Spring 2014

Energy efficient security and privacy management in sensor clouds

Vimal Kumar

ENERGY EFFICIENT SECURITY AND PRIVACY MANAGEMENT IN SENSOR CLOUDS

by

VIMAL KUMAR

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

in Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

2014

PUBLICATION DISSERTATION OPTION

This dissertation consists of four articles prepared in the style required by the journals or conference proceedings in which they were published:

Pages 21 to 43, “Secure Hierarchical Data Aggregation in Wireless Sensor Networks: Performance Evaluation and Analysis”, was published in 2012 13th IEEE International Conference on Mobile Data Management (MDM 2012), Bengaluru, India.

An earlier version, “A Test-bed for Secure Hierarchical Data Aggregation in Wireless Sensor Networks”, was published in 2010 7th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2010), San Fransico, California.

Pages 44 to 78, “PIP: Privacy and Integrity Preserving Data Aggregation in Wireless Sensor Networks”, was published in 2013 32nd IEEE International Symposium on Reliable Distributed Systems (SRDS 2013), Braga, Portugal.

Pages 79 to 114, “Distributed Attribute Based Access Control of Aggregated Data in Sensor Clouds”, was submitted to 2014 IEEE International Conference on Sensing, Communication, and Networking (SECON 2014), Singapore.

Pages 115 to 151, “Efficient and Secure Code Dissemination in Sensor Clouds”, was submitted to 2014 15th IEEE International Conference on Mobile Data Management (MDM 2014), Brisbane, Australia.

A part of the literature review appeared as a book chapter, “Secure Data Aggregation in Wireless Sensor Networks”, in Wireless Sensor Network Technologies for the Information Explosion Era, (Springer 2012).

ABSTRACT

Sensor Cloud is a new model of computing for Wireless Sensor Networks, which facilitates resource sharing and enables large scale sensor networks. A multi-user distributed system, however, where resources are shared, has inherent challenges in security and privacy. The data being generated by the wireless sensors in a sensor cloud need to be protected against adversaries, which may be outsiders as well as insiders. Similarly the code which is disseminated to the sensors by the sensor cloud needs to be protected against inside and outside adversaries. Moreover, since the wireless sensors cannot support complex, energy intensive measures, the security and privacy of the data and the code have to be attained by way of lightweight algorithms.

In this work, we first present two data aggregation algorithms, one based on an Elliptic Curve Cryptosystem (ECC) and the other based on symmetric key system, which provide confidentiality and integrity of data against an outside adversary and privacy against an in network adversary. A fine grained access control scheme which works on the securely aggregated data is presented next. This scheme uses Attribute Based Encryption (ABE) to achieve this objective. Finally, to securely and efficiently disseminate code in the sensor cloud, we present a code dissemination algorithm which first reduces the amount of code to be transmitted from the base station. It then uses Symmetric Proxy Re-encryption along with Bloom filters and HMACs to protect the code against eavesdropping and false code injection attacks.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude towards my advisor Dr. Sanjay Madria for his continuous support of my Ph.D. study and research. I have greatly benefitted from his, knowledge, advice and guidance, which was instrumental in the completion of this dissertation.

Special thanks go to Dr. Sriram Chellappan for the engaging discussions we had and his invaluable encouragement and practical advice. I would also like to thank the members of my dissertation committee, Dr. Wei Jiang, Dr. Jagannathan Sarangapani and Dr. Fikret Ercal, for their constructive comments and feedback.

I would like to acknowledge the contributions of the current and former members of the research group especially, Brijesh Kashyap Chejerla, Dylan McDonald, Dr. Nayot Poolsappasit and Dr. Roy Cabaniss, with whom I spent countless hours in the lab and engaged in numerous fruitful discussions. I am thankful to my friends especially Abhinav Saxena, Shashank Kumar, Sachin Sharma and Bhanu Pratap Singh Kanwar who provided the much needed humor in otherwise challenging times and made my stay in Rolla enjoyable.

I owe a very important debt to my father Ram Narayan Baghel, my mother Sushma and my brother Prashant for their patience and support at all stages of my life. I would particularly like to thank Monica, who has been a constant source of motivation and a beacon of light when there was none.

Finally I would like to dedicate this work to my late brother Ashish Kumar, who left us too soon. I hope this would make you proud.

TABLE OF CONTENTS

PAPER

LIST OF ILLUSTRATIONS

LIST OF ALGORITHMS

LIST OF TABLES

1. INTRODUCTION

While the use of wireless sensors is growing, they continue to be tightly connected with the internal IT system or the network owner. In this traditional notion of Wireless Sensor Networks (WSN), a user needs to own a wireless sensor network, deploy it and maintain it, in order to use it. This model of computing in the WSN domain has been a restrictive force against its widespread adoption. A new model has recently been introduced, based on the successful Cloud Computing model for IT resources, which provides sensing as a service and decouples the user and the network owner.

Sensor Cloud Computing is a heterogeneous computing environment, which brings together multiple WSNs, each of which may contain many wireless sensors owned by different entities. In this computing paradigm, the users do not need to own the sensor network before using it. They can simply buy the sensing services from the sensor cloud. Since the amount of investment goes down, usage of large scale sensor networks becomes affordable. Similar to cloud computing, resources in sensor clouds can be dynamically provisioned and de provisioned on demand, providing greater flexibility of operations. However, unlike cloud computing, the resources in the sensor cloud i.e. wireless sensors may be owned by different entities. These entities perform the duties of deploying and maintaining their WSNs and lease them out to the sensor cloud.

The benefits of using a sensor cloud though, come with their own set of security issues. A multi-user, distributed system always poses challenges in security, and in the wireless sensor network domain, these challenges need to be handled in an energy efficient manner. The primary use of a WSN is in collecting sensory data from wireless sensors. This data is aggregated in-network, to reduce energy and bandwidth consumption. The wireless sensors though, typically work unattended and are prone to eavesdropping and false data injection attacks. When the data is routed through multiple WSNs in the sensor cloud, it also becomes important to protect the privacy of the data and the nodes from each other. In a multi-user scenario, like a sensor cloud, the users can query the network directly. The network then responds with relevant data which is aggregated in-network on the nodes. It is necessary in such a setting, to have an appropriate mechanism in place which provides access control according to the privilege granted to each user. Finally, since wireless sensors in a sensor cloud are dynamically provisioned for users; we also require a secure code dissemination algorithm, which is also efficient in terms of energy consumption.

We first discuss the issue of collecting data securely from a wireless sensor network in Section I. Wireless sensors are severely constrained in terms of bandwidth, energy resources and computational power, which calls for energy efficient and lightweight algorithms. The simplest method of data collection is by having each node's data forwarded by other nodes to the base station. This simple method however, incurs huge amount of wireless transmission which is expensive in terms of energy. To reduce the wireless transmissions and hence the energy consumption, we perform the aggregation of data at intermediate nodes. When data is aggregated, it becomes easier for an adversary to inject false data stealthily. To prevent that from happening we use Additive Digital Signatures in the form of a modified version of Elliptic Curve Digital Signature Algorithm (ECDSA). We also use Elliptic Curve El Gamal (ECEG) homomorphic encryption for confidentiality. Homomorphic encryption and additive digital signatures allow us to replace energy intensive decryption and verification operations with a few lightweight addition operations, thereby helping in conserving energy. Additive digital signatures further reduce the energy consumption by providing integrity verification without the requirement of a separate verification phase. Further in the section we discuss the implementation of our algorithm on Mica2 and TelosB sensor platforms and the experiments to analyze the performance.

In Section II we discuss the PIP algorithm which along with confidentiality and data integrity, also takes into account privacy of data and further reduces the energy consumption. PIP is based upon the recursive secret sharing scheme in [1]. Recursive secret sharing scheme [1] allows us to store additional data in shares of a secret. We use this additional space to store a linear combination of the data and the integrity key, in the shares of the data. A scrambling key is then used to scramble the shares in a manner which preserves the homomorphic property of the shares. We then enhance the algorithm to be able to localize a rogue sensor to a certain extent. Further in the section, we provide the proof of security of our construction and analyze its performance by implementing it on TelosB sensor platform.

In Section III, we focus on controlling the access to the aggregated data in sensor clouds. Traditionally access control schemes for wireless sensors have been proposed, keeping communication with a single sensor (acting as a data repository) in mind. The network topology is assumed to be fixed and the access control is only enforced at the data repository. The network topology in sensor cloud however, is not fixed and a new query tree is formed dynamically for each user query. Moreover, there are multiple parties in a sensor cloud; the user, sensor cloud administrator and network owners. While the sensor cloud administrator imposes access control, the network owners may also want control over, who is able to access data from their sensors. We use Key Policy Attribute Based Encryption (KP-ABE) to provide attribute based encryption for access control. Paillier Encryption is used to support aggregation of access control keys. To provide the network owners the flexibility to control the access, we introduce authorizations, by way of a one way key chain.

In Section IV, we turn our attention to secure code dissemination in sensor clouds. In a sensor cloud, wireless sensors are frequently de-provisioned and provisioned for new users. The sensor cloud needs to disseminate code to these newly provisioned sensors. Since the provisioning of sensors happens very often, an efficient way of disseminating code is required to reduce the expenditure of energy on code forwarding. It also needs to be secure since, the code will likely be forwarded through many sensor networks on the way to its destination. Our approach here is to reduce the amount of code needed to be transmitted by discovering the common fragments of code, in various application codes. The common fragments of code are distributed on the sensors during deployment, so that they can be picked up whenever a new application is to be deployed. The confidentiality of the code is provided using the symmetric proxy re-encryption scheme discussed in [2] and the false code injection is checked using a combination of Bloom Filter and HMAC.

2. LITERATURE REVIEW

Sensor cloud computing is a recent phenomenon, but a great deal can be learnt from prior research in the field of Wireless Sensor Networks. A large body of work exists for security and privacy in wireless sensor networks in general. Our focus here is on the aggregation of sensory data in secure and privacy preserving manner and in securely disseminating application code in sensor clouds. We thus divide this review in three parts namely, (i) Preliminaries, (ii) Secure and privacy preserving data aggregation, and (iii) Secure code dissemination. In Preliminaries, we discuss some security concepts which are useful in the context of wireless sensor networks. In the latter two parts we review the related work on secure and privacy preserving data aggregation and secure code dissemination in sensor cloud respectively.

2.1. PRELIMINARIES

2.1.1. Elliptic Curve Cryptography. Public key cryptography, also known as asymmetric cryptography is widely used in distributed environments. However, public key algorithms generally are power hungry and clearly not suitable for wireless sensors. Over the last few years Elliptic Curve Cryptography (ECC) has emerged as an attractive and viable public key system for constrained environments. ECC offers considerably greater security for a given key size, compared to the first generation public key systems. The smaller key size also makes possible more compact implementations, for a given level of security. ECC offers the equivalent security of a public key system for much lesser key size and also solves the problem of key distribution over insecure channel found in symmetric key cryptography. A small key size is not the only advantage of ECC, it is computationally inexpensive too. In [3] authors compared the energy cost of RSA and ECC on an 8-bit microcontroller and found ECC to have significant advantage over RSA in terms of energy required to complete the operations. The authors compared RSA signature scheme with ECDSA and RSA key exchange with ECDH key exchange scheme. Their results are compactly presented in Table 2.1. It is evident that ECC operations require much less energy overall than corresponding RSA operations. The comparatively high cost of verification is not an issue as mostly the verification is done at the base station.

Table 2.1. Energy cost of digital signature and key exchange [mJ]

	Signature		**Key Exchange**	
Algorithm	**Sign**	**Verify**	**Client**	**Server**
RSA-1024	304	11.9	15.4	304
ECDSA-160	22.83	45.09	22.3	22.3
RSA-2048	2302.7	53.7	57.2	2302.7
ECDSA-224	61.54	121.98	60.4	60.4

2.1.2. Homomorphic Encryption. An encryption algorithm is said to be homomorphic if it allows for the following property to hold.

$$enc(a) \oplus enc(b) = enc(a \oplus b)$$

The above equation implies that in a homomorphic encryption scheme an operation performed on the encrypted data produces the same result as when the encryption is done after the operation has been performed on the plaintext first. There are two types of homomorphic encryption schemes, additive homomorphic schemes where we have $enc(a+b) = enc(a) + enc(b)$ and multiplicative homomorphic schemes where we have $enc(a*b) = enc(a)*enc(b)$.

2.1.3. Additive Digital Signatures. Given n signatures on n distinct messages by n distinct users, it is possible to aggregate all these signatures into a single signature. This single signature will convince the verifier that the n users signed the n original messages

[4]. In a wireless sensor network, this single signature can replace the individual signatures of the sensors and thus help us save the overhead of sending all the signatures along with the aggregated data. Each sensor in the network generates a digital signature on the data it produces and forwards both the data and the signature to its parent. The parent aggregates the data as well as the digital signatures and passes it to its parent. This process continues till the base station obtains the final aggregated data and the aggregate signature. Aggregated digital signature techniques [4] are of two kinds, one where each sensor shares it signing key with the base station so that the base station is able to verify the digital signature. The other is where along with the data and the signature the keys are also aggregated, so that at the end we get an aggregate key which can be used to verify the aggregate data using the aggregate signature.

2.1.4. Secret Sharing. Secret sharing is a technique which enables us to divide a secret in n parts such that any k parts can be used to regenerate the secret. It was first described by Shamir [5] and Blakley [6] independently in 1979. Blakley's secret sharing scheme [6] is based upon the properties of hyper-planes. The secret is chosen as a point in a k-dimensional space. The shares of this secret are constructed as n, $k-1$ degree hyper-planes, which intersect at the secret. Any k chosen hyper-planes out of the n, can then be used to regenerate the secret. Shamir's secret sharing scheme [5], on the other hand is based on the properties of polynomials. Any $k-1$ degree polynomial can be generated using k points. Thus, in Shamir's secret sharing scheme, a random $k-1$ degree polynomial is generated such that the secret is the 0^{-th} degree coefficient. n points are then generated on this polynomial, which act as the shares. Any k of the n shares can then be used to regenerate the polynomial and the secret. Recursive secret sharing [1] is a space efficient secret sharing technique based on Shamir's [5]. Recursive secret sharing (RSS) scheme can carry up-to $k-2$ secrets in k shares. In the RSS scheme, the secrets are taken as fixed points in the space and a polynomial is then created to fit the points. The randomness in the scheme is introduced by choosing only one random point. This is in contrast with Shamir's secret sharing scheme where the polynomial is generated by taking only one fixed point, while the rest of the points are random.

2.1.5. Key Policy Attribute Based Encryption. Attribute based encryption was first proposed by Sahai and Waters in [7]. The attributes within the context of a sensor cloud can be sensor type, region of interest, sensor owner, etc. In attribute based encryption either the ciphertext or the secret key are generated so that they are dependent upon the attributes of the user. Each attribute is associated with a private and a public key component such that, only the user holding those attribute will be able to decrypt the data encrypted under those attributes.

In key policy attribute based encryption (KP-ABE) [8], each ciphertext is associated with a set of attributes. The access policy is defined by an access tree and the private decryption key is generated based on this access tree, hence the name key policy. A ciphertext can be decrypted with a key only if the attributes associated with the ciphertext satisfy the key's access tree. The KP-ABE [8] is composed of four algorithms, *Setup, Encryption, Key Generation, and Decryption*.

- *Setup:* The Setup algorithm defines the attributes in the system and outputs a public key PK and a master key MK. The public key PK is used for encryption while the master key MK along with PK is used to generate user keys.

- *Encryption:* The encryption algorithm takes as input a message m, a set of attributes γ and the public key PK and produces the cipher-text E

- *Key Generation:* The key generation algorithm takes as input an access tree $\mathcal{T}$, the master key MK and the public key PK to produce a secret key SK, such that SK can decrypt E iff $\mathcal{T}$ matches γ.

- *Decryption:* The decryption algorithm takes as input the ciphertext E, the decryption key SK and the public key PK and decrypts the ciphertext iff $\mathcal{T}$ matches γ, otherwise it produces $\perp$.

2.1.6. Proxy Re-Encryption. Proxy re-encryption is a cryptographic primitive that enables us to re-encrypt a given ciphertext, without decrypting it. As illustrated in Figure 2.1, proxy re-encryption involves four parties; an encryptor, a proxy, and two receivers. A re-encryption key is generated, usually by using the secret keys of the receivers and given to the proxy. When the encryptor, encrypts a plaintext for one of the recievers, the proxy can use the re-encryption key on the ciphertext, to re-encrypt the ciphertext for the other receiver. This re-encrypted ciphertext can be decrypted by the other receiver using it's secret key.Proxy re-encryption is generally used for delegating decryption rights of sensitive information. In this work, however, it has been used for secure dissemination of code in wireless sensor networks.

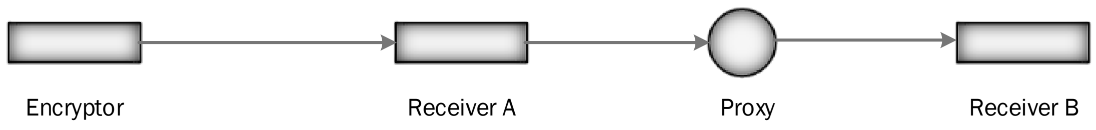

Figure 2.1. Proxy Re-encryption

2.2. SECURE AND PRIVACY PRESERVING DATA AGGREGATION

Early secure data aggregation schemes were *hop by hop* schemes. In *hop by hop* schemes, data is decrypted and verified at each hop. These schemes such as the one by Hu and Evans [9] mostly dealt with either the issue of data confidentiality or integrity in the face of a single compromised node. Hu and Evans scheme employs the idea of delayed aggregation. A compromised node could either corrupt the MACs or the data but not both. Thus a single compromised node could always be identified; however the algorithm fails in the event when multiple nodes are compromised. Schemes tackling the issue of multiple compromised nodes were introduced later, for example the scheme by Chan et. al. [10]. This algorithm is resilient to any number of malicious nodes but deals only with attacks on data integrity. Chan et. al. [10] intended to reduce the congestion around nodes high up in the hierarchy during the integrity verification phase. Their integrity verification algorithm is distributed through the network which reduces the communication load on certain nodes and increases the time to first node failure. Schemes such as SecureDAV [11] and SDAP [12] also provided for data integrity by making use of threshold cryptography and Merkle hash trees respectively. In [13] the authors proposed two *hop by hop* secure data aggregation algorithms; CPDA and SMART to provide confidentiality of data. These algorithms could also provide the privacy of a single node's data to some extent but not data integrity. In CPDA, nodes hide their data in a random polynomial of order $k-1$, where k is the number of nodes in a cluster and send it to all the nodes in the cluster securely using pair wise keys. These k polynomials are then added up and sent to the cluster head, along with a secret which was used to generate the polynomials. The authors then discuss another privacy preserving data aggregation scheme SMART which is less communication intensive than CPDA. In SMART each node in an aggregation tree slices its data into a fixed number of parts say k, and sends $k-1$ parts to $k-1$ neighboring sensors, keeping one for itself. Sensors add all of the slices together and send them to the aggregator along the path of the tree. In CPDA, the aggregate is revealed to the cluster head while, in SMART, because positive slices of data are distributed, some amount of information about the data is always leaked. [14] and [15], were proposed by the same authors as improvements upon this scheme, to additionally provide integrity of data. [14] uses the concept of disjoint aggregation trees, where the data is simultaneously aggregated in two disjoint trees. The scheme however is ineffective if the adversary is able to compromise just two nodes, one each in the two aggregation trees. The algorithm in [15] uses peer monitoring, to provide data integrity. This algorithm however, suffers from high bandwidth and energy requirements.

Hop by hop schemes though leave the data exposed to aggregators. A malicious aggregator can compromise the security of the scheme. Consequently *end to end* secure data aggregation schemes were proposed. CDA [16] is an end to end scheme which uses homomorphic encryption in the form of Domingo Ferrer privacy homomorphism (DFPH). It also uses aggregate signatures to provide data integrity. The scheme however is, not secure against chosen plaintext attacks. Moreover, the solution provided in [16] is not scalable. The algorithm in [17] uses ECC for homomorphic encryption but does not provide data integrity. Castelluccia et al.'s algorithm (EDA) in [18] uses modular addition as the homomorphic encryption algorithm. Modular addition is a straightforward operation which has the advantage of simplicity over other homomorphic encryption algorithms. However, the algorithm requires the keys be pre-distributed and there is no provision for key updates which is a serious security flaw. A similar scheme was presented in [19], which utilizes secret perturbation to address the issue of data confidentiality and privacy. The authors in [19] propose four different variations of their perturbation scheme to minimize bandwidth requirement. This scheme is lightweight, *end to end* and prevents the aggregator from knowing the aggregate, however, provides no data integrity. The algorithm presented in [21] uses additive digital signatures to provide *end to end* confidentiality, privacy and data integrity. [16] presented a similar algorithm, while fundamental flaws limit its applicability in wireless sensor networks, we improve upon and increase the efficiency of the algorithm presented in [21]. The energy efficiency however, can be further improved, by using homomorphic constructs which are based on symmetric key encryption such as in [18]. The scheme in [22] uses random perturbations to accomplish this. In this scheme secrets are deployed in the network tree hierarchy. These secrets are used to perturb data, which is in the form of histograms. The scheme achieves its objectives, however, it is restrictive in the sense that, only an approximation of the data can be provided. Table 2.2 provides a comparison of the various secure data aggregation scheme, discussed in this subsection.

Table 2.2. A comparison of secure data aggregation schemes

Scheme	HBH	ETE	Sym. key	Pub. Key	Confidentiality	Integrity	Privacy
Hu et.al.	X		X			X	
Chan et.al.	X		X			X	
SecureDAV	X			X		X	
SDAP	X		X			X	
CPDA	X				X		X
SMART	X				X		X
IPDA	X				X	X	X
ICPDA	X				X	X	X
CDA		X		X	X	X	X
Bahi et.al.		X		X	X		
EDA		X	X		X		X
Feng et.al.		X	X			X	X
Wang et.al.		X	X			X	X
Albath et.al.		X		X	X	X	X

2.3. USER ACCESS CONTROL

Schemes such as broadcast authentication and symmetric key as well as public key encryption have been used for access control since a long time. One of the first works, however, purely discussing access control and authentication of users in wireless sensors was by Benenson et. al [23]. In the proposed authenticated querying protocol in this paper, each user is authenticated by multiple nearby sensor nodes. This authentication is accomplished by using public key cryptosystem. To forward this query to a remote node, all the nodes which authenticated the user send additional information such as a MAC to the node. If the remote node receives the information from more than a fixed number of nodes, the user is authenticated. Wang and Li in [24] use a similar scheme which first authenticates the user on local sensors and uses the local sensor's commitment for authentication by remote sensors. Both the schemes use Elliptic Curve Cryptography for authentication on local nodes. However, while [23] uses SHA-1 for authentication by remote nodes, [24] uses t-degree bivariate polynomials to establish symmetric keys between the nodes for securely communicating authentication and access information. In [25], a symmetric key based approach to user access control is presented. In this approach the user approaches the trusted authority with its credentials. Based on the credentials, the trusted authority determines the constraints for the particular user. A key is then generated using the constraint, the user's identity and the symmetric key of the sensor node the user wants to access. When the user wants to access data from the sensor it can contact the sensor node with its identity and the credentials. The sensor node generates a symmetric key using this information. The communication between the sensor node and the user is done using this symmetric key. The paper further also explores delegation of access rights. In [26] also, a combination of symmetric key cryptography and MAC has been used to provide role based access control. In this paper however, the problem is viewed from a service oriented architecture point of view, where each node provides a service and the access is dependent upon the role of the user in the system. All of these approaches however suffer from the problem that access control is all-or-none. A user either has the access to all the data being generated by a node or none of the data being generated by a node.

A more fine grained access control of sensor data was presented in [27]. The scheme proposed in this paper used the key policy attribute based scheme (KP-ABE). In [27] each node and the nodes data was associated with a set of attributes. The attributes themselves are associated with public key components. An access tree based on the attributes of required data is created for a given user. The access tree is then used to generate the private key which is provided to the user. The user then provides the access tree to the sensor and the sensor provides data to the user according to the access tree. The data is encrypted such that it can only be decrypted with a private key generated based on that access tree. This scheme was further enhanced in [28], which included multiple base stations. This paper also introduced a revocation scheme which was energy efficient on the sensor node while consuming more energy on the user side. The system model in [29] looks at the same problem from a different point of view. In this scheme the device itself, which is generating the data, controls the access by encrypting it according to the access policies. The scheme uses CP-ABE [30] for access control of data, where the access policy is embedded in the ciphertext itself. This scheme however, is computationally inefficient and consumes a lot of energy on resource constrained wireless sensors.

None of the schemes, in the reviewed literature however work with data which is aggregated in network. In a sensor cloud, the sensor networks can be large and data aggregation is preferred when collecting data to save energy and bandwidth on the nodes. Thus we require a scheme which can work with multiple sensor nodes which are aggregating data in-network. In section III, we present and discuss such a scheme.

2.4. SECURE CODE DISSEMINATION

It has long been understood, that the best way to reprogram a wireless sensor network is through wireless reprogramming. The XNP protocol [31] was one of the first algorithms designed for this purpose. The XNP protocol however, could only reprogram the nodes within a single hop from the base station. Each node thus, needed to be within the base station's communication range to receive the code image. Deluge [32] provided one of the first fully functional, efficient way of programming wireless sensors remotely. Deluge [32] implemented an Advertise-Request-Receive model, where nodes advertise if they have a new code image. Nodes which are running the older versions of the code image request new image upon receiving the advertisement and then receive the packets for the code image. This algorithm however, did not take into account, the similarities in application code to reduce energy consumption. The algorithm in [33] by Reijers et. al. discussed a difference-based code dissemination scheme. This algorithm attempted to reduce the amount of transmitted code by sending only the difference between the new and the old code image. The paper introduced commands such as *insert, copy, repair, repair dbl* and *patch list* to generate an edit script. Instead of wirelessly sending the complete new code, the base station would only transmit the edit script. The wireless sensors would then transform their code image according to the edit script to generate the new version of the code. The edit script however made the algorithm, platform dependent. A platform independent scheme was introduced in [34]. In this algorithm the code image was divided into small fixed size blocks and hashes of each block were calculated. This was done for each new version of the code image too. A difference script consisting of *copy* and *download* commands was created using the difference in hashes. Algorithms [33] and [34] however, only work for small code changes, recently the QDiff algorithm [35] was proposed which could handle much greater code changes than previous algorithms. The QDiff algorithm [35] works on the ELF file level and hence is platform independent. It uses slop regions to maintain similarity between two versions of the code. If no slop region exists the new code is moved to the end of the file. A high level of similarity between the codes at the ELF file level ensures a small patch size. Moreover, the patch can be directly applied in the RAM, which eliminates the need of a reboot, thus saving a large amount of energy.

The above schemes, tackled the problem of reducing the energy consumption in code dissemination, they however, did not touch upon security. In a sensor cloud, security of the transmitted code is also equally important, since the code and along with it, all the keying material is disseminated via multiple networks. One of the early schemes, which combined security with code dissemination was Seluge [36]. It attempted to tackle the code image integrity and various DoS attacks on Deluge [32]. For each code image, it hashes the code packets of the last page and concatenates the hashes with the packets of the previous page. This process is performed recursively till all the packets of the first page are hashed. The hashes from the first page are then used to create a hash tree, the root of which is signed by the base stations private key. A page 0 is then constructed which consists of the information to verify the root hash. In the end a signature packet is constructed which consists of the root hash, the meta data about the code and the signature over the root hash. Seluge [36] however, did not offer confidentiality of code. The algorithm in [37] discussed confidentiality in code dissemination. This scheme handles the code image differently than [36]. All packets are considered to be in a sequence rather than as pages. The hash of a packet is generated in much the same way as [36], except that the last packet is concatenated with an L-byte nonce. The first L-bytes of the hash of a packet are also used as the key for encrypting the packet. The whole sequence of packets is thus encrypted and the hash of the first packet is then used to construct a cipher puzzle and is signed by the base station. These solutions protect the code against an outside adversary. In sensor cloud, on the other hand the adversary is taken as inside the network and hence these scheme fail to adapt adequately. In section IV we discuss a code dissemination scheme which is both efficient and well suited to sensor clouds.

3. BIBLIOGRAPHY

[1] A. Parakh and S. Kak, Recursive secret sharing for distributed storage and information hiding. In *Proceedings of the 3rd international conference on Advanced networks and telecommunication systems*, ANTS'09, pages 88–90, Piscataway, NJ, USA, 2009. IEEE Press.

[2] A. Syalim, T. Nishide, and K. Sakurai, Realizing proxy re-encryption in the symmetric world In *Informatics Engineering and Information Science*, ser. Communications in Computer and Information Science, A. Abd Manaf, A. Zeki, M. Zamani, S. Chuprat, and E. El-Qawasmeh, Eds., 2011, vol. 251, pp. 259–274.

[3] A.S. Wander, N. Gura, H. Eberle, V. Gupta, and S.C. Shantz. Energy analysis of public-key cryptography for wireless sensor networks. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 324–328, 2005.

[4] D. Boneh, C. Gentry, B. lynn, and H. Shacham. A Survey of Two Signature Aggregation Techniques. In *Cryptobytes, 2003*, 2003.

[5] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[6] G. Blakley. Safeguarding cryptographic keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, volume 48, pages 313–317, June 1979.

[7] A. Sahai and B. Waters. "Fuzzy identity-based encryption," in *Advances in Cryptology EUROCRYPT 2005*, ser. Lecture Notes in Computer Science, R. Cramer, Ed. Springer Berlin Heidelberg, 2005, vol. 3494, pp. 457–473.

[8] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 89–98.

[9] L. Hu and D. Evans, "Secure aggregation for wireless networks," in *Workshop on Security and Assurance in Ad hoc Networks*. IEEE Computer Society, 2003.

[10] H. Chan, A. Perrig, and D. X. Song, "Secure hierarchical in-network aggregation in sensor networks," in *Computer and Communications Security*, ser. CCS '06, 2006, pp. 278–287.

[11] A. Mahimkar and T. S. Rappaport, “SecureDAV: A secure data aggregation and verification protocol for sensor networks,” in *Proceedings of the IEEE Global Telecommunications Conference*, 2004, pp. 2175–2179.

[12] Y. Yang, X. Wang, S. Zhu, and G. Cao, “SDAP: a secure hop-by-hop data aggregation protocol for sensor networks,” in *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, ser. MobiHoc ’06. ACM, 2006, pp. 356–367.

[13] W. He, X. Liu, H. Nguyen, K. Nahrstedt, and T. Abdelzaher. Pda: Privacy-preserving data aggregation for information collection. *ACM Trans. Sen. Netw.*, 8(1):6:1–6:22, August 2011.

[14] W. He, H. Nguyen, X. Liuy, K. Nahrstedt, and T. Abdelzaher. ipda: An integrity-protecting private data aggregation scheme for wireless sensor networks. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1 –7, nov. 2008.

[15] W. He, X. Liu, H. Nguyen, and K. Nahrstedt. A cluster-based protocol to enforce integrity and preserve privacy in data aggregation. In *Distributed Computing Systems Workshops, 2009. ICDCS Workshops ’09. 29th IEEE International Conference on*, pages 14 –19, june 2009.

[16] H.-M. Sun, Y.-C. Hsiao, Y.-H. Lin, and C.-M. Chen, “An efficient and verifiable concealed data aggregation scheme in wireless sensor networks,” in *Proceedings of the 2008 International Conference on Embedded Software and Systems*. IEEE Computer Society, 2008, pp. 19–26.

[17] J. Bahi, C. Guyeux, and A. Makhoul, “Efficient and robust secure aggregation of encrypted data in sensor networks,” in *Fourth International Conference on Sensor Technologies and Applications*, ser. SENSORCOMM ’10, July 2010, pp. 472–477.

[18] C. Castelluccia, E. Mykletun, and G. Tsudik, “Efficient aggregation of encrypted data in wireless sensor networks,” in *Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, ser. Mobiquitous ’05. IEEE Computer Society, 2005, pp. 109–117.

[19] T. Feng, C. Wang, W. Zhang, and L. Ruan. Confidentiality protection for distributed sensor data aggregation. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 56–60, 2008.

[20] C. Castelluccia, A. C-F. Chan, E. Mykletun, and G. Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Trans. Sen. Netw.*, 5(3):20:1–20:36, June 2009.

[21] J. Albath and S. Madria, "Secure hierarchical data aggregation in wireless sensor networks," in *WCNC*. IEEE, 2009, pp. 2420–2425.

[22] C. Wang, G. Wang, W. Zhang, and T. Feng. Reconciling privacy preservation and intrusion detection in sensory data aggregation. In *INFOCOM, 2011 Proceedings IEEE*, pages 336 –340, april 2011.

[23] Z. Benenson, N. Gedicke, and O. Raivio, "Realizing robust user authentication in sensor networks" *Real-World Wireless Sensor Networks (REALWSN)*, vol. 14, 2005.

[24] H. Wang and Q. Li, "Achieving distributed user access control in sensor networks," *Ad Hoc Networks*, vol. 10, no. 3, pp. 272 – 283, 2012.

[25] D. Liu, "Efficient and distributed access control for sensor networks," in *Distributed Computing in Sensor Systems*, ser. Lecture Notes in Computer Science, J. Aspnes, C. Scheideler, A. Arora, and S. Madden, Eds. Springer Berlin Heidelberg, 2007, vol. 4549, pp. 21–35.

[26] J. Maerien, S. Michiels, C. Huygens, D. Hughes, and W. Joosen, "Access control in multi-party wireless sensor networks," in *Wireless Sensor Networks*, ser. Lecture Notes in Computer Science, P. Demeester, I. Moerman, and A. Terzis, Eds. Springer Berlin Heidelberg, 2013, vol. 7772, pp. 34–49.

