2008

Protocol design, implementation and integration for the protection of sensor data confidentiality and integrity

Santosh Kumar Panchapakesan
Iowa State University

Protocol design, implementation and integration for the protection of sensor data confidentiality and integrity

by

Santosh Kumar Panchapakesan

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Wensheng Zhang, Major Professor
Johnny S. Wong
Daji Qiao

Iowa State University

Ames, Iowa

2008

TABLE OF CONTENTS

LIST OF TABLES iv

LIST OF FIGURES v

ACKNOWLEDGEMENTS vi

ABSTRACT vii

CHAPTER 1. INTRODUCTION 1

1.1 Background and Research Motivations 1

1.2 Research Objectives 4

1.3 Research Challenges and Methodologies 4

1.4 Research Results 6

CHAPTER 2. RELATED WORK 7

2.1 Node Revocation schemes 9

2.2 Confidentiality and Integrity Protocols 10

2.2.1 APB - Adaptive Polynomial Based Scheme for Confidentiality 10

2.2.2 MAF 12

CHAPTER 3. PROTECTING DATA INTEGRITY AND CONFIDENTIALITY AT USER NODE 16

3.1 Background 16

3.2 DKVP 17

3.2.1 Network Assumptions 17

3.2.2 Security Assumptions 17

3.2.3 Design Goals 18

3.2.4 Proposed Privilege Deprivation Scheme 19
3.2.5 Overview 19
3.2.6 Preparation at Network Controller 20
3.2.7 Protection of Vulnerable Data at Sensor Nodes 23
3.2.8 Protection of Vulnerable Keys at Sensor Nodes 25
3.2.9 Key Updating at Innocent Users 27
3.2.10 Data Retrieval by Innocent Users 30
3.2.11 Security Analysis 33

CHAPTER 4. INTEGRATING SECURITY PROTOCOLS 35

4.1 Background 35
4.2 High-level Design 37

CHAPTER 5. SYSTEM DEPLOYMENT AND USAGE MODELS: CASE STUDIES 49

5.1 Model I : Protocols Executing Individually 49
5.2 Model II: Protocols Executing Together 50

CHAPTER 6. IMPLEMENTATION AND RESULTS 53

6.1 Programming Limitations 53
6.2 Network Model Assumed 54
6.3 Results 54
6.3.1 Storage Overhead 54
6.3.2 Computation Overhead 55
6.3.3 Communication Overhead 64
6.4 Implementation on live network 66

CHAPTER 7. CONCLUSIONS AND FUTURE WORK 67

BIBLIOGRAPHY 69

LIST OF TABLES

Table 4.1 Security Level for APB 39

Table 6.1 Storage Overhead 55

Table 6.2 Time taken for operations 66

LIST OF FIGURES

Figure 1.1 Application Scenario describing the problem. 3

Figure 2.1 Illustration Example for Polynomial Based Encryption. 8

Figure 2.2 An Example of Disseminating A New Seed Polynomial (r = 4, z = 0). 12

Figure 2.3 Illustration of the Integrity protocol. 15

Figure 3.1 Working of DKVP. 21

Figure 3.2 Illustration example. 32

Figure 4.1 Design of the $Integration - Suite$. 36

Figure 4.2 High-level Design of the APB. 38

Figure 5.1 Example Illustrating the 3 protocols working together. 52

Figure 6.1 Computation Overhead at Sensor/Storage Node. 57

Figure 6.2 Computation Overhead at User Node for APB. 58

Figure 6.3 Computation Overhead at User Node for DKVP. 59

Figure 6.4 Computation Overhead at User Node for Both APB and DKVP. . . . 59

Figure 6.5 Energy consumed at Sensor/Storage Node. 60

Figure 6.6 Computation Overhead of DKVP (Degree of Polynomial = 13). . . . 61

Figure 6.7 Computation Overhead of APB (Degree of Polynomial = 13). 62

Figure 6.8 Computation Overhead of Both (Degree of Polynomial = 13). 63

Figure 6.9 Energy Consumed at the Storage node. 63

Figure 6.10 Processing Time for Communication. 65

Figure 6.11 Energy Consumed for Communication. 65

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Wensheng Zhang for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Johnny S. Wong and Dr. Daji Qiao. I would also like to thank Nalin V. Subramanian, for his valuable inputs to this work. I would additionally like to thank my parents for supporting me emotionally throughtout my stay at Iowa State University. I would also like to acknolwdge the support given to me by my friends whose encouragement have lifted my spirits to the level of completing my thesis. Finally, I would like to thank Linda Dutton and Gloria Cain, for the patience shown in hearing me out all all the times during my stay.

ABSTRACT

Wireless sensor networks are data centric because in many applications, sensor nodes are required to generate data, collect data, storage data and process data queries. Meanwhile, wireless sensor networks are vulnerable to security attacks because they are deployed in unattended (often hostile) environments and do not have tamper resistant hardware. Therefore, secure and efficient data management schemes are necessary to sensor networks. In this thesis work, we study how to secure a representative type of sensor data management approach called data centric storage based (DCS) schemes, with focus on protecting data confidentiality and integrity.

Considerable efforts have been made for securing DCS, however, existing work has the limitations of (i) not considering user node compromise, (ii) lack of studies on real system implementation and detailed experiments, and (iii) lack of studies on integrating security schemes to defend against multiple attacks simultaneously. To overcome these limitations, we have conducted the following research: Firstly, we have designed a new data confidentiality protocol called DKVP (data and key vulnerability protection) scheme to protect sensor data confidentiality in presence of user node compromise. Secondly, we have implemented three polynomial-based sensor data confidentiality and integrity protection schemes, namely, the adaptive polynomial-based scheme for secure data storage and query (APB), the message authentication function based schemes for data integrity (MAF), and the DKVP scheme, on top of TinyOS/Mote platform. Thirdly, we have developed a prototype system that consists of (i) integrated data confidentiality and integrity protection modules (i.e., the APB, MAF and DKVP schemes), (ii) effective and friendly interfaces to application developers to facilitate inclusion of security features into application programs, and (iii) example programs to

demonstrate the integration suite developed by us.

Extensive experiments have been conducted to study the feasibility and performance of the above designs and implementations. The results show that, if system parameters are properly chosen, desired security level can be achieved which is cost affordable by the current generation of sensor nodes such as MICA motes. In particular, our study shows that running the three integrated protocols together consumes only 27 msec of processing time and 60% of CPU usage.

