing systems* (pp. 575–578). Washington, DC, USA: IEEE Computer Society.

Krishnamurthy, B., & Wills, C. E. (2006). Generating a privacy footprint on the internet. In *Proceedings of the 6th ACM SIGCOMM conference on internet measurement* (pp. 65–70). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1177080.1177088` doi: http://doi.acm.org/10.1145/1177080.1177088

Kristinsson, B. (2011, December). *Ardrand: The arduino as a hardware random-number generator.* Independent studies final report, Reykjavik University.

Kristinsson, B. (2013, Janurary). *Reasoning about security properties of the TSense protocols with ProVerif.* UROP independent studies project report, Reykjavik University.

Kursawe, K. (2011, January). *From trusted systems to the smart grid.* Presentation, KTH. (Retrieved May 2011)

Kursawe, K., Schellekens, D., & Preneel, B. (2005). Analyzing trusted platform communication. In *ECRYPT Workshop, CRASH – CRyptographic Advances in Secure Hardware* (p. 8).

Lagerholm, S. (2012). *The IPv4 depletion site.* Retrieved from `http://www.ipv4depletion.com` (Accessed May 2012)

Lampson, B. W. (1969). Dynamic protection structures. In *AFIPS '69 (fall): Proceedings of the november 18-20, 1969, fall joint computer conference* (pp. 27–38). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1478559.1478563

Lampson, B. W. (1974). Protection. *SIGOPS Oper. Syst. Rev.*, *8*(1), 18–24. doi: http://doi.acm.org/10.1145/775265.775268

Levin, D., Douceur, J. R., Lorch, J. R., & Moscibroda, T. (2009). TrInc: small trusted hardware for large distributed systems. In *NSDI'09* (pp. 1–14). Berkeley, CA, USA: USENIX Association.

Levy, S. (2001). *Hackers: Heroes of the computer revolution.* Penquin.

Lim, K.-S., & Stadler, R. (2001). A navigation pattern for scalable Internet management. In *IFIP/IEEE INM* (p. 405-420). doi: 10.1109/INM.2001.918056

Lim, K.-S., & Stadler, R. (2005, 15-19 May). Real-time views of network traffic using decentralized management. *9th IFIP/IEEE International Symposium on Integrated Network Management*, 119-132. doi: 10.1109/INM.2005.1440778

Liu, A., & Ning, P. (2008). TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In *IPSN '08: Proceedings of the 7th international conference on information processing in sensor networks* (pp. 245–256). Washington, DC, USA: IEEE Computer Society. doi: http://dx.doi.org/10.1109/IPSN.2008.47

Liu, D., & Ning, P. (2003). Establishing pairwise keys in distributed sensor networks. In *CCS '03: Proceedings of the 10th ACM conference on computer and communications security* (pp. 52–61). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/948109.948119

Liu, Y., & Das, S. (2006, November). Information-intensive wireless sensor networks: potential and challenges. *IEEE Communications Magazine*, *44*(11), 142-147.

Madden, S., Franklin, M., Hellerstein, J., & Hong, W. (2002b). TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *5th Symposium on Operating Systems Design and Implementation* (p. 131-146).

Madden, S., Franklin, M. J., Hellerstein, J. M., & Hong, W. (2002a). The design of an acquisitional query processor for sensor networks. In *ACM SIGMOD* (pp. 491–

502). ACM Press.

Malan, D. (2004). *Crypto for tiny objects* (Tech. Rep.). Harvard University.