[27] S. Yu, K. Ren, and W. Lou, "Fdac: Toward fine-grained distributed data access control in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 4, pp. 673–686, 2011.

[28] S. Ruj, A. Nayak, and I. Stojmenovic, "Distributed fine-grained access control in wireless sensor networks," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, 2011, pp. 352–362.

[29] G. Bianchi, A. T. Capossele, C. Petrioli, and D. Spenza, "Agree: exploiting energy harvesting to support data-centric access control in {WSNs}," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2625 – 2636, 2013.

[30] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, 2007. SP '07. IEEE Symposium on*, 2007, pp. 321–334.

[31] J. Jeong, S. Kim, and A. Broad, "Network Reprogramming," *University of California at Berkeley*, 2003.

[32] J. Hui and D. Culler, “The dynamic behavior of a data dissemination protocol for network programming at scale,” in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 81–94.

[33] N. Reijers and K. Langendoen, “Efficient code distribution in wireless sensor networks,” in *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, ser. WSNA ’03. New York, NY, USA: ACM, 2003, pp. 60–67.

[34] J. Jeong and D. Culler, “Incremental network programming for wireless sensors,” in *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, oct. 2004, pp. 25 – 33.

[35] N. Bin Shafi, K. Ali, and H. Hassanein, “No-reboot and zero-flash over-the-air programming for wireless sensor networks,” in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th Annual IEEE Communications Society Conference on*, june 2012, pp. 371 –379.

[36] S. Hyun, P. Ning, A. Liu, and W. Du, “Seluge: Secure and dos-resistant code dissemination in wireless sensor networks,” in *Proceedings of the 7th international conference on Information processing in sensor networks*. IEEE Computer Society, 2008, pp. 445–456.

[37] H. Tan, D. Ostry, J. Zic, and S. Jha, “A confidential and dos-resistant multi-hop code dissemination protocol for wireless sensor networks,” *Computers & Security*, vol. 32, no. 0, pp. 36 – 55, 2013.

PAPER

I. SECURE HIERARCHICAL DATA AGGREGATION IN WIRELESS SENSOR NETWORKS: PERFORMANCE EVALUATION AND ANALYSIS

Vimal Kumar∗, Sanjay K Madria∗

∗ Department of Computer Science,

Missouri University of Science and Technology, Rolla, Missouri 65401

Secure data aggregation in wireless sensor networks has two contrasting objectives, i) Efficiently collecting and aggregating data and ii) Aggregating the data securely. Many schemes do not take into account the possibility of corrupt aggregators and allow the aggregator to decrypt data in hop by hop algorithms. On the other hand using public key cryptography for providing end to end security is not energy efficient. In this paper we present and analyze the performance of the secure hierarchical data aggregation algorithm which uses an efficient public key cryptosystem (elliptic curve cryptography) to achieve end to end security. Unlike many other secure data aggregation algorithms which require separate phases for secure aggregation and integrity verification, the secure hierarchical data aggregation algorithm does not require an additional phase for verification. This saves energy by avoiding additional transmissions and computational overhead on the sensor nodes. We present and implement the secure data aggregation algorithm on Mica2 and TelosB sensor network platforms and measure the execution time and energy consumption of various cryptographic functions. We have also simulated our algorithms to analyze how an end to end scheme increases the network life time. We experimentally analyze our algorithms based on parameters like throughput, end to end delay and resilience to node failures.

1. INTRODUCTION

The small size of wireless sensors allows them to be easily deployed in hostile environment in large numbers without being noticed however it presents a few constraints namely limited memory, available processing power and the size of battery. The presence of these constraints calls for judicious use of the available power to maximize sensor lifetime and in turn network lifetime.

A sensor expends most of its energy in communicating with other sensors. Our objective therefore is to minimize the amount of communication by using data aggregation. However, data aggregation introduces new issues with respect to data security. When data is aggregated, it loses its inherent identification information. As a result of this, if proper mechanisms are not in place, an intermediate node can corrupt the data surreptitiously. We discuss this issue of *corrupt aggregator* along with the issues of *confidentiality* and *data integrity* in detail in this paper.

Two different types of secure data aggregation schemes have been proposed by researchers, *hop by hop* and *end to end* secure data aggregation schemes. End to end schemes provide better security and require lesser amount of computation which helps in reducing energy consumption. However, most previous secure data aggregation algorithms were two phase algorithms; first phase for secure data aggregation and second phase for verification. Secure hierarchical data aggregation in wireless sensor networks [1] provides an overview of the possibility of a single phase end to end scheme for both aggregation and verification. Though our paper carries forward the preliminary ideas proposed by us in [1], we have modified those as follows. We replace Elliptic Curve Integrated Encryption Scheme (ECIES) algorithm used in [1] with Elliptic Curve El Gamal (ECEG) to make it truly homomorphic and also provide a secure and robust tree construction algorithm for reliable passage of data. We demonstrate that our algorithm saves energy and increases network life time in two ways. First, it replaces energy intensive decryption and verification operations by light weight ciphertext, digital signatures and public key addition operations on the aggregators. Second, it does not require a separate integrity verification phase. We make use of aggregate digital signature to relieve the network of the extra communication cost. In addition, we have modified these algorithms further for efficient execution and implementation on sensor network test-bed consisting of Mica2 and TelosB motes to evaluate their performance. We report in-depth experimental results of the algorithm using sensor test-bed to measure energy, execution time and throughput under various conditions of failures and attacks. To the best of our knowledge this is the first paper which provides design, implementation, performance and analysis of a single phase secure data aggregation algorithm on two different sensor motes platform.

2. RELATED WORK AND BACKGROUND

The biggest challenge while working with wireless sensors is the limited available battery power. Radio communication consumes a large amount of energy on a sensor [11]. Minimizing communication between nodes therefore is vital in wireless sensor networks, owing to which, data aggregation has been a topic of interest to researchers in this field. Early secure data aggregation schemes were *hop by hop* schemes. Schemes like [2] mostly deal with the issue of data confidentiality in the face of a single compromised node. In this scheme, while a parent aggregates the MACs sent by its children, it doesnt aggregate the data. Aggregation of data is done by the grandparent. A compromised node can either corrupt the MACs or the data but not both. Thus a single compromised node will always be identified; however the algorithm fails when both the parent and the grandparent are compromised. Often when an adversary strikes, he captures a handful of nodes in the attacked region and not just a single node. Schemes tackling the issue of multiple compromised nodes were introduced later, for example the scheme by Chan et al [3]. This algorithm is resilient to any number of malicious nodes but deals only with attacks on data integrity. The integrity verification algorithm here is distributed through the network which reduces the communication load on certain nodes and increases the time to first node failure. Schemes like SecureDAV [4] and SDAP [5] also provided for data integrity by making use of threshold cryptography and Merkle hash trees. SecureDAV [4] like [3] does not have confidentiality protection. It uses threshold cryptography for proving authenticity of the messages and Merkle hash trees for integrity. Similar to [3], SDAP[5] also aims to reduce the congestion around the nodes which are placed high up in the hierarchy. To accomplish this, the algorithm logically partitions the aggregation tree into sub trees of similar sizes called groups. Data integrity is provided by using a commit and attest technique. *Hop by hop* schemes though leave the data exposed to aggregators. A malicious aggregator can compromise the security of the scheme. Consequently *end to end* secure data aggregation schemes were proposed, some of which are discussed in [1], [6], [7] and [8]. These algorithms use the concept of homomorphic encryption for adding encrypted data. An encryption algorithm is said to be homomorphic if it allows for the following property to hold.

$$Enc(a) \oplus Enc(b) = Enc(a \oplus b)$$

Similar to homomorphic encryption an aggregate digital signature algorithm provides the functionality to aggregate n signatures on n distinct messages by n distinct users, into a single signature. The algorithm in [6] uses Elliptic Curve Cryptography (ECC) for homomorphic encryption but does not provide data integrity. Castelluccia et al.s algorithm in [7] uses modular addition as the homomorphic encryption algorithm. Modular addition is a straightforward operation which has the advantage of simplicity over other homomorphic encryption algorithms. However, the algorithm requires the keys be pre-distributed and there is no provision for key updates which is a serious security flaw. While [6] and [7] do not provide data integrity, both [1] and [8] use aggregate signature protocols for the same. The algorithm in [8] is closest to our algorithm which uses Elliptic Curve Digital Signature Algorithm (ECDSA) since it tries to provide both end to end confidentiality and data integrity. However, fundamental flaws limit its usability. The algorithm encodes the data in a special form before aggregation. While encoding enables the algorithm to retrieve individual sensors data at the base station, it limits the scalability of the scheme. The size of data increases with number of nodes which makes the algorithm unsuitable for large networks. Algorithm in [9] provides confidentiality but forgoes data integrity. A detailed comparison of the related work can be found in [10]. There exists no prior work which has evaluated the performance of a secure data aggregation scheme by implementing it on real motes. This paper comprehensively discusses the implementation, simulation and performance analysis of the secure hierarchical data aggregation algorithm. Table 2.1 presents a comparison of the number of transmissions required in the verification phase of various secure data aggregation algorithms. Our scheme uses aggregate digital signatures which eliminate the need of a data verification phase. This is advantageous as a separate data verification phase means a number of extra wireless transmissions which consumes energy as shown in the table. In the table n is number of nodes in the tree, d is degree of the tree h is the height of the tree in hops.

Table 2.1. Number of transmissions in the integrity verification phase

Scheme	**Number of messages**
SDAP	$h/2+\sum_{i=1}^{h/2}(h/2+i)2.$
SecureDAV	h
Secure aggregation for wireless sensors	$h/2$
Secure hierarchical in network aggregation	$(2^{h-1}-1)\sum_{i=2}^{h}2^{i-1}$

3. SECURE HIERARCHICAL DATA AGGREGATION ALGORITHM

The secure hierarchical data aggregation algorithm employs homomorphic encryption and aggregate digital signatures for end to end confidentiality and data integrity. The original algorithm [1] specifies the use of ECIES for encryption and a modified version of ECDSA for signing. ECIES however is not suitable for homomorphic encryption and hence we replaced it with ECEG in our work. We designed a tree construction algorithm shown in Algorithm 1 which is secure and favors strong bidirectional links and organizes the nodes in a tree hierarchy. The tree construction algorithm consists of a reconfiguration procedure shown in Algorithm 2 for nodes which get disconnected from the tree for any reason. This procedure helps in increasing the connectivity of the network when under a node capture or denial of service attack. After the nodes have been organized in a tree, each node generates a reading x. The reading is signed using the aggregate signature algorithm and SIG(x) is generated. The reading is then encrypted using the homomorphic encryption algorithm and the ciphertext ENC(x) is produced. The leaf nodes then send the encrypted data, signature and the public key corresponding to the private key used for generating signature to their parent. After a node has received data from all its children, it sums up all the encrypted readings, which is possible because homomorphic encryption is being used. It sums up the signature using the aggregate signature algorithm and all the public keys. SUM-ENC, SUM-SIG and SUM-PK are then sent to the nodes parent. This process is repeated at every node until the data reaches the base station. In the remainder of this section we discuss the signature and encryption algorithms in more detail.

3.1. MODIFIED ECDSA SIGNATURE ALGORITHM

The signature algorithm in [1] assumes the sensors are preloaded with the appropriate elliptic curve parameters, the base station's public key and a network wide random integer.

Algorithm 1 Tree construction algorithm

```
Require: Parameters MAX_CHILDREN, NUM_NODES, Secret key $k$ deployed on each node
 1: The base station starts by broadcasting a HELLO message, which consists of a random number $r$.
 2: If a sensor which has not yet elected its parent receives a HELLO message, it calculates HMAC$_k(r)$ and sends it in a PARENT REQUEST to the originator of the HELLO message.
 3: if a PARENT REQUEST is received then
 4:     Calculate HMAC$_k(r)$ and check whether it matches the HMAC in the packet.
 5:     Check whether RSSI value of the request packet is greater than a minimum threshold.
 6:     Check that the number of children is less than the MAX_CHILDREN limit.
 7:     The number of requests from a particular node is less than the allowed limit.
 8:     If the above four checks are satisfied, send an ACCEPTED message otherwise send REJECTED.
 9: end if
10: Upon receiving an ACCEPTED message a node elects the sender of the ACCEPTED message as its parent and broadcasts a HELLO message.
11: If a sensor has not been able to elect a parent after a certain period of time it invokes the HELP procedure.
12: In case of a disconnection the HELP procedure is invoked by the affected nodes.
```

Algorithm 2 HELP procedure

```
1: The sensor invoking the help procedure broadcasts a HELP request with a random number $r_1$.
2: Nearby sensors who are higher in the hierarchy than the HELP invoking sensor and can accept more children reply to the HELP request with HMAC$_k(r_1)$ and a random number $r_2$.
3: The sensor calculates HMAC$_k(r_1)$, compares it with the HMAC$_k(r_1)$ in the incoming packets and chooses one of the replying nodes as its parent. It then sends HMAC$_k(r_2)$ to the parent.
4: The parent verifies HMAC$_k(r_2)$ and upon verification accepts the sensor as a child.
```

The random integer is used to compute a new k for each round. At the start of each new round, sensors choose their private key z and generate a corresponding public key $Q = zT$ where T is the base point. In the ECDSA algorithm a signature is a tuple (r,s) such that

$r = (r(x) \bmod p)$, where $(r(x), r(y)) = k\mathrm{T}$ and $s = k^{-1}(h(m) + z * r(x)) \bmod p$. Here h is a secure hash function and p is a prime. When two signatures $d_1 = (r_1, s_1)$ and $d_2 = (r_2, s_2)$ on two messages m_1 and m_2 are added, r_1 and r_2 are equal while s_1 and s_2 can be written as $s_1 = k^{-1}(h(m_1) + z * r(x))$ and $s_2 = k^{-1}(h(m_2) + z * r(x))$. ECDSA is not an aggregate signature scheme because when these two signatures are added $h(m_1)$ and $h(m_2)$ need to be added. Hashing is not homomorphic so $h(m_1) + h(m_2) \neq h(m_1 + m_2)$ hence an aggregate signature will not be the same as the signature on the sum of messages. On the other hand if we replace the hash of the message by the message itself in the formula the signature becomes additive since we are summing up integers. The signature (r, s) in the modified signature scheme is $r = (r(x) \bmod p)$ and $s = k^{-1}(m + z * r(x)) \bmod p$. Not using hashing can make ECDSA vulnerable to existential forgery attack, where the attacker can substitute a suitable message/signature pair in place of the original message and signature. We tackle this by only allowing the sensors to send a part of the signature. The sensors send the s component while the r component is generated at the base station. The attacker will not be able to modify the complete signature and hence will not be able to launch a successful existential forgery attack. This is discussed in detail in Section 4.1.

3.2. EC ELGAMAL ENCRYPTION

We use the additive homomorphic elliptic curve elgamal encryption (ECEG) encryption scheme for confidentiality. Before we encrypt a message using ECEG we first need to map the plaintext data to a point on the elliptic curve. We use a simple homomorphic mapping technique where we multiply the plaintext message m by the base point T, to get the elliptic curve point mT. Each message m_i maps to a point M_i on the elliptic curve. The M_is are added, and the addition of the elliptic curve points is equivalent to the addition of the plaintext data. The plaintext can be found by reverse mapping the final result. This process can be seen in Algorithm 3.

```
Algorithm 3 Elliptic Curve Elgamal
Require: Elliptic curve parameters $D = (q, FR, a, b, T, p, h)$, sensor reading $m_i$, base station's private key $z_e$ and base station public key $Q_e = z_e T$
    Encryption
1:  Map the message $m$ to an elliptic curve point $M$ using a mapping function.
2:  Generate a random integer $a_i$.
3:  Calculate $C_1 = a_i T$ and $C_2 = M + a_i Q_e$.
4:  $(C_1, C_2) = (a_i T, M + a_i Q_e)$ is the ciphertext.
    Decryption
5:  Calculate $(-z_e * C_1)$ and add it to $C_2$.
6:  The decrypted message $M$ is the addition $(-z_e * C_1) + C_2$.
```

$$
\begin{aligned}
M_1 + M_2 + ... + M_n &= map(m_1) + ... + map(m_n) \\
&= m_1 T + m_2 T + ... + m_n T \\
&= (m_1 + m_2 + ... + m_n) T \\
&= (\sum m_i) T
\end{aligned}
$$

The inverse of this mapping function is a brute force attack on the elliptic curve point mT given point T. In our algorithm decryption is only done at the base station. Since we assume the base station to be a powerful machine, reverse mapping the point using a brute force method aided by knowledge of the range of sensed data should not be an issue.

4. PERFORMANCE ANALYSIS

4.1. IMPLEMENTATION

We implemented our secure data aggregation scheme on two different mote platforms, Mica2 and TelosB. The specifications of these platforms are shown in Table 4.1. The coding was done using the TinyOS platform. We made use of the TinyECC library [12] for cryptographic operations. TinyECC [12] is implemented over the prime field $\mathbb{F}_p$ where p is a large prime number. The library consists of routines for large natural number operations and ECC operations. For simulation of our testbed we used 160 bit ECC keys which provide security equivalent to 1024 bit RSA keys. We measured the time taken and the corresponding energy consumption of our crypto functions on both Mica2 and TelosB platforms for 128, 160 and 192 bit keys. We calculate the energy consumption for cryptographic operations as follows. The time taken t to complete an operation is multiplied by the voltage supplied V and the current drawn I during active mode. The energy consumed is thus calculated as $E = V \text{ x } I \text{ x } t$.

Table 4.1. Specifications of Mica2 and TelosB

Specification	**Mica2**	**TelosB**
Processor	MPR400CB 8 bit Microcontroller	TI MSP430 16 bit Microcontroller
Program flash	128kB	48kB
RAM	4kB	10kB
Clock Speed	7.3827 MHz	8MHz
Baud Rate	38.4Kbaud	250Kbaud

The execution times of various operations for Mica2 and TelosB motes is shown in Figure 4.1 and Figure 4.2 respectively. The corresponding energy consumption is shown in

Figure 4.3 and Figure 4.4 respectively. The decryption and verification functions are only performed at the powerful base station and are replaced by ciphertext addition, signature addition and key addition on the sensor motes. As can be seen, these three operations are very lightweight and incur very little energy, thus reducing end to end delay and increasing the network life time.

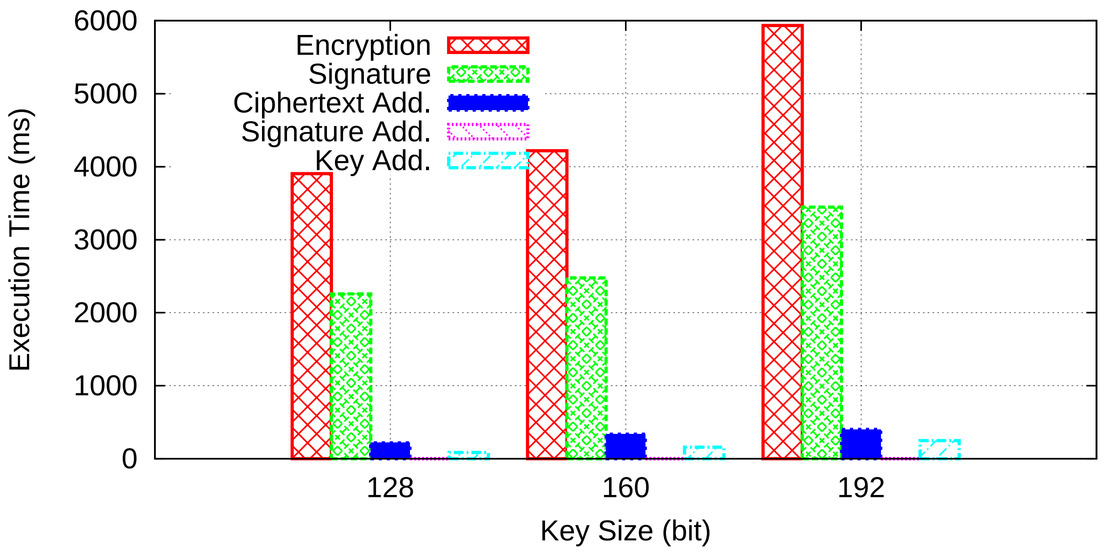

Figure 4.1. Execution times on Mica2 for 128, 160 and 192 bit keys

In TinyOS 1.x the maximum packet size on TOSSIM is 119 bytes whereas on a Mica2 mote the maximum packet size is 241 bytes. In our algorithm each sensor needs to transmit the ciphertext, the digital signature and its public keys. The ciphertext consists of two components (C_1, C_2) which are points on the elliptic curve, the digital signature consists of two components (r, s). The size of these packets however can be optimized. We start by noting that the requirement for the modified version of ECDSA to be additive is that the random number k chosen by all the sensors should be same. In our algorithm a new k is generated in each round of data sensing such that all the sensors and the base station generate the same k. The digital signature in ECDSA is a tuple (r, s) such that r is the

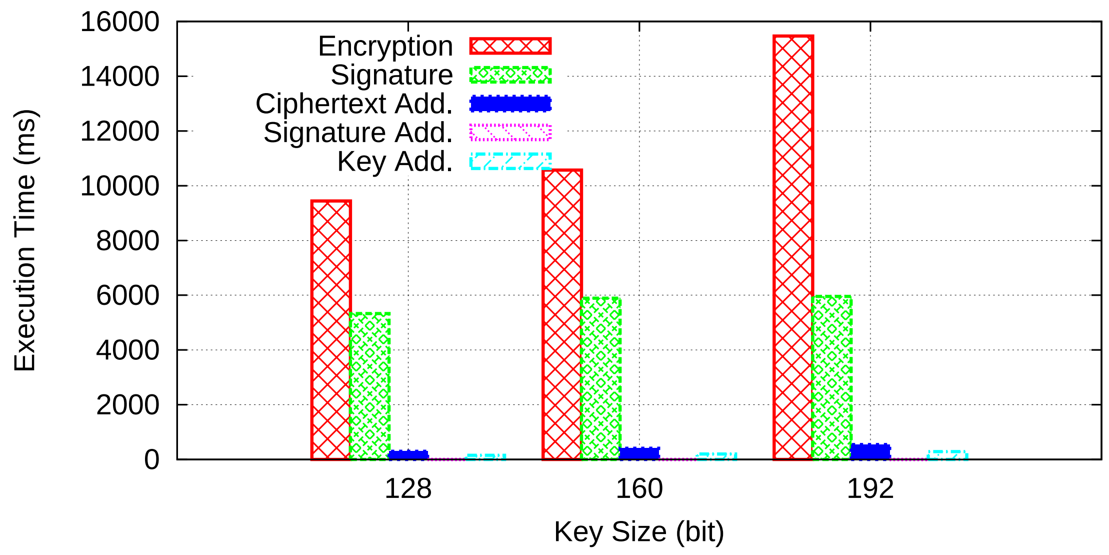

Figure 4.2. Execution times on TelosB for 128, 160 and 192 bit keys

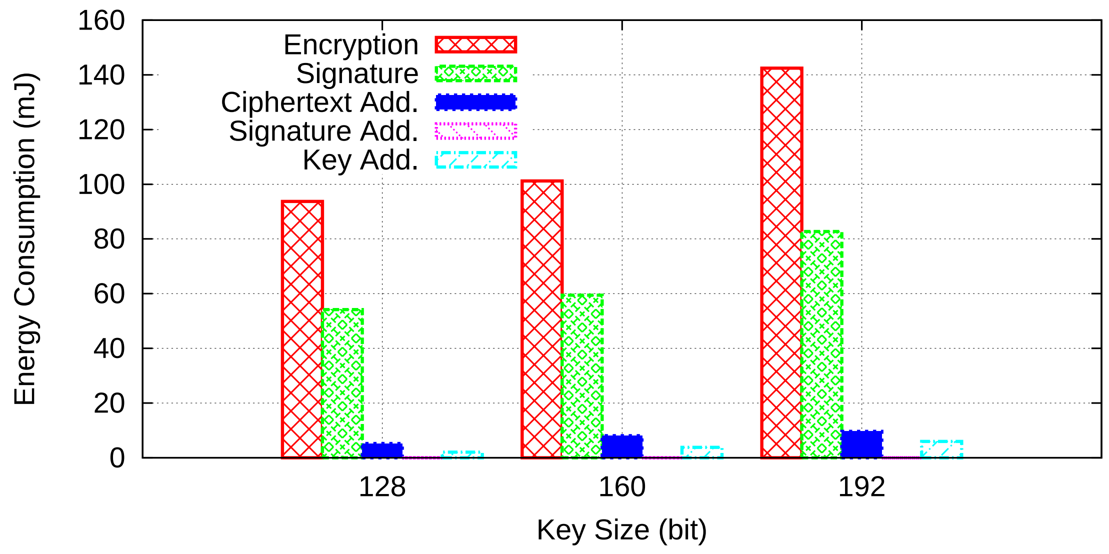

Figure 4.3. Energy consumption on Mica2 for 128, 160 and 192 bit keys

x-component of the product of k and the base point T, i.e $r = r(x)$ where, $(r(x), r(y)) = kT$ and $s = k^{-1}(m + zr)$. Since we are using the same k at all sensors in any particular iteration, the r component of the signature will also be same at all the sensors. This implies that we

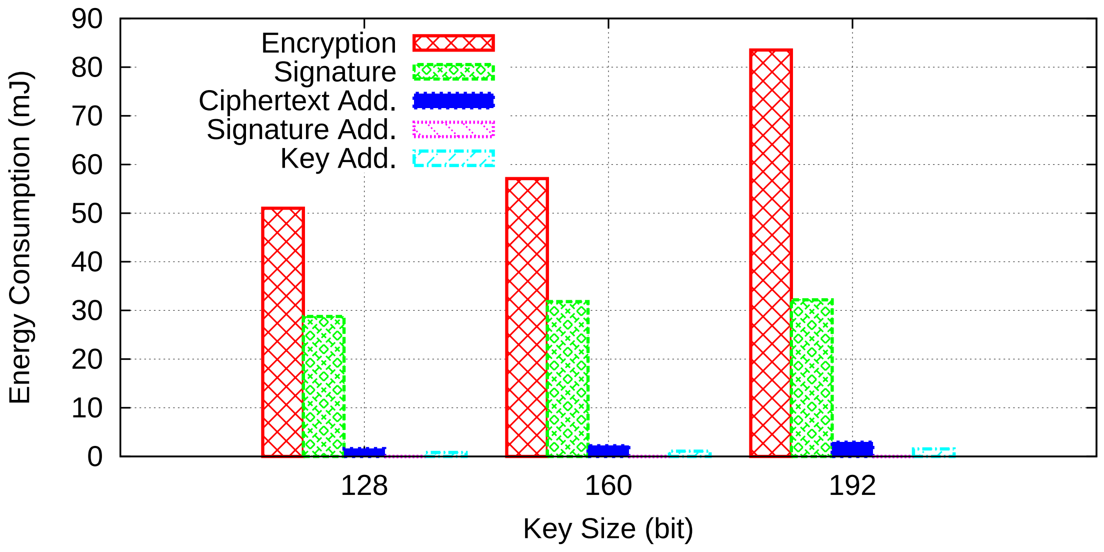

Figure 4.4. Energy consumption on TelosB for 128, 160 and 192 bit keys

can generate the r component at the base station if we know k, and thus it does not need to be transmitted. As discussed before, this also helps in avoiding existential forgery attacks To optimize the size of the ciphertext to be sent we note that the ciphertext in Elgamal encryption consists of two components C_1 and C_2. $C_1 = a_i * T$ and $C_2 = M + a_iQ_e$ where a_i is a random integer. However if we take a_i to be a pseudo random integer, generated using a seed shared with the base station, we will not need to transmit the C_1 component of the ciphertext. Since the base station knows the seed and can generate the pseudo random integer a_i, it can calculate C_1 easily and use it to decrypt the ciphertext. Thus, we were able to eliminate the r-component of the digital signature and the C_1 component of the ciphertext from the packet thus reducing the total to 105 bytes. We use a one byte field to indicate the total number of nodes involved in the aggregation and 1 byte for indicating the position of sender in the aggregation tree. Thus the total size of the packet is 107 bytes which is now well within the limit of 119 bytes for TOSSIM. The energy consumed in transmission and reception of this packet on Mica2 and TelosB mote platforms is shown in Table 4.2.

Table 4.2. Energy consumed during reception and transmission of packets

Mode	**Power (dBm)**	**Current Drawn (mA)**	**Energy Consumed (mJ)**	
			Mica2	**TelosB**
Rx		7.0	0.498	0.077
Tx	-20	3.7	0.247	0.040
Tx	-19	5.2	0.349	0.057
Tx	-15	5.4	0.361	0.059
Tx	-8	6.5	0.434	0.071
Tx	-5	7.1	0.473	0.078
Tx	0	8.5	0.569	0.093
Tx	4	11.6	0.774	0.127
Tx	6	13.8	0.920	0.151
Tx	8	17.4	1.160	0.190
Tx	10	21.5	1.434	0.235

4.2. SIMULATION

For further performance analysis we simulated a network of sensor nodes in TOSSIM. The nodes are organized in a grid and are placed at 5 feet from each other. We used the TinyOS lossy radio model to simulate a noisy environment. For each link between any two nodes, the model generates a bit error probability which represents the probability of a bit being flipped during the transmission. Packet error probability depends on the size of the packet and can be derived using the bit error probability. The key size for security operations is 160 bit as stated before.

Figures 4.5 and 4.6 show the throughput of the algorithm for a network of 25 and 50 nodes respectively. We define throughput as the ratio of number of packets received by the aggregators to the number of packets sent by the aggregators. Figure 4.5 shows the throughput of a network of 25 nodes for 4 different cases and packet sizes. As expected the no security case provides the best throughput. Tinysec [13], which is a link layer symmetric encryption scheme has a throughput of 97.75%. Tinysec [13] uses symmetric ciphers RC5 and Skipjack with a 64 bit key. However, it uses a network wide key, which although good for in network processing, cannot protect against node capture attacks. If the adversary is able to compromise one node, he will be able to eavesdrop on all the communication in the system. When only confidentiality protection is used in our algorithm the throughput reduces to 95.62%. The reduction in throughput is due to increased packet size which in this case is 86 bytes. As the size of the packet increases, the probability of packet collision and CRC corruption also increases which amounts to packet loss and thus reduces the throughput. The last case is when both confidentiality protection and integrity verification are used. The packet size in this case increases to 107 bytes and the throughput decreases to 92.7%. A similar trend can be seen with a network of 50 nodes in Figure 4.6. The throughput in all the cases is lower than in the previous case due to larger number of nodes, however the general trend remains the same. Secure hierarchical data aggregation algorithm provides greater security at the cost of a very nominal drop in throughput.