CHAPTER 1. INTRODUCTION

1.1 Background and Research Motivations

A sensor network [1] is typically comprised of unattended-deployed sensor nodes (also called *motes*) capable of performing simple computations, gathering information, and communicating with other nodes.

With these attractive features, sensor networks are deployed in a wide variety of applications, including health care, military surveillance, smart homes, inventory management and disaster rescue [1]. In these applications, the major duties of sensor nodes are monitoring their direct environment, generating sensing data, collecting data and storing data, and answering queries for data. To efficiently accomplish these duties, many data management schemes have been proposed. In this thesis work, we focus on the data centric storage (DCS) [4] based data management, in which data is stored in the network and storage nodes are responsible for servicing queries from network users. It has been shown that this management approach can reduce the overhead of communicating data from nodes to the base station [4]. It adopts the concepts of geographic routing and the peer-to-peer lookup to facilitate storage and query.

Meanwhile, sensor nodes are often deployed in unattended (and sometimes hostile) fields. If a field is invaded by an adversary, sensor nodes deployed there could be captured and compromised. Hence, data transmitted or stored in the network are at risk if some nodes of the network are compromised. Therefore, securing data communication and storage is of critical importance. Figure 1.1 depicts an example where sensor nodes are deployed in a forest and the DCS approach is adopted for data management. Here, the network consists of a centralized *base station*, some *storage nodes* (i.e., nodes storing sensor data) and some *user nodes* (i.e., nodes that query sensor data). Other nodes either generate data (i.e., *data sources*)

or are on paths connecting data sources and storage nodes (i.e., *en-route nodes*). a, b and c denote a compromised en-route node, a compromised user node and a compromised storage node, respectively. After these nodes have been compromised, the following threats to data security are posed in the network:

- *Violating Data Confidentiality*: If the data stored in a storage node is not properly protected, the data can be revealed once it is compromised. If the data in transmission is not properly protected (e.g., encrypted), it could be revealed by a comopromised en-router node.

- *Violating Data Integrity*: Without proper integrity protection, the data in transmission can be modified by a compromised en-router node without being detected. A compromised en-router node can even inject false data.

- *Violating Authorized Data Access*: It is important that every piece of data is accessed only by the users that are authorized to so. If a user is compromised but is not properly revoked, the desired property of authorized data access will no longer hold.

The problem of data security has attracted lots of attention and many schemes [7, 5, 19, 20] have been proposed to protect data confidentiality and integrity. However, these efforts have not or have not adequately addressed the following issues:

- Protecting data confidentiality when one or more user nodes are compromised.

- Implementing the schemes in a real sensor network testbed to evaluate its feasibility and identify practical issues.

- Integrating multiple schemes together to defend against multiple simultaneous attacks.

- Providing effective and friendly interfaces to application developers to facilitate the developers to include security protection features in their newly developed applications

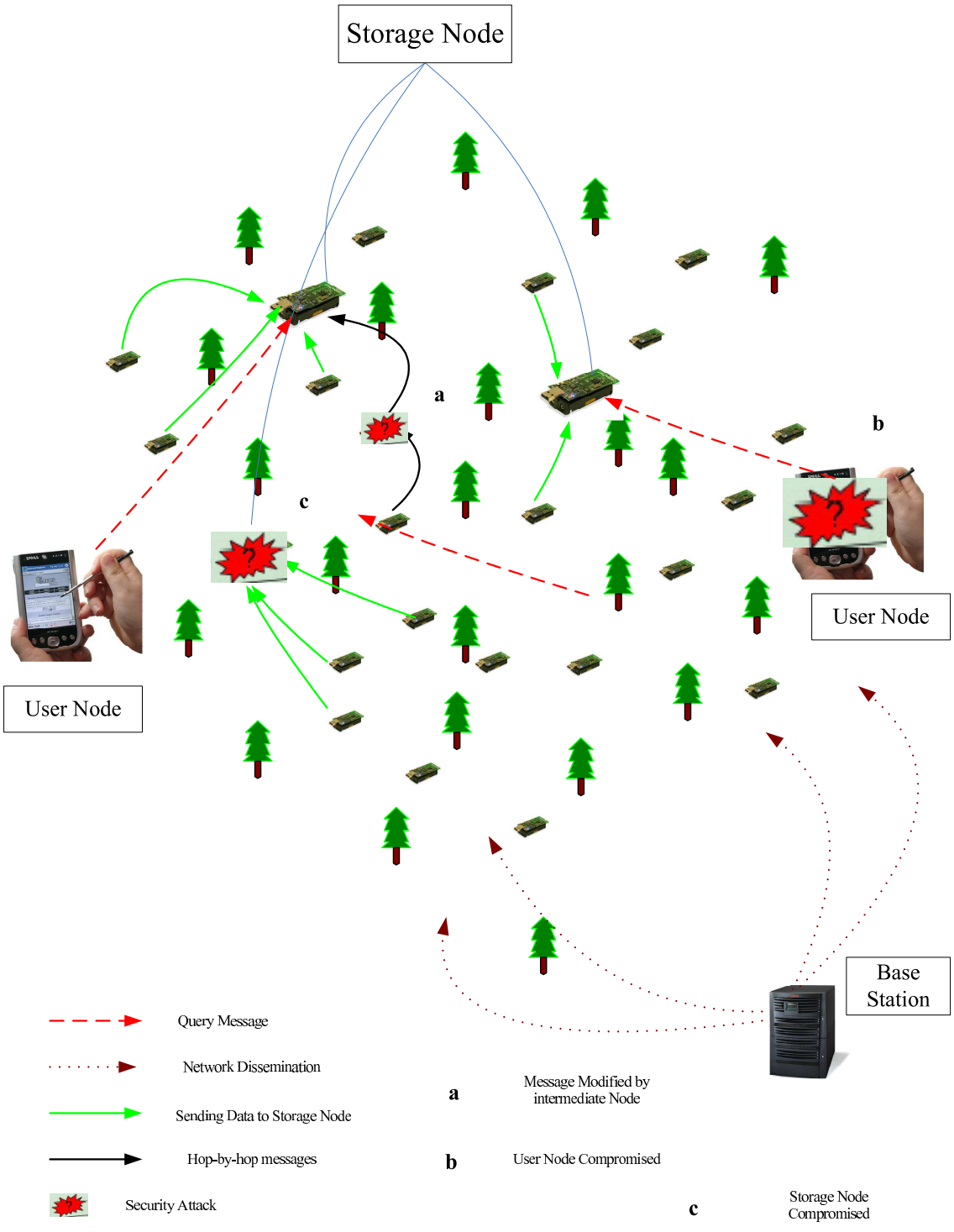

Figure 1.1 Application Scenario describing the problem.