Malan, D. J., Welsh, M., & Smith, M. D. (2008). Implementing public-key infrastructure for sensor networks. *ACM Trans. Sen. Netw.*, *4*(4), 1–23. doi: http://doi.acm.org/10.1145/1387663.1387668

Matrosov, A., Rodionov, E., Harley, D., & Malcho, J. (2010). *Stuxnet under the microscope.* Whitepaper (rev. 1.31). (Retrieved June 2012)

Maurer, U. (2006). Secure multi-party computation made simple. *Discrete Appl. Math.*, *154*(2), 370–381. doi: http://dx.doi.org/10.1016/j.dam.2005.03.020

McAfee Labs. (2012). *McAfee threats report: First quarter 2012.* http://www.mcafee.com. (Retrieved May 2012)

McEvoy, R., Tunstall, M., Murphy, C. C., & Marnane, W. P. (2007). Differential power analysis of HMAC based on SHA-2, and countermeasures. In *Information security applications* (Vol. 4867/2007). Springer Berlin / Heidelberg.

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of applied cryptography.* CRC Press.

Merkle, R. C. (1978, April). Secure communications over insecure channels. *Commun. ACM*, *21*(4), 294–299. Retrieved from `http://doi.acm.org/10.1145/359460.359473` doi: 10.1145/359460.359473

Moallemi, C. C., & Van Roy, B. (2006). Consensus propagation. *IEEE Transactions on Information Theory*, *52*, 4753–4766.

Moehrke, J. (2011). *SSL is not broken, browser based PKI is.* Retrieved from `http://healthcaresecprivacy.blogspot.com/2011/04/ssl-is-not-broken-browser-based-pki-is.html` (Accessed 8. march 2012)

Mortier, R., Isaacs, R., & Barham, P. (2005, May). *Anemone: using end-systems as a rich network management platform* (Tech. Rep. No. MSR-TR-2005-62). Cambridge, UK: Microsoft Research.

Mun, M., Reddy, S., Shilton, K., Yau, N., Burke, J., Estrin, D., ... Boda, P. (2009). PEIR, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proceedings of the 7th international conference on mobile systems, applications, and services* (pp. 55–68). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1555816.1555823` doi: http://doi.acm.org/10.1145/1555816.1555823

Mykletun, E., Narasimha, M., & Tsudik, G. (2004). Signature bouquets: Immutability for aggregated/condensed signatures. In *ESORICS* (pp. 160–176).

Narasimha, M., & Tsudik, G. (2006). Authentication of outsourced databases using signature aggregation and chaining. In *Database systems for advanced applications (DASFAA)* (p. 420-436).

Nath, S., Gibbons, P. B., Seshan, S., & Anderson, Z. (2008). Synopsis diffusion for robust aggregation in sensor networks. *ACM Trans. Sen. Netw.*, *4*(2), 1–40. doi: http://doi.acm.org/10.1145/1340771.1340773

Nature. (2008, September). Community cleverness required. *Nature*, *455*(1). Retrieved from `http://www.nature.com/nature/journal/v455/n7209/full/455001a.html`

Needham, R. M., & Schroeder, M. D. (1978). Using encryption for authentication in large networks of computers. *Commun. ACM*, *21*(12), 993–999. doi: http://doi.acm.org/10.1145/359657.359659

Needham, R. M., & Schroeder, M. D. (1987). Authentication revisited. *SIGOPS Oper. Syst. Rev.*, *21*(1), 7–7. doi: http://doi.acm.org/10.1145/24592.24593

Needham, R. M., & Wheeler, D. J. (1997). *TEA extensions* (Tech. Rep.). Computer Laboratory, University of Cambridge.

Neuman, B. C., & Ts'o, T. (1994, September). Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*.

Nielson, S. J., Crosby, S. A., & Wallach, D. S. (2005, February). A taxonomy of rational attacks. In *4th. IPTPS*.

*OWASP top ten project.* (2012). Retrieved from `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project` (Retrieved May 2012)

Palmskog, K., Gonzalez Prieto, A., Meirosu, C., Stadler, R., & Dam, M. (2010). Scalable metadata-directed search in a network of information. In *Future Network & Mobile Summit*.