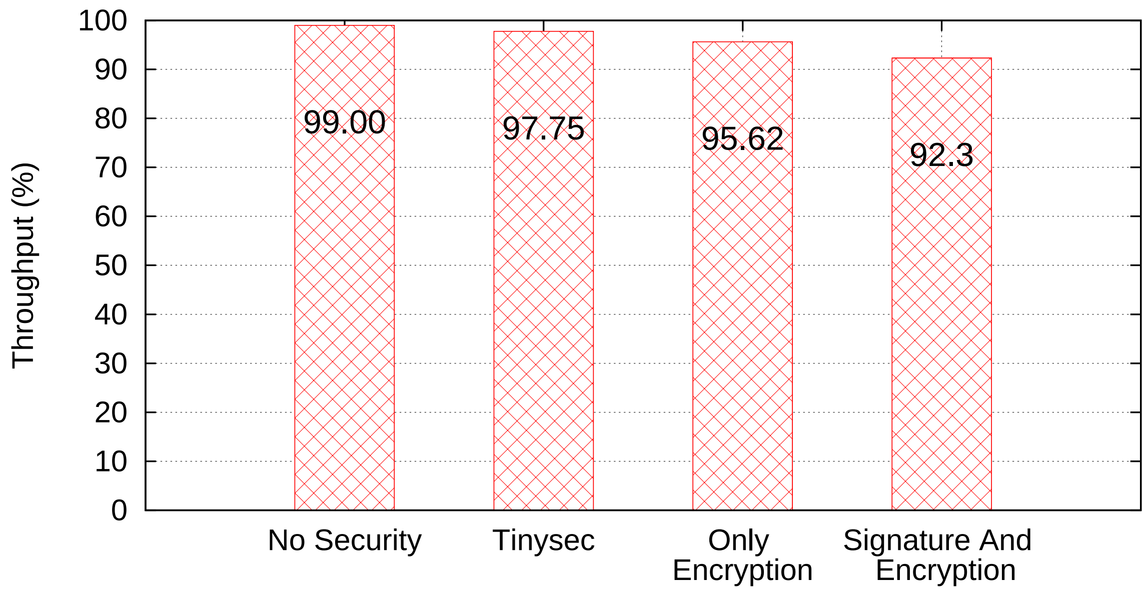

Figure 4.5. Throughput for a network of 25 nodes

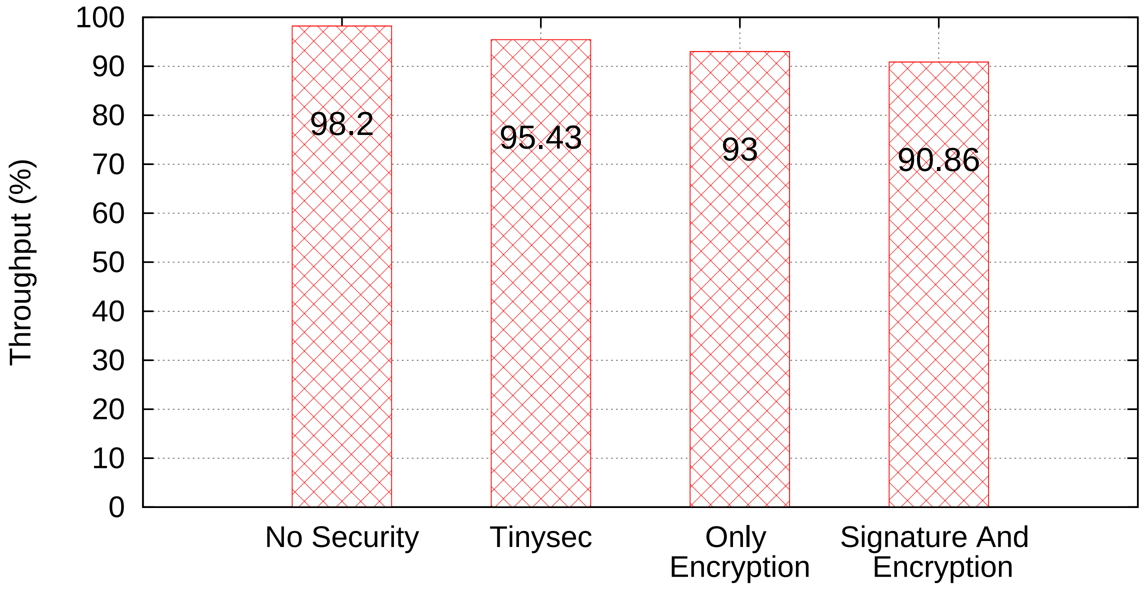

Figure 4.6. Throughput for a network of 50 nodes

Figure 4.7 shows throughput when the number of nodes in the network is varied. Probability of packet collision increases with number of nodes, which reduces the throughput. Figure 4.8 shows the connectivity of the network under a node capture or a denial of service attack. The network consists of 50 nodes placed in a 10 x 5 grid 5 feet apart from each other. We assume that under such attacks a node gets completely disconnected from the network. Since the network is organized in a tree hierarchy, if an aggregator under attack is disconnected, all nodes in the subtree under the aggregator get disconnected. This can be clearly seen in the figure. For 5, 10, 15 and 20 % of nodes under attack the corresponding percentage of nodes disconnected is 12.87, 26.05, 40.4, and 53.94. The reconfiguration mechanism of our algorithm alleviates this issue which directly affects availability. Aggregator nodes when connected to the tree periodically send a beacon message. When child nodes stop receiving these messages from their parent they initiate the reconfiguration procedure and find an alternate parent. Figure 4.8 shows how reconfiguration helps in increasing the connectivity of the network. Even after reconfiguration some nodes are not recovered. This happens for two reasons. First, in the reconfiguration procedure, only those nodes whose position is higher in the tree than the disconnected node can become its parent. This condition has to be satisfied to avoid cycles in the tree, where the whole cycle is disconnected but the nodes in the cycle think they are connected. If there are no higher level nodes in the communication range of a disconnected node, it can never make it to the tree in the reconfiguration phase. Second, if all the higher level nodes in the communication range of a disconnected node already have maximum number of allowed children connected to them, they will not accept new nodes.

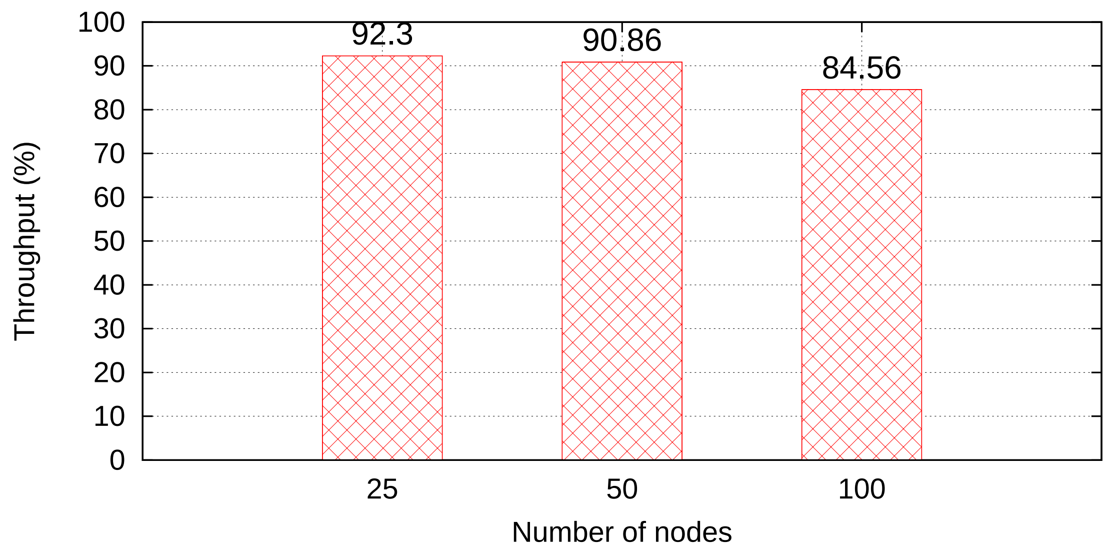

Figure 4.7. Throughput vs number of nodes

In an aggregation tree, a parent node needs to wait for all of its children to send data to it before starting aggregation. How long a parent waits for its children directly impacts the end to end delay and the throughput of the network. Intuitively, if a parent waits for a long duration of time the throughput will be high but the corresponding end to end delay would also be very high. On the other hand if the waiting time is too small the throughput will suffer since a lot of packets would be dropped but the end to end delay would also be very small. Figure 4.9 shows this tradeoff between throughput and end to end delay of the network. The figure shows the throughput of the network and the end to end delay when wait time at the aggregators is varied. The time axis shows the simulation time in seconds. When the wait time at the aggregator is 0.5 seconds, most of the packets are dropped since the timer expires even before the children can send their packets as a result of which the throughput is very low. The corresponding end to end delay is also very low in this case. At 1 second, the throughput jumps to 80.15 % this is because most of the packets can make it to the aggregator within this time. However there is only a slight, almost linear increase in end to end delay. Thus between the 0.5 and 1 second mark, a large gain in the throughput is achieved at the expense of a very small increase in the end to end delay. After the 1 second mark the gain in throughput slows down becoming almost constant after the 4 seconds mark. The end to end delay on the other hand keeps increasing in an almost linear fashion. Between the 1 and 4 seconds marks, there is a small gain in throughput at the expense of high end to end delay. Increasing the wait time beyond 4 seconds increases the end to end delay with no substantial gain in throughput.

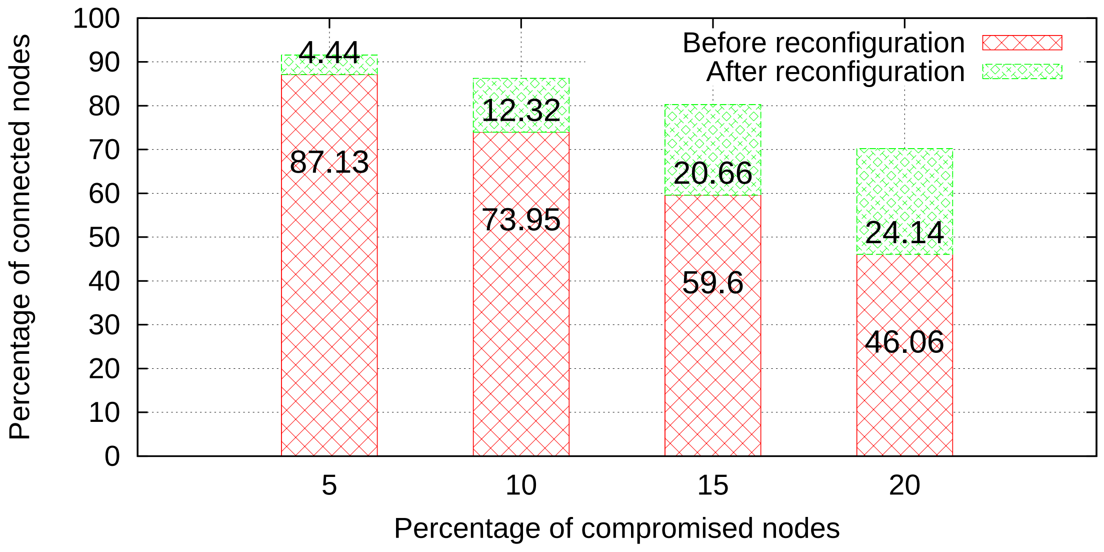

Figure 4.8. Connectivity in case of node capture

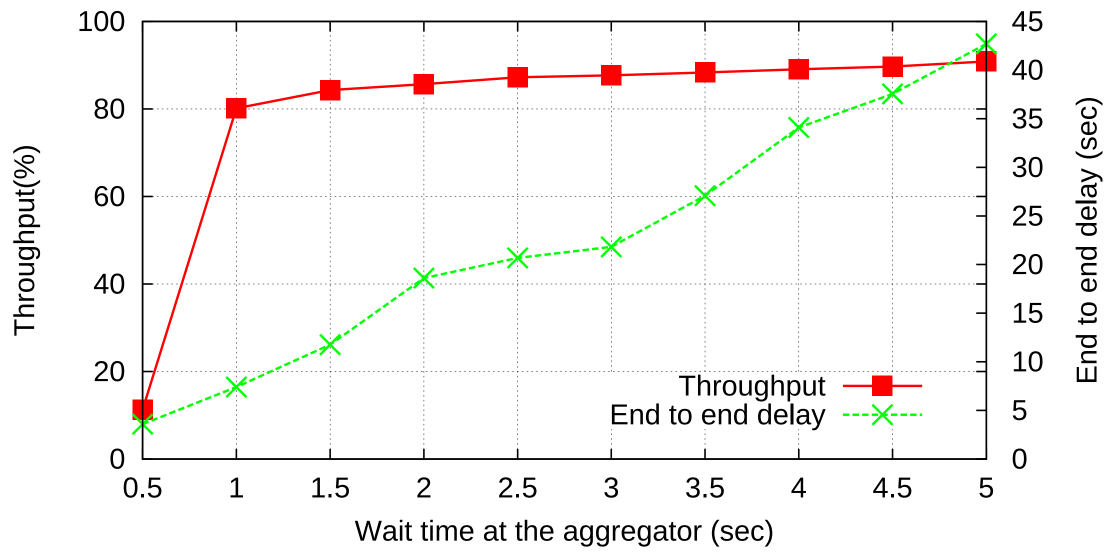

Figure 4.9. Throughput vs wait time at the aggregators vs end to end delay

5. CONCLUSIONS

This paper presents in-depth performance evaluation, analysis and simulation of secure hierarchical data aggregation algorithm. We demonstrated that our algorithm saves energy and increases network life time in two ways. First, it replaces energy intensive cryptographic operations by the light weight addition operations on the aggregators. Second, it does not require a separate integrity verification phase. Our implementation shows that this algorithm reduces the aggregator's work load which results in an increase in the aggregator's and overall network's lifetime. Additionally we performed the performance analysis of the algorithms by simulations and saw that it is capable of handling denial of service attacks and also node failure situations. From the throughput experiment we also established that the algorithm does well even under a noisy environment.

6. BIBLIOGRAPHY

[1] J. Albath and S. Madria, “Secure hierarchical data aggregation in wireless sensor networks,” in *WCNC*. IEEE, 2009, pp. 2420–2425.

[2] L. Hu and D. Evans, “Secure aggregation for wireless networks,” in *Workshop on Security and Assurance in Ad hoc Networks*. IEEE Computer Society, 2003.

[3] H. Chan, A. Perrig, and D. X. Song, “Secure hierarchical in-network aggregation in sensor networks,” in *Computer and Communications Security*, ser. CCS ’06, 2006, pp. 278–287.

[4] A. Mahimkar and T. S. Rappaport, “SecureDAV: A secure data aggregation and verification protocol for sensor networks,” in *Proceedings of the IEEE Global Telecommunications Conference*, 2004, pp. 2175–2179.

[5] Y. Yang, X. Wang, S. Zhu, and G. Cao, “SDAP: a secure hop-by-hop data aggregation protocol for sensor networks,” in *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, ser. MobiHoc ’06. ACM, 2006, pp. 356–367.

[6] J. Bahi, C. Guyeux, and A. Makhoul, “Efficient and robust secure aggregation of encrypted data in sensor networks,” in *Fourth International Conference on Sensor Technologies and Applications*, ser. SENSORCOMM ’10, July 2010, pp. 472–477.

[7] C. Castelluccia, E. Mykletun, and G. Tsudik, “Efficient aggregation of encrypted data in wireless sensor networks,” in *Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, ser. Mobiquitous ’05. IEEE Computer Society, 2005, pp. 109–117.

[8] H.-M. Sun, Y.-C. Hsiao, Y.-H. Lin, and C.-M. Chen, “An efficient and verifiable concealed data aggregation scheme in wireless sensor networks,” in *Proceedings of the 2008 International Conference on Embedded Software and Systems*. IEEE Computer Society, 2008, pp. 19–26.

[9] M. Groat, W. He, and S. Forrest, “KIPDA: k-indistinguishable privacypreserving data aggregation in wireless sensor networks,” in *INFOCOM’11*. 2011, pp. 2024–2032.

[10] V. Kumar, and S. Madria, “Secure data aggregation in wireless sensor networks,” in *Wireless Sensor Network Technologies for the Information Explosion Era*. Springer, 2010.

[11] S. Peter, K. Piotrowski, “On concealed data aggregation for wireless sensor networks,” in *Proceedings of the 4th IEEE Consumer Communications and Networking Conference*, ser. CCNC ’07. IEEE, January 2007.

[12] A. Liu and P. Ning, “Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks,” in *Proceedings of the 7th international conference on Information processing in sensor networks*, ser. IPSN ’08. IEEE Computer Society, 2008, pp. 245–256.

[13] C. Karlof, N. Sastry, and D. Wagner, “Tinysec: a link layer security architecture for wireless sensor networks,” in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys ’04. ACM, 2004, pp. 162–175.

II. PIP: PRIVACY AND INTEGRITY PRESERVING DATA AGGREGATION IN WIRELESS SENSOR NETWORKS

Vimal Kumar∗, Sanjay K Madria∗

∗ Department of Computer Science,

Missouri University of Science and Technology, Rolla, Missouri 65401

With the exponential rise of pervasive computing applications, data privacy has become much more of an important issue than before. When data is aggregated at each hop in a sensor network, it becomes harder to protect its privacy. A number of privacy preserving data aggregation algorithms have recently appeared for wireless sensor networks (WSNs), very few of them however also address the issue of data integrity along with privacy. Data privacy and integrity are two contrasting objectives to achieve in general. In a privacy preserved data aggregation, it becomes easier for an attacker to inject false data hence, we suggest that both privacy and integrity of data should be treated together. In this paper, we present an energy efficient, privacy preserving data aggregation algorithm which also preserves data integrity in WSNs. We analyze the security of the algorithm and provide proofs for confidentiality and integrity. We enhance this algorithm further to localize, to a certain degree, the corrupt aggregator. We provide the results of our implementation of the algorithm on TelosB motes, illustrating that both the computational overhead and the energy consumption are very low. Finally, we compare our algorithm with other schemes having similar objectives demonstrating that our algorithm performs better in terms of bandwith usage and energy consumption in a WSN environment.

1. INTRODUCTION

A wireless sensor network (WSN) consists of a large number of small, low cost, battery operated, wireless sensors. WSNs have a multitude of applications such as environmental sensing, health-care monitoring, traffic monitoring, law enforcement, and military, among others. In-network processing techniques such as sensor data aggregation are used to reduce communication overhead in WSNs. Data aggregation refers to using an aggregation function f such as SUM, AVG, MIN/MAX, MULT, etc. on n data items, at intermediate nodes, to produce one aggregate value $v = f(x_1, x_2, ..., x_n)$. Aggregating data on intermediate nodes not only reduces the energy consumption, but also minimizes end to end delay and bandwidth usage as well as increase the throughput of the network.

Increasing proliferation of wireless sensors in our daily lives has given rise to the need of privacy preserving and secure data aggregation. An excellent motivating example of the application of privacy preserving data aggregation was provided in [1] which we reproduce with a modification for the need of integrity. *A utility company may want to monitor the aggregate electricity and water consumption of a neighborhood by placing wireless sensors in the houses. This aggregate information may be helpful for the company and the community however individuals run the risk of their privacy being exposed as the data is communicated via multiple hops to the base station. This private information may reveal a lot about day to day activities of a household and could be used maliciously; hence individuals may only be willing to participate if the privacy of their data is guaranteed [1]. In addition to this, malicious insider nodes may also inject false data in the aggregate for personal benefits and hence it is also required that integrity of data be preserved.*

In this paper, we address the issues of *privacy* and *integrity* in data aggregation in wireless sensor networks. By *privacy* we essentially mean data privacy and thus it pertains to hiding the details of a node's data from other nodes in the network. *Integrity* refers to preventing the modification of data by corrupt aggregators. *Privacy* is typically obtained by techniques (e.g., encryption) that produce output resembling random bits. On the contrary, *integrity* is usually obtained by peer monitoring, which does not work with randomized data. Our work differs from the work of many others in that we provide a scheme that combines these two seemingly contrasting objectives of *privacy* and *integrity*.

2. PROBLEM STATEMENT

We consider a WSN with n nodes, organized in a tree hierarchy, and rooted at the base station *(B.S)*. Each sensor S_i generates a reading δ_i, the aggregate of k such readings is $\sum_{i=1}^{k}\delta_i$. Our definition of privacy is based on the well-known concept of semantic security, that the adversary cannot determine any information about the plaintext from a given ciphertext in polynomial time. We define the set of nodes which have access to the sensor reading δ_i as the privacy set Π_i. A sensor reading is private iff its privacy set consists of the node generating the reading itself and the base station. Similarly, an aggregate is private iff its privacy set consists of only the base station. Thus, we define the following conditions for privacy preserved data aggregation.

- For δ_i to be private, $\Pi_i = \{S_i, B.S\}$
- For $\sum_{i=1}^{k}\delta_i$ to be private, $\Pi_{1\ldots k} = \{B.S\}$

If k out of n nodes participate in data collection and each node S_i contributes a reading δ_i, the aggregate is $\sum_{i=1}^{k}\delta_i$. A data integrity violation occurs if the aggregate received by the base station $\sum_{i=1}^{k}\delta_i' \neq \sum_{i=1}^{k}\delta_i$.

Most works on privacy preserving data aggregation focus solely on privacy while completely ignoring the data integrity aspect. Privacy preserving data aggregation enables an aggregator to aggregate k data items without knowing what they are. This leaves the aggregation operation prone to false data injection attacks by corrupt aggregators. This can be overcome by using aggregate signatures such as in [2]. Signing and verification however, have a high energy cost and our requirement is of an algorithm which can provide integrity of the private data while also being energy efficient. Based on the above discussion, we define our objectives as below:

- To present a scheme for energy efficient privacy preserving data aggregation

- To provide energy efficient mechanism for the integrity of data aggregated privately.
- Simulation and implementation of the algorithm on real motes to verify its performance and energy usage.

3. RELATED WORK

A number of privacy preserving algorithms for data aggregation in WSNs have been proposed. Some of them, such as [2] and [3] make use of asymmetric homomorphic encryption to provide privacy of aggregate data. Asymmetric homomorphic cryptography provides the required privacy but also consumes a great deal of energy as shown in the comparison provided later in section 8. This approach may not always be viable on energy constrained wireless sensors and thus, there is a need for more efficient solutions.

Two privacy preserving algorithms, CPDA and SMART are proposed in [1]. In CPDA, nodes hide their data in a random polynomial and send it to all other nodes in the cluster using pair wise keys. These polynomials are added up and sent to the cluster head, along with a secret. The authors then discuss SMART which is less communication intensive than CPDA. In SMART each node in an aggregation tree slices its data into a fixed number of parts and sends each part to a neighboring node, keeping one for itself. Nodes add all of the received slices together and send them to the aggregator along the path of the tree. In CPDA, the aggregate is revealed to the cluster head while, in SMART, because positive slices of data are distributed, some amount of information about the data is always leaked. Moreover the privacy of data depends on key pre-distribution. If an adversary is able to capture a sufficient number of nodes, the privacy in both schemes suffers heavily. The scheme in [4] utilizes secret perturbation to address the issue of privacy. This scheme prevents the aggregator from knowing the aggregate, and no privacy is lost when a few nodes are compromised. In PASKOS and PASKIS algorithms proposed in [5] each node is assigned a subset of keys from a key ring. In PASKOS, nodes use all of their keys to create hashes, and randomly either add or subtract them from the data. This randomized data and a list indicating whether the hashes were added or subtracted is sent to the aggregator. The authors further improve the algorithm such that even the base station is oblivious to the aggregate in the PASKIS protocol. These protocols protect the aggregate, even from the intermediate nodes. As in [1] however, the privacy depends on key pre-distribution. Moreover, because a list, whose size is the size of the key ring, must be sent by every node, the bandwidth consumption of the protocols is also very high. In [6] each node establishes a twin key with at least one other node in a cluster. The twin key is used to either add or remove data from the aggregate. Each node, prior to sending data to the next node, creates a shadow value, encrypts it with the twin key, and adds it to the aggregate. Twin key has the property that if used again it will cancel out the effect of the first use. Thus, when a shadow is added to the aggregate, a node sharing that twin key will remove the shadow when the aggregate comes to it during aggregation. This scheme again suffers from the problem that a sufficient number of captured nodes can reveal the privacy of a number of other nodes. [7] proposed a novel solution for multi-dimensional data aggregation using a bilinear pairing based public key encryption. This scheme saves bandwidth when n-dimensional data has to be aggregated. The downside, however, of this scheme is that it is not very energy efficient.

Schemes in [1] [4] [5] [6] [7] address the issue of privacy of data. They do not however, provide the integrity of data. In many cases, the goal of the adversary is to have the base station accept false readings which these schemes fail to protect against. Schemes in [8] [9] [10] were proposed to handle an adversary that tries to inject false data in the system. iPDA [9] handles data integrity along with the privacy of data. It uses the SMART [1] algorithm and preserves the integrity of data by using two parallel but disjoint aggregation trees rooted at the base station. The algorithm assumes that it will be difficult for an attacker to inject data and affect both the aggregation trees equally. An adversary however, which has captured at least two nodes, one in each tree, can easily subvert this scheme. iCPDA[10] uses peer monitoring to protect integrity of data. In this scheme, the privacy of individual sensor data is protected, while the privacy of partial aggregates is not. Moreover, an adversary can simply eavesdrop on the cleartext communication of nodes and obtain the aggregate. The aggregators are assumed to be trustworthy and aggregates are revealed to them. If a node determines an aggregator is misbehaving, it sends a report to the base station. The base station, however, faces the dilemma of whether or not to trust the reporting node. The scheme proposed in [8] also provides both integrity and privacy. In this scheme, secrets are pre-deployed on intermediate nodes to assess the integrity of the data. The scheme is based on random perturbation and therefore is very energy efficient. Though, it can only provide an approximation of the data as it is based on histogram of the sensor data. Unlike [8] our scheme provides accurate data aggregation. Our goals are similar to the goals in [9] and [10], i.e. the privacy and integrity of data. Additionally, our algorithm does not reveal partial aggregates to intermediate nodes. Due to the similarity of goals we will compare our privacy and integrity preserving algorithm PIP to iPDA[9] and iCPDA[10].

4. SYSTEM AND ADVERSARY MODELS

Our network consists of n wireless sensors deployed in the field and a base station. Wireless sensors have limited energy and communicate in a small radius and therefore, resort to hop-by-hop communication. The base station has unlimited energy and is secure. Each sensor S_i generates a reading δ_i the aggregate, of which needs to be sent to the base station. Aggregator nodes perform data aggregation, and the base station receives $\sum \delta_i$. We assume that the adversary only tries to corrupt the aggregators and not the leaf nodes, as a corrupt aggregator will have a larger impact on the aggregated value. We limit ourselves to the SUM aggregation function in this paper.

Our goal is to provide privacy and integrity preserving aggregation of data. Our algorithm defends against an adversary who will attempt to.

- Compromise the data privacy of the aggregators in the network.

  In this case, for a sensor reading δ_i of a node S_i, the privacy set $\Pi_i = \{S_i, B.S, S_i, ..., S_{i+p}\}$ or for an aggregate $\sum_{i=1}^{k} \delta_i$, $\Pi_{1...k} = \{B.S, S_i, ..., S_{i+p}\}$, where the data is accessible to nodes $S_i, ..., S_{i+p}$.

- Inject false data into the network.

  In this case the aggregate data received by the base station, $\sum_{i=1}^{k} \delta'_i$ and the actual aggregate data $\sum_{i=1}^{k} \delta_i$ do not match i.e $\sum_{i=1}^{k} \delta'_i \neq \sum_{i=1}^{k} \delta_i$.

5. PRELIMINARIES

5.1. SECRET SHARING

In secret sharing, a data item δ is divided into n parts such that any k parts can be used to reconstruct the data and the knowledge of $k-1$ parts does not provide any information.

To create shares of a data item δ using Shamir's [11] scheme, a prime number p is chosen such that $p > max(\delta, n)$. A random $k-1$ degree polynomial is then created, where the coefficients $a_1, \ldots, a_{k-1}$ are chosen randomly from a uniform distribution over the integers in $[0, p]$ and $a_0 = \delta$. This polynomial $\omega(x) = a_0 x_0 + a_1 x_1 + \ldots + a_{k-1} x_{k-1}$ is then sampled at points $x = 1, \ldots, n$ to create shares $\omega(1), \ldots, \omega(n)$.

Any k out of the n shares created above can be used to create a $k-1$ degree polynomial $\omega'(x)$ using polynomial interpolation. To reconstruct the original data $\omega'(x)$ is evaluated at $x = 0$.

5.2. RECURSIVE SECRET SHARING

Parakh and Kak [12] proposed a recursive secret sharing scheme based on Shamir's secret sharing [11]. Recursive secret sharing is proposed for efficient storage, where the shares of the data item δ are used to store $k-2$ additional pieces of information. A node with all the shares can easily reconstruct δ and the $k-2$ pieces of hidden information.

We denote recursive secret sharing by RSS and Shamir's secret sharing by SSS, respectively. In RSS Parakh and Kak, modify SSS such that the shares are generated using polynomial interpolation rather than by generating random polynomial and sampling it. For example, suppose secrets $\delta_1, \ldots, \delta_{k-1}$ are to be stored in the shares. A random number y_{11} is chosen and using $(0, \delta_1)$ and $(1, y_{11})$, a 1-degree polynomial $\omega^1(x)$ is interpolated. This polynomial $\omega^1(x)$ is then sampled at $x = 2$ and $x = 3$ to obtain $\omega^1(2)$, $\omega^1(3)$. Next,

$(0,\delta_2)$ and the points $(1,\omega^1(2))$ and$(2,\omega^1(3))$ are used to interpolate a 2-degree polynomial $\omega^2(x)$. The polynomial $\omega^2(x)$ is sampled at $x = 3,4$ and 5 to obtain $\omega^2(3),\omega^2(4)$ and $\omega^2(5)$. These three points are used as y coordinates for $x = 1,2,3$ along with the point $(0,\delta_3)$ to interpolate the 3-degree polynomial $\omega^3(x)$. This process is repeated until the last secret δ_{k-1} is used to create a $k-1$ degree polynomial $\omega^{k-1}(x)$. This polynomial can then be sampled at n points to creates n shares, any k of which can be used to regenerate δ_{k-1}. In the reconstruction phase, δ_{k-1} is regenerated using the original method discussed in the previous subsection by constructing $\omega^{k-1}(x)$ using the k shares. This polynomial $\omega^{k-1}(x)$ is then sampled at $x = 1,\ldots,k-1$ to obtain $\omega^{k-1}(1),\ldots,\omega^{k-1}(k-1)$. The points $(k-1,\omega^{k-1}(1))$,...,$(2(k-1),\omega^{k-1}(k-1))$ are then used to interpolate a $k-2$ degree polynomial $\omega^{k-2}(x)$. δ_{k-2} is recovered by evaluating $\omega^{k-2}(x)$ at $x = 0$. This process is repeated for all δ_i until secret δ_1 is recovered.

6. PROPOSED ALGORITHM

In this section, we present PIP: Privacy and Integrity Preserving Data Aggregation Algorithm, which fulfills the objectives stated in section 2. We begin by addressing the energy efficient privacy and integrity preservation objectives. We base our algorithm on the recursive secret sharing algorithm described in section 5.2. The authors in [12] proposed the scheme for efficient storage, where the shares of the data δ are used to store $k-2$ additional pieces of information. A node with at least k shares can easily reconstruct all of the $k-1$ pieces of hidden information. We provide a construction in which we can prevent a node which has all the shares from reconstructing and retrieving the hidden data. Our construction preserves the homomorphic property of SSS and RSS which is used to aggregate data in a privacy preserving manner.

The basic methodology is to create shares for a given sensor reading using RSS. Each node then sends all of the shares to its parent. Because RSS is based on SSS, it is additively homomorphic and so the parent node will be able to aggregate the shares to aggregate the data. However, because each node is sending all the shares to its parent, it is straightforward for a curious aggregator to regenerate the data and thus subvert its privacy. To prevent this, each node scrambles the shares by subtracting modulo prime, a scrambling key shared between each node and the base station from the shares. This scrambling procedure is very light weight and preserves the homomorphic property of RSS. We could also use homomorphic encryption here. Scrambling however, is preferred since it consumes negligible energy when compared to encryption while still being secure, as will be seen in the security proof later. Finally, because we also want to preserve the integrity of the data, the intuition is to have some additional information, along with the shares which will help the base station to detect any changes in the data. We use an integrity key, which each node shares with the base station and embed this key in the generated shares using

RSS. It should be noted that Verifiable Secret Sharing (VSS) could also be used to ensure data integrity. VSS schemes however, typically require energy intensive cryptographic constructs and are not efficient communication wise. Our scheme as we show in section 8 is both computationally light weight and efficient in terms of communication overhead.

6.1. BASIC PIP ALGORITHM

Basic PIP consists of two algorithms, one for generating the shares from the data and the keys and the other for regenerating data and checking its integrity. We assume that the sensor nodes are randomly deployed in a field and they self organize into a tree hierarchy using the tree construction algorithm given in [2]. The base station preloads each sensor with a pseudo random function (PRF) $f(.)$, a network wide nonce n, three distinct random seeds r_1, r_2, r_3, a network-wide random seed r_4 and the network-wide prime number p. Before every iteration, a sensor node uses the PRF $f(.)$ on the nonce n, using the random seeds r_1, r_2, r_3, r_4 as keys, to generate the perturbation key η, a scrambling key Ψ and an integrity key $I = \{I', I''\}$ as shown in Algorithm 4. The newly generated keys will be used as random seeds for the next iteration. All the operations are performed mod p. In the basic PIP algorithm, a sensor S_k uses the perturbation key η_k, and the sensor reading δ_k to generate perturbed data $\delta_{\eta k}$. Recursive secret sharing is then used and a linear combination of the integrity key $I_k^{'} + I_k^{''}\delta_{\eta k}$ and data, $\delta_{\eta k}$ are encoded to create the shares $\omega_k^2(3), \omega_k^2(4)$, and $\omega_k^2(5)$. If these shares are sent to the aggregator, then a malicious aggregator would be able to easily replace the data in the shares with corrupt data, keeping the integrity key intact. We use the scrambling key Ψ_k to scramble the shares and prevent this from happening. The scrambled shares λ_k^1, λ_k^2, and λ_k^3 are generated as shown in the algorithm. Each leaf node sends the scrambled shares λ_k^1, λ_k^2, and λ_k^3 to its parent, which aggregates the shares received from all of its children. The aggregate shares $\lambda_{agg}^1, \lambda_{agg}^2$, and λ_{agg}^3 are calculated as $\lambda_{agg}^1 = \sum\lambda^1, \lambda_{agg}^2 = \sum\lambda^2, \lambda_{agg}^3 = \sum\lambda^3$, where the summation is over all children. The base station then uses the sum of the scrambling keys $\sum\Psi$ to unscramble the aggregate shares to obtain $\omega^2_{agg}(3), \omega^2_{agg}(4)$ and $\omega^2_{agg}(5)$. These shares are then used to decode received perturbed data $\omega^1_{agg}(0)$ and the linear combination of integrity keys and the perturbed data $\omega^2_{agg}(0)$. If the equation $\omega^2_{agg}(0) - \omega^1_{agg}(0)I^{''}_k = \sum_{k=1}^{n} I^{'}_k$ holds, data has not been tampered with and is recovered after removing the perturbation. Otherwise, the data has been tampered with and the base station would need to take appropriate steps. Algorithmic details of the share verification and data regeneration process can be seen in Algorithm 5.