1.2 Research Objectives

In order to address the afore-identified problems, this thesis work has the following objectives:

- Designing a novel scheme, called *DKVP*, to protect data confidentiality and integrity in the presence of User Node compromise
- Designing, implementing and evaluating (through experiments) a prototype system for resilient protection, data confidentiality and Integrity
  - Implementing *the adaptive polynomial based (APB) scheme* [7], the DKVP scheme and *the message authentication function (MAF) scheme* [5].
  - Providing effective and friendly interfaces to application developers. The system hides all the internals of the security protocols, and facilitates the application developers to include security features in the applications developed by them.
  - Developing a test-bed (including both security protocol implementations and various test problems) and using the same to evaluate the protocols for their feasibility and performance.

1.3 Research Challenges and Methodologies

In conducting this thesis work, we have met and addressed the following challenges:

- Efficiency: Due to stringent resource constraints in sensor networks, security protocols should be light-weight and involve as little computation, communication and storage as possible. For example, one of the main issues in sensor networks is that of memory management. With 4KB of RAM and 40KB of ROM, it is important that on integrating several protocols the usage is as low as possible. This is an addendum to the point mentioned above.
- Resilience to attacks: Due to unattended deployment fields and lack of tamper resistance, the designed protocols should be resilient to various kinds of attacks, especially node

compromises. In addition, the security protocols have to be flexible in the sense that, on detecting node compromises, the system should be properly updated with fresh security parameters.

- Ease of integration: Due to the possibility of coexistence of multiple attacks, security protocols may not be used alone; instead, when adopted into other applications, multiple protocols could be deployed together. Considering this, a protocol should be designed or enhanced to facilitate seamless integration.

- Generalness: The implementation should be easy to adapt to various scenarios. Hence it is important to abstract the features such that, applications need not worry too much about the inherent implementation of the protocols.

To address the challenges of efficiency and resilience to attacks, we adopt the polynomial-based approaches [3] which have been studied for sensor networks in detail in [7, 6]. The protocol design of DKVP, polynomial-based encryption is adopted.

To address the challenges in integrating multiple protocols and providing general, effective and friendly interfaces to application developers, we investigate security protocols and how they should be included in applications, and then conduct the following steps:

- Identify the commonalities in the protocols developed to ensure that there is code re-use to make the implementation efficient

- Abstract the functionalities into interfaces so that the inherent implementation is hidden from the application developer

- Provide an infrastructure to the application developer to choose the security level at which the protocols must be executed to ensure flexibility in implementation. This is achieved by maintaining a configuration file where the initial setup values are registered.

1.4 Research Results

We have designed DKVP protocol, which has satisfactory performance in terms of low storage and computation overhead. The results obtained from PowerTossim indicate that our protocols when executed together use at most 27 msec of processing time and also indicated the use of at most 60% of CPU usage. These results were obtained for very high values of security parameters. Also the protocols used 75% of the ROM for program space, again when executed with highest values of security parameters. Hence for a practical system that uses nominal security parameters, the system would fare very efficiently. During the course of this work we were also able to provide various results which can be used as guidelines for applications that use our protocols.

In addition, we have implemented and integrated three security protocols for protecting data confidentiality and security. These protocols were implemented in nesC and were executed in TOSSIM and telosb motes for evaluation. We used the PowerTossim tool to evaluate the work. We varied various security parameters to analyze the impact of the protocols and hence provide guidelines to indicate, when to use necessary parameters for different hardware architectures. Our experiments show that these protocols, run alone or together, have low computation, communication and storage overhead while satisfying security requirements. We also developed test programs to show how the protocols can be integrated into application programs. As demonstrated, these protocols can be effectively and conveniently integrated into application programs.

CHAPTER 2. RELATED WORK

Wireless sensor networks as mentioned earlier are resource constrained. There have been various security solutions to address different aspects of the network. Unlike traditional networks, strength of the security protocol is not the only metric when used in sensor networks, it is also important that the security protocol is feasible for the sensor network. For example, it is not advisable to use *Public-key cryptography* in Wireless sensor networks. This is because of the complexity and the computation overhead this would cause. For this purpose authors of [7] and [6], had studied and designed protocols using *Polynomial Base encryption*, [3].

Polynomial based encryption works on the philosophy of pseudo-public-key cryptography. Here there is an authentication server that generates a polynomial. Depending on number of interacting entities the degree of the polynomial is set. For example if the interactions is always dual then the degree could be set to 2. The authentication server evaluates one of the variables of the polynomial and sends the coefficients to the interacting nodes. Based on the identifiers of the interacting entities, the encryption key and decryption key is evaluated. This can be illustrated as shown in Figure. 2.1. Here Authentication Server generates a 2-variable polynomial of degree 2, *f(x,y)*. It then disseminates the *f(x,node_id)*. For the sake of illustration we use *f(x,node_id)* for encryption and *f(node_id,y)* for decryption. This is not a convention. When a particular node u has to send data to v it evaluates *f(u,v)* and encrypts the data with the evaluated value and sends the encrypted data to v. On receiving the decrypted data node v evaluates *f(u,v)* and decrypts the data using that.