Panetto, H., & Molina, A. (2008). Enterprise integration and interoperability in manufacturing systems: Trends and issues. *Computers in Industry*, *59*(7), 641 - 646. Retrieved from `http://www.sciencedirect.com/science/article/B6V2D-4SJ9504-1/2/bc0c65595577f763181899d3874c16bb` doi: DOI:10.1016/j.compind.2007.12.010

Parno, B., Perrig, A., & Gligor, V. (2005, May). Distributed detection of node replication attacks in sensor networks. In *IEEE Symp. Security and Privacy* (pp. 49–63). Oakland, CA.

Paulos, E., Honicky, R., & Goodman, E. (2007, 6–9 November). Sensing atmosphere. In *ACM conference on embedded networked sensor systems (SenSys)*. Sydney, Australia.

Peleg, D. (2000). *Distributed computing: A locality-sensitive approach.* Society for Industrial and Applied Mathematics (SIAM).

Perlman, R. (1985). An algorithm for distributed computation of a spanning tree in an extended LAN. *SIGCOMM Comput. Commun. Rev.*, *15*(4), 44–53. doi: http://doi.acm.org/10.1145/318951.319004

Perrig, A., Chan, H., Przydatek, B., & Song, D. (2007). SIA: Secure information aggregation in sensor networks. *Journal of Computer Security*, *15*(1), 69-102.

Perrig, A., Stankovic, J., & Wagner, D. (2004). Security in wireless sensor networks. *Commun. ACM*, *47*(6), 53–57. doi: http://doi.acm.org/10.1145/990680.990707

Perrig, A., Szewczyk, R., Tygar, J. D., Wen, V., & Culler, D. E. (2002). SPINS: security protocols for sensor networks. *Wirel. Netw.*, *8*(5), 521–534. doi: http://dx.doi.org/10.1023/A:1016598314198

Plotkin, G. D. (2004). The origins of structural operational semantics. *Journal of Logic and Algebraic Programming*, *60–61*, 3–15.

Przydatek, B., Song, D., & Perrig, A. (2003). SIA: Secure Information Aggregation in Sensor Networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems* (pp. 255–265). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/958491.958521

R Development Core Team. (2011). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from `http://www.R-project.org/` (ISBN 3-900051-07-0)

Rajagopalan, R., & Varshney, P. (2006). Data-aggregation techniques in sensor networks: a survey. *Communications Surveys & Tutorials, IEEE*, *8*(4), 48-63. doi: 10.1109/COMST.2006.283821

Rankl, W., & Effing, W. (2001). *SmartCard Handbook* (2nd edition ed.). John Wiley and Sons Ltd.

van Renesse, R. (2003). The importance of aggregation. In A. Schiper, A. Shvatsman, H. Weatherspoon, & B. Zhao (Eds.), *Lecture notes in computer science* (Vol. 2584, p. 87-92). Springer Verlag.

Rogaway, P., Bellare, M., & Black, J. (2003). OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, *6*(3), 365–403. doi: http://doi.acm.org/10.1145/937527.937529

Roosta, T., Shieh, S., & Sastry, S. (2006). Taxonomy of security attacks in sensor networks and countermeasures. In *First IEEE international conference on system integration and reliability improvements* (pp. 13–15). Hanoi.

Roush, W. (2003, February). 10 emerging technologies that will change the world – sensor networks. *Technology Review.* Retrieved from `http://www`

.technologyreview.com

Roy, S. (2008). *Secure data aggregation in wireless sensor networks.* Doctoral dissertation, George Mason University, Fairfax, VA.

Roy, S., Conti, M., Setia, S., & Jajodia, S. (2008). Securely computing an approximate median in wireless sensor networks. In *SecureComm '08: Proceedings of the 4th international conference on security and privacy in communication networks* (pp. 1–10). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1460877.1460885

Rúnarsson, K., Kristinsson, B., & Jónsson, K. V. (2010, September). *TSense: Trusted sensors and support infrastructure.* Reykjavík University, Rannís NSN project report.