Algorithm 4 Share Generation Algorithm

```
Require: Sensor reading $\delta_k$, PRF $f(.)$, nonce $n$, random seeds $r_1$, $r_2$, $r_3$, $r_4$.
 1: Generate Perturbation key as $\eta_k = f_{r_1}(n)$, Scrambling key $\Psi_k = f_{r_2}(n)$, Integrity key
    $I_k = \{I^{'}_k, I^{''}_k\} = \{f_{r_3}(n), f_{r_4}(n)\}$ .
 2: Update $r_1 = \eta_k$, $r_2 = \Psi_k$, $r_3 = I^{'}_k$, $r_4 = I^{''}_k$.
 3: Generate perturbed data $\delta_{\eta k} = \delta_k + \eta_k$.
 4: Choose a random number $y$ uniformly from $\mathbb{Z}_p$.
 5: Interpolate $(0, \delta_{\eta k})$ and $(1, y)$ to obtain a degree 1 polynomial $\omega^1_k$.
 6: Sample the polynomial $\omega^1_k$ at $x = 2,3$ and interpolate $(0, I^{'}_k +$
    $I^{''}_k \delta_{\eta k}), (1, \omega^1_k(2)), (2, \omega^1_k(3))$ to obtain a degree 2 polynomial $\omega^2_k$.
 7: Sample $\omega^1_k$ at $x = 3,4,5$ to obtain $(3, \omega^2_k(3)), (4, \omega^2_k(4)), (5, \omega^2_k(5))$.
 8: $\omega^2_k(3), \omega^2_k(4), \omega^2_k(5)$ are the shares which consist of the sensor reading $\delta_k$ and the in-
    tegrity key $I_k$ of the sensor $k$.
 9: These shares are then scrambled using the scrambling key $\Psi_k$ as follows.
        $\lambda^1_k = \Psi_k - \omega^2_k(3)$
        $\lambda^2_k = \omega^2_k(3) - \omega^2_k(4)$
        $\lambda^3_k = \omega^2_k(4) - \omega^2_k(5)$
10: Final shares are the scrambled shares $\lambda^1_k, \lambda^2_k, \lambda^3_k$.
```

Algorithm 5 Data Regeneration and Integrity Check

```
Require: PRF $f(.)$, nonce $n$, random seeds $r_1$, $r_2$, $r_3$, $r_4$.
1: B.S generates the scrambling, perturbation and the integrity keys for all the sensors and
   computes $\sum_{k=1}^{n}\Psi_k$, $\sum_{k=1}^{n}\eta_k$ and $\sum_{k=1}^{n}I_k^{'}$ .
2: Root node unscrambles the shares using $\sum_{k=1}^{n}\Psi_k$ as follows.
       $\omega_{agg}^2(3) = \sum_{k=1}^{n}\Psi_k - \lambda_{agg}^1$
       $\omega_{agg}^2(4) = \omega_{agg}^2(3) - \lambda_{agg}^2$
       $\omega_{agg}^2(5) = \omega_{agg}^2(4) - \lambda_{agg}^3$
3: Interpolate $(3,\omega_{agg}^2(3)),(4,\omega_{agg}^2(4))$ and $(5,\omega_{agg}^2(5))$ to obtain a degree 2 polynomial
   $\omega_{agg}^2$.
4: Sample $\omega_{agg}^2$ at $x = 1,2$.
5: Interpolate $(2,\omega_{agg}^2(1)),(3,\omega_{agg}^2(2))$ to obtain a degree 1 polynomial $\omega_{agg}^1$.
6: Calculate $\omega_{agg}^1(0)$ and $\omega_{agg}^2(0)$
7: if $\omega_{agg}^2(0) - \omega_{agg}^1(0)I_k^{''} == \sum_{k=1}^{n}I_k^{'}$ then
     Accept the aggregate as $\omega_{agg}^1(0) - \sum_{k=1}^{n}\eta_k$.
8: else
     Aggregate is corrupt.
9: end if
```

6.2. NUMERICAL EXAMPLE

We provide a working example of the basic PIP algorithm below. Let us consider a small aggregation tree consisting of 3 nodes, as shown in Figure 6.1. Node A aggregates the data from nodes B, C and itself and sends the aggregate to the base station. Each node generates the scrambling key Ψ, the integrity key $I = \{I', I''\}$ and the perturbation key η. Using these keys, the nodes sense data, perturb it and generate scrambled shares. To illustrate the share generation process, let us take node B as an example, which has the scrambling key $\Psi_B = 40$, the integrity key $I_B = 49,4$ and the perturbation key $\eta_B = 2$. All calculations are done mod 131, which is a network parameter. Node B has data $\delta_B = 22$, to which it adds the perturbation key $\eta_B = 2$ and generates perturbed data $\delta_{\eta B} = 24$. It chooses a random number $y = 7$ and interpolates $(0,24)$ and $(1,7)$ to generate the polynomial $\omega_B^1(x) = 114x + 24$. This polynomial is sampled at $x = 2$ and 3 to obtain the points $(2,121),(3,104)$. Next we compute $I_B^{'} + I_B^{''}\delta_{\eta B}$ = 14 and interpolate $(0,14),(1,121)$ and $(2,104)$ to generate the polynomial $\omega_B^2(x) = 69x^2 + 38x + 14$. We sample this polynomial at $x = 3,4,5$ to generate the recursive shares $94,91$, and 95. We then use the scrambling key $\Psi_B = 40$ to obtain the final scrambled shares $\lambda_B^1 = 77, \lambda_B^2 = 3$ and $\lambda_B^3 = 127$. Node B sends it's scrambled shares to node A, as does node C. Node A, being the aggregator, aggregates its own shares with the shares it receives to obtain the aggregate shares $\sum\lambda_{agg}^1 = 28, \sum\lambda_{agg}^2 = 78, \sum\lambda_{agg}^3 = 9$. The aggregate shares are then sent to the base station. Because the keying material of all the nodes is shared with the base station, the base station can calculate $\sum\Psi$, $\sum I'$, I'' and $\sum\eta$ which, in this case, are 88, 39, 4 and 6 respectively. The base station then unscrambles the aggregate shares using $\sum\Psi$. The original shares after unscrambling come out to be 60, 113, and 104. These shares are interpolated to obtain the polynomial $\omega_{agg}^2(x) = 100x^2 + 8x + 53$. The polynomial $\omega_{agg}^2(x) = 100x^2 + 8x + 53$ is then sampled at $x = 1,2$ to obtain points $(1,30)$ and $(2,76)$ respectively. Points $(2,30)$ and $(3,76)$ are then interpolated, and the polynomial $\omega_{agg}^1(x) = 46x + 69$ is obtained. The base station then checks whether the equation $\omega_{agg}^2(0) - \omega_{agg}^1(0)I_k^{''} = \sum_{k=1}^n I_k^{'}$ holds. Since we can see that the equation is satisfied in the example, $\omega_{agg}^1(0) = 69$ is taken as the sum of the perturbed data $\sum\delta_\eta$ from which $\sum\delta$ can be derived as $\sum\delta = \omega_{agg}^1(0) - \sum\eta$.

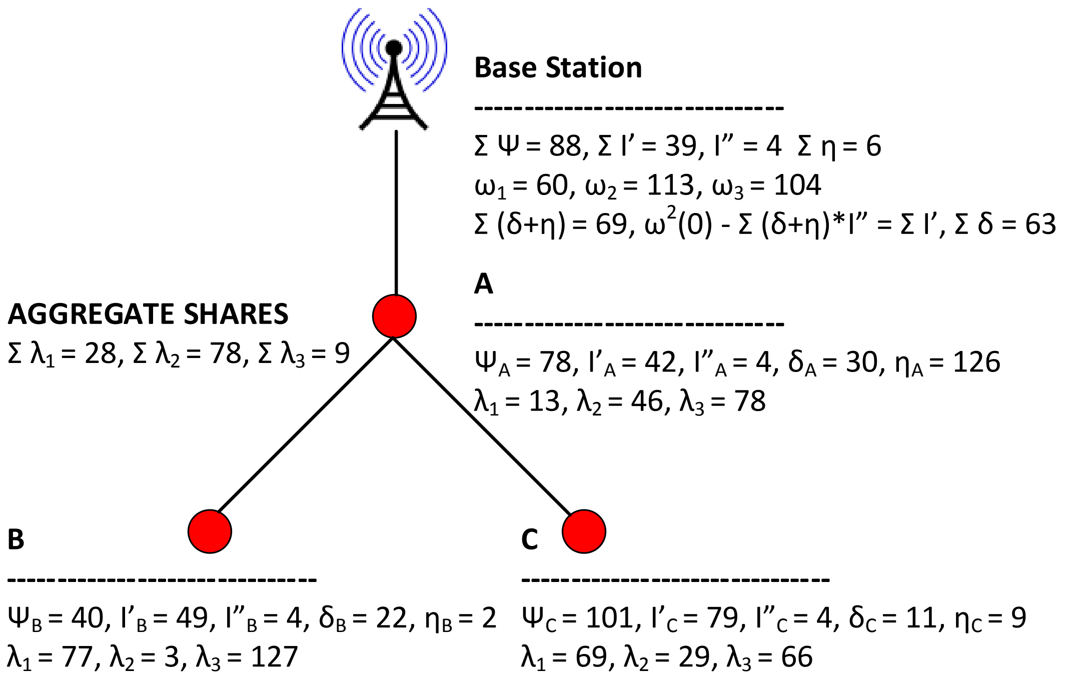
Base Station
Σ Ψ = 88, Σ I' = 39, I" = 4 Σ η = 6
ω1 = 60, ω2 = 113, ω3 = 104
Σ (δ+η) = 69, ω2(0) - Σ (δ+η)*I" = Σ I', Σ δ = 63
A
ΨA = 78, I'A = 42, I"A = 4, δA = 30, ηA = 126
λ1 = 13, λ2 = 46, λ3 = 78
AGGREGATE SHARES
Σ λ1 = 28, Σ λ2 = 78, Σ λ3 = 9
B
ΨB = 40, I'B = 49, I"B = 4, δB = 22, ηB = 2
λ1 = 77, λ2 = 3, λ3 = 127
C
ΨC = 101, I'C = 79, I"C = 4, δC = 11, ηC = 9
λ1 = 69, λ2 = 29, λ3 = 66

Figure 6.1. Data aggregation using Basic PIP

6.3. SECURITY ANALYSIS

6.3.1. Confidentiality. The basic PIP algorithm can be reduced to the CMT scheme described in [13], which is proven to be semantically secure. The CMT scheme is based on pseudo random functions (PRFs). Given a nonce r, a data point $\delta \in [0, M-1]$, where M is an integer, a PRF $f(.)$, an encryption key k and a length matching function $h(.)$, the CMT scheme encrypts the data point δ as below.

$$Enc_k(\delta) = (\delta + h(f_k(r)))mod\ M \qquad (1)$$

The basic PIP scheme can be stated formally as below. Given an input of a nonce r, a data point $\delta \in [0, p-1]$, where p is a large prime , the scrambling key Ψ and the perturbation key

η, the scheme first generates perturbed data.

$$\delta_\eta = (\delta + \eta) mod\ p \tag{2}$$

The scheme then uses the RSS(.) function to create the shares. This can be represented as

$$RSS(\delta_\eta, f_k(r)) = \lambda_1, \lambda_2, \lambda_3 \tag{3}$$

Using $f_k(r)$ gives the shares a non-deterministic character. Using a PRF with a nonce and a sensor specific key is the practical way of generating the random number y in step 2 of Algorithm 4. We see from Algorithm 5 that if the first share is correctly recovered, we can recover the other shares easily. Thus the security of the scheme hinges on the security of the first share. We define a function *g(.)*, such that

$$g(\lambda_1) = -\lambda_1 \tag{4}$$

then, using equations (2), (3) and (4), the basic PIP scheme can be written as

$$PIP_k(\delta) = (\Psi + g(\lambda_1)) mod\ p \tag{5}$$

Equation (5) is in the same form as (equation 1), where the randomness generated by $f_k(r)$ in equation (1) is generated by $g(\lambda_1)$ in equation (5). By this reduction, we can say that the basic PIP scheme is at least as secure as the CMT scheme in [13].

6.3.2. Integrity. We say that a scheme protects the integrity of a message if the probability of an adversary performing existential forgery is negligible. The integrity of the basic PIP scheme is based upon its confidentiality, since the integrity key is wrapped in the shares along with the perturbed data. An adversary that is able to break the confidentiality of the scheme will also be able to break the integrity of the scheme. Thus, we say that, for every adversary A attacking the basic PIP scheme for integrity, $\exists$ an adversary B, attacking its confidentiality. For an adversary that possesses messages δ_0 and δ_1 and is given Λ_b, where $b \in 0,1$(From here on, for the sake of convenience we denote the scrambled shares of node k by Λ_k), we define a PPT distinguisher $D(\Lambda_0, \Lambda_1, \delta_0, \delta_1)$ which outputs 1 if it is able to distinguish between the two Λs. The advantage of the adversary can then be written as

$$Adv_{PIP_C}[B] = Pr[D(\Lambda_0, \Lambda_1, \delta_0, \delta_1) = 1] \tag{6}$$

Using (6) the advantage of an adversary attacking integrity of the scheme can be written as

$$Adv_{PIP_I}[A] \leq Adv_{PIP_C}[B] + (1/2^{|\Psi|} * 1/2^{|\eta|}) \tag{7}$$

In (7), the first term corresponds to the adversary attacking the confidentiality of the scheme. The second term corresponds to an attacker randomly guessing the keys Ψ and η, where $|\Psi|$ and $|\eta|$ are the respective key sizes. We showed earlier that the scheme is semantically secure. The term $Adv_{PIP_C}[B]$ is therefore negligible.

$$Adv_{PIP_I}[A] \leq 1/2^{|\Psi|} * 1/2^{|\eta|} \tag{8}$$

$$Adv_{PIP_I}[A] \leq 1/2^{2b} \tag{9}$$

where b is the size of the shares and $b = |\Psi| = |\eta|$. We can determine from (9) that the advantage of the adversary attacking the integrity is negligible and the scheme protects the integrity of the data as long as b (the share size) is sufficiently large.

6.4. DISCUSSION ON PRIVACY AND INTEGRITY IN BASIC PIP

In semantic security, an adversary in possession of Λ_1 and Λ_2 will be unable to differentiate between the two. Thus, with shares of size b, there are 2^b equally likely possibilities and no information is leaked to the aggregator regarding either the original shares or data. Thus, in basic PIP, for sensor S_i, which has data δ_i, the privacy set consists only of S_i, however in the following section on enhanced PIP, we will see that a node may have to reveal its shares to the base station for verification, thus $\Pi_i = \{S_i, B.S\}$. The same reasoning can be applied to aggregate data of k nodes. An aggregator receives private data from $k-1$ nodes, and aggregates the scrambled shares. Again, there are 2^b equally likely possible original aggregate shares. Each node however, only knows its own Ψ and so the knowledge of aggregate scrambled shares would not be enough to gain any information regarding the aggregate data. In this case $\Pi_{i \in Agg} = \{B.S\}$, where Agg is the set of nodes which took part in data aggregation.

In recursive secret sharing, if k shares are known, original aggregate $\sum \delta_\eta$ can be replaced by false data $\sum \hat{\delta}_\eta$ in the shares without altering the integrity key. However, we established above that a node not in possession of the scrambling key Ψ cannot obtain the original shares. This means that, to inject false data into the system, a node will have to change the scrambled shares. Changing the scrambled shares, amounts to changing all the embedded data in the shares. In this case the embedded data is the aggregate data $\sum \delta_\eta$ and $\sum I' + I'' \sum \delta_\eta$. Thus we conclude that in recursive shares $\Delta \sum \delta \cong \Delta I$. This means that if the B.S discovers that $\sum I' \neq \sum \hat{I}'$, where $\sum \hat{I}'$ is the sum of the integrity keys $\hat{I}'$ derived from the received shares, it is equivalent to $\sum \delta_\eta \neq \sum \hat{\delta_\eta}$.

7. EXTENSIONS TO BASIC PIP ALGORITHM

7.1. ENHANCED PIP FOR CORRUPT AGGREGATOR DETECTION

The basic PIP algorithm allows us to determine whether or not the aggregate data is corrupt. It, however, does not enable us to detect which malicious aggregators injected false data. To identify such malicious aggregators we perform a two fold check in the enhanced PIP algorithm. To begin with, we make an additional assumption that the base station can reach all the nodes in the network using a secure channel. As before, we assume that each sensor node is preloaded with a pseudo random function (PRF) $f(.)$, a network wide nonce n, three distinct random seeds r_1, r_2, r_3 and a network-wide random seed r_4 which are used to generate the scrambling, integrity and the perturbation keys along with the network-wide prime number p. This time however, the nodes are provided with another random seed r_5 to generate an additional scrambling key such that the nodes have two scrambling keys Ψ^1 and Ψ^2. The network hierarchy is known to the base station and hence it can calculate $\sum\Psi^2$ for any aggregator in the network. Additionally, each parent child pair (S_p, S_k) shares a commitment key γ_{pk}. Leaf nodes begin the data collection by generating the scrambled shares through the process shown in Algorithm 4, using the scrambling key Ψ^1. The leaf nodes then recursively encode the commitment key γ_{pk}, (shared with their parent node) into the scrambled shares. These shares are then scrambled once again using the scrambling key Ψ^2 before being sent to the parent. The parent node receives Λ from all of its children and computes $\Lambda_{agg} = \sum\Lambda$. It then computes $\sum\gamma_{pc}$, the sum of the commitment keys it shares with the children that sent their Λs. It also calculates its own scrambled shares Λ_p using both the scrambling keys Ψ^1 and Ψ^2 and $\gamma_{gp} - \sum\gamma_{pc}$ as the commitment key. γ_{gp} is the commitment key shared between the parent and the grandparent. Λ_{agg} and Λ_p are then sent to the grandparent node. The grandparent adds Λ_{agg} and Λ_p. This addition results in the subtraction of $\sum \gamma_{pc}$ from the shares leaving only γ_{gp}. The process is shown in Algorithm 6.

Algorithm 6 Data Aggregation using enhanced PIP

```
Require: Sensor reading $\delta_k$, PRF $f(.)$, nonce $n$, random seeds $r_1$, $r_2$, $r_3$, $r_4$, $r_5$, commitment keys $\gamma_{kc}$ and $\gamma_{pk}$ shared with the child and parent nodes respectively.
1: Generate Perturbation key $\eta_k = f_{r_1}(n)$, Scrambling keys $\Psi_k^1 = f_{r_2}(n)$, $\Psi_k^2 = f_{r_2}(n)$, Integrity key $I_k = \{I_k^{'}, I_k^{''}\} = \{f_{r_3}(n), f_{r_4}(n)\}$ .
2: Update $r_1 = \eta_k$, $r_2 = \Psi_k^1$, $r_3 = I_k^{'}$, $r_4 = I_k^{''}$, $r_5 = \Psi_k^2$.
3: Generate scrambled shares $\lambda_k^1, \lambda_k^2, \lambda_k^3$ using Algorithm 4 with $\Psi_k^1$ as the scrambling key.
4: Leaf nodes interpolate $(0,\gamma_{pk}),(1,\lambda_k^1),(2,\lambda_k^2),(3,\lambda_k^3)$ to generate the polynomial $\omega_k^3(x)$ and then sample it to obtain shares $\omega_k^3(4),\omega_k^3(5),\omega_k^3(6),\omega_k^3(7)$, which are then scrambled using $\Psi_k^2$ to obtain final scrambled shares $\Lambda_k$.
5: Parent nodes aggregate the shares sent by their children and compute $\sum \Lambda_{agg}$.
6: Parent nodes compute $\sum_{i=0}^{j} \gamma_{kc_i}$, s.t. $j$ is the number of children and $\gamma_{kc_i}$ is the key shared with child $i$ .
7: Parent nodes use $\gamma_{pk} - \sum_{i=0}^{j} \gamma_{kc_i}$ as the commitment key and generate their scrambled shares $\Lambda_k$. $\sum \Lambda_{agg}$ and $\Lambda_k$ are then sent to the grand parent node.
```

7.1.1. Corrupt Aggregator Detection.

We assume that the attacker can capture a certain percentage of aggregators in the network and all the captured nodes inject false data in the network. With this assumption, various scenarios are possible, under a false data injection attack (See Figure 7.1).

The base station detects corrupt data by following the approach in Algorithm 7. When corrupt data is received from a node, the base station queries this node for its shares Λ, the shares Λ_c received from its child nodes, and the commitment keys it shares with the children. As discussed in section 6.4, an intermediate node can change the aggregate and the data encoded in the shares in two ways. If the scrambling key is known, the intermediate node can make changes to the data without making any change to the integrity key. If the scrambling key is not known, both the aggregate and other encoded data, can be changed

Algorithm 7 Corrupt aggregator detection

```
Require: Corrupt shares $\sum_{k=1}^{j} \Lambda_k$, $\Psi^2$ for all sensor nodes.
 1: Set $R = \{\phi\}$
 2: Base station queries the node $S_{ca}$ from which corrupt aggregate was received for its
    commitment keys shared with the children and $\Lambda$, $\Lambda_c$ received from the children, where
    $c \in C$ and $C$ is the set of the child nodes.
 3: repeat
 4:   Base station chooses a previously unselected child node $S_i$, computes appropriate
      $\sum\Psi^2$, and decodes the commitment key $\gamma'_{kc_i}$
 5:   if $\gamma'_{kc_i} \neq \gamma_{kc_i}$ then
        Perform step 1 for child $i$.
 6:     if $\Lambda \neq \Lambda_i$ then
          $R = \{S_{ca}, S_i\}$.
 7:     else
          Commitment keys are checked for all children of the child node.
 8:       if All keys match then
            $R = \{S_{ca}\}$.
 9:       else
            Go to step 5.
10:       end if
11:     end if
12:   end if
13: until All child nodes in $C$ are selected.
14: Output $R$ as the set of corrupt nodes.
```

by making random changes in the shares. Since a node does not share Ψ^2 with any other node, corrupt nodes do not know $\sum\Psi^2$. Thus, they will have to make random changes in the shares. Any random change, however, would cause unpredictable changes in all the enoded data in the shares. When an aggregator receives a query from the base station, it responds with, its own final shares and the shares received from its children, as well as the commitment keys it shares with each of its children. The base station then uses appropriate $\sum\Psi^2$ to extract the commitment key from the shares, verifying it against the one sent by the queried node. If the commitment check fails, the children of this node are queried further for their shares and their children's shares. If the shares sent by the queried node and the ones sent by a child node fail to match, both the nodes are removed from the system, since either the child or the parent is trying to frame the other. If the shares do match and the shares sent by the child node pass the commitment check, the queried node is removed from the network. If the shares sent by the child node fail further, the network is probed further in a similar fashion until a corrupt node is detected.

Security Analysis: An analysis similar to the one performed for basic PIP can be performed for enhanced PIP with same results.

Note: While it may seem that the enhanced PIP algorithm is expensive in terms of communication, we argue that because the detection of corrupt nodes in a network is vital such message complexity can be tolerated. Moreover, the detection phase takes place infrequently and is only needed once corrupt data is detected.

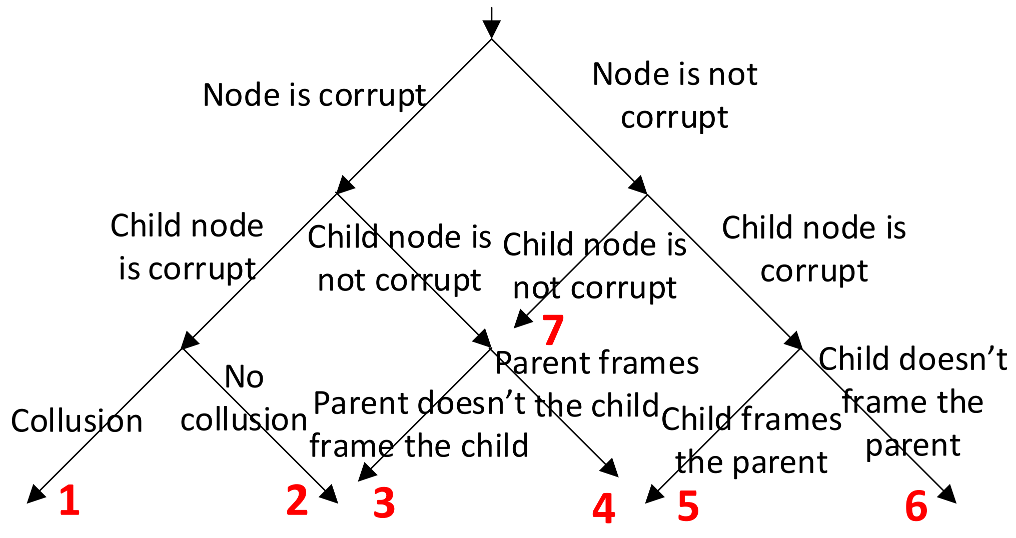

Figure 7.1. Attack scenarios

7.1.2. Attack Scenarios. In a network where multiple nodes are captured by an adversary, captured nodes may try to affect the network in six different ways, as illustrated in Figure 7.1. Enhanced PIP algorithm can handle all the six scenarios. In some cases, however, it is impossible to differentiate between a corrupt node and a good one. In such cases, the good node along with the corrupt ones must be removed from the network. In this section we discuss all the six scenarios and how enhanced PIP deals with these cases.

- Case 1. Corrupt nodes only injects false shares

  This case consists of scenarios 2, 3, and 6 in Figure 7.1. A standalone corrupt node which only injects false shares will be detected in step 6 of Algorithm 7.

- Case 2. Corrupt child frames a good parent node

  This case is seen in scenario 5. A child node may inject false shares into the system, but when queried for the shares, it submits the original shares, thus framing the parent node. Our algorithm detects this change. It, however, is not possible to determine which node indeed is lying and hence the child node and its parent both will be removed from the network.

- Case 3. Parent node frames the child node

  This case is seen in scenario 4. The parent node changes the shares received from a child node before aggregation, and presents these altered shares when queried. The child node, when queried presents the original shares. Since, for the base station, this case is indistinguishable from Case 2, both the nodes are removed from the system.

- Case 4. Corrupt nodes collude

  This is scenario 1 in Figure 7.1. In our algorithm unlike [6], [1],[5], node collusion does not result in any privacy loss. A node can only affect one other node at a time like in cases 1 & 2. Even when the nodes collude there is no more damage than when they are not colluding.

7.2. MULTIDIMENSIONAL PIP

Most motes these days have multiple sensors and in many cases, we need these motes to use more than one of them. This gives rise to multidimensional data. The challenge with multidimensional data is to ensure its privacy and integrity and be energy and bandwidth efficient. A naive solution is to protect the privacy and integrity of each dimension of the data individually. However, because the provenance of all of these dimensions is same more efficient solutions can be achieved. Multidimensional privacy preserving data aggregation was treated in [7]. The authors in [7] did not consider data integrity, we, however, observe that integrity can be built into their solution, in the same manner as ours. Their scheme assumes a heterogeneous network, where the aggregators nodes are few and are more powerful than the rest. This limits the flexibility of their solution as every node must be within the vicinity of a powerful aggregator failing which, it cannot be a part of the data aggregation process.

Both the basic and enhanced versions of PIP can be extended easily to accommodate multidimensional data. Unlike the solution in [7], our solution works in a homogenous environment without the assumption of any powerful aggregator node. We begin with the basic PIP, taking the first dimension as data, adding other dimensions just like the integrity key was added in the basic PIP algorithm. Integrity key is added similarly at the end. For each dimension, this solution adds one share. Thus, the bandwidth usage after the first dimension increases by one share per dimension, compared to the naive solution in which each dimension would require 3 shares.

8. PERFORMANCE ANALYSIS

We implemented the algorithms on the TelosB mote platform using TinyOS 2.x. The TelosB mote platform has a TI MSP430 16 bit microcontroller, with 10 kB of RAM and a clock speed of 8MHz. We measured the execution time of the share generation algorithm and the data regeneration and integrity check algorithm for different data/share sizes and the results are shown in Figure 8.1. Figure 8.2 illustrates the corresponding energy consumption for varying data/share sizes. To measure energy consumption we used the formula $E = V * I * t$, where V is the voltage supplied, I is the current drawn, and t is the time taken for the operation. From Figures 8.1 and 8.2, we can deduce that PIP introduces very little delay in the network and consumes very little energy while providing privacy and integrity of the data. As discussed previously, asymmetric homomorphic encryption and signatures such as in [2] have been used to provide privacy and integrity. Table 8.1 shows a comparison between the homomorphic SDA algorithm and PIP scheme for two different key sizes. SDA uses Elliptic Curve Cryptography (ECC). According to the 2011 ECRYPT report, a 128-bit ECC key has security equivalent to a 64-bit symmetric key, while the 192-bit ECC key is equivalent to a 96-bit symmetric key. As previously discussed, a b bit share size in PIP has security equivalent to $2b$ bit symmetric key, by virtue of using two b bit keys, the scrambling key and the perturbation key. Therefore, we compare the 128-bit ECC in SDA with 32-bit PIP and 192-bit ECC with 48-bit PIP. It can be seen that, PIP consumes far less energy for equivalent security than SDA. The difference is approximately an order of 4.

Table 8.1. Energy efficiency of PIP compared to SDA[2]

Scheme	**32/128 bit**	**48/192 bit**
SDA	79.77 mJ	115.7 mJ
PIP	0.047 mJ	0.068 mJ

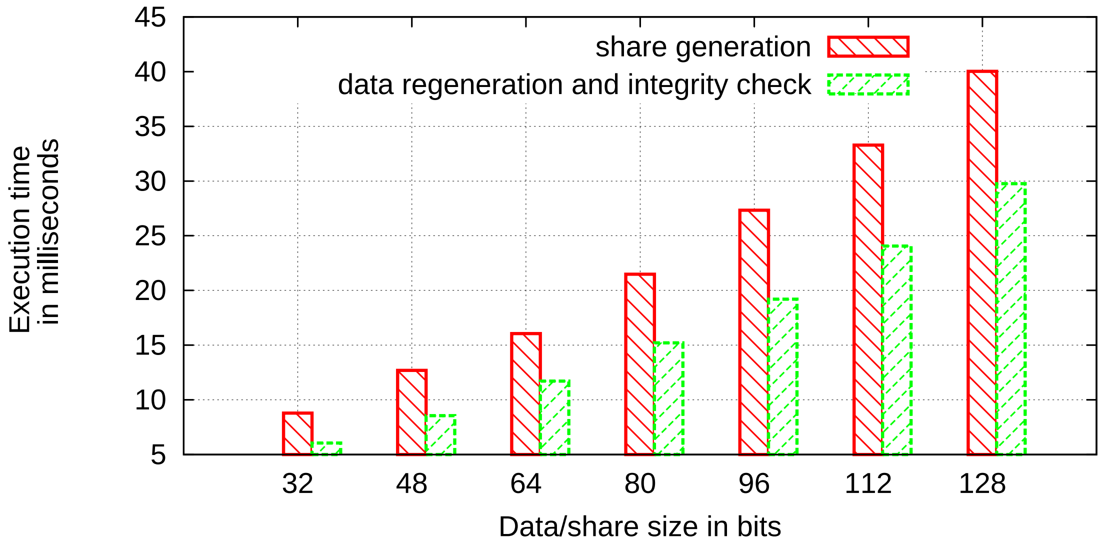

Figure 8.1. Execution time of Basic PIP on TelosB motes

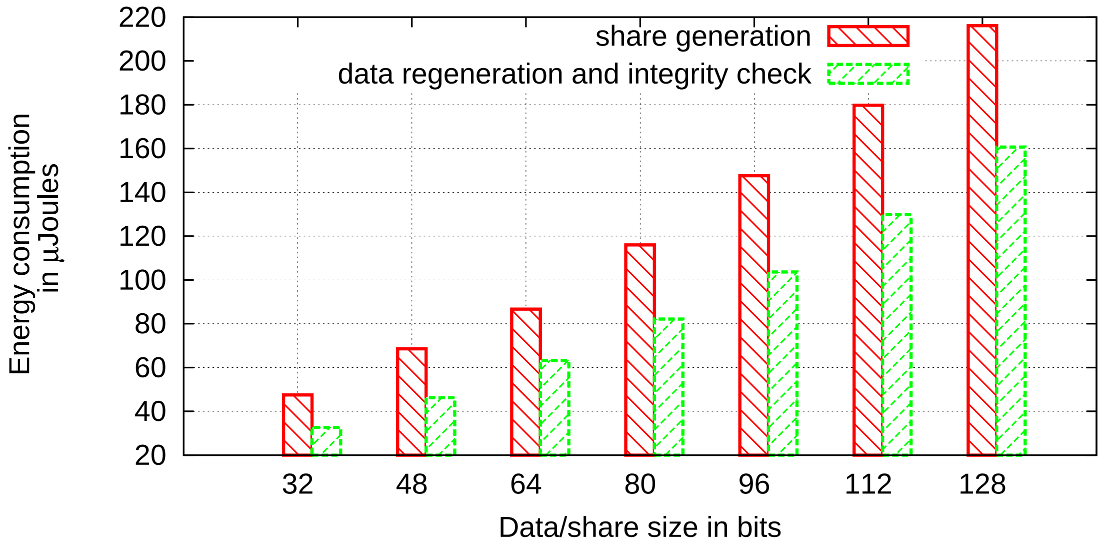

Figure 8.2. Energy consumption Basic PIP on TelosB motes

We have also implemented enhanced PIP on TelosB motes. Because the calculation of higher degree polynomials is a time-consuming process, we pre-computed several repeatedly used coefficients and placed them in the init function. The execution time for varying data/share sizes in enhanced PIP is shown in Figure 8.3. Even after pre-computing the repeatedly used coefficients, we can see that the time required to generate shares in enhanced PIP is much higher than in basic PIP. The energy consumed in enhanced PIP, although not shown, will also be proportionally higher. The memory requirements of basic and enhanced PIP algorithms are shown in Figure 8.4. We see that the RAM required by the algorithms is modest, ranging from under 1kB for both basic and enhanced versions for 32-bit share size to a maximum of 1.54 kB for enhanced PIP with a 128-bit share size. Thus both versions of PIP are easy to load on most low memory wireless sensors.