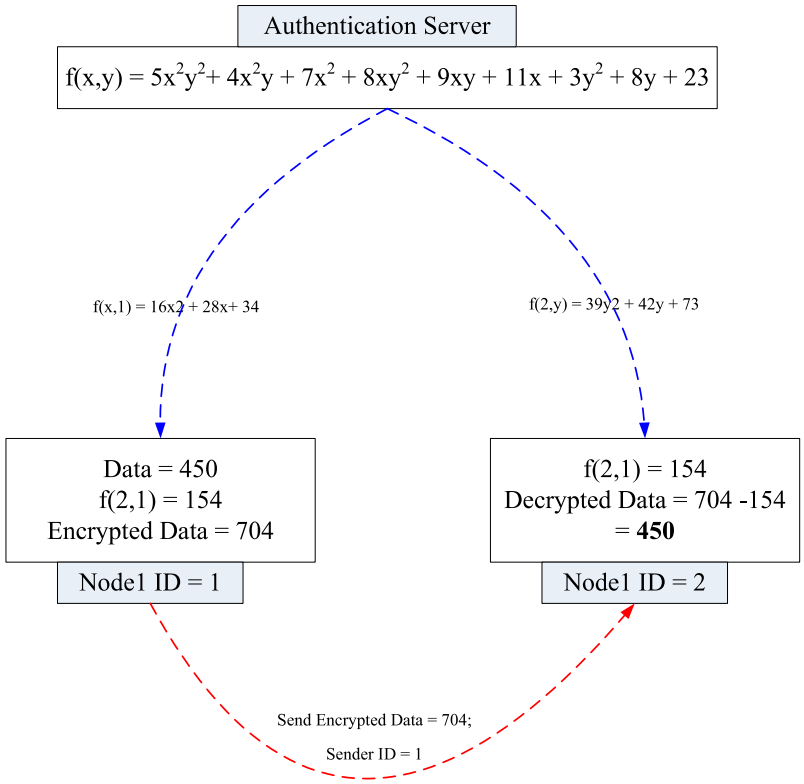

Figure 2.1 Illustration Example for Polynomial Based Encryption.

2.1 Node Revocation schemes

For the purpose of designing and developing DKVP, we surveyed a lot of work on Node revocation schemes. Our main goal was to analyze the schemes to ensure that, when adopted they will be feasible for wireless sensor networks. The problem of revoking privilege and denying access has been studied in other scenarios but not in sensor networks. Some of the existing solutions take the explicit revocation approach, in which a revocation list is broadcast to all the related entities such that they are aware of the entities which should be revoked and others take the implicit approach with group-keys that are updated at all entities except those whose privileges have to be revoked.

Group key based revocation is an efficient method to revoke users as studied in [14]. Here the authors propose a scheme to distribute a group decryption key that is shared by n users in a group so that all but d users are revoked. The scheme uses a *Leader node* that distributes revocation group keys based on the system parameters. The main disadvantage of this scheme is that it is effective only when the key length is high of the order of 1024 bits, etc. This may not be a very viable option for sensor networks.

In [13], the authors propose a novel mechanism to revoke users by generating polynomials based on the list of revoked users. In this scheme, on obtaining the ID(s) of the nodes that needs to be revoked, a list is prepared and hence a revocation polynomial based on the list. The authors also introduce a concept of *group-manager* which generates a key based on 2 polynomials, say key-polynomials. The key-polynomials are embedded with the revocation polynomial such that when a node that needs to be revoked obtains the key-polynomial, it does not retrieve the original key. The essence of this mechanism is been adopted and modified to design $DKVP$, which will be explained later. [10], [8] and [9] were few of the works that we referred to analyze the solutions for node and key revocations. These techniques could not be adapted directly into sensor network applications as they address different scenarios and problems.

2.2 Confidentiality and Integrity Protocols

An integrated suite of products for the purpose of security infrastructure, should have several individual components. In our case we implement two such protocols, namely APB [7] and MAF [5]. Few of the parameters that will be used all through the protocols are

2.2.1 APB - Adaptive Polynomial Based Scheme for Confidentiality

This scheme addresses the security issue of confidentiality. Here the network controller disseminates a polynomial periodically, which gives its name of being adaptive. In this protocol there are mainly three communicating entities - *Network Controller, Storage Node and User Node*. The network controller disseminates a network-wide polynomial which enables encryption and decryption. The encryption and decryption keys are based on the network-wide polynomial.

- F_q : A finite field over which all the coefficients are used.
- l : The total number of bits which determines the F_q.
- r : The number of bits which are removed from the evaluated polynomials, remainder of which will be used as key.

The protocol has mainly has 3 parts as described below -

- *Dissemination of Network-wide Seed Polynomial*

  To disseminate the network-wide seed polynomial, the seed polynomial is perturbed. The network controller firstly generates a polynomial which is used for encryption, let's call it $f(x, y)$. This encryption polynomial is perturbed with another polynomial say $p(x, y, z)$ where x is substituted with the storage node id, y is substituted with dissemination interval and z is substituted with perturbation interval. Here the dissemination interval is less than perturbation interval. For all practical reasons, the perturbation interval is divided into several slices of dissemination interval units. The perturbation polynomials

are generated as

$$\overline{p_u(y,z)} = p(u,y,z) - \alpha_u$$

These coefficients of $p_u(y,z)$ are evaluated at the storage node.

Based on the encryption polynomial and perturbation polynomial, a network-wide seed polynomial is generated as follows:

$$w(x,y) = f(x,y) + p(x,y,v)$$

where v is the perturbation interval.

The network-side polynomial is disseminated to the storage nodes where the key to encrypt is generated. Upon receiving $w(x,y)$ the storage node derives its own seed polynomial:

$$\overline{f_u(y)} = w(u,y) - \overline{p_u(y,v)}$$

Figure 2.2, which featured in [7], describes the dissemination part of the protocol as an illustration.

$\overline{f_u(y)}$ the storage node evaluates the keys for various intervals says i_s to i_e. For a phase i and initial dissemination interval, the key $K_{u,i,0}$ is constructed by evaluating $\overline{f_u(i)}$ which was obtained at the first dissemination interval. The data that needs to be securely stored is encrypted with $l-r$ bits of $K_{u,i,0}$. The hash of the key that was used to encrypt is also stored along with the encrypted data. On securely encrypting the data, the key that was used is erased from the mote.

- *Securing Data Retrieval* The user nodes (querying entities) are preloaded with $\underline{f_i(x)}$ which is derived as:

$$\underline{f_i(x)} = f(x,i) + \beta_i$$

When a data is required the user node queries a storage node, the storage node sends the encrypted data along with the hash of the key that was used to encrypt the data. On receiving the data and the hash, the user node obtains the key by evaluating $f(u,i)$. On evaluating the key it checks for 3 combinations to see which could have been used to