Saroiu, S., & Wolman, A. (2010). I am a sensor, and I approve this message. In *Hot-Mobile '10: Proceedings of the eleventh workshop on mobile computing systems & applications* (pp. 37–42). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1734583.1734593

Scheibelhofer, K. (2007). A bit-slice implementation of the Whirlpool hash function. In *CT-RSA.*

Schneier, B. (1996). *Applied cryptography* (2nd ed.). John Wiley & Sons.

Schneier, B. (2009, July). *Schneier on security: Another new AES attack.* Retrieved from http://www.schneier.com/blog/archives/2009/07/another_new_aes.html (Retrieved May 2012)

Schneier, B. (2012, May). *Backdoor found (maybe) in Chinese-made military silicon chips.* Retrieved from http://www.schneier.com/blog/archives/2012/05/backdoor_found.html (Retrieved May 2012)

Shi, E., & Perrig, A. (2004, December). Designing secure sensor networks. *IEEE Wirel. Netw.*.

Shneidman, J., & Parkes, D. C. (2004). Specification faithfulness in networks with rational nodes. In *Proceedings of the twenty-third annual ACM symposium on principles of distributed computing* (pp. 88–97). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1011767.1011781 doi: 10.1145/1011767.1011781

Shrivastava, N., Buragohain, C., Agrawal, D., & Suri, S. (2004). Medians and beyond: new aggregation techniques for sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on embedded networked sensor systems* (pp. 239–249). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1031495.1031524

Simha, R., Narahari, B., Zambreno, J., & Choudhary, A. (2006). Secure execution with components from untrusted foundries. In *Proceedings of the advanced networking and communications hardware workshop (ANCHOR).*

Simpson, Jr., C. R., & Riley, G. F. (2004, April). NETI@home: A distributed approach to collecting end-to-end network performance measurements. In *PAM - A workshop on Passive and Active Measurements.*

Slagell, A., & Yurcik, W. (2005). Sharing computer network logs for security and privacy: A motivation for new methodologies of anonymization. In *Proceedings of SECOVAL: The Workshop on the Value of Security through Collaboration* (pp. 80–89).

Sophos. (2012). *Security threat report.* Retrieved from `http://www.sophos.com/medialibrary/PDFs/other/SophosSecurityThreatReport2012.pdf` (Retrieved May 2012)

Sourcefire. (2012). SNORT users manual 2.9.2 [Computer software manual]. Retrieved from `http://manual.snort.org/` (Retrieved May 2012)

Stadler, R., Dam, M., Gonzalez, A., & Wuhib, F. (2008). Decentralized real-time monitoring of network-wide aggregates. In *LADIS '08: Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware* (pp. 1–6). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1529974.1529984

Stajano, F. (2003). Security for whom? the shifting security assumptions of pervasive computing. In *Proceedings of the Mext-NSF-JSPS international conference on software security: theories and systems* (pp. 16–27). Berlin, Heidelberg: Springer-Verlag. Retrieved from `http://portal.acm.org/citation.cfm?id=1765533.1765536`

Stajano, F., & Anderson, R. (1999). The resurrecting duckling: Security issues for adhoc wireless networks. In *7th international workshop on security protocols* (pp. 172–194). Cambridge, England: Springer-Verlag.

Stallman, R. (2007). *Can you trust your computer?* Retrieved from `http://www.gnu.org/philosophy/can-you-trust.html` (Retrieved May 5, 2011)

Stann, F., & Heidemann, J. (2003, April). RMST: Reliable data transport in sensor networks. In *Proceedings of the first international workshop on sensor net protocols and applications* (pp. 102–112). Anchorage, Alaska, USA: IEEE. Retrieved from `http://www.isi.edu/~johnh/PAPERS/Stann03a.html`

Sterling, B. (1993). *The hacker crackdown: Law and disorder on the electronic frontier.* Bantam.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., & Balakrishnan, H. (2003). Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, *11*(1), 17–32. doi: http://dx.doi.org/

10.1109/TNET.2002.808407

Stutzbach, D., & Rejaie, R. (2006, October). Understanding churn in peer-to-peer networks. In *IMC.* Rio de Janeiro, Brazil.