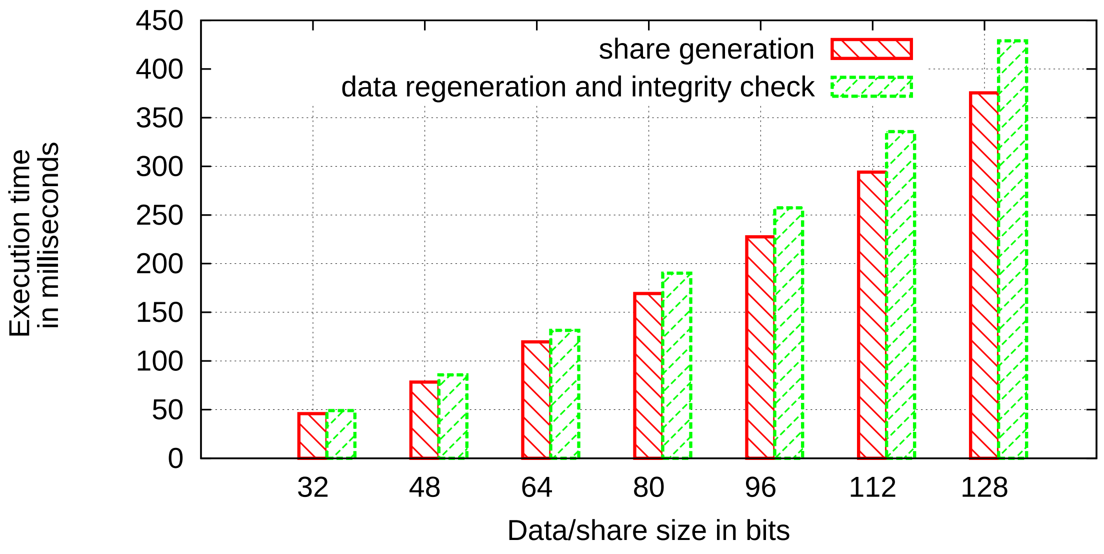

Figure 8.3. Execution time of enhanced PIP on TelosB motes

We compare our algorithm with iPDA and iCPDA in terms of bandwidth consumption per node. We consider the header size of the network packet to be 18 bytes, which is the case for TelosB on TinyOS 2.x. For iPDA, to ensure a fair comparison, we assume that the parameter J, which dictates the number of slices of the data item is 3. In iPDA, each node slices its data into J parts and sends out $2J-1$ packets to other nodes in the network. For iCPDA, we take the average size of the cluster $k=3$. The parameter p_c, which is the

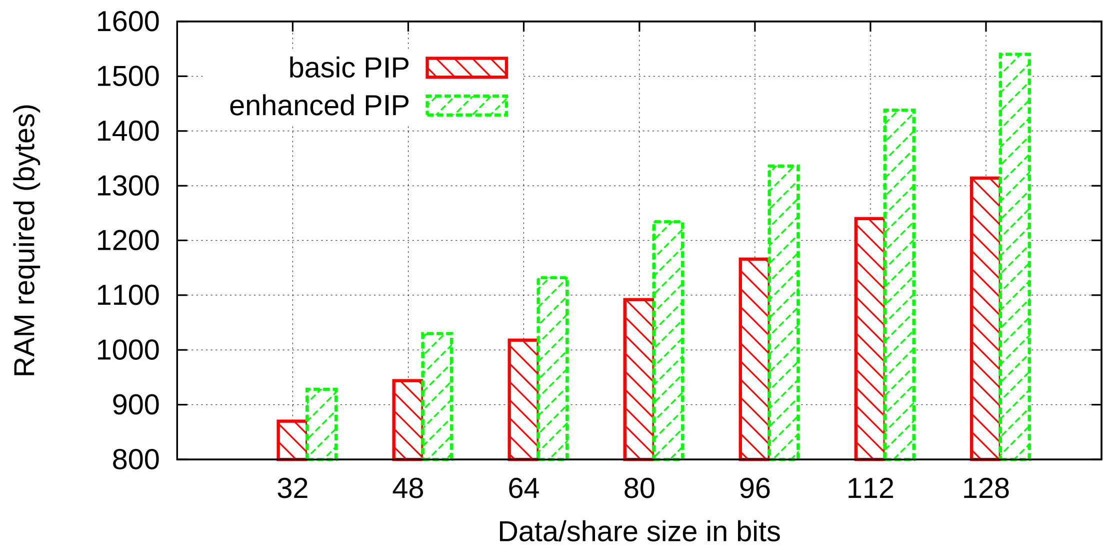

Figure 8.4. Memory required for basic and enhanced PIP on TelosB

number of cluster heads is $1/k$. Thus 1/3 of the nodes in the network are cluster heads. We also assume that the cluster tree of iCPDA has degree 2. In this case then, each node on average sends $(k^2+k+1)/2$ packets and receives on average $(3k^2-1)/k$ packets. In contrast, in our algorithm, regardless of the number of children each aggregator had, each node sends out 3 shares in a single packet and each aggregator receives d such packets, where d is the degree of the tree taken to be 3 here. The comparison of the bandwidth required per node is illustrated in Figure 8.5. Figure 8.6 shows the comparison of energy consumption of the three algorithms. Energy on a node is mainly consumed in the encryption (share generation) operation and in the wireless transmission and reception of the data. For this comparison we vary the number of nodes whose data has to be aggregated. In iPDA, this parameter is J, which governs the number of slices in which a data has to be divided. A node sends the slices to $2J-1$ other nodes. Each slice of the data is encrypted with a shared symmetric key between the pair of nodes. Thus, every node in the network, on average, both sends and receives $2J-1$ encrypted slices of data. This amounts to $2J-1$ encryptions and decryptions on every node. In iCPDA, this parameter is the cluster size k. Here, each node sends out $k-1$ encrypted messages to $k-1$ other nodes in the cluster. All other communication in the cluster is in cleartext. In our algorithm, this parameter may be chosen as the degree of the tree. Our algorithm, however, is agnostic to this metric and hence has a constant computational overhead. The aggregator node however, receives a varying number of packets from its child nodes depending on its degree. This consumes an amount of energy which varies with the number of child nodes each aggregator has. This energy spent in transmission and reception of packets however, is small when compared with iPDA and iCPDA due to lesser amount of wireless communication as shown in Figure 8.5. In the experiment, we chose AES-256 encryption for iPDA and iCPDA. In our implementation on TelosB motes, each AES-256 encryption consumes .01 millijoule and decryption consumes .016 millijoule of energy. We assume that the computational overhead in slicing and adding the data in iPDA and iCPDA, and addition of aggregates in PIP is negligible. In both iPDA and iCPDA, the privacy of the algorithm depends on the number of nodes taking part in aggregation. The computational overhead, however, increases linearly with the number of nodes and hence a tradeoff must be made between privacy and computational overhead. In our case though, the number of nodes in aggregation does not affect the privacy preserving capability of the algorithm and hence, no such tradeoff is required. As discussed previously, the security of a b bit share in PIP is equivalent a *2b* bit symmetric key. We thus, choose to compare the two algorithms with the 128 bit version of PIP. In Figure 8.6, we can see that PIP performs better in terms of energy consumption when aggregating data for 3 or more nodes in case of iPDA and at all times in case of iCPDA.

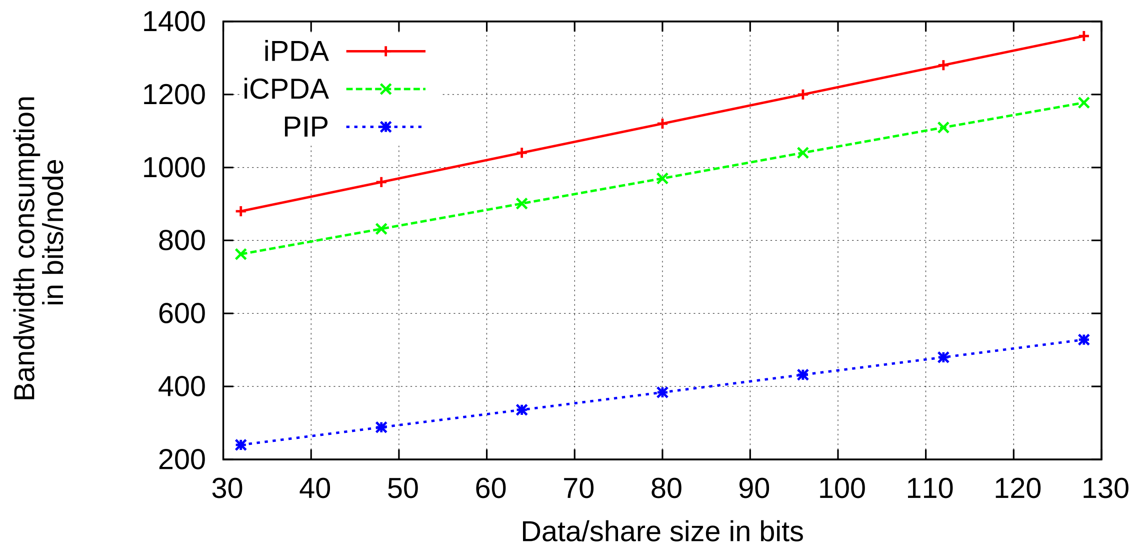

Figure 8.5. Bandwidth consumption per node for varying share size

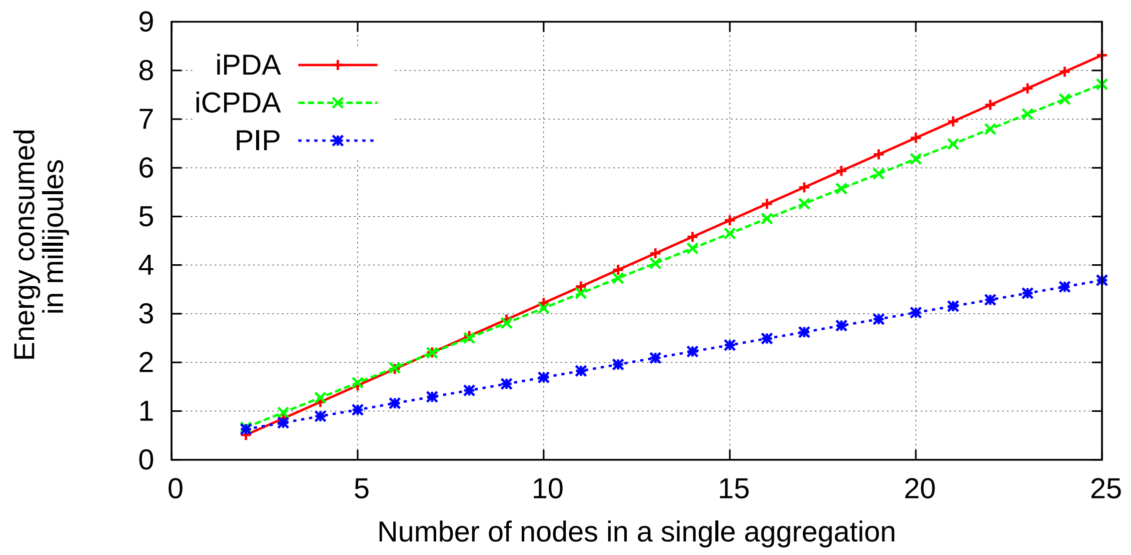

Figure 8.6. Energy consumed as a function of nodes involved in aggregation

9. CONCLUSION

In this paper, we first presented the basic PIP algorithm, to perform privacy and integrity preserved data aggregation in wireless sensor networks. We then provided the enhanced PIP algorithm which can detect corrupt aggregators that inject false data into the system. We implemented both algorithms on TelosB motes and measured the execution time and energy consumption. The results establish that the algorithms are lightweight in terms of energy and introduce very little delay in the network. We also compared the basic PIP algorithm's bandwidth requirement with two other existing algorithms with similar objectives and found that basic PIP performs better than both. Finally, we noted that while the privacy in other algorithms depended on the number of nodes taking part in aggregation, in our algorithms it is not so and this also helps us save energy at the aggregators.

10. BIBLIOGRAPHY

[1] W. He, X. Liu, H. V. Nguyen, K. Nahrstedt, and T. Abdelzaher, “Pda: Privacy-preserving data aggregation for information collection,” *ACM Trans. Sen. Netw.*, vol. 8, no. 1, pp. 6:1–6:22, Aug. 2011.

[2] V. Kumar and S. K. Madria, “Secure hierarchical data aggregation in wireless sensor networks: Performance evaluation and analysis,” *Mobile Data Management, IEEE International Conference on*, vol. 0, pp. 196–201, 2012.

[3] J. Girao, D. Westhoff, and M. Schneider, “Cda: concealed data aggregation for reverse multicast traffic in wireless sensor networks,” in *Communications, IEEE International Conference on*, vol. 5, May 2005, pp. 3044 – 3049 Vol. 5.

[4] T. Feng, C. Wang, W. Zhang, and L. Ruan, “Confidentiality protection for distributed sensor data aggregation,” in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008, pp. 56–60.

[5] L. Zhang, H. Zhang, M. Conti, R. Pietro, S. Jajodia, and L. Mancini, “Preserving privacy against external and internal threats in wsn data aggregation,” *Telecommunication Systems*, pp. 1–14, 2011.

[6] M. Conti, L. Zhang, S. Roy, R. Di Pietro, S. Jajodia, and L. V. Mancini, “Privacy-preserving robust data aggregation in wireless sensor networks,” *Security and Communication Networks*, vol. 2, no. 2, pp. 195–213, 2009.

[7] X. Lin, R. Lu, and X. S. Shen, “Mdpa: multidimensional privacy-preserving aggregation scheme for wireless sensor networks,” *Wirel. Commun. Mob. Comput.*, vol. 10, no. 6, pp. 843–856, Jun. 2010.

[8] C. Wang, G. Wang, W. Zhang, and T. Feng, “Reconciling privacy preservation and intrusion detection in sensory data aggregation,” in *INFOCOM, 2011 Proceedings IEEE*, 2011.

[9] W. He, H. Nguyen, X. Liu, K. Nahrstedt, and T. Abdelzaher, “ipda: An integrity-protecting private data aggregation scheme for wireless sensor networks,” in *MILCOM 2008. IEEE*, nov. 2008, pp. 1 –7.

[10] W. He, X. Liu, H. Nguyen, and K. Nahrstedt, “A cluster-based protocol to enforce integrity and preserve privacy in data aggregation,” in *Distributed Computing Systems Workshops, 2009. ICDCS Workshops ’09. 29th IEEE International Conference on*, June 2009, pp. 14 –19.

[11] A. Shamir, “How to share a secret,” *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[12] A. Parakh and S. Kak, “Recursive secret sharing for distributed storage and information hiding,” ser. ANTS’09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 88–90.

[13] C. Castelluccia, A. C.-F. Chan, E. Mykletun, and G. Tsudik, “Efficient and provably secure aggregation of encrypted data in wireless sensor networks,” *ACM Trans. Sen. Netw.*, vol. 5, no. 3, pp. 20:1–20:36, Jun. 2009.

III. ATTRIBUTE BASED ACCESS CONTROL OF AGGREGATED DATA IN SENSOR CLOUDS

Vimal Kumar∗, Sanjay K Madria∗,

∗ Department of Computer Science,

Missouri University of Science and Technology, Rolla, Missouri 65401

Sensor clouds are large scale wireless sensor networks (WSNs), built by connecting a number of smaller WSNs together. Each of these smaller individual WSNs may be owned by different owners. Sensor clouds are dynamic in nature, where wireless sensors can be provisioned and de-provisioned for the users on demand. In such a multi-user, multi-owner system, user access control is a significant problem. Previous user access control schemes have been designed for standalone sensors or smaller networks and do not take large networks into consideration.

In large networks, data is generally aggregated in-network during data collection. In this paper, we present our user access control scheme, which works on aggregated data in a sensor network. Our scheme, which is based on attribute based encryption is able to differentiate between users which require data with the same set of attributes. This is necessary in a commercial sensor cloud system for billing and management purposes. Our scheme gives the flexibility to sensor network owners, to control user access of data from their sensors. Finally, we compare our scheme to related schemes, in terms of computation and communication overhead to show its effectiveness.

1. INTRODUCTION

Sensor Cloud [1], [2] is an emerging paradigm of computing for Wireless Sensor Networks, which facilitates resource sharing and enables large scale sensor network deployment. It brings together multiple WSNs owned by different entities, each of which may contain many wireless sensors. Sensor clouds are characterized by dynamic provisioning and de-provisioning of wireless sensors for the users based on demand.

Sensor clouds share some of the same security issues as WSNs and introduce some new ones owing to multiple users and multiple WSN owners [3]. The security issue we consider in this paper is access control of sensor data. Sensor clouds are distributed systems which generate data in real time and on demand. The data generally is aggregated in network which requires that the access control mechanism should support secure data aggregation in the event of nodes being captured. To aggravate the issue, in sensor clouds, the network topology is not known in advance since the nodes are provisioned and de-provisioned on demand for each user. Moreover, a sensor cloud composed of multiple WSNs may be owned by different users each of which may place certain restrictions on who can and cannot access the data being generated by their sensors. This possibility is unique to sensor clouds and has not been sufficiently explored before. Thus, what is required is an access control scheme for sensor clouds, where only authorized users are able to access data from the sensors. The access control scheme should be able to work in an environment where the data is aggregated in network and the topology of sensor nodes can vary for each user and each query. This access control scheme should also be able to integrate the WSN owner permissions on who can access data from their sensor nodes.

A solution which has been proposed in [4] and [5] is to encrypt data such that only the authorized users are able to decrypt it. The solutions in [4] and [5] though have considered only a partially distributed system. The system model in these works does not consider an in-network data aggregation scenario.

Keeping the above requirements of sensor cloud in mind as well as shortcomings of some of the related work, in this paper, we propose an attribute based user access control scheme, which takes into account that data is usually aggregated in wireless sensor networks to save energy and bandwidth. We also consider that each WSN in the sensor cloud may impose authorization restriction on the data usage. In our scheme, data instead of being stored on a node is collected in response to the user's query and is aggregated during the process. We also provide a scheme where the network owner or the sensor network itself can change the authorization level of the data during run time under special conditions. In our scheme, the sensors are able to differentiate between different users who might have the same query, attributes and authorization level. This is of particular help in a commercial system which needs to bill the users according to their usage and so needs to differentiate between users running the same query. We also discuss a method to increase the flexibility in the kind of queries that can be run on the sensor cloud by manipulating the access tree.

Formally, our contributions in this paper are as follows.

- A fully distributed, fine grained access control scheme for aggregated data in sensor clouds.
- A scheme which considers runtime modification of authorizations.
- Integration of the access control scheme with a secure data aggregation scheme.

2. RELATED WORK

User access control in wireless sensors has received considerable attention in recent years. Using Access Control Lists (ACLs) in wireless sensors where each sensor verifies the access allowed for each user by validating them against its ACL is not a scalable and suitable in WSN environment where users can leave and new users can join. In [6], a user receives a symmetric key K_i for each sensor node SN_i it wants to access. This key is generated by a trusted authority and is based on the user's credentials. The drawback of this scheme is that the user needs to know the identities of all the sensor nodes it wants to access. The scheme also does not support data aggregation, which implies that to access data from multiple nodes, the user will need to individually communicate and get authenticated with each node. A similar approach was presented in the context of service oriented architecture in [7]. This approach has the same drawbacks as that in [6]. Moreover, the access is not fine grained; users are grouped by roles and all users in a particular role have the same access.

In [8], an Elliptic Curve Cryptography based approach, the trusted authority issues an ACL to the user, based on its credentials and issues a public private key pair along with a certificate. This certificate is needed to be verified by a local sensor node before granting access rights. To access data from a remote node, the user needs to get endorsements from k local nodes. These endorsements are verified by the remote nodes before granting access. In this approach, the user need not know the identities of the nodes in advance but like the previous scheme, this scheme too does not support data aggregation.

A more fine grained access control approach has been presented in [9], [4] and [5]. These solutions are based on the attribute based encryption (ABE) methods. The system model in [9] is different from the conventional system models used for WSNs. In this paper, the access is controlled by the wireless device itself rather than being co-ordinated by a trusted authority. The scheme uses CP-ABE [10] for access control of data, where the access policy is embedded in the ciphertext itself. However, it is computationally very expensive, compared to other ABE methods. In [4], each attribute of the data is associated with a public key component. An access tree based on the attributes of required data is created for a given user. The access tree is then used to generate the private key which is provided to the user. The user then provides the access tree to the sensor and the sensor provides data to the user according to the access tree. The data is encrypted such that it can only be decrypted with a private key generated based on that access tree. This scheme was further enhanced in [5] to include multiple base station each of which controls a set of attributes.

The concept of ABE was first introduced in [11]. Goyal et. al. then proposed key policy based ABE in [12]. The ABE schemes were proposed for wired systems, where the data is stored at a server and the access is provided according to the attributes held by various users. The schemes in [4] and [5] are direct adaptations of KP-ABE in wireless sensor networks and do not take the distributed nature of WSNs into account. In the system model presented in [4] and [5], users access data either directly from sensor nodes or from a storage node which collects data from other sensors. While this is a feasible model, the heterogeneity limits the flexibility of the network. We, in this paper, propose a dynamic attribute based access control mechanism for wireless sensor networks which works under the general assumption of a homogenous network.

3. MODELS

3.1. SYSTEM MODEL

The sensor cloud system as illustrated in Figure 3.1 consists of three parties, the Users ($\mathcal{U}$), the Sensor Cloud Administrator (*SCA*) and the Sensor nodes (*SN*). The sensor nodes are in the form of individual WSNs which are maintained by the WSN owners. To access data from the sensors, a user first contacts the sensor cloud administrator (*SCA*) with a query over the attributes of the data it wants to access. It then receives a secret key over these attributes from the *SCA*. This secret key is then used to access data from the wireless sensors, which has been encrypted based on the said attributes.

3.2. ADVERSARY MODEL

We assume that the adversary is capable of capturing a certain percentage of sensor nodes. The adversary has the following goals:

- To try to get access to the data, for which it does not have the secret key.
- To try to access data for which the user does not have authorization.
- To tamper with the keys and data meant for other users, so as to disrupt the protocol.

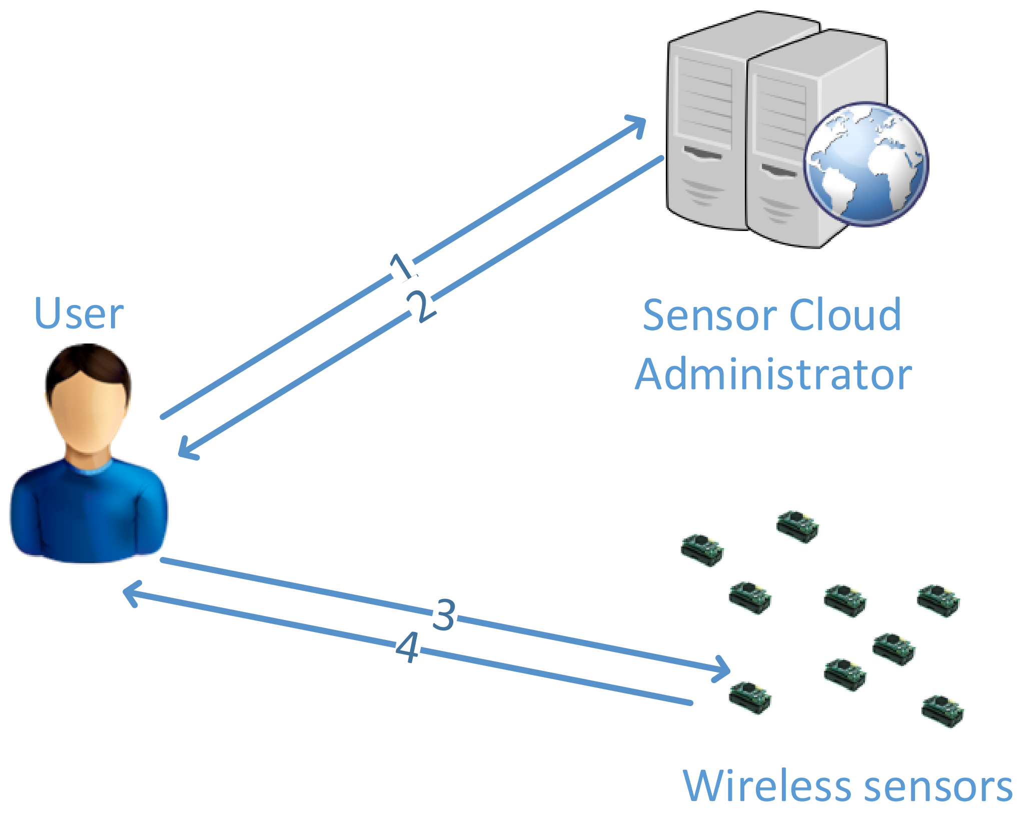

Figure 3.1. System Model

4. PRELIMINARIES

4.1. BILINEAR MAPS

Let G_1 and G_T be cyclic groups of the prime order q. Typically, G_1 is an elliptic curve group and G_T is a finite field. We, therefore, denote G_1 using additive notation and G_T using multiplicative notation. Let P and Q be two generators of G_1. A bilinear map then is an injunctive function $e : G_1 \times G_1 \rightarrow G_T$, which has the following properties.

- Bilinearity: $\forall P, Q \in G_1, \forall a, b \in \mathbb{Z}_q^*$, such that we have $e(aP, bQ) = e(P, Q)^{ab}$
- Non-degeneracy: If $P \neq 0$, then $e(P, P) \neq 1$
- Computability: There exists an efficient algorithm for computing $e(P, Q), \forall P, Q \in G_1$

4.2. KEY POLICY ATTRIBUTE BASED ENCRYPTION

In key policy attribute based encryption (KP-ABE) [12], each ciphertext is associated with a set of attributes. The access policy is defined by an access tree and the private key is generated based on this access tree, hence the name key policy. A cipher-text can only be decrypted with a key, if the attributes associated with the ciphertext satisfy the key's access tree. The KP-ABE [12] is composed of four algorithms, *Setup, Encryption, Key Generation, and Decryption*.

- *Setup:* The Setup algorithm defines the attributes in the system and outputs a public key PK and a master key MK. The public key PK is used for encryption while the master key MK along with PK is used to generate user keys.
- *Encryption:* The encryption algorithm takes as input a message m, a set of attributes γ and the public key PK and produces the cipher-text E

- *Key Generation:* The key generation algorithm takes as input an access tree $\mathcal{T}$, the master key MK and the public key PK to produce a secret key SK, such that SK can decrypt E iff $\mathcal{T}$ matches γ.

- *Decryption:* The decryption algorithm takes as input the ciphertext E, the decryption key SK and the public key PK and decrypts the ciphertext iff $\mathcal{T}$ matches γ, otherwise it produces $\perp$.

4.3. PAILLIER ENCRYPTION

The Paillier cryptosystem [13] is a public key encryption scheme based on the Decisional Composite Residuosity Assumption (DCRA). The DCRA states that given a composite $n = n_1 \times n_2$, for primes n_1 and n_2, and an integer z, it is hard to find out whether there exists a y such that $z \equiv y^n \bmod n$. A message $M \in \mathbb{Z}_n$ in Paillier encryption is encrypted as,

$$C = g^M.r^n \bmod n^2$$

Where $g \in \mathbb{Z}^*_{n^2}$ is a public element, while $r \in \mathbb{Z}^*_n$ is chosen randomly by the encryptor. The ciphertext C can be decrypted as.

$$M = L(C^\lambda \bmod n^2).\mu \bmod n$$

Where $\lambda = lcm(n_1 - 1, n_2 - 1)$, $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$ and $L(x) = (x-1)/n$

The property of Paillier encryption we are most interested in is its additive homomorphism. Given C_{m_1} and C_{m_2}, paillier ciphertexts for m_1 and m_2. The ciphertext for $m_1 + m_2$ can be generated by multiplying C_{m_1} and C_{m_2}.

$$C_{m_1+m_2} = C_{m_1}.C_{m_2}$$

$C_{m_1+m_2}$ can then be decrypted by the private key to retrieve $m_1 + m_2$.

4.4. PIP ALGORITHM FOR DATA AGGREGATION

PIP [14] is a privacy and integrity preserving data aggregation algorithm for wireless sensors. PIP uses three different keys, perturbation key η, integrity key (I', I'') and scrambling key Ψ. While η, I', and Ψ are generated pseudo-randomly and independently by each sensor node, I'' is common across the network. In the scheme, first the perturbation key is used to perturb the data. The perturbed data is then used to generate random shares. The scheme then uses recursive secret sharing to embed the augmented integrity key into the shares of the perturbed data. The resulting shares are then scrambled using the scrambling key. The scheme is homomorphic, so the scrambled shares can be added together. The data regeneration protocol then uses the summation of the keys to regenerate the data.

We use PIP as the representative secure data aggregation protocol in our scheme. However, any secure data aggregation protocol which uses a summation of individual sensor keys at the base station and provides confidentiality and integrity can be used.

5. ACCESS CONTROL POLICY

To obtain fine grained access control of data, a tree based access structure was proposed in [12]. The basic idea behind such an access control structure is that each data item can be described by certain attributes attached to it. In the case of sensor clouds, the data being generated by the sensors has certain descriptive attributes such as the type of sensor which generated the data, type of mote on which the sensor is mounted, sensor's location, sensor's owner etc. When users require data from the sensor cloud, they can formulate complex queries based on these attributes to exactly specify the kind of data they want. An example of such a query can be a user requesting data which is either of sensor type T_1 or T_2, from sensors which belong to any two of the three sensor owners O_1, O_2 and O_3 and which are deployed in region, R_1 . Such a query can be represented by a tree, called an access tree $\mathcal{T}$ as shown in Figure 5.1. In the access tree $\mathcal{T}$, leaf nodes are associated with the attributes while each non-leaf node represents a threshold gate. The logic gate AND can be represented by a $2-of-2$ threshold gate, while an OR can be represented by a $1-of-n$ threshold gate.

Such an access tree is very expressive and can be used to define a number of queries.

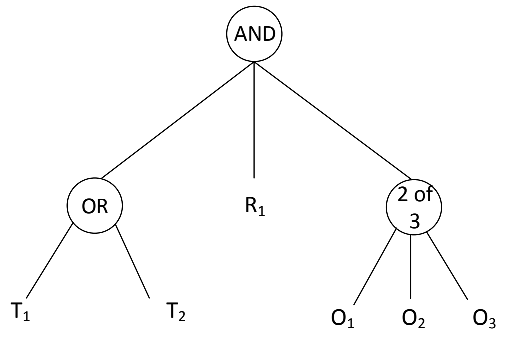

Figure 5.1. Example access tree

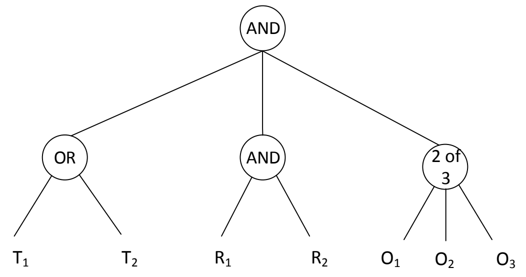

Figure 5.2. Unacceptable access tree

In a distributed environment however, there are certain types of queries which cannot be expressed by this access tree. For example if in the previous query, the user requests data from both regions R_1 and R_2, then an access tree as shown in Figure 5.2 would be formed. This access tree may work in the centralized models of [4] and [5], since the data has already been collected and stored on a single node. In a distributed model though, no node would satisfy the criteria of being in regions R_1 and R_2 simultaneously and thus, no data would be returned.

We propose a modification which improves the access tree in order to satisfy such kind of queries. Our modification also allows partial satisfaction of access trees. First, we classify the attributes into two types.

- **Type A** attributes are those attributes whose multiple instances can be satisfied by a single sensor, e. g. sensor type attribute with instances humidity and temperature can be satisfied by a node which has both humidity and temperature sensors mounted on it.

- **Type B** attributes are the attributes which can only be satisfied one instance at a time, e.g region attribute. A sensor can only be in one region at a given point of time.

We modify the access tree of Figure 5.2 in such a manner that each top level subtree can be satisfied individually by a sensor. To convert the access tree of Figure 5.2 to an acceptable and partially satisfiable form we present a procedure shown in Algorithm 8. The partially satisfiable form of the access tree is shown in Figure 5.3. Figure 5.3 has two top level subtrees, each of which can be individually satisfied by a sensor which is either in region R_1 or R_2.