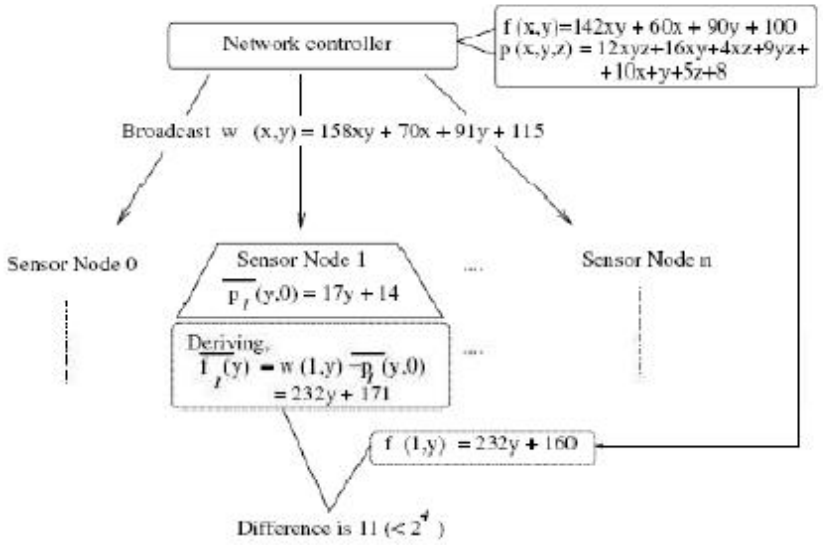

Figure 2.2 An Example of Disseminating A New Seed Polynomial (r = 4, z = 0).

determine which key was used. The key could be one of $(l-r)$ bits of $f(u,i)$, $(l-r)$ bits of $f(u,i) + 2^r$ or $(l-r)$ bits of $f(u,i)$ - 2^r.

The security analysis and detailed proofs can be obtained from [7].

2.2.2 MAF

This scheme addresses the problem of integrity in data management. Unlike APB the protocol here is not adaptive and the Network controller does not disseminate security information periodically. In this protocol there are mainly 2 entities with respect to the security aspect, namely - *Network Controller and Sensor node.* The *Network Controller* disseminates the unique identifier to the sensor nodes along with the necessary polynomials. For all practical purpose this part of the Network controller's involvement can be eliminated by pre-loading all the security information, but for the sake of reducing initialization setup and not using the flash, we implement the additional communication between the Network controller and sensor node. The network controller constructs a 3-variable polynomial f(x,y,z) which will be used as

verification and authorization. The security analysis and the theoritical analysis of this work can be found in [5]. This protocol introduces a new security parameter γ for the purpose of generating an ID space.

- F_q : A finite field over which all the coefficients are used.
- l : The total number of bits which determines the F_q.
- r : The number of bits which are removed from the evaluated polynomials, remainder of which will be used as key.

This protocol can be divided into 4 phases as described below:

- *Constructing a perturbation polynomial for message verification and authentication purpose, and ID space for sensors [5]:* The Network controller randomly picks a d_x variable polynomial $\alpha(x)$ over F_q. Based on the $\alpha(x)$ F_q is divided into $2^{l-(r-\gamma-1)}s$ sets

$$S_i = \{x | x \in F_q, \alpha(x) - i * 2^{r-\gamma-1} \in 0, ..., 2^{r-\gamma-1} - 1\}$$

for i = 0,.....,$2^{l-(r-\gamma-1)}s$ Let S_k be the largest set among all the S_i's. Let I_s be the largest set S_k and use $\overline{\alpha(x)}$ given by:

$$\overline{\alpha(x)} = \alpha(x) - k * 2^{r-\gamma-1}$$

$\overline{\alpha(x)}$ is used for the verification purpose and I_s is used to choose the identifier u_s. Similar to S_i, another polynomial R_i is generated with another polynomial $\beta(y)$, such that:

$$R_i = \{y | y \in F_q, \beta(y) - i * 2^{r-\gamma-1} \in 0, ..., 2^{r-\gamma-1} - 1\}$$

for i = 0,.....,$2^{l-(r-\gamma-1)}s$ Similar to S_k another set $R_{k'}$ is chosen. Let I_r be the largest set $R_{k'}$ and use $\overline{\beta(y)}$ given by:

$$\overline{\beta(y)} = \beta(y) - k' * 2^{r-\gamma-1}$$

$\overline{\beta(y)}$ is used for the verification purpose and I_r is used to choose the identifier u_r.

- *Node Initialization:* The Network controller, preloads each sensor node u with the following information as mentioned in [5].
  - a unique sender ID, u_s where $u_s \in I_s$
  - a unique receiver ID, u_r where $u_r \in I_r$
  - a polynomial $auth_u(y, z)$ for authenticating outgoing messages where

$$auth(y, z) = f(u_s, y, z) + r_{u,0} * \overline{\beta(y)} + r_{u,1}$$

    $r_{u,0}$ is randomly picked from $0, ..., 2^{\gamma}$ and $r_{u,1}$ is randomly picked from $0, ..., 2^{r-2}$
  - a polynomial $verf_u(x, z)$ for authenticating outgoing messages where

$$verf(x, z) = f(x, u_r, z) + r'_{u,0} * \overline{\alpha(x)} + r'_{u,1}$$

    $r'_{u,0}$ is randomly picked from $0, ..., 2^{\gamma}$ and $r'_{u,1}$ is randomly picked from $0, ..., 2^{r-1}$
  - secure one-way hash $h(.)$ and
- *Messaging Sending at Senders* When a sender u wants to send a message m, it constructs a message authentication function:

$$MAF_{u,m}(y) = f(x, u_r, z) + r'_{u,0} * \overline{\alpha(x)} + r'_{u,1}$$

  The Sender sends $\langle \mathbf{m}, \mathbf{MAF}(\mathbf{y}), \mathbf{u_s} \rangle$ to the receiver nodes.
- *Message Verification at Receivers:* On receiving the message m and MAF(y) the verifying node first evaluates the hash of the message by calling $h(m)$. It then evaluates $verf_u(u_s, h(m))$ and $MAF(u_r)$. If $verf_u(u_s, h(m))$ - $MAF(u_r) \in 0, ..., 2^{r-1}, q - (2^{r-1})....q$ then the message is verified. The proof of this can be found in [5].

An illustration of this can be found in Figure 2.3 adapted from [5]

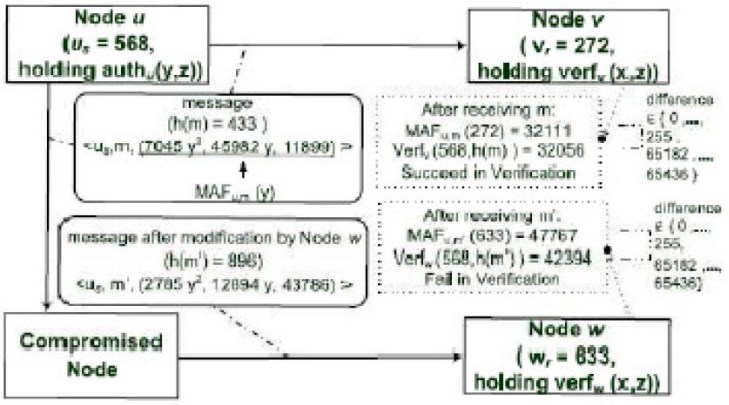

Figure 2.3 Illustration of the Integrity protocol.