Sundmaeker, H., Guillemin, P., Friess, P., & Woelfflé, S. (2010, march). *Vision and challenges for realising the Internet of Things.* CERP-IoT Report.

Synergist SCADA. (2012). *The top 5 SCADA security threats for 2012.* Retrieved from `https://www.synergistscada.com/the-top-5-scada-security-threats-for-2012/` (Retrieved May 2012)

Tasker, P. S. (1981). Trusted computer systems. In *IEEE Symposium on Security and Privacy* (Vol. 0, p. 99). Los Alamitos, CA, USA: IEEE Computer Society. doi: http://doi.ieeecomputersociety.org/10.1109/SP.1981.10020

TCSEC. (1985, December). *Trusted computer system evaluation criteria (Orange Book).* U.S. Department of Defense (DoD).

Thornton, F., Haines, B., Das, A. M., Bhargava, H., Campbell, A., & Kleinschmidt, J. (2006). *RFID security.* Syngress.

Trappe, W., Zhang, Y., & Nath, B. (2005). MIAMI: methods and infrastructure for the assurance of measurement information. In *DMSN '05: Proceedings of the 2nd international workshop on data management for sensor networks* (pp. 11–17). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1080885.1080888

Trossen, D., & Pavel, D. (2005). Building a ubiquitous platform for remote sensing using smartphones. In *Proceedings of the second annual international conference on mobile and ubiquitous systems: Networking and services (MobiQuitous).*

Trusted Computing Group. (2011, March). *TPM main specification. part 1 design principles.* (Version 1.2, Revision 116)

Tryggvason, K. (2012, December). *Verifying authentication protocol for distributed aggregation systems.* Reykjavik University, Course project report, T-701-REM4 Research Methodologytr.

Tuyls, P., & Batina, L. (2006). RFID-tags for anti-counterfeiting. In *Topics in cryptology* (Vol. 3860, pp. 115–131). Springer Verlag.

Tuyls, P., Schrijen, G.-J., Škorić, B., van Geloven, J., Verhaegh, N., & Wolters, R. (2006). Read-proof hardware from protective coatings. In *Proceedings of the 8th international conference on cryptographic hardware and embedded systems* (pp. 369–383). Berlin, Heidelberg: Springer-Verlag.

Uckelmann, D., Harrison, M., & Michahelles, F. (2011). An architectural approach towards the future Internet of Things. In D. Uckelmann, M. Harrison, & F. Michahelles (Eds.), *Architecting the Internet of Things.* Springer.

Upadhyayula, S., & Gupta, S. (2007). Spanning tree based algorithms for low latency and energy efficient data aggregation enhanced convergecast (DAC) in wireless sensor networks. *Ad Hoc Netw.*, *5*(5), 626–648. doi: http://dx.doi.org/10.1016/j.adhoc.2006.04.004

Varga, A. (2001, June). The OMNeT++ discrete event simulation system. In *European Simulation Multiconference (ESM'2001).*

Vaudenay, S. (2002). Security flaws induced by CBC padding applications to SSL, IPSEC, WTLS... In *EUROCRYPT.*

Venere, E., & Gardner, E. K. (2008, January). *Cell phone sensors detect radiation to thwart nuclear terrorism.* Purdue University News. Retrieved from `http://news.uns.purdue.edu/x/2008a/080122FischbachNuclear.html` (Retrieved May 2012)

Vermorel, J., & Mohri, M. (2005). Multi-armed bandit algorithms and empirical evaluation. In *Proceedings of the 16th european conference on machine learning* (pp. 437–448). Berlin, Heidelberg: Springer-Verlag. Retrieved from `http://dx.doi.org/10.1007/11564096_42` doi: 10.1007/11564096_42

Wagner, D. (2004). Resilient aggregation in sensor networks. In *SASN '04: Proceedings of the 2nd ACM workshop on security of ad hoc and sensor networks* (pp. 78–87). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1029102.1029116

Wan, C.-Y., Campbell, A. T., & Krishnamurthy, L. (2004). Reliable transport for sensor networks: PSFQ - pump slowly fetch quickly paradigm. *Wireless Sensor Networks*, 153–182.