Algorithm 8 Access Tree Conversion

```
Require: Access Tree $\mathcal{T}$, Tree arrays $X, B$
 1: Remove all the Type B attribute subtrees from $\mathcal{T}$
 2: for each subtree $\in T$ do
 3:     if subtree is a Type $B$ subtree then
 4:         Remove the subtree from $\mathcal{T}$
 5:         Add the subtree to $B$
 6:     end if
 7: end for
 8: Add $\mathcal{T}$ to $X$
 9: for each subtree $\in B$ do
10:     Create as many copies of the access trees in $X$ as there are attributes in the subtree
11:     Concatenate one attribute to each copy
12:     Store the concatenated access tree back in $X$
13: end for
14: Connect all the new trees through an AND gate
```

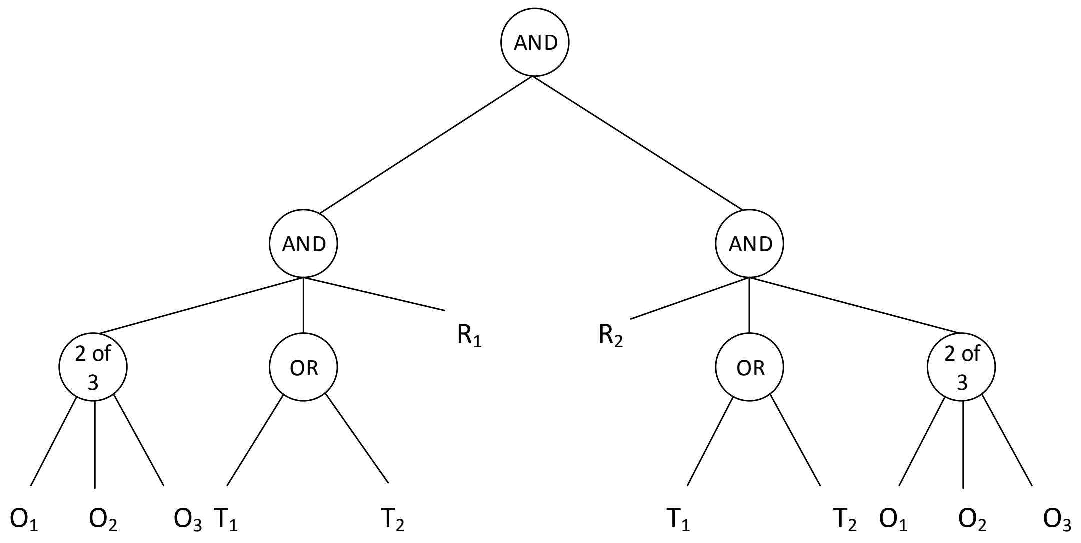

Figure 5.3. Acceptable access tree

6. OVERVIEW OF THE SCHEME

As discussed before, the system consists of a set of users $\mathcal{U}$, sensor cloud administrator SCA and the senor nodes SN. In the setup phase, the SCA generates the public and the master keys. The public key is assumed to be known by the users as well as the sensors. When a user $\mathcal{U}_j$ wants to access data from the sensor cloud, it approaches SCA, with its query on the data attributes. SCA uses the query to create an access tree $\mathcal{T}_j$ and a secret key SK_j over $\mathcal{T}_j$ and $\mathcal{U}_j$'s temporary identity. $\mathcal{T}_j$, SK_j and the public and private components of the user's temporary identity are then given to the user.

A user may contact any of the sensor nodes for data. We designate the node which receives the user request as the Gateway Node (GN_j). After receiving the query, in the form of the access tree $\mathcal{T}_j$, GN_j floods the network with this query, creating a query tree Q_j. Each node in, Q_j, which satisfies the access tree $\mathcal{T}_j$, then generates a random session key and encrypts this key using the key policy attribute based encryption. The encrypted random session keys of the query tree Q_j, are aggregated and sent to the user via GN_j. The user then decrypts the aggregated session key, since it has the secret key SK_j. Once the user is able to compute the aggregated session key, it derives the symmetric keys to be used in data aggregation and proceeds with data collection.

7. ACCESS CONTROL SCHEME

7.1. SYSTEM SETUP

The Sensor Cloud Administrator (SCA), defines the set of all attributes $\mathcal{A}$. For each attribute $i \in \mathcal{A}$, a number t_i is chosen uniformly at random from $\mathbb{Z}_q^*$. A number y is also chosen randomly from $\mathbb{Z}_q^*$. SCA then generates a Paillier public-private key pair (K_{pub}, K_{pr}) and creates the public key as,

$$PK = (G_1, P, T_1 = t_1P, T_2 = t_2P, ..., T_{|\mathcal{A}|} = t_{|\mathcal{A}|}P, K_{pub}, Y = e(P,P)^y)$$

The master secret key is,

$$MK = (t_1, t_2, ..., t_{|\mathcal{A}|}, y)$$

Each sensor is then loaded with, the public key PK, a PRF $f(.)$ and a nonce x.

7.2. ACCESS CONTROL SECRET KEY GENERATION

As illustrated in Figure 3.1, when a user $\mathcal{U}_j$ wants to collect data from the sensor cloud, it first contacts the SCA for access. Along with it's request, $\mathcal{U}_j$ also provides a query based on the attributes of the data it wants to access. The SCA then creates an access tree $\mathcal{T}_j$, based on the user's query and the attributes, such as the one in Figure 5.1. The access tree is then augmented by ANDing a new node, "ID" to the root node as shown in Figure 7.1. This makes sure that the secret key is always bound to an ID. This would help in easy revocation at a later time. To create a unique identity attribute for the user, the SCA then randomly generates a previously unused number t_j from $\mathbb{Z}_q^*$ and its public component t_jP. Once the access tree $\mathcal{T}_j$ is created, SCA proceeds to generate the secret key SK_j as follows. Starting from the root node, SCA constructs a random polynomial u_x of degree

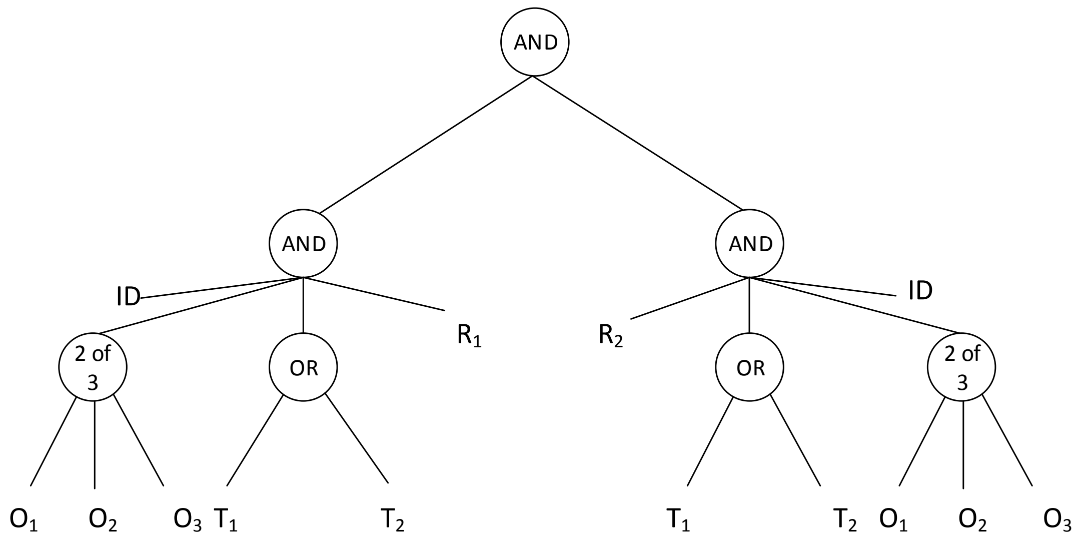

Figure 7.1. Access tree augmented with ID

$d_x + 1$ for each node x in $\mathcal{T}_j$, where d_x is the degree of that node. For the root node r it sets $u_r(0) = y$ and chooses the rest of the points randomly. For all other nodes it sets $u_x(0) = u_{parent(x)}(index(x))$, where $parent(x)$ denotes the parent of node x and $index(x)$ returns an enumeration on the children of the parent of node x. All other points are chosen randomly. The secret key SK is then defined as,

$$SK_j = (D_k = \frac{u_k(0)}{t_k}P, k \in \gamma)$$

Where, γ is the set of leaf nodes in $\mathcal{T}_j$ including the node ID and D_k is calculated for all the leaf nodes k in $\mathcal{T}_j$. SCA then gives the secret key SK_j, the access tree $\mathcal{T}_j$, nonce b and T_j to the user $\mathcal{U}_j$.

7.3. DATA AGGREGATION KEY GENERATION

When a user $\mathcal{U}_j$ contacts GN_j for data, it provides the query as the tuple $< \mathcal{U}_j, \mathcal{T}_j, T_j, r >$, where $\mathcal{T}_j$ is the access tree for the query, T_j is the public component of the user's identity and r is a user generated pseudorandom number. GN_j floods this query in the sensor cloud which results in a query tree Q_j. Each node i whose attributes satisfy the access tree $\mathcal{T}_j$, uses the PRF $f(.)$ on the nonce b with r as the key to generate a value $s_i = f_r(b) \in \mathbb{Z}_q$. Each node then generates a partial session key $p_i \in G_T$. which is then encrypted using Paillier encryption to obtain C_{p_i}. This encrypted partial session key C_{p_i} is then again encrypted as,

$$E = (E' = C_{p_i} Y^{s_i}, \{E_k = s_i T_k\}, k \in \gamma) \tag{1}$$

Leaf nodes then send the encrypted partial session key to their parent, where it is aggregated as,

$$E = (E' = \prod C_{p_i} Y^{s_i}, \{E_k = \sum s_i T_k\}, k \in \gamma)$$

which is equal to,

$$E = (E' = \prod C_{p_i} Y^{\sum s_i}, \{E_k = T_k \sum s_i\}, k \in \gamma)$$

The final aggregate ciphertext of the query tree Q_j at the gateway node is,

$$E_{Q_j} = (E' = CY^s, \{E_k = sT_k\}, k \in \gamma) \tag{2}$$

where, $s = \sum_{i \in Q_j} s_i$ and $C = \prod_{i \in Q_j} C_{p_i}$ is the Paillier encrypted aggregate session key.

7.4. DATA AGGREGATION KEY ESTABLISHMENT

The ciphertext in equation 2 which consists of the encrypted session key is given to the user, who proceeds to decrypt the ciphertext. The decryption process works in a bottom up manner, starting from the leaf nodes of the access tree Q_j. For each leaf node k the value F_k is calculated using the below formula.

$$F_k = \begin{cases} e(D_k, E_k) = e(P,P)^{su_k(0)} & if, k \in \gamma_j \\ \perp & otherwise \end{cases}$$

For each non leaf node z, if z is a d out of n gate and if more than $n-d$ children return $\perp$ then F_z is $\perp$ otherwise F_z is calculated as,

$$\begin{aligned} F_z &= \prod_{i \in S_z} F_i^{\delta_i(0)} \\ &= \prod_{i \in S_z} e(P,P)^{su_i(0)\delta_i(0)} \\ &= e(P,P)^{su_z(0)} \end{aligned}$$

where S_z denotes the set of node $z's$ children and $\delta_i(0)$ denotes the Lagrange coefficient. The user $\mathcal{U}_j$ then computes F_z for the root node, which returns $e(P,P)^{ys} = Y^s$, if the operation is successful. Since E' in the received ciphertext in eq. 2 is CY^s, the user simply divides E' by Y^s to obtain C, which is the aggregate Paillier cipertext of the partial session keys. C can be decrypted to obtain the aggregate of the session keys, S (We would call it the session key). Once the user $\mathcal{U}_j$ obtains the session key S, it is used to derive keys for the secure data aggregation algorithm. The secure data aggregation algorithm in PIP [14] previously discussed, uses three keys; perturbation key η, integrity key (I', I'') and scrambling key Ψ. The keys η, I' and Ψ can be easily derived once S is known while I'' is assumed to be public knowledge. One way to derive the keys can be by using simple modular functions as shown in equations, 3, 4, 5. k_1, k_2, k_3 here, are assumed to be known to the user as well as the sensors.

$$\eta = S \bmod k_1, \quad (3)$$

$$I' = S \bmod k_2, \quad (4)$$

$$\Psi = S \bmod k_3, \quad (5)$$

7.5. DATA AGGREGATION

The sensors, use the same formulae in equations, 3, 4, 5, on their individual partial session keys to derive their individual data aggregation keys. The user will have the sum of the perturbation, integrity and scrambling keys of the nodes in the query tree Q_j. Nodes then encrypt data using the PIP scheme and send the ciphertext to their parent. The encrypted data gets routed along the edges of Q_j, being aggregated at each intermediate node. Finally, the encrypted aggregated data is communicated to the user by the gateway node. The user can then use its data aggregation keys to retrieve the data.

8. DISCUSSION

In a sensor cloud, there can be multiple users who request the same attribute based data. In previous approaches, no distinction was made between various users if their query was same. In a sensor cloud though, it is essential to keep track of each user's usage so that they can be billed accordingly. We, therefore, augment the tree with an identity element which makes each tree unique even though the attribute based query may be the same. In the access control secret key generation phase, the secret key is generated using the private key component of this identity element along with the other attributes in the access tree. Since the sensors would need the public component of this identity to be able to encrypt data for it, the *SCA* gives the public component of the identity T_j to the user, which is passed on to the sensors along with the query. The sensors use T_j as the unique identifier for the user's query for billing purposes.

In the data aggregation key generation phase, the sensors generate a partial session key and encrypt it using KP-ABE [12]. We encrypt the partial session key with Paillier encryption [13] to make use of its homomorphic property. In equation 1, we see that for two sensors SN_1 and SN_2, $E'_1 = p_1 Y^{s_1}$ and $E'_2 = p_2 Y^{s_2}$ respectively. We need to aggregate both the random numbers s_1 and s_2 and the partial session keys p_1 and p_2. Since p_i is a Paillier encrypted ciphertext of the partial session key, we can simply multiply E'_1 and E'_2 to obtain $E' = p_1 p_2 Y^{s_1+s_2}$. This adds the random numbers s_1 and s_2 as the exponents of Y and multiplies the Paillier ciphertexts p_1 and p_2. From the homomorphic property of Paillier encryption, we know that when the ciphertexts are multiplied, the plaintext is added up. Thus, we are able to aggregate both the random numbers and the partial session keys.

It needs to be noted that we are not using Paillier encryption for security. The security is provided by the Decisional Bilinear Diffie-Hellman (DBDH) Assumption, we have used Paillier encryption for its ability to sum the plaintexts when ciphertexts are multiplied. This introduces the overhead of an extra Paillier encryption during key establishment. In section 12, we discuss how to reduce this overhead.

9. REVOCATION OF USERS

In FDAC [4], a user revocation is handled by updating the master key secret y embedded in the user secret key SK. To accomplish this, the authority SCA includes an additional updatable component in the user secret key SK and a corresponding component in the public key PK. To revoke a user, a new master secret y' and the corresponding public component $Y' = e(P,P)^{y'}$ is generated. The new public component Y' is broadcasted to the sensor nodes, while the differential between the new secret components is broadcasted to all users except the one whose access is to be revoked. The selective broadcasting is done by using ciphertext based attribute based encryption (CP-ABE) [10].

Our scheme also uses an additional secret key and corresponding public key component for user revocation. In section 7.2, it was explained how the access tree is augmented with an ID node and a unique secret key component t_j and public key component T_j is computed for each user $\mathcal{U}_j$. The public component T_j is used by the sensors when encrypting data for the user $\mathcal{U}_j$. Each sensor generates s_iT_j for the user $\mathcal{U}_j$ during the data aggregation key generation. s_iT_j is then used in key establishment. This added component hence serves two purposes. First it binds the key to the user. The data aggregation key would be established only if the user possesses the corresponding secret key component t_j. This can be used to keep track of individual data usage of the users and is important in commercial systems like sensor clouds. Second, it helps in easy revocation of users. To revoke a user, the SCA has to simply broadcast $\mathcal{U}_j$ in the sensor cloud. Once the sensor nodes receive $\mathcal{U}_j$, they simply stop sending s_iT_j in further data aggregation key generations. This makes sure that the revoked user $\mathcal{U}_j$ would not be able to generate data aggregation keys required to decrypt sensor data.

Our user revocation scheme only incurs one broadcast in the sensor network and the maintenance of a list of revoked users. On the other hand the revocation scheme in FDAC

[4] requires one broadcast in the sensor network and one selective broadcast to the users. The selective broadcast is done using CP-ABE [10] which requires computational overhead at the *SCA* and for each user $\mathcal{U}_j$ for every revocation. Moreover, the users may be mobile and they may be online as well as offline. If the selective broadcast fails to reach a user, that user will not be able to receive data from the sensor network in the future.

10. MODIFYING ACCESS AT RUNTIME

Data generated by a sensor network may hold different degrees of importance at different times. Sensor data which under normal circumstances is allowed to be accessed by everyone, may become more important under special circumstances and may require special privileges to be accessed. Two examples below illustrate the point.

Example 1: A network owner has deployed A WSN in a building which keeps track of the number of people coming in and out of the building. This information is public and may be accessed by anyone. In case of an event however, the network owner may escalate the authorization level of this information so that only authorized personnel may access the information about the event.

Example 2: A network owner has deployed seismic sensors in a wide field. This data is public and may be accessed to study the seismic activity of the area. In case the sensors sense a sudden short high intensity shock wave, it may by an indication of an explosion or an earth quake. The sensors themselves escalate the authorization level in this case, to avoid panic in public.

10.1. ENCRYPTION SCHEME FOR MODIFYING ACCESS AT RUNTIME

We build the network owner's or sensor's control on the access of data by introducing authorization in the key establishment phase. We specify authorization by authorization levels which are assumed to be hierarchical in nature. In our scheme authorization level l is higher than authorization level $l+1$, such that $AL_0 > AL_1 > ... > AL_{n-1}$. A user which has an authorization level of AL_l can access data which belongs to any authorization level AL_m, where $m \geq l$ but cannot access data which belongs to authorization levels, where $m < l$. We model the hierarchical authorization levels by a one way key chain.

$$K_0 \rightarrow K_1 \rightarrow K_2 \rightarrow ... \rightarrow K_{n-1}, \text{ where } K_l = h(K_{l-1})$$

In the key chain, K_0 is the root key and all other keys can be derived by repeated application of the hash function $h(.)$ on the root key. A key can derive other keys in the direction of the hash chain but not the opposite direction. The key chain is such, that data meant for an authorization level AL_l can only be decrypted if the user has a corresponding key K_m, where $m \leq l$. The root key K_0, corresponds to the absolute authorization over all data.

To allow run time modification of access the scheme is modified as follows. Additional public key components $h_l = Y^{K_l}$ are created for each of the authorization keys K_l, such that the public key becomes,

$$PK = (G_1, P, T_1, T_2, ..., T_{|\mathcal{A}|}, Y, h_l \text{ for every key } K_l)$$

The secret key for each authorization level are the key K_is which are only given to authorized people for that authorization level. To encrypt a message for a certain authorization level l, the public key component of the authorization level is raised to the random number s_i and multiplied with the partial session key. The ciphertext generated by each node in equation 1 then becomes.

$$E = (E' = C_{p_i} h_l^{s_i}, \{E_k = s_i T_k\}, k \in \gamma) \tag{6}$$

The aggregate ciphertext at the gateway node would be

$$E_{Q_j} = (E' = CY^{sK_l}, \{E_k = sT_k\}, k \in \gamma) \tag{7}$$

As discussed before, the user $\mathcal{U}_j$ can use it's secret key SK_j on E_{Q_j} to obtain F_z for the root node, which is Y^s. If the user has the correct authorization key, it can calculate $(Y^s)^{K_i} = Y^{sK_i}$, which can be used to retrieve C from 7. Once C is obtained, the user can proceed to generate the data aggregation keys.

10.2. PROTOCOL FOR MODIFYING ACCESS AT RUNTIME

Once the symmetric keys for data aggregation have been established using the session key, the data collection becomes operational. To modify access to the data, after the data collection is operational, nodes follow the below protocol.

We assume nodes $SN_1,...,SN_N$ are collecting data and encrypting it securely using keys derived from partial session keys $p_1,...,p_N$. The authorization level is AL_l and the aggregate session key is P. We assume that some event $\mathcal{E}$ occurs and in response to $\mathcal{E}$, nodes $SN_k,...,SN_{k+k'}$ escalate their authorization level to $AL_{l'}, l' < l$. The nodes $SN_k,...,SN_{k+k'}$ now generate new partial session keys p', which will be encrypted with the escalated authorization level key $h_{l'}$ as in equation 6. This ciphertext is sent to the user along the query tree $\mathcal{Q}$. To however, enable the user to keep decrypting the data it receives from the rest of the nodes, the sensor nodes also send their previous partial key p, encrypted with the old authorization level. Both the new and the old partial keys of the nodes $N_k,...,N_{k+k'}$ which have escalated their authorization level are aggregated and sent to the user. The user decrypts the old aggregate partial keys $\sum p$ and generates the new session key as $P - \sum p$. This new session key is then used to derive the new scrambling, integrity and perturbation keys for continuing to receive aggregate data from the rest of the nodes at the old authorization level. $\sum p'$ can only be decrypted by the user if it has the key for the escalated authorization level. Thus, if the user does not have the required authorization to decrypt data from a set of nodes, it continues to only receive data from the rest of the nodes. If the user has the private key for the new authorization level, then it can decrypt $\sum p'$ and generate new scrambling, integrity and the perturbation keys. The nodes after escalating the access level, now encrypt their data with the new partial keys. A tag tg is attached to this encrypted data to indicate the new authorization level. Encrypted data which has the same tag is aggregated. Thus the user now receives two aggregates of data, one encrypted with keys derived from the new session key and one with keys derived from the old session key. The user can decrypt one or both the aggregates depending upon the authorization it has.

11. SECURITY ANALYSIS

As discussed in our adversary model, the adversary (which is also a malicious user) has the following three goals.

- To try to get access to the data, for which it does not have the secret key.
- To try to access data for which the user does not have authorization.
- To tamper with the keys and data meant for other users, so as to disrupt the protocol.

To achieve these goals, the adversary can collude with other users and compromise some sensor nodes. We show in this section that our scheme prevents the adversary from accomplishing these goals in the presence of user collusion and node compromise.

Each node in our scheme encrypts a randomly generated partial session key using KP-ABE. KP-ABE is secure under the Decisional Bilinear Diffie-Hellman (DBDH) Assumption (The security proof can be found in [12]). With KP-ABE, even though the users collude, they cannot decrypt data which has not been encrypted for their individual secret keys. In equation 7, the session key is encrypted as $Ce(P,P)^{sK_l}$. Assuming, the user has the correct authorization key K_l, in order to recover C, the user will have to calculate $e(P,P)^s$, which it cannot calculate unless it has the correct secret key. The adversary may also compromise sensor nodes. Because each sensor node generates an encrypted partial session key C_{p_i} individually, compromising sensor nodes, does not give the the adversary any advantage too.

Authorization in our scheme is provided in the form of private keys K_l, for authorization level AL_l. Since the authorization keys are in the form of a one way hash chain, a lower authorization level's key can be derived from a higher authorization key but not vice versa. That is, if the underlying hash function is secure. For a particular authorization level AL_l, the sensor nodes encrypt the session key as $Ce(P,P)^{sK_l}$. Nodes with the required

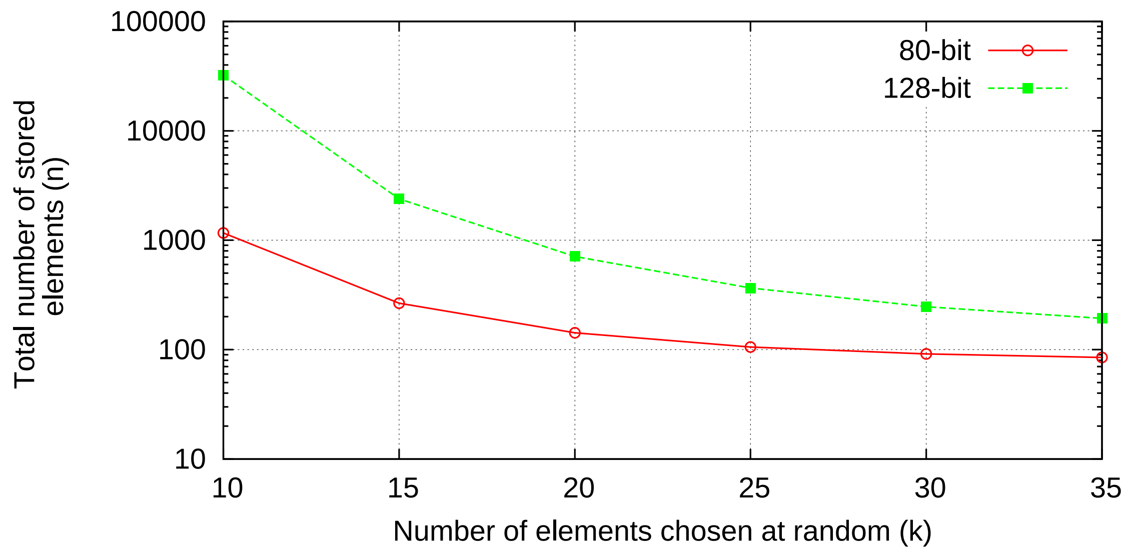

Figure 11.1. Relationship between number of elements chosen (k) and stored (n)

authorization level can calculate $e(P,P)^s$ and then $(e(P,P)^s)^{K_l}$ to compute C and hence the session key S. User who does not have the correct authorization will not be able to compute $(e(P,P)^s)^{K_l}$ and hence cannot derive the correct session key to access data.

An adversary can compromise sensor nodes, which can corrupt the ciphertext, $Ce(P,P)^{sK_l}$ to disrupt the key establishment process. If the user receives corrupted ciphertext, the key generation process will still go through and the user will generate incorrect data aggregation keys. This is where PIP algorithm's integrity preserving nature comes in. Since the user has derived incorrect data aggregation keys, PIP will raise an alarm when the user tries to decrypt sensor data using the incorrect keys. When this happens the user can inform the *SCA*, which will take appropriate steps.

12. PERFORMANCE ANALYSIS

We have implemented our scheme on Mica2 mote platform using TnyOS2.x. The Mica2 mote platform is based on the Atmel ATmega 128L 8-bit microcontroller, which has 4 kB of RAM and a clock speed of 8MHz. The external flash storage is 512KB. To implement bilinear pairing, we have used the TinyPairing library of [15]. TinyPairing is an efficient and lightweight pairing based library, which uses elliptic curves to implement the pairing. The underlying finite field is $\mathbb{F}_{3^{97}}$, which results in fast multiplication and cubing in the extension field. A 160-bit elliptic curve (EC) is chosen for the implementation. Point compression algorithms are used to compress the EC representation to 20 bytes per point. The extension field size is 156 bytes.

We begin this section by discussing an optimization to the implementation of our scheme which was presented in section 7. During the data aggregation key generation phase, each sensor node generates a random partial session key and encrypts it using Paillier encryption before multiplying it with $e(P,P)^{s_i}$. This process involves one Paillier encryption operation which is expensive in terms of energy. To reduce the energy consumption on the sensor nodes, we take the following approach.

During the system setup phase the *SCA* generates n Paillier encrypted elements in G_T and pre-deploys the elements on the sensors. In the data aggregation key generation phase, the sensors then randomly choose $k \leq n$ elements from the stored elements and multiply them together to generate one random Paillier encrypted partial key. The number of unique keys which can be generated with this method are dependent on the values of n and k and are given by,

$$\text{Number of unique keys} = \frac{n!}{k!(n-k)!}$$

The relationship between n and k for 80 and 128 bit symmetric key security is shown in Figure 11.1. A smaller value of k implies less number of extension field multiplications,

Table 12.1. Execution time and energy consumption on Mica2 motes

Operations	**Exec. Time (ms)**	**Energy Con. (mJ)**
Scalar Multiplication	2551	61.224
Ext. Field Multiplication	89	2.136
Ext. Field Cubing	3	0.072
Ext. Field Exponentiation	689	16.536
Paillier Key Generation	453	10.872
Point Addition	116	2.784
Optimised Scalar Mult.	857	20.568

however as can be seen in the figure, the total number of stored elements n needs to be very large. For larger k values, the space complexity decreases. Thus, there is a tradeoff between computation and space complexities. The system designer can choose appropriate parameters depending upon which motes are available. In our implementation we have chosen $k = 15$ and $n = 266$ for 80-bit symmetric key security level. We further make use of the efficient extension field cubing and multiplication algorithms of [15] to implement the generation of random Paillier encrypted partial key, the execution time of which comes out to be 453 ms. We have also implemented the time consuming scalar multiplication operation on EC points in a similar way and have traded space complexity for time complexity. Our implementation of the optimized scalar multiplication takes 857 ms compared to 2551 ms if no optimization is used. The execution times and energy consumption of various operations needed for the scheme on Mica2 motes is shown in the Table 12.1. To measure energy consumption we used the formula $E = V * I * t$, where V is the supplied voltage, I is the current drawn in active mode and t is the execution time of the operation. Using a low power mote such as Mica2 results in higher execution times but the energy consumption on these motes is considerably lower than high-end motes such as the iMote2.

A comparison of the computation involved in our scheme and the schemes in [4] and [5] is given in Table 12.2. Our scheme requires an additional Paillier key generation operation. The overhead of this operation, however, compared to $|A_j| + 1$ scalar operations,

Table 12.2. Comparison of computation complexity

Scheme	**Scalar Mul.**	**Extension Field Mul.**	**Extension Field Exp.**	**Paillier Key-Gen**
Yu et. al	$\|A_j\|+1$	1	1	0
Ruj et. al	$\|A_j\|+1$	1	1	0
Ours	$\|A_j\|+1$	1	1	1

Table 12.3. Comparison of communication complexity

Scheme	**Access Control**	**User Revocation**
Yu et. al	$(\|A_j\|+1)\mathbb{G}_1+1\mathbb{G}_T$	$1\mathbb{G}_T+1\mathbb{G}_1$
Ruj et. al	$(\|A_j\|)\mathbb{G}_1+1\mathbb{G}_2+1\mathbb{G}_T$	$\|A_j\|\mathbb{G}_1$
Ours	$(\|A_j\|+1)\mathbb{G}_1+1\mathbb{G}_T$	$1\|identifier\|$

where $|A_j|+1$ is the number of attributes in the user access tree (including the ID attribute), is considerably low. A comparison of the communication overhead of the three schemes is shown in Table 12.3. The overhead for establishing the access control is the same in all three of the schemes. In the user revocation process, Yu et. al's scheme in [4] requires two broadcasts; a broadcast of one extension field element in the sensor network and a broadcast of one EC point to the users. Ruj et. al's scheme in [5] the broadcast of $|A_j|$ EC points (one for each attribute), while our scheme only needs to broadcast an identifier element to the sensors.

13. CONCLUSION AND FUTURE WORK

In this paper, we have presented a user access control scheme for sensor clouds. Our access control scheme considers large sensor networks where sensor nodes collaborate and aggregate data in the network to save energy and bandwidth. The scheme also provides the opportunity to the network owners to modify the access control policies at run time and provides an efficient revocation strategy. Finally, our scheme is able to distinguish between different users with the same query, important for sensor cloud applications. We show that the computation complexity of our scheme is marginally higher than other approaches, while the communication complexity remains the same. Improving the computation complexity further is left as a future work. We conclude with the observation that although the scheme is designed for sensor networks, it can also be adopted in other settings such as wired and mobile networks, where the goal is to provide access control in a collaborative and data aggregation scenario.

14. BIBLIOGRAPHY

[1] S. Madria, V. Kumar, and R. Dalvi, "Sensor cloud: A cloud of virtual sensors," *IEEE Software*, March 2014, to appear.

[2] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure-physical sensor management with virtualized sensors on cloud computing," in *Network-Based Information Systems (NBiS), 2010 13th International Conference on*. IEEE, 2010, pp. 1–8.

[3] N. Poolsappasit, V. Kumar, S. Madria, and S. Chellappan, "Challenges in secure sensor-cloud computing," *Secure Data Management*, pp. 70–84, 2011.

[4] S. Yu, K. Ren, and W. Lou, "Fdac: Toward fine-grained distributed data access control in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 4, pp. 673–686, 2011.

[5] S. Ruj, A. Nayak, and I. Stojmenovic, "Distributed fine-grained access control in wireless sensor networks," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, 2011, pp. 352–362.

[6] D. Liu, "Efficient and distributed access control for sensor networks," in *Distributed Computing in Sensor Systems*, ser. Lecture Notes in Computer Science, J. Aspnes, C. Scheideler, A. Arora, and S. Madden, Eds. Springer Berlin Heidelberg, 2007, vol. 4549, pp. 21–35.

[7] J. Maerien, S. Michiels, C. Huygens, D. Hughes, and W. Joosen, "Access control in multi-party wireless sensor networks," in *Wireless Sensor Networks*, ser. Lecture Notes in Computer Science, P. Demeester, I. Moerman, and A. Terzis, Eds. Springer Berlin Heidelberg, 2013, vol. 7772, pp. 34–49.

[8] H. Wang and Q. Li, "Achieving distributed user access control in sensor networks," *Ad Hoc Networks*, vol. 10, no. 3, pp. 272 – 283, 2012.

[9] G. Bianchi, A. T. Capossele, C. Petrioli, and D. Spenza, "Agree: exploiting energy harvesting to support data-centric access control in {WSNs}," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2625 – 2636, 2013.

[10] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, 2007. SP '07. IEEE Symposium on*, 2007, pp. 321–334.

[11] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology EUROCRYPT 2005*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, vol. 3494, pp. 457–473.

[12] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-based encryption for fine-grained access control of encrypted data,” in *Proceedings of the 13th ACM conference on Computer and communications security*, ser. CCS ’06. ACM, 2006, pp. 89–98.

[13] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *Advances in Cryptology EUROCRYPT 99*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1999, vol. 1592, pp. 223–238.

[14] V. Kumar and S. Madria, “Pip: Privacy and integrity preserving data aggregation in wireless sensor networks,” in *Reliable Distributed Systems (SRDS), 2013 IEEE 32nd International Symposium on*, 2013, pp. 10–19.

[15] X. Xiong, D. S. Wong, and X. Deng, “Tinypairing: A fast and lightweight pairing-based cryptographic library for wireless sensor networks,” in *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, 2010, pp. 1–6.