CHAPTER 3. PROTECTING DATA INTEGRITY AND CONFIDENTIALITY AT USER NODE

3.1 Background

As discussed earlier, the APB scheme protects data at the storage node by periodically updating the keys. In this scheme the user can periodically query the storage node. The user node is pre-loaded with a polynomial, which can be used to generate the key to decrypt the data that was retrieved from the storage node. The *APB* scheme however, does not deal with malicious *User Nodes*, that is, if a user node is compromised, the keys that it has cannot be revoked and hence it has access to all the data generated at the storage node. To address this limitation of the *APB* scheme we propose a novel mechanism to solve the problem. Our solution is comprised of four components - Protection of vulnerable data at the storage node, Protection of vulnerable keys at the storage node, Key updating to innocent user nodes and data retrieval by innocent user nodes. In our scheme, once a compromised user is detected, a revocation process is invoked. Firstly, a polynomial to re-encrypt the data is generated and disseminated to the storage nodes. On receiving the polynomial the storage nodes generate their own perturbations to encrypt the data/keys. The re-encryption polynomial is selectively broadcast to user nodes other than the revoked users. The process mentioned above ensures that the data can be secured from malicious user-nodes.

3.2 DKVP

3.2.1 Network Assumptions

We consider a sensor network that is composed of a large number of sensor nodes, each of which has a unique ID, and a network controller (NC), which may be online or offline. Each sensor node has a certain capacity to store sensor data. The sensor network adopts the distributed data storage and retrieval type approach for data management. Specifically, once a sensor node has generated some data, the data are stored locally or at some designated storage node. The stored data can be retrieved by authorized users. Similar to the APB scheme, we assume that data are retrieved based on time for simplicity, though the scheme can be extended to support more complicated modes of data retrieval. In addition, sensor nodes maintain loose time synchronization. The lifetime of the network is divided into phases, denoted as Phase 0, Phase 1, and so on, where each phase has the same time duration denoted as τ.

3.2.2 Security Assumptions

We assume that the Network controller is trustworthy and cannot be compromised. Any sensor node is originally innocent before deployment, but it may get compromised after deployment. Users can be compromised and the compromise of user nodes can be detected eventually; further, user compromise will not happen frequently. Because of potential large number of sensor nodes, we do not assume the detection of sensor node compromise. Also, all user nodes w not know the identity of the other user nodes. Once a (sensor or user) node is compromised all the information stored in the node can be read out by the attackers. Furthermore, the compromised nodes can be reprogrammed and thus fully controlled by the attackers. The adversary can pose several threats for the system: i) all data encrypted in the past time phases for which a compromised user is authorized to access will be exposed the adversary; ii) the data that will be generated in the future time phases for which a compromised user is authorized will also be open for the adversary.

Based on the assumptions we consider the following types of attacks:

- Outsider attacks: They are launched by attackers that have not compromised any sensor or user nodes and therefore do not know any secret of the network. In particular the attackers may overhear the communication channel and try to compute the secrets used in the network. The analysis of the APB scheme [7] has shown that the success of such attacks will require prohibitively high overhead. Hence we will not consider such attacks in this thesis.
  - *[*type-I attacks] do not permit the information broadcast to the user node or to the sensor node to actually reach the desired destination.
- Insider attacks: They are launched by the attackers that have compromised a few sensor or user nodes and therefore know some secrets preloaded to these compromised nodes. Also, if the insider is a sensor node, then it can always receive valid information being broadcast. Insider attackers can also launch the attack. In addition
  - *[type-II attacks]* collect all the secret owned by the compromised nodes, and attempt to derive secret held by innocent node.
  - *[type-III attacks]* collect all the secrets owned by the compromised storage and user nodes and try to collude, eventually decrypt the data stored.

3.2.3 Design Goals

Our scheme is designed with the following goals.

- Confidentiality: The data that is stored in the sensor node cannot be decrypted by the adversary even if the user nodes are compromised or the sensor nodes are compromised.
- Authorization: Only authorized user nodes can access the data or shares stored; once a user node is compromised it will not be able to access the data or share from other innocent nodes.
- Efficiency: Even after implementing our protocol the overhead induced is low and acceptable.

In this thesis, the above design goals are achieved via accomplishing the following more specific objectives:

- An innocent user can only retrieve the data that he/she is authorized to access.
- A compromised user should be deprived of the access privilege that he/she was granted. In other words, it includes:
  - A compromised user should be prevented from retrieving data that have been generated in the past.
  - A compromised user should be prevented from retrieving data that have not been generated yet but had been given access privilege before being compromised.
  - Completely revoke the compromised user, such that the secrets revealed will be of no use to the adversary.
- Colluding compromised user and sensor node should not be able to fetch protected data either from innocent sensor nodes or from compromised sensor node.

3.2.4 Proposed Privilege Deprivation Scheme

3.2.5 Overview

We assume some users have been compromised, and these compromised users have been granted the privileges to access the data that were or will be generated during certain time phases (call *risky time phases*) thereafter:

- For any past risky time phase, the data associated with it becomes vulnerable and therefore are called *vulnerable data*. If sensor nodes are not aware of these user nodes being compromised, the user nodes can still retrieve the vulnerable data; if some sensor nodes are compromised, the vulnerable data stored in these nodes can be retrieved since the compromised nodes can derive the keys and decrypt the data.
- For any future risky phase, the data associated with it have not been generated yet, but the keys associated with it may have already been generated. These keys become

vulnerable and therefore are called *vulnerable keys*. Note that, if sensor nodes still use the vulnerable keys to encrypt data, the data can still be retrieved by compromised users.

Protecting vulnerable data and keys is important to make the system secure.

After the compromise of some users has been identified, the network controller (NC) will launch a revocation process. The process consists of three steps:

- First, the network controller (NC) prepares a polynomial to be used for vulnerable data and/or vulnerable keys.
- Second, the polynomial is securely broadcast to all sensor nodes, and each sensor node that receives it then derives certain perturbation number or perturbation polynomial to protect the vulnerable data and/or keys it has.
- Third, the polynomial is securely broadcast only to innocent users, and based on the received polynomial innocent users can update the information necessary for retrieving the data they are authorized to.