Watro, R., Kong, D., Cuti, S.-f., Gardiner, C., Lynn, C., & Kruus, P. (2004). TinyPK: Securing sensor networks with public key technology. In *SASN '04: Proceedings of the 2nd ACM workshop on security of ad hoc and sensor networks* (pp. 59–64). New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1029102.1029113

Wilkes, M. V., & Needham, R. M. (1979). *The cambridge CAP computer and its operating system.* Elsevier North Holland.

Winkler, T., & Rinner, B. (2011). Securing embedded smart cameras with trusted computing. *EURASIP Journal on Wireless Communications and Networking*, *2011.*

Wood, A. D., & Stankovic, J. A. (2002). Denial of service in sensor networks. *Computer*, *35*(10), 54–62. doi: http://dx.doi.org/10.1109/MC.2002.1039518

Wood, A. D., Stankovic, J. A., & Son, S. H. (2003). JAM: A jammed-area mapping service for sensor networks. In *24th IEEE real-time systems symposium* (pp. 286–297).

Worm targets industrial-plant operations. (2010, nov.). *IEEE Computer*, *43*(11), 15 -18. doi: 10.1109/MC.2010.333

Wuhib, F., Dam, M., & Stadler, R. (2008, February). Decentralized detection of global threshold crossings using aggregation trees. *Computer Networks*, *52*(9), 1745–1761. doi: http://dx.doi.org/10.1016/j.comnet.2008.02.015

Wuhib, F., Dam, M., Stadler, R., & Clemm, A. (2005, October). Decentralized computation of threshold crossing alerts. In *DSOM 2005: 16th IFIP/IEEE international workshop on distributed systems: Operations and management* (Vol. LNCS 2775, p. 220-232).

Wuhib, F., Dam, M., Stadler, R., & Clemm, A. (2007, May). Robust monitoring of network-wide aggregates through gossiping. In *10th IFIP/IEEE International Symposium on Integrated Management (IM 2007).* Munich, Germany. (2008)

Yao, Y., & Gehrke, J. (2002). The Cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, *31*(3), 9–18. doi: http://doi.acm.org/10.1145/601858.601861

Yao, Y., & Gehrke, J. (2003). Query processing for sensor networks. In *First conference on innovative data systems research (CIDR).* San Fransisco, CA.

Yedidia, J. S., Freeman, W. T., & Weiss, Y. (2002, January). *Understanding belief propagation and its generalizations* (Tech. Rep. No. TR-2001-22). Mitsubishi Electric Research Laboratories.

Yick, J., Mukherjee, B., & Ghosal, D. (2008). Wireless sensor network survey. *Computer Networks*, *52*(12), 2292 - 2330.

Yu, L., Wang, N., & Meng, X. (2005, sept.). Real-time forest fire detection with wireless sensor networks. In *International conference on wireless communications, networking and mobile computing* (Vol. 2, p. 1214 - 1217). doi: 10.1109/WCNM.2005.1544272

Zhao, Y. J., Govindan, R., & Estrin, D. (2003, May 11). Computing aggregates for monitoring wireless sensor networks. In *The first IEEE international workshop on sensor network protocols and applications (SNPA).* Anchorage, AK, USA.

Zug, S., Dietrich, A., & Kaiser, J. (2007). An architecture for a dependable distributed sensor system. *IEEE Transactions on Instrumentation and Measurement*, *6*(1).

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is
ISSN 1670-8539