IV. EFFICIENT AND SECURE CODE DISSEMINATION IN SENSOR CLOUDS

Vimal Kumar∗, Sanjay K Madria∗,

∗ Department of Computer Science,

Missouri University of Science and Technology, Rolla, Missouri 65401

In this paper, we present an efficient and secure code dissemination technique aimed at sensor clouds. Previous code dissemination techniques were geared toward traditional wireless sensor networks. They did not take into account, the dynamic nature of a sensor cloud, where the applications running on the motes may not just be updated but changed completely in successive code disseminations. The technique presented in this paper is based upon the observation that a large amount of code is common between applications in wireless sensor networks. Our technique first discovers the code common across various wireless sensor applications. It then distributes this code in the form of functions *a priori* into the network. During code dissemination, these common functions are picked up by the sensors from the network. Only a part of the code needs to be transmitted from the base station. This reduces the overall transmitted code and hence the energy consumption. Since, security is important in sensor clouds, we further present a security scheme based on proxy re-encryption to provide confidentiality and integrity of the code. We have implemented our scheme using two different proxy re-encryption algorithms, on Mica2 and TelosB mote platforms to measure its energy consumption. We have also evaluated our scheme in terms of disseminated code size and bandwidth usage to illustrate its efficiency compared to a popular secure code dissemination technique, Seluge.

1. INTRODUCTION

The code running on the sensors may need to be either updated or changed completely several times throughout a Wireless Sensor Network's (WSN's) lifetime. WSNs are typically very large in size, which makes manual updates to each sensor node impractical. A more feasible alternative involves disseminating the code wirelessly in the network. Sensor nodes receive the code packet-by-packet and then rebuild the code image, once all of the code has been received. In wireless code dissemination, code images are communicated via the wireless channel which is inherently in-secure and prone to attacks from adversaries. A secure code dissemination technique enables the code dissemination to be confidential and also protected against malicious code injection attacks.

A large amount of work has been done to reduce the amount of code to be transmitted from the base station to different sensor nodes [1], [2], [3], [4],[5]. These efforts, however, have been focused on traditional WSNs, that support only one application. In such networks, the code updates happen infrequently. Most often, these updates are minor so most of the code remains unchanged. Several schemes, such as [1],[3] and [5] created a difference script between the old and the updated code. The base station disseminates the script rather than the entire code. Doing so not only reduces the number of packets but also saves energy along the forwarding nodes. In a sensor cloud, however, clusters of nodes are provisioned dynamically to the user to support several applications on demand, [6], [7]. Dynamic provisioning implies that the code on the wireless sensors is changed entirely as a new application is installed. The difference script mechanism cannot be applied in this scenario because the script itself would be the size of the code. Thus, there exists a need for an efficient code dissemination scheme that is well-suited for a sensor cloud scenario. Efficiency of code dissemination is an especially important issue in sensor clouds because the frequency of code change is high to support different applications.

High frequency of code change implies that the sensors spend a good deal of their energy on forwarding and installing new code. Any reduction in the amount of total code transferred therefore gets multiplied by high frequency resulting in great reduction in energy consumption. Moreover, clusters of sensors in a sensor cloud are dynamically provisioned to users, which means that at any given point in time, various clusters can be working for various users. In such a scenario, the security of the code (in terms of confidentiality and integrity) also becomes very important. Code disseminated from the base station will inevitably be forwarded by many sensors on its way to its destination cluster. Code confidentiality is thus a critical pre-requisite since the code may be carrying keying material which must be protected against eavesdropping. Another pre-requisite is the integrity of code, which will make sure that an adversary has not injected malicious code packets during the code dissemination process. To summarize, there is a need for an efficient code dissemination scheme, which is well suited to a dynamically provisioned sensor cloud scenario. The scheme needs to minimize the number of code packets transmitted and should provide confidentiality and protect the integrity of the disseminated code. In this paper, we describe our efficient and secure code dissemination scheme which addresses the above concerns. Our contributions in this paper are as follows:

- A code dissemination algorithm which reduces the total number of packets sent from the base station to a cluster of sensors in a sensor cloud scenario.
- A security mechanism which provides confidentiality and integrity of code packets while they are disseminated in a sensor cloud.

2. RELATED WORK

Deluge [8] provided an efficient way to reprogram motes wirelessly. It divides the code image into fixed size pages and then divides each page into packets with a size that is network dependent. Because Deluge [8] was created for traditional WSNs, it advertises new code images using an epidemic protocol. Nodes can then request for individual pages and new code images by listening to the advertisement packets. Although Deluge [8] tried to reduce the wireless traffic, it did not take into account the similarities between various code updates.

Reijers et al. [5] proposed an approach that addressed incremental code updates. This approach took a UNIX **diff** like approach for determining the difference between two versions of the same code. They introduced commands such as *insert, copy, repair, repair dbl* and *patch list* to generate an edit script. Instead of wirelessly sending the complete new code, the base station would only transmit the edit script. The wireless sensors would then transform their code image according to the edit script to generate the new version of the code. The authors, however, only discussed the encoding mechanism of the code image without any code distribution algorithm. Moreover, the edit script was platform dependent. A platform independent incremental code update algorithm was described in [3]. This algorithm divided a program image into small, fixed-size blocks and calculated the hash of each block. The same was done for the new code image and a difference script was created by comparing the hashes of the code images. The difference script consisted of copy and download commands where copy meant the wireless sensor could just copy the block from the old code image and download meant the block contained changes and thus had to be downloaded from the base station.

Approaches in [3] and [5], however, work particularly for small code changes. Any code change that produces an address shift results in an extremely inefficient script. To overcome this, [4] introduced slop regions around functions. Functions are allowed to grow into the slop regions. Thus, small changes in code do not produce address shifts. If the function grows bigger than the allowed slop region, it is moved to an area with a bigger slop region and linked to its previous location. This approach however, wastes a large amount of space on the sensors. Additionally, the efficiency of code dissemination depends on the amount of memory available to be sacrificed. QDiff [1] presented an optimized patch creation technique. The size of the patch created in QDiff [1] is small compared to other schemes. The algorithm works on the ELF file level and hence, is platform independent. QDiff [1] uses slop region to maintain similarity between two versions of the code. If no slop region exists, the new code is moved to the end of the file. A high level of similarity between the codes at the ELF file level ensures a small patch size. Moreover, the patch can be directly applied in the RAM, eliminating the need for a reboot, thus saving a large amount of energy.

All of these approaches targeted the traditional wireless sensor network model in which the changes between successive versions of the application code are small. On the contrary, in a sensor cloud, a code update implies that the entire application must be changed. As a result, the application code needs to be completely updated. Code dissemination in a sensor cloud like scenario was first discussed in [9]. The authors discussed a code dissemination algorithm for multi-application WSNs, where various groups of sensors support different WSN applications. This algorithm is based on the idea that WSN applications share a lot of common code. This common code can be disseminated from other sensors in the network instead of disseminating everything from the base station. The authors presented the idea and demonstrated the effectiveness through simulations. They did not, however, offer implementation specific details, such as the handling of code shifts and the introduction of new global variables. Instead, the focus was on adaptive buffer management for such a code dissemination algorithm. Our algorithm exploits this same idea. Many applications in WSNs may share a good chunk of code. Unlike [9] however, we provide a scheme of how this idea can be implemented in a secure fashion in the context of a sensor cloud.

Another important issue in a sensor cloud scenario is security. Seluge [10] was one of the early attempts to build a secure code dissemination algorithm. Seluge [10] aimed to tackle the code image integrity and various DoS attacks on Deluge [8]. For each code image, it hashes the code packets of the last page and concatenates the hashes with the packets from the previous page. This process is performed recursively until all of the packets from the first page are hashed. The hashes from the first page are then used to create a hash tree, the root of which is signed by the base station's private key. Page 0 is then constructed which consists of the information needed to verify the root hash. In the end, a signature packet is constructed which consists of the root hash, the meta data about the code, and the signature over the root hash. Secure Deluge [11] also applies similar techniques for secure code dissemination. Neither [10] nor [11], however, offer confidentiality of code. Confidentiality of code has been discussed in [12]. In [12] all packets are considered to be in a sequence. The hash of a packet is generated in the same way as in [10]. The last packet, however, is concatenated with an L-byte nonce. The first L-bytes from the hash of a packet are also used as the key for encrypting the packet. The whole sequence of packets is thus encrypted. The hash of the first packet is then used to construct a cipher puzzle signed by the base station. The eavesdropper in [12] is an outsider. The confidentiality of the code is protected against such an adversary, while the nodes in the network are easily able to decrypt the encrypted code. In a sensor cloud scenario, on the other hand, the adversary is inside the network since different clusters of sensors may belong to different users. The confidentiality of the code must be protected against such sensors belonging to other users. This problem is exacerbated by the fact that we also want to store commonly used code on the sensors in the sensor cloud. We propose to use proxy re-encryption [13], [14] to solve this problem. Proxy re-encryption is discussed in section 5.1 and we present our algorithm in section 7.3.

3. SYSTEM MODEL AND ASSUMPTIONS

The sensor cloud consists of a large number of wireless sensors. We consider that a clustering algorithm such as [15] has been run and the sensors have been grouped into clusters. These sensors are provisioned to the users in terms of clusters. At any given point in time, the sensor cloud may have many users, each holding one or more clusters of sensors. Such a model has been discussed in [6][7] and is visualized in Figure 3.1. Sensors in a cluster provisioned to a particular user collect data for that user. They may, however, act as forwarding nodes for other clusters, for transferring data and code. In previous models [1], [2], [3], [4],[5], the code updation ocurred on the scale of the WSN. In our model, on the other hand, the code change occurs at the cluster scale. The code is updated when either individual users install new applications in their sensor clusters or a cluster is provisioned to a new user and a new application is installed. We assume that a routing structure is in place, using which the base station can route the code to any particular cluster. Each cluster has a cluster key (CK), known to the cluster members and the base station. The adversary in our model lies inside the network. The sensors in the clusters, provisioned to other users are assumed to be curious and may want to eavesdrop on the code being transferred. The sensors storing common code, may want to inject malicious code by modifying the code they store and making other sensors accept this modified code.

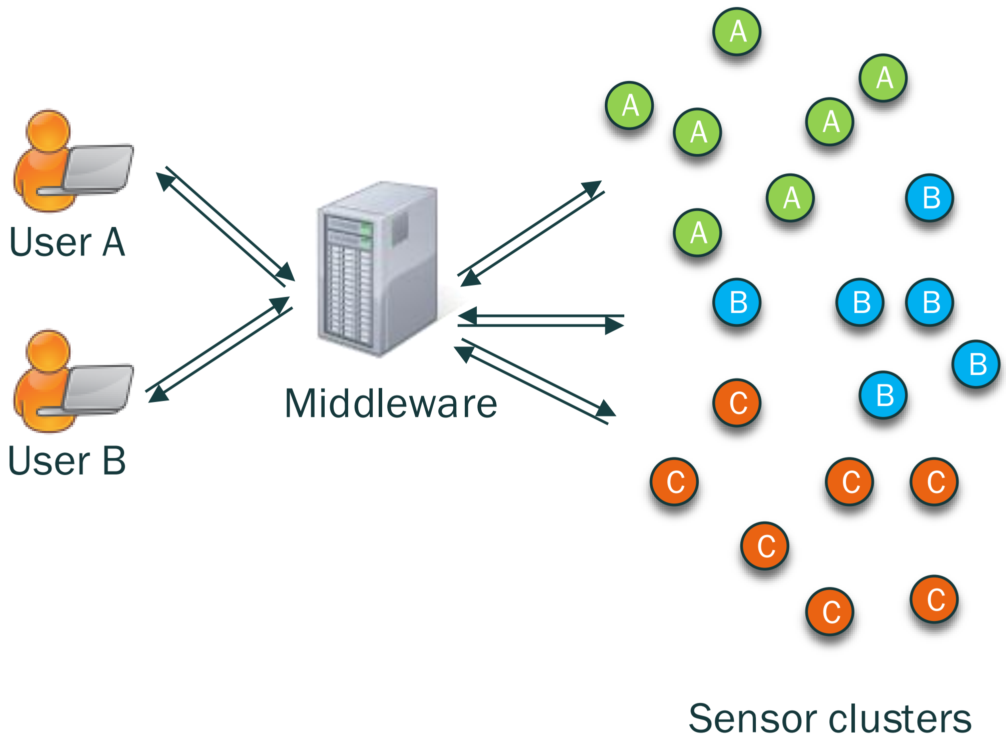

Figure 3.1. System Model

4. PROPOSED APPROACH

Our approach is based on the observation that the executable code which runs on the wireless sensors consists entirely of subroutines and objects. These subroutines and objects have a one-to-one correspondence with the functions and global variables in a high level language such as nesC in TinyOS. Many of these functions and global variables are common across a number of wireless sensor applications. In a sensor cloud environment which has a number of WSN applications running simultaneously in various clusters of sensors, many applications may share parts of the same code. Some applications may have the same security code, while others may share the same routing subroutines. Still others may share the sensing code, and so on. All applications also share the same operating system code. The basic idea, therefore, is to first identify the commonly used functions and global variables across all of the given applications. These functions and global variables are then distributed throughout the network so that every sensor node probabilistically stores a few of them. When the code on a cluster of sensors needs to be changed, the base station first determines which of the functions and objects can the sensors request from the other sensors in the network. Only the part of code not already present in the network is sent from the base station. The remaining code is requested from nearby sensors. Moreover, since the provisioning of sensors in a sensor clouds is always in the form of clusters, the security related tasks such as decryption, authentication and verification can be distributed among the sensors to reduce energy consumption. The security challenges in the scheme are enumerated as follows.

- Because the functions are stored on sensors, a request for a specific function will leak information about the code. To avoid such information leaks, the functions must be stored encrypted on the sensors. Encryption, however, presents a problem because the encryption keys will need to be revealed to the requesting sensors. Once the requesting sensors know the encryption keys, they can send spurious requests and retrieve all of the encrypted code.

- When sensors reply to function requests with encrypted functions, they need to make sure that the functions have not been tampered with. This authentication needs to be done as soon as the functions are received to thwart energy draining attacks.

5. PRELIMINARIES

In this section we discuss the concepts of Proxy re-encryption and Bloom filter and provide a list of the symbols used throughout the paper in Table 5.1.

5.1. PROXY RE-ENCRYPTION

Proxy re-encryption allows third parties called proxies to re-encrypt a ciphertext generated using a secret key so that it can be decrypted using a different public key. Proxy re-encryption (PRE) was first presented in [14]. This scheme known as the BBS scheme is a very simple PRE scheme based on Elgamal encryption. We present the elliptic curve version of the BBS scheme in Algorithm 9.

The concept of symmetric key proxy re-encryption (SRE) has been explored by Syalim et. al in [13]. The symmetric proxy re-encryption scheme of [13] makes use of an All Or Nothing Transform (AONT) along with a symmetric cipher. AONT has the properties that it's output is pseudo-random, and the transformation of output back to input requires all the output blocks be in their correct positions. These properties are used along with a weak permutation cipher to develop a secure symmetric proxy re-encryption scheme. For algorithmic details, reader can refer to [13].

The EC-BBS and the SRE [13] schemes presented here have the often undesired property of bi-directionality, where it is relatively easy to derive $RK_{j\rightarrow i}$ from $RK_{i\rightarrow j}$. As we will see, however, in our algorithm, the knowledge of $RK_{j\rightarrow i}$ is not useful for an attacker in gaining any information about the encrypted code. While any PRE scheme can be used in our algorithm, we consider EC-BBS and SRE because of their energy efficiency. We use EC-BBS to present the algorithm in section 7. To use SRE, a single symmetric key can be used to replace the secret and the public keys in the algorithm.

Algorithm 9 EC-BBS Algorithm

```
Require: Elliptic curve parameters, base point $T$, message $m$.
    KeyGen
 1: Generate a random integer $a$. The secret and public key pair then is $(a, a*T)$.
    Encryption
 2: Map the message $m$ to an elliptic curve point $M$ using a mapping function.
 3: Generate a random integer $r$.
 4: Calculate $C_1 = M + r*T$ and $C_2 = a*r*T$.
 5: $(C_1, C_2) = (M + r*T, a*r*T)$ is the ciphertext.
    Decryption
 6: Calculate $M' = C_1 - (a^{-1} * C_2)$.
 7: Reverse map point $M'$ to message $m'$.
    Re-encryption KeyGen
 8: For nodes $A$ and $B$, with the secret and public key pair $(a, a*T)$ and $(b, b*T)$, the
    re-encryption key $RK_{A \to B}$ can be generated as $b/a$
    Re-encryption
 9: Node A's ciphertext: $C_a = (M + r*T, a*r*T)$.
10: Calculate $C_b = (M + r*T, a*r*T*RK_{A \to B})$.
11: Node B's ciphertext: $C_b = (M + r*T, b*r*T)$.
```

5.2. BLOOM FILTER

Bloom filter is a versatile space efficient probabilistic data structure used primarily to verify set membership [16]. The Bloom Filter consists of an array of bits, initialized to 0 and k different hash functions. To add an element to the Bloom Filter, the k hash functions are used to hash the element to obtain k array positions. These array positions are then set to 1. To verify an element's membership, it is hashed using the k hash functions and each resulting array position is checked. If any of the bits at these positions are found to be 0, the element is not a member. If all the bits are set to 1, it is a member. In this work we use the Bloom Filter as a means of verification. The usage of Bloom Filter in our algorithm is discussed in section 7.3.

Table 5.1. Symbols used in the algorithm

Symbol	**Description**
FID	Unique identifier for each fuction
CFL	List of common functions
A_i	Authentication key
$h()$	Hash function
$g()$	A pseudo random function (PRF)
$f()$	A function in the code (Not a mathematical function)
$f_j()$	Function with identifier j
Q_i	EC-BBS public key
K_i	EC-BBS secret key corresponding to Q_i
$E_{Q_0}(f_j())$	Function $f_j()$, with FID j encrypted using key Q_0
$RK_{i\rightarrow j}$	Re-encryption key to re-encrypt a ciphertext encrypted with key Q_i to one encrypted with key Q_j
RK_i	Re-encryption key to re-encrypt a ciphertext encrypted with key Q_0 to one encrypted with key Q_i
CK	A cluster's group key
n	Nonce
S_n	Sesion key to encrypt code packets for a session created using the nonce n
P	Number of pages of code
N	Number of packets in each page
SN	Denotes the sensor network
SN_i	i^{-th} node in the sensor network

6. DETECTING COMMON FUNCTIONS

We assume that the base station has a tentative list of sample applications that may be used in the sensor cloud in future. It must be noted that not all the applications are needed to be known beforehand. Rather a small sample size would be sufficient to detect the common functions across the applications. We follow a procedure similar to Qdiff [1] to detect common functions in the applications. The ELF files were dumped using the *msp430-objdump* utility for TelosB and the *avr-objdump* utility for Mica2 platforms. Bauhaus-toolkit [17] was then used to compare the C files generated from the nesC code of the various applications. Bauhaus-toolkit [17] has a clone detection utility that can detect Type I and Type II clones in different applications, at the source code level. Type I clones are fragments of code which are exactly identical while Type II clones are copies which are structurally identical but may have the identifiers changed. At this point of time, we only consider Type I clones. In future, we will develop techniques to take Type II clones into account as well. The Type I clones found in the C code by the clone detection utility of the Bauhaus toolkit [17], can be further divided into two different types at the ELF file level.

Definition 1. *We define **Type 1a** clones as the true Type 1 clones, where the two codes are exactly the same and they may or may not have been shifted in memory.*

Definition 2. *Some Type 1 clones may also contain calls to functions and global variables which have shifted in memory. We define such clones, which have calls to functions and refer global variables, which have shifted in memory as **Type 1b** clones.*

Once the common functions are detected the base station rearranges the application code in the ELF file in a way which facilitates further use of common functions. The reordering of the functions is done so that the placement of the common functions is consistent across all the applications. Beginning with the Type 1a functions, the base station places the common functions at the end of the *.text* section. This is in contrast to QDiff [1], which places the new code at the end of the *.text* section. The code will now grow towards the beginning of the file. After all the Type 1a functions are moved, Type 1b functions are moved in the same manner. This results in the reordering of other functions and changes in function references. The base station then fixes the changes in function references throughout the code. We place the common functions at the end of the *.text* section and not at the beginning to avoid situations in which the base station receives applications with large *.data* and *.bss* sections. This will move the beginning of the *.text* section further down. In such a case, the common functions placed in the beginning of the *.text* section of a smaller sample application cannot be used. Therefore, it is necessary to ensure that there are no common functions present at the beginning of the *.text* section. This process of detecting common functions and rearranging them in the application code is illustrated in Algorithm 10 and Figure 6.1.

Algorithm 10 Rearrange Application Code (RAC)

```
1: Create a List $L$ of Type I clones in the application codes
2: Mark the elements of $L$ as $Ta$ or $Tb$, depending on whether they are Type 1a or Type
   1b
3: for each $i$ in $L$ do
4:     if $Ta$ then
5:         Move to the last possible location
6:     end if
7: end for
8: for each $i$ in $L$ do
9:     if $Tb$ then
10:        Move to the last possible location
11:    end if
12: end for
13: Reorder the remaining functions
14: Fix the function calls
```

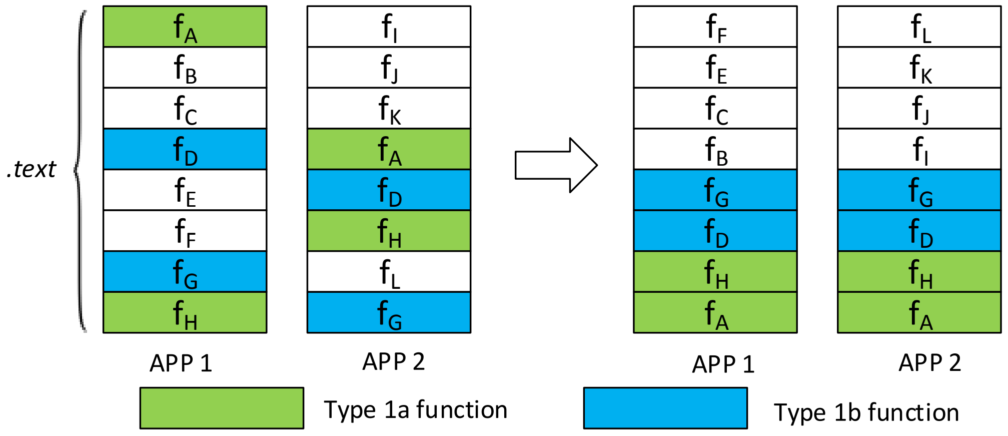

Figure 6.1. Rearranging application code

7. PROPOSED ALGORITHM

7.1. PRE-DEPLOYMENT PHASE

The pre-deployment phase consists of two parts; code processing and crypto pre-processing. The common function detection as explained in the previous section happens in the code processing part. The base station begins by going through all the sample applications, assigning a unique identifier known as the FID to each new function it encounters. The FIDs and the corresponding functions are stored in a table called the function table. Once all the functions and the common functions are identified, the base station uses Algorithm 10 to rearrange the code for each application. In the crypto pre-processing phase, the base station creates a one-way hash chain, $A_0, A_1, .., A_t$ which we call the authentication hash chain. t is taken to be sufficiently large so as to cover the entire lifetime of the network operations. The hash chain has the following rules:

1. $A_i = h(A_{i+1})$
2. A_0 is the root of the chain, which is obtained by applying the hash function $h()$, t times on A_t.

For each sensor, some FIDs are randomly selected. The base station then generates a random secret key K_0 and the corresponding elliptic curve public key $Q_0 = K_0 * T$. The functions are encrypted with this key Q_0 using the proxy re-encryption scheme EC-BBS. Pairs of FID and the associated encrypted function $(j,\ E_{Q_0}(f_j()))$, authentication key A_0, hash function $h()$ and a pseudo random function $g()$ are pre-deployed on the sensors. This process is shown in Algorithm 11.

Algorithm 11 Pre-Deployment

```
1: Associate each function with a unique FID and build the function table
2: Call Alg RAC to create L and rearrange application codes
3: for each $i$ in $SN$ do
4:     Randomly select $k$ FIDs from $L$
5:     for each $j$ in $k$ do
6:         Encrypt $f_j()$ using $Q_0$
7:         Store tuple $(j, E_{Q_0}(f_j()))$ on $SN_i$
8:     end for
9: end for
```

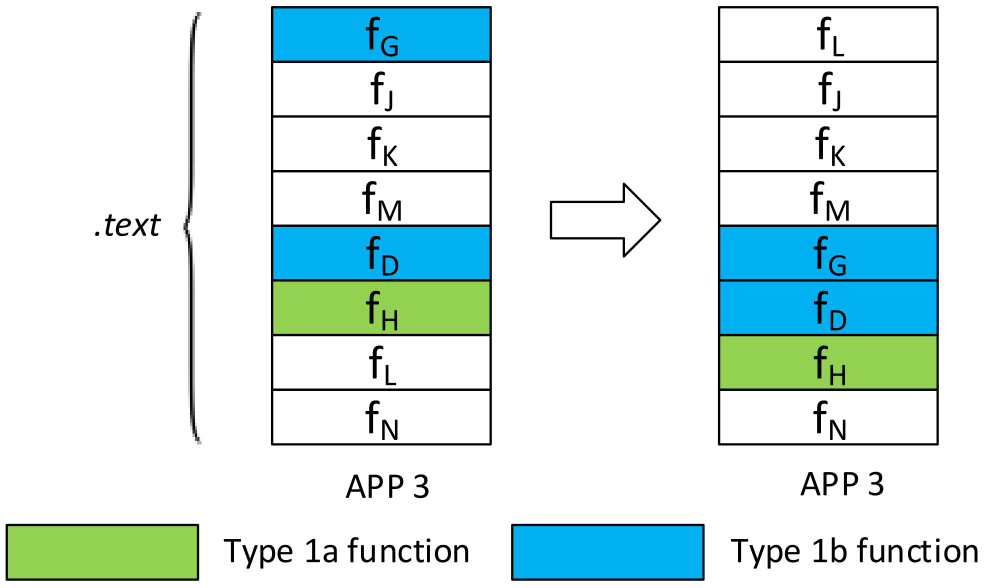

Figure 7.1. Rearranging application code for a new application

7.2. PRE-DISSEMINATION

When the base station has to disseminate a new application code in the network, it first identifies those functions of the new application which can be found in the network,

Algorithm 12 Pre-Dissemination

```
1: Create CFL
2: For the $i$-th iteration calculate the re-encryption key $RK_i$ from keys $K_i$ and $K_{i-1}$
3: Compute $h(CFL)$ and $HMAC_{A_i}(h(RK_i)||(h(CFL)))$
4: Disseminate $HMAC_{A_i}(h(K_i)||(h(CFL)))$ in the network before the the code is dissem-
   inated
```

stored on the nodes. It then rearranges the functions of the new application code such that the common functions reside in the same memory location as the code which was distributed in the network. Rest of the code is then placed around these functions. This can be seen in Figure 7.1. The global variables of this new application are also arranged according to the common functions' need in the *.data* and *.bss* sections. The base station then creates a list called the common functions list (CFL) which is in the form of FIDs along with the size of the functions and their memory location in the compiled code.

Before disseminating the code, the base station generates a random secret key K_i and the corresponding elliptic curve public key $Q_i = K_i * T$ for the i^{-th} iteration of code change. It then computes the re-encryption key RK_i from K_0 and K_i as K_i/K_0. A pre-dissemination packet is constructed which consists of an HMAC of hash of the re-encryption key concatenated with hash of the CFL, i.e $HMAC_{A_i}(h(RK_i)||h(CFL))$. The key A_i used to generate the HMAC is the next key in the authentication key chain. The HMAC is then disseminated in the network just prior to the code. This is a broadcast packet and all the nodes in the network save the contents of this packet to authenticate the CFL and the re-encryption key at a later stage. This process is illustrated in Algorithm 12.

7.3. CODE DISSEMINATION

The base station (BS) prepares for code dissemination by creating a Bloom Filter (BF) of an appropriate length. It uses a hash function to hash the common functions on the CFL one-by-one to populate the BF. It then combines the CFL, the BF, the new code, the next re-encryption key (RK_i) and the next decryption key (K_i) together. The BS also creates an index on the code dissemination content to help the nodes recover everything; it appends this index to the contents.

Once the total code dissemination content as shown in Figure 7.2 is known, it is divided into pages. These pages are further divided into packets. The BS then uses a nonce (n) and the clusters group key (CK) with a pseudo random function $g()$ to generate a session key, $S_n = g_n(CK)$. This key is used to encrypt the packets and provide confidentiality. Each packet is encrypted individually with the same key. To provide code integrity, we used a process similar to that used by Seluge [10]. For the sake of continuation, we use the same nomenclature as that used by Seluge [10]. We assume there are P pages and that each page has N packets. The pages are denoted as Page 1 to Page P, while the packets for Page i are denoted as $Pkt_{i,1}$ to $Pkt_{i,N}$. Packets in Page P are hashed and the hash of packet i is appended to packet i in page $P-1$. The packets in Page $P-1$ then consist of the concatenation of the hashes of the corresponding Page P packet and the original packets of Page $P-1$. This process is continued until all of the packets of Page 1 are hashed. A Merkle Hash Tree is created over the packets in Page 1 as shown in Figure 7.3, we call this the Vertical Hash Tree (VHT). Seluge [10] created a digital signature over the root of the Merkle Hash Tree. Verification of a signature is a public key cryptography operation and consumes a large amount of energy. Our implementation of Elliptic Curve Digital Signature Algorithm (ECDSA) signatures over TelosB motes shows that verification of one signature needs 28.771 mJ of energy [18]. On the other hand one AES-256 bit encryption costs .01mJ of energy. In a traditional wireless sensor network, the digital signature is necessary because the entire network needs to be updated. In a sensor cloud, because only one cluster needs to be updated at a time, symmetric key cryptography can be used in place of public key cryptography. Instead of signing the root of the hash with the it's private key, the BS uses the session key (S_n) as a signature key. A signature packet (which includes the VHT root hash and the nonce n) is created and the signature is produced by encrypting (VHT root hash $||$ n). The nodes in the cluster can derive the session key (S_n) from the cluster key and the nonce and verify the root hash.

Index	CFL	BF	New Code	RK_i	K_i

Figure 7.2. Content disseminated by the base station

We observe that since code updation in a sensor cloud happens in a cluster with the sensors are physically close together, energy intensive tasks such as decryption of packets can be done in a distributed manner. Thus, instead of every sensor decrypting all the code, each sensor can decrypt a few packets, and thus conserve a large amount of energy. This process, however, makes it necessary that nodes within a cluster are protected against malicious code injection from each other. To accomplish this, the base station creates another hash tree on the same dissemination contents, which we call the Horizontal Hash Tree (HHT). For this hash tree, each page of the code is hashed and the hashes of the pages $h(Page)$ are used as leaf nodes. The root hash of HHT is encrypted using the session key (S_n) and included in the signature packet. The code dissemination algorithm is given in Algorithm 13. The Vertical and the Horizontal Hash Trees are illustrated in Figure 7.3. Just before beginning the code dissemination, BS broadcasts the next authentication key (A_i), which was used to create the HMAC in the pre-dissemination phase.

Algorithm 13 Code Dissemination

```
1: Populate BF by hashing the functions in CFL
2: Create Index on $CFL||BF||NewCode||RK_i||K_i$
3: Create session key $S_n = g_n(CK)$
4: Encrypt packets of $Index||CFL||BF||NewCode||RK_i||K_i$ using $S_i$
5: Hash the encrypted packets and create VHT with Page 0
6: Hash the pages and create HHT
7: Create the signature packet
8: Broadcast Authentication Key $A_i$
9: Disseminate Code
```

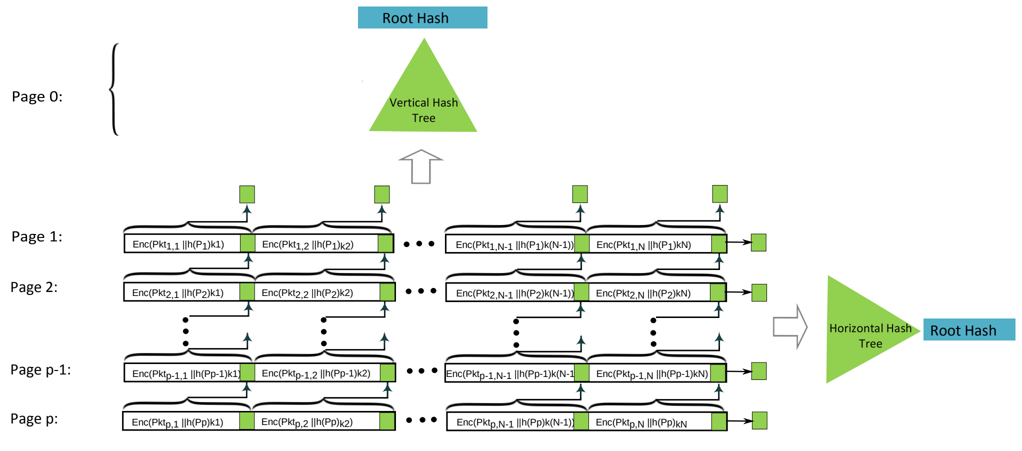

Figure 7.3. Vertical and Horizontal hash trees

7.4. ACTIVITY ON THE NODES

After receiving the next authentication key (A_i), the nodes verify this key by determining or not whether $h(A_i) = A_{i-1}$. The one way property of the hash chain ensures that any malicious node, which has obtained previous authentication keys, A_{i-1},A_{i-2},..., etc. cannot predict the key A_i, with non-negligible probability. This implies that an adversary that makes any changes to the contents of the pre-dissemination packet, will be caught with a very high probability, which ensures the delivery of both the un-tampered $h(RK_i)$ and

$h(CFL)$. The $h(RK_i)$ and $h(CFL)$ would be used to verify the re-encryption key and the CFL, which would be explained later in this subsection.