3.2.6 Preparation at Network Controller

The functionality of the deprivation scheme is enabled, by preparing the system with the following: The network controller arbitrarily constructs a two-variable polynomial $S(x, y)$ called broadcast perturbing polynomial over a finite field F_q, where q is a large prime number. We define three other integers l, r, and t, where l is the smallest integer such that $2^l > q$ (every element of the finite field F_q can be represented with l binary bits), r is smaller than l, and t represents the degree of all the polynomials constructed by the network controller (NC).

Before each sensor node is deployed,

- a unique ID v is assigned to every node, where v is an element of F_q
- the node v is preloaded with a perturbed share of $S(x, y)$, i.e.,

$$\widehat{S(v, y)} = S(v, y) - \beta_v,$$

where β_v is an element of F_q randomly picked from $\{0, \cdots, 2^{(r-2)} - 1\}$.

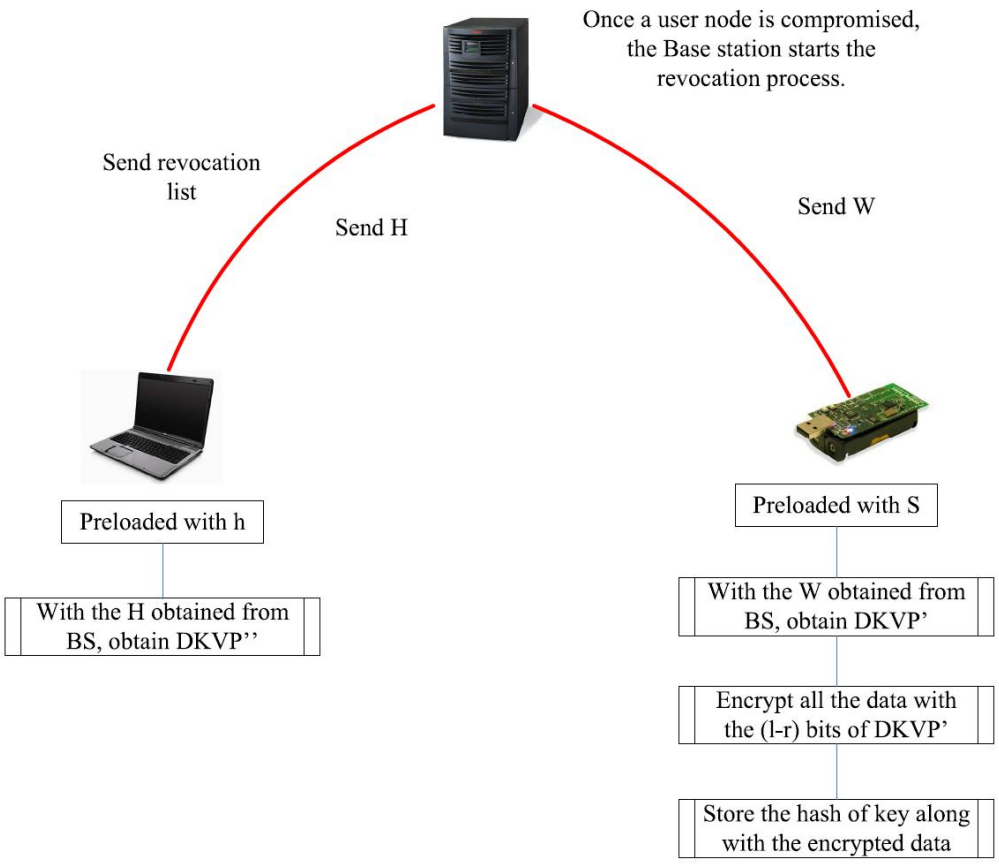

Figure 3.1 Working of DKVP.

When the sensor node is deployed, node v derives m (system parameter) perturbed shares of $S(v, y)$:

$$\begin{aligned} S'_{1,v} &= \widehat{S(v,1)} - \gamma_1, \\ S'_{2,v} &= \widehat{S(v,2)} - \gamma_2, \\ &\cdots \\ S'_{m,v} &= \widehat{S(v,m)} - \gamma_m \end{aligned}$$

where $\gamma_1, \gamma_2, \cdots, \gamma_m$, are elements of F_q randomly picked from $\{0, \cdots, 2^{(r-2)} - 1\}$. $\widehat{S(v,y)}$ is removed after the node derives these shares.

Suppose compromise of some users has been detected, the network controller (NC) prepares a revocation list. Let the immediate risky time phase be j (also can be referred as, the total number of revocations so far), for $1 \leq j \leq m$. The revocation list R_j contains list of compromised users between two subsequent risky time phases, $j-1$ and j. Then R represent,

$$\begin{aligned} R &= \{R_1 \cup R_2 \cup \cdots \cup R_j\} \\ &= \{w_1, w_2, \cdots, w_p\} \end{aligned}$$

where w_i represents the ID of a compromised user, for $i = 1, 2, \cdots, p$, and p is the total number of compromised users at the starting of risky time phase j.

Once a user node is compromised, the network controller arbitrarily constructs a univariate polynomial called *vulnerable data/key protection (*$DKVP_j(x)$*)* polynomial over F_q, which provides the basis for:

(i) protecting vulnerable data at sensor nodes,

(ii) protecting vulnerable keys at sensor nodes,

(iii) key updates at innocent users,

(iv) enabling data retrieval at innocent users.

The network controller arbitrarily constructs a uni-variate polynomial $h(y)$ called selective broadcast perturbing polynomial, over F_q. Before a user node is deployed:

- a unique ID u is assigned, u is an element of F_q
- it is preloaded with $h'_u(= h(u) - \omega)$, where ω is randomly picked from $\{0, \cdots, 2^{(r-1)} - 1\}$..
- it is preloaded with perturbed set of all $DKVP$ polynomials, i.e, $\{\underline{DKVP_1(x)}, \underline{DKVP_2(x)}, \cdots, \underline{DKVP_e(x)}\}$.
- it is preloaded with R.