After the pre-dissemination phase is complete, the cluster for which the code dissemination was intended receives the encrypted contents. The contents are authenticated using the Vertical Hash Tree, in a manner similar to that used by Seluge [10]. The session key (S_n) is derived by using the PRF $g()$ on the cluster key CK, with nonce n received in the signature packet. To enable distributed decryption and authentication of the contents, the cluster head in each cluster creates virtual ids ranging from 1 to N and gives each sensor one of the virtual ids, where N is the number of packets in a page. The sensors, instead of dealing with all the packets only store the packets which are multiples of their virtual id. Thus, a node with virtual id 1, decrypts $pkt1$ in all the pages. Likewise, the node with virtual id 2, decrypts $pkt2$ in all the pages and so on. When the number of nodes in the cluster is less than N, nodes can be given additional virtual ids. Each node can decrypt and authenticate the packet in page i from the hash in the packet in page $i+1$ and finally packets in page 1 can be authenticated from the VHT. After all of the packets have been received and authenticated, the nodes encrypt their decrypted packets again and broadcast for other nodes to receive. The encryption is done in large blocks, reducing the number of encryption and decryption operations a node must perform and thus conserving energy on nodes.

Once the nodes receive all the packets from the code dissemination, they first verify that the cluster members have not tampered with the code. This is done by hashing each page and verifying the root of the Horizontal Hash Tree. Since, nodes are only allowed to decrypt a part of each page, any change in the code by a malicious node will always be identified. After the verification phase is complete, the nodes use the index to extract the CFL, the Bloom filter, the new code, the re-encryption key RK_i and the encryption key K_i. The cluster head then broadcasts RK_i and CFL in clear.

When a node receives this broadcast packet, it checks if it has one or more of the requested functions in the CFL by comparing the FIDs. If the node finds that it has some of the requested functions, it verifies the validity of the CFL and RK_i by creating an HMAC over $h(CFL)$ and $h(RK_i)$ using the authentication key A_i. This HMAC is compared with the HMAC received in the pre-dissemination phase. If both of the HMACs match, CFL and the RK_i are valid. The requested functions are then re-encrypted with RK_i and sent back to the requesting nodes. The encryption key (K_i) received by the cluster nodes in the code dissemination is used to decrypt the received encrypted functions. The received functions are then verified using the Bloom Filter. The functions are hashed and the resulting positions in the filter are checked against the already existing entries in the Bloom Filter. If the hashes of a received function result in positions which are unset in the Bloom Filter, the function is rejected, otherwise it is accepted. Once all functions pass through the Bloom Filter, the nodes are ready to build the code image from its various parts. To build the code image, the nodes use the CFL to plug the common functions into their appropriate position in the code. The code image is stored and built in the flash memory. Once the build is complete, the bootloader can boot the node up using this image. The entire process is illustrated in Algorithm 14.

Algorithm 14 Image build

```
1:  Cluster head assigns virtual ids to sensors
2:  Nodes in the cluster store dissemination packets corresponding to their virtual ids
3:  Encrypted packets are verified using hashes and then decrypted
4:  Decrypted packets with hashes are combined in large blocks and securely broadcasted
    in the cluster.
5:  All nodes receive all the transmitted code
6:  VHT and HHT are verified
7:  Base station broadcasts the CFL
8:  Nodes receive encrypted common functions from other nodes
9:  Functions are decrypted and verified through BF
10: if functions are verified then
11:     Build code image
12:     if All functions are verified then
13:         Reboot with new code
14:     end if
15: end if
```

8. A DISCUSSION ON SECURITY

Our algorithm divides an application's code into two parts, new code and the common code. Different mechanisms are used for the security of both these categories, which are discussed in the following subsections.

8.1. CONFIDENTIALITY OF CODE

8.1.1. New Code. Confidentiality of the new code is provided by encrypting the dissemination content with the session key (S_n). This key is created uniquely for each session by using the cluster's key CK and a randomly generated nonce (n). This nonce is sent in the signature packet with the VHT root hash, and the signature over (VHT root hash$||n$). The signature guarantees the correct reception of both the VHT root hash and the nonce n.

8.1.2. Common Code. Confidentiality of the common code is provided by proxy re-encryption. The following discussion is based on the EC-BBS proxy re-encryption scheme, although a similar argument can also be made using SRE [13].In the pre-deployment phase, the base station encrypts randomly chosen functions with the encryption key Q_0 using EC-BBS and stores them on the nodes. Neither this key Q_0 nor the corresponding secret key K_0 are revealed to any node. This provides the confidentiality of the common functions from the nodes on which the functions are stored. Prior to disseminating any application code, the base station first broadcasts $HMAC_{A_i}(h(RK_i)||h(CFL))$ to all nodes. The first packet in the code dissemination is the packet that contains the authentication key A_i. The validity of the authentication key can be verified by determining whether or not $h(A_i) = A_{i-1}$. After code dissemination, when node j requests a function from node k, j needs to send the CFL and the re-encryption key RK_i to node k. Node k checks the validity of the request by generating an HMAC on the received CFL and RK_i, and comparing it with the HMAC received in the pre-dissemination phase. It then re-encrypts the requested function using the received RK_i. This process allows node j to receive re-encrypted functions without revealing them to node k or any other unauthorized node.

8.2. INTEGRITY OF CODE

8.2.1. New Code. For integrity of the new code we use two hash trees, the Vertical Hash Tree (VHT) and the Horizontal Hash Tree (HHT). The hashes of packets in Page 1 are used as leaf nodes and the resulting tree is taken as the VHT. The VHT is used in the same manner as that described by Seluge [10]. For the HHT each page is hashed and the hashes of pages are taken as the leaf nodes and a tree is built. When the nodes in a cluster perform distributed decryption of the disseminated code, corrupt nodes can inject malicious code into the code image. The HHT, however, makes sure that such a code injection will always be detected.

8.2.2. Common Code. When node j requests a function from node k, a corrupt k can return corrupt or malicious code back to j. Node j verifies all the received functions against the Bloom Filter it received during code dissemination to thwart such attacks.

9. PERFORMANCE ANALYSIS

We implemented our algorithm on the Mica2 and TelosB mote platforms with the TinyOS operating system and simulated it using the TOSSIM simulator. The TinyECC [19] library was used for elliptic curve implementation, and AES-256 was used as the symmetric key encryption algorithm. The S4 [20] routing algorithm was used to route code through the clusters.

The Bloom Filter used to verify the integrity of the received common function has a definite false positive probability p_f, defined as,

$$p_f = (1 - e^{-km/n})^k \tag{1}$$

where k is the number of hash functions used, m is the size of the Bloom Filter and n is the number of functions hashed. For a given p_f and k, the number of functions n that can be supported are calculated as,

$$n = \frac{-m}{k} \ln(1 - p_f^{1/k}) \tag{2}$$

In our experiments, we have taken p_f to be 0.01 and m as 256. Figure 9.1 illustrates the relationship between n and k for the given values of p_f and m. It can be seen from the figure that the maximum number of functions that can hashed in the Bloom Filter, for a maximum false positive probability of 0.01 is 27 at k = 7. We have therefore taken k as 7 in our experiments.

We implemented both the EC-BBS proxy re-encryption scheme and the SRE scheme [13] on Mica2 and TelosB motes to test their energy efficiency. In Algorithm 9, we can see that decryption requires one point addition (subtraction) and one point multiplication. The results of our implementation can be seen in Figure 9.2 and 9.3 which show the execution time and the energy consumption resepectively of various operations on the motes. The block size was taken to be 128 bits. The operations shown in the figure are encryption, decryption, re-encryption, re-encryption key generation and key generation. While the encryption, key generation and re-encryption key generation operations are performed on the base station, the nodes perform re-encryption and decryption operations. We can see that all of the SRE operations are very lightweight consuming less than 1.2 mJ of energy on Mica2 mote and less than 0.33 mJ of energy on a more optimized TelosB mote. In the case of EC-BBS, as expected the execution times and the corresponding energy consumption is slightly higher than that seen for SRE. The motes, however, only perform the decryption and the re-encryption operations, which consume 6.1 mJ and 3.12 mJ, respectively, on Mica2 and 1.86 mJ and 0.95 mJ on TelosB.

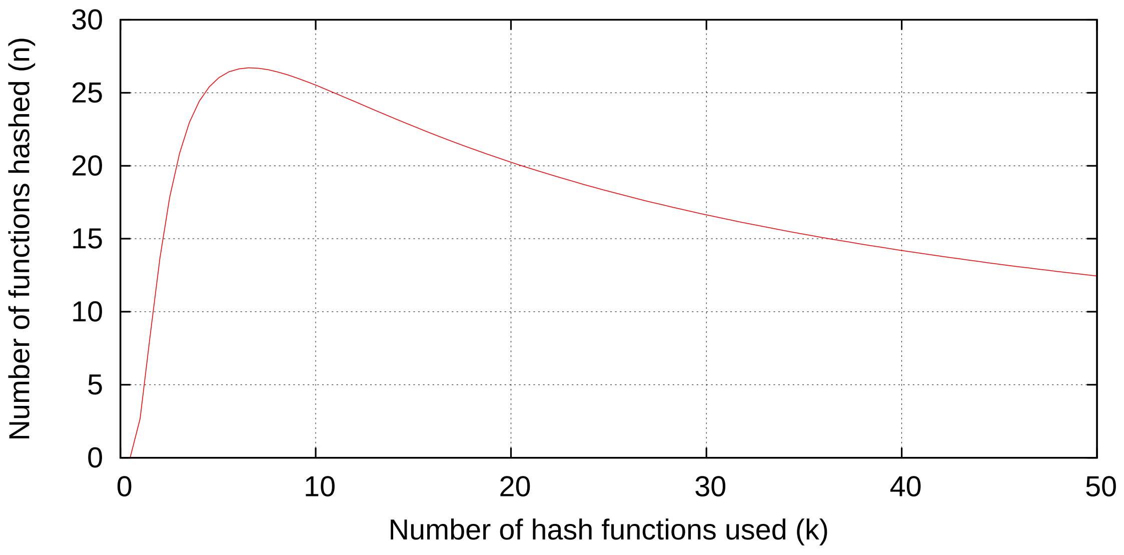

Figure 9.1. Relationship between k and n for $p_f = 0.01$

In the pre-deployment phase, the base station chooses some FIDs randomly and stores the corresponding encrypted functions on the sensors. The more functions the base station could store on the nodes the more energy efficient the scheme would be, however, there is a trade off with privacy. Malicious nodes can decrypt their stored functions with the combination of the re-encryption key RK_i and the decryption key K_i, during an iteration of code dissemination. We define privacy lost ρ as the fraction of code stored on compromised nodes which is common with the next iteration of code dissemination. The tradeoff between storage capacity and privacy for varying percentages of compromised nodes can be seen in Figure 9.4. For the rest of our experiments, we have taken the storage capacity to be 512 bytes.

$$\rho = \frac{\left(\frac{\textit{Code stored on compromised nodes in a cluster}}{\textit{Total number of application}}\right)}{\textit{Average amount of code per application}}$$

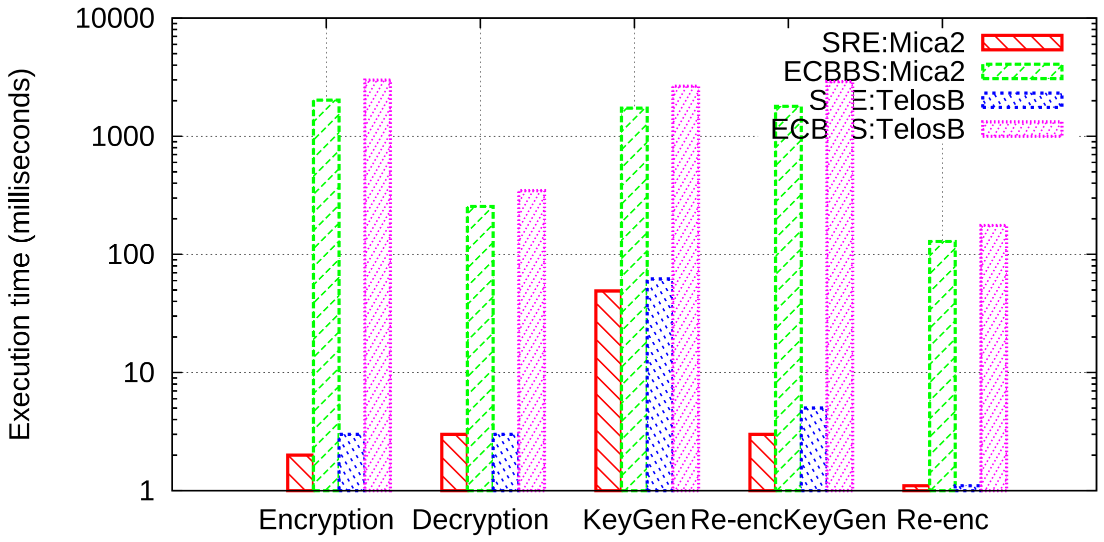

Figure 9.2. Execution times of proxy re-encryption operations

Figure 9.5 shows the reduction in the size of disseminated code for various applications when compared to Seluge [10]. For this experiment, we chose five applications; the standard TinyOS apps, `Blink, Sense, Oscilloscope, RadioSenseToLeds(RSTL)`

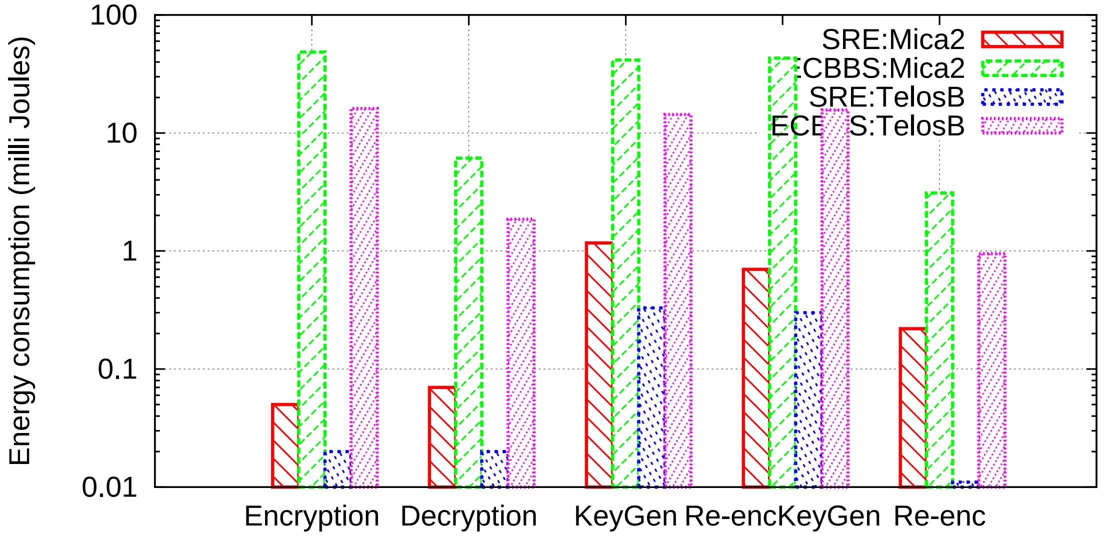

Figure 9.3. Energy consumption of proxy re-encryption operations

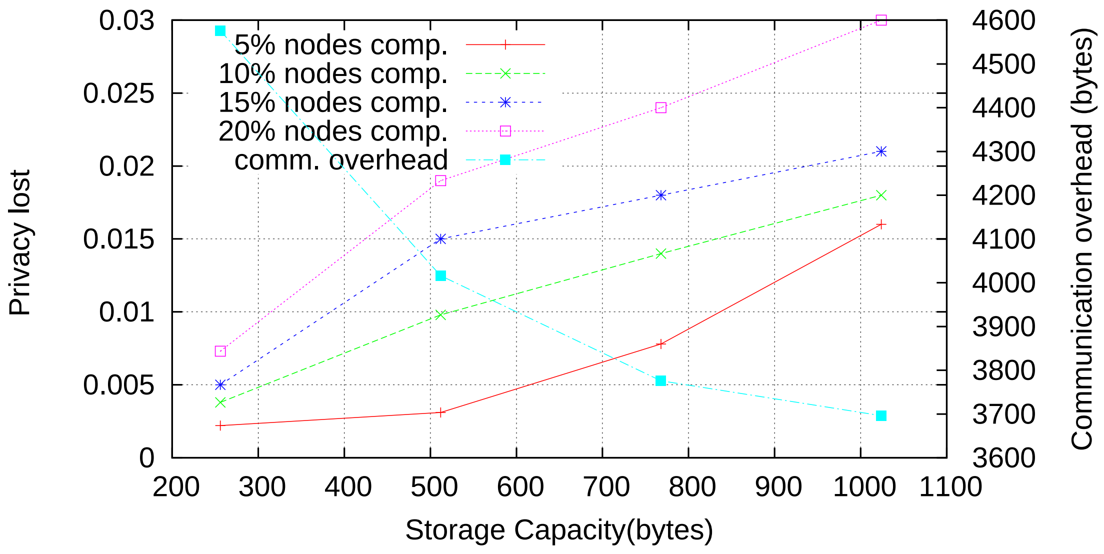

Figure 9.4. Tradeoff between privacy lost and communication overhead

and an application `Tree`, which is a tree construction application. The common functions between these applications were discovered and distributed in a simulated network. The network was taken to be a 30x30 grid of 900 nodes placed 2 meters apart from each other.

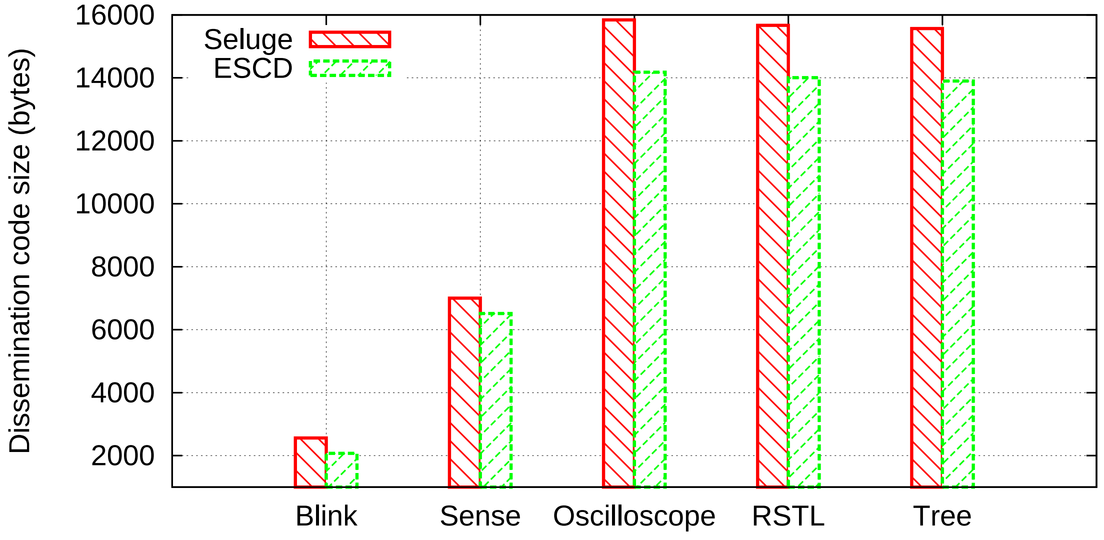

Figure 9.5. Percentage reduction in overall code size

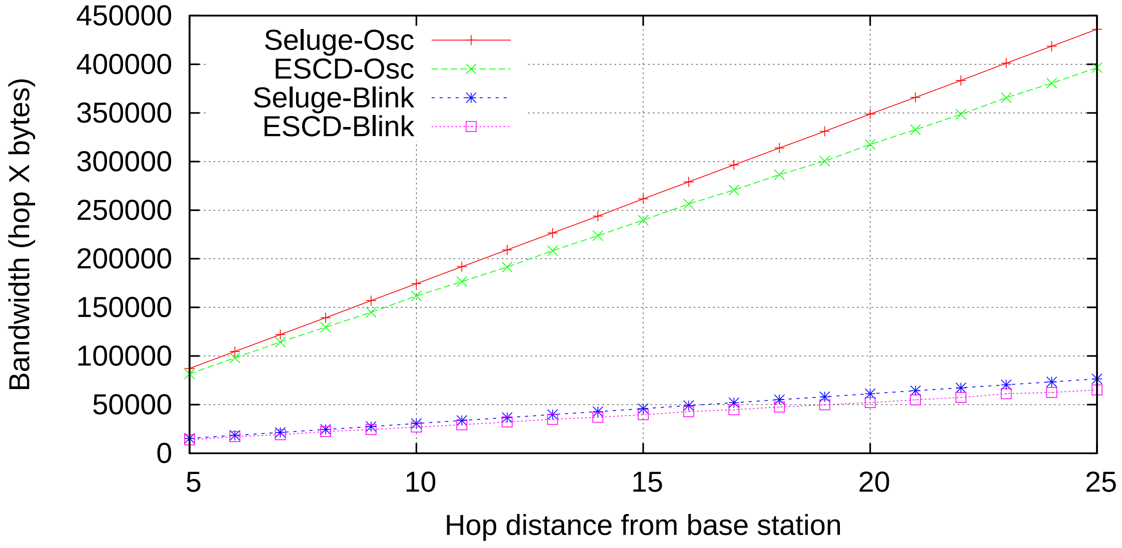

Figure 9.6. Bandwidth consumption

The dissemination code for our algorithm consisted of the new code along with the overhead of the index, the CFL, Bloom Filer, re-encryption key and the decryption key. The amount of reduction in disseminated code for each application is illustrated in Figure

9.5. Our algorithm is denoted by ESCD. The reduction is highest in the `Blink` application. Here the algorithm disseminated 19.06% less code than Seluge [10]. The code reduction in `Sense` was 6.9%, while in `Oscilloscope, RadioSenseToLeds` and `Tree` it was approximately 10.5%.

Figure 9.6 illustrates this difference in terms of bandwidth for the `Oscilloscope` and `Blink` applications. We define the product of the size of code and the distance it travels in hops as the metric of bandwidth. In ESCD, once the reduced code reaches a cluster head, the CFL is broadcasted and the nodes reply back with the functions in the CFL. We designed a controlled flooding method, where each node which receives the broadcast, re-broadcasts it further with the probability of $1-(0.5+0.1*hop)$. Doing so ensures that flooding of the CFL in the network, stops on average after the third hop. Figure 9.6 considers both, the bandwidth consumed in this controlled flooding as well as the bandwidth consumed in sending the requested functions to the requesting cluster head. The flash memory available to store common functions was assumed to be 512 bytes. As can be seen, our algorithm reduces the total number of wireless transmissions in the network and hence the total bandwidth consumed. The reduction is more prominent when the cluster to be updated lies far away from the base station in terms of hop distance. Figure 9.7 illustrates the energy overhead of our algorithm as compared to Seluge [10], for each of the applications. This energy overhead is due to the decryption of code packets, the verification of HHT and the verification of common functions through Bloom Filter. When SRE was used on the Mica2 motes, the overhead varied between 1.224 mJ for `Blink` and 2.304 mJ for `Tree`, with EC-BBS the overhead increased to 61.72 mJ for `Blink` and 63.5 mJ for `Tree`. In the case of TelosB motes, the overhead was reduced to between 0.462 mJ for `Blink` and 0.774 mJ for `Tree` for SRE, and between 18.862 mJ for `Blink` and 19.174 mJ for `Tree` for EC-BBS. A large portion of this overhead is the cost of confidentiality of code packets, which is not provided by Seluge [10].

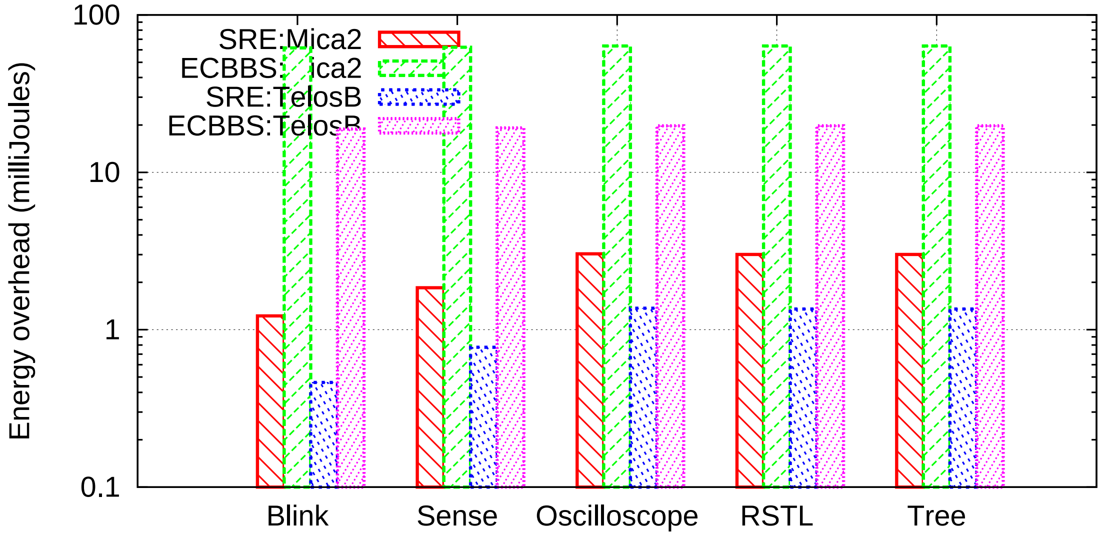

Figure 9.7. Energy overhead compared to Seluge

10. CONCLUSION

Sensor clouds are an emerging paradigm for sensor networks. These are very dynamic in nature with nodes being constantly provisioned and de-provisioned for users. In such a scenario, an efficient code dissemination algorithm that is also secure becomes necessary. In this paper, we have presented a novel code dissemination algorithm that is both efficient and secure. Our code dissemination algorithm considers the similarities that exist between codes across applications. The basic idea, therefore, is to communicate only the new code to the sensors while the common code can be picked up from the sensors in the network. This process reduces the amount of code that needs to be communicated. Reduced amount of code results in energy efficient code dissemination. Our security framework is based around proxy re-encryption, hash trees and Bloom Filters, which combine to provide confidentiality and integrity to the code dissemination algorithm. We have designed and implemented our algorithm on TelosB motes and from the experiments, it can be concluded that this algorithm reduces the amount of communication and energy required, while also providing confidentiality and integrity of code. In this paper we have focussed only on Type 1 clones in the application code. In future, we intend to include Type 2 clones as well, which will greatly increase the efficiency of the algorithm.

11. BIBLIOGRAPHY

[1] N. Bin Shafi, K. Ali, and H. Hassanein, "No-reboot and zero-flash over-the-air programming for wireless sensor networks," in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th Annual IEEE Communications Society Conference on*, june 2012, pp. 371 –379.

[2] W. Dong, Y. Liu, C. Chen, J. Bu, C. Huang, and Z. Zhao, "R2: Incremental reprogramming using relocatable code in networked embedded systems," *IEEE Transactions on Computers*, vol. 99, no. PrePrints, 2012.

[3] J. Jeong and D. Culler, "Incremental network programming for wireless sensors," in *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, oct. 2004, pp. 25 – 33.

[4] J. Koshy and R. Pandey, "Remote incremental linking for energy-efficient reprogramming of sensor networks," in *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, jan.-2 feb. 2005, pp. 354 – 365.

[5] N. Reijers and K. Langendoen, "Efficient code distribution in wireless sensor networks," in *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, ser. WSNA '03. New York, NY, USA: ACM, 2003, pp. 60–67.

[6] N. Poolsappasit, V. Kumar, S. Madria, and S. Chellappan, "Challenges in secure sensor-cloud computing," *Secure Data Management*, pp. 70–84, 2011.

[7] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure-physical sensor management with virtualized sensors on cloud computing," in *Network-Based Information Systems (NBiS), 2010 13th International Conference on*. IEEE, 2010, pp. 1–8.

[8] J. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 81–94.

[9] W. Li, Y. Du, Y. Zhang, B. R. Childers, P. Zhou, and J. Yang, "Adaptive buffer management for efficient code dissemination in multi-application wireless sensor networks," in *IEEE/IPIP International Conference on Embedded and Ubiquitous Computing*, 2008, pp. 295–301.

[10] S. Hyun, P. Ning, A. Liu, and W. Du, “Seluge: Secure and dos-resistant code dissemination in wireless sensor networks,” in *Proceedings of the 7th international conference on Information processing in sensor networks*. IEEE Computer Society, 2008, pp. 445–456.

[11] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler, “Securing the deluge network programming system,” in *Proceedings of the 5th international conference on Information processing in sensor networks*, ser. IPSN ’06. New York, NY, USA: ACM, 2006, pp. 326–333.

[12] H. Tan, D. Ostry, J. Zic, and S. Jha, “A confidential and dos-resistant multi-hop code dissemination protocol for wireless sensor networks,” *Computer Security*, vol. 32, no. 0, pp. 36 – 55, 2013.

[13] A. Syalim, T. Nishide, and K. Sakurai, “Realizing proxy re-encryption in the symmetric world,” in *Informatics Engineering and Information Science*, ser. Communications in Computer and Information Science, A. Abd Manaf, A. Zeki, M. Zamani, S. Chuprat, and E. El-Qawasmeh, Eds., 2011, vol. 251, pp. 259–274.

[14] M. Blaze, G. Bleumer, and M. Strauss, “Divertible protocols and atomic proxy cryptography,” in *EUROCRYPT*, 1998, pp. 127–144.

[15] J. Albath, M. Thakur, and S. Madria, “Energy constrained dominating set for clustering in wireless sensor networks,” *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 812–819, 2010.

[16] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[17] T. Eisenbarth, R. Koschke, and D. Simon, “Aiding program comprehension by static and dynamic feature analysis,” in *Software Maintenance, 2001. Proceedings. IEEE International Conference on*, 2001, pp. 602 –611.

[18] V. Kumar and S. K. Madria, “Secure hierarchical data aggregation in wireless sensor networks: Performance evaluation and analysis,” *Mobile Data Management, IEEE International Conference on*, vol. 0, pp. 196–201, 2012.

[19] A. Liu and P. Ning, “Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks,” in *Information Processing in Sensor Networks, 2008. IPSN ’08. International Conference on*, 2008, pp. 245–256.

[20] Y. Mao, F. Wang, L. Qiu, S. Lam, and J. Smith, “S4: Small state and small stretch compact routing protocol for large static wireless networks,” *Networking, IEEE/ACM Transactions on*, vol. 18, no. 3, pp. 761–774, 2010.

SECTION

4. CONCLUSION

This document first presents algorithms to perform secure and privacy preserving data aggregation in sensor clouds. Further an algorithm for controlling user access while using such data aggregation algorithm is also provided. Finally, it also presents an algorithm for secure dissemination of code in sensor clouds.

The scheme proposed in Section I performs energy efficient and secure data aggregation on wireless sensor networks. It makes use of the Elliptic Curve El Gamal (ECEG) encryption scheme to achieve data confidentiality and a modified version of Elliptic Curve Digital Signature Algorithm (ECDSA) to achieve integrity of aggregated data. The homomorphic property of ECEG and the additive property of our version of ECDSA allows us to simply add ciphertext and digital signatures, which saves energy on aggregators. Unlike previous approaches our algorithm does not need a separate integrity verification phase, which also helps to save energy on the nodes. Additionally our robust tree construction algorithm handles denial of service attacks and node failure events.

The proposed approach in Section II presents a more efficient way of performing secure and privacy preserving data aggregation using recursive secret sharing and symmetric key homomorphic encryption constructs. Recursive secret sharing is used to store data and a linear combination of data and a key, which is used for verifying the integrity of data. The shares are perturbed and scrambled using a simple symmetric key encryption scheme which supports homomorphism. An enhanced version of the scheme is also presented which can be used to detect malicious nodes in the network, which inject false data.

In Section III a distributed attribute based access control scheme was presented which can work on aggregated data. To accomplish this key policy attribute based encryption is used to provide access control and paillier encryption to provide homomorphism for aggregating keys and data. The access tree used in attribute based encryption was modified to support a wider range of queries. Additionally this scheme provides the flexibility to a network owner to modify the authorization required to access data at run time. It was shown that this approach has a marginally high computational complexity, while keeping the communication complexity same as previous schemes. Further the scheme also provides an efficient method to revoke user access.

In Section IV an efficient and secure method of disseminating code to wireless sensors in a sensor cloud is presented. This approach has two parts, in the first part, the total amount of code transmitted from the base station to the wireless sensors is reduced. This reduction in code is based on the fact that, applications written for wireless sensor share fragments of code. These common fragments of code are deployed *a priori* in the network, so that any code transmission from the base station in future can be done without these fragments. In the second part of the approach, security, in terms of confidentiality of code and protection against malicious code injection attacks is discussed. The approach uses Symmetric Re-Encryption, Bloom Filters and HMACs to accomplish these objectives.

VITA

Vimal Kumar was born in New Delhi, India in 1984. His initial schooling took place in seven different schools, spread across five states in India. He earned his bachelors degree in Computer Science & Engineering from Guru Gobind Singh Indraprastha University at New Delhi in 2006. After his graduation, he worked as a software engineer at the GTS division of Computer Sciences Corporation at the Hyderabad and Noida offices.

Vimal came to the Missouri University of Science and Technology in 2008, where he earned his Doctorate of Philosophy in Computer Science, in May, 2014. While there, he worked as a research assistant with Dr. Sanjay Madria, focusing on security and privacy aspects of data aggregation in Wireless Sensor Networks and Sensor Clouds. Vimal also served as a teaching assistant for the Cloud Computing course and mentored undergraduate students on topics of localization and wireless sensor network security.