3.2.7 Protection of Vulnerable Data at Sensor Nodes

In order to protect the data at the sensor nodes, the polynomials have to be securely broadcast to the sensor nodes first, followed by the sensor nodes performing encryption of the data. The network controller, on generating $DKVP_j(x)$ for any risky time phase j, executes the following two-phase procedure to securely broadcast the perturbed shares to all the sensor nodes:

(i) *Construction of network-wide dissemination polynomial.*

To hide $DKVP_j(x)$ from the adversary, it is perturbed with $S_j(x)$, the network controller constructs a $network - wide$ dissemination polynomial $W_j(x)$ as given below,

$$W_j(x) = DKVP_j(x) + S_j(x) + \alpha_j$$

where α_j is an element of F_q randomly picked from $\{0, \cdots, 2^{(r-1)} - 1\}$.

(ii) The network controller broadcasts $W_j(x)$ to all sensor nodes.

Upon receiving the $network - wide$ dissemination polynomial $W_e(x)$ (for risky time phase e), the sensor node v executes the following steps to protect vulnerable data:

(i) *Computing Perturbed share of DKVP polynomial.*

The Node v executes the following procedure using the stored secret $S'_{e,v}$,

$$\begin{aligned}\overline{DKVP_e(v)} &= W_e(v) - S'_{e,v} \\ &= (DKVP_e(v) + S_e(v) + \alpha_e) \\ &\quad -(S_e(v) - \beta_v - \gamma_e) \\ &= DKVP_e(v) + \alpha_e + \beta_v + \gamma_e\end{aligned}$$

From the previous definitions we can derive that, $\alpha_e + \beta_v + \gamma_e \in \{0, 1, \cdots, 2^r - 1\}$. The network controller uses $S_j(x)$ and α_j to secure the broadcast channel from: (a) an adversary gaining access to $DKVP_j(x)$ polynomial, and (b) an intermediate sensor node from obtaining access to shares of other sensor nodes. This allows the $DKVP$ shares to be different for each sensor node.

(ii) *Encrypting the vulnerable data.*

Let the vulnerable data at sensor node be $Data_j, \forall j = 1, 2, \cdots, e$, where e is the current time period. To protect the vulnerable data, sensor nodes encrypt the data using the newly received shares of $DKVP$ as follows:

$$\begin{aligned}Data'_j &= Data_j + [l-r]\overline{DKVP_e(v)}, \\ &\forall j = 1, 2, \cdots, e\end{aligned}$$

where $[l-r]\overline{DKVP_e(v)}$ represents the decimal equivalent of left most $(l-r)$ binary bits of $\overline{DKVP_e(v)}$. After m revocations, the data stored at the sensor node will take the form as given below,

$$\begin{aligned}Data'_j &= Data_j + \Sigma_{k=1}^{m}[l-r]\overline{DKVP_k(v)}, \\ &\forall j = 1, 2, \cdots, e\end{aligned}$$

(iii) *VD Store-and-Erase.*

The sensor node maintains a 1-bit variable $VD_j - bit$, to identify whether or not, the

data $Data_j$ is protected with $DKVP$-shares. $VD_j - bit$ is 0 by default, and set to 1 upon execution of step(ii). To enable data retrieval at the user node, the sensor node also maintains $hash(Data_j)$ and $hash([l-r]\overline{DKVP_k(v)})$ for all $j = 1, 2, \cdots, e$ and $k = 1, 2, \cdots, m$ respectively. The $hash(.)$ is a security hash function which generates the message authentication code (MAC) for the input. After protecting the vulnerable data, the sensor node erases $\overline{DKVP_k(v)}$, and $S'_{j,v}$, to prevent adversary from retrieving protected data on compromising the same.

Example I.

Let system parameters be $l = 8, r = 3, t = 2$. Suppose some user compromise has been detected for any risky time phase $e = 10$. NC constructs new DKVP polynomial $DKVP_{10}(x) = 3x^2 + 4x+7$. Along with $DKVP_{10}(x)$, the broadcast perturbing polynomial $S_{10}(x) = 19x^2+18x+27$, and the arbitrarily chosen random numbers $\alpha_{10} = 1$, NC constructs $W_{10}(x) = 22x^2 + 22x + 35$ and broadcasts to all sensor node. Let us consider sensor $v(= 1)$ receives $W_{10}(x)$. Sensor node v uses the disseminated polynomial and the preloaded secret $S'_{10}(v = 1) = 60$ to derive $\overline{DKVP_{10}(1)} = W_{10}(1) - S'_{10}(1) = 19$. Let the vulnerable data at the sensor node v be, $Data_{10} = 44$. The left most $(l - r)$ bits of $\overline{DKVP_{10}(1)}$, i.e, $[8 - 3]\overline{DKVP_{10}(1)} = 16$ is used to protect $Data_{10}$, such that, $Data'_{10} = Data_{10} + 16 = 60$ is replaced for $Data_{10}$ and VD-bit is set to 1. Sensor node stores $hash([8 - 3]\overline{DKVP_{10}(1)}) = hash(16)$. Sensor v then erases $W_{10}(v), W_{10}(x), S'_{10}(v)$.

3.2.8 Protection of Vulnerable Keys at Sensor Nodes

Protecting vulnerable keys is an extension of the previous section. The network controller does not perform any additional functionality from what has been mentioned in the previous section. However, the sensor node executes a few additional steps to protect the vulnerable keys. The steps that do not repeat from the previous section are:

(-) the network controller broadcasts the $network - wide$ disseminating $W_j(x)$ (as in Sec. 3.2.7),

(-) the sensor node compute the perturbed share of $DKVP$ polynomial (as in Sec. 3.2.7),

The variation in the method from the previous section is as given below:

(i) *Encrypting the Vulnerable keys.* Let the vulnerable key at sensor node be $K_j, \forall j = e+1, \cdots, m$, where e is the current time period or current risky time phase. To protect the vulnerable key, the sensor node encrypts the key using the newly received shares of $DKVP$ polynomial as follows:

$$K'_j = K_j + [l-r]\overline{DKVP_e(v)},$$
$$\forall j = e, e+1, \cdots, m$$

At the end of m revocations, the protected key stored at the sensor node will be as given below,

$$K'_j = K_j + \Sigma_{k=1}^{m}[l-r]\overline{DKVP_k(v)},$$
$$\forall j = e+1, \cdots, m$$

(ii) *VK Store-and-Erase.*

The sensor node maintains a 1-bit variable $VK_j - bit$ for each key K_j. $VK_j - bit$ is 0 by default, and set to 1 upon execution of step (i). Once a key K_j is used to encrypt the data $Data_j$, the key K_j is erased and the corresponding $VK_j - bit$ is set to 1. This step, enables users to identify during retrival of data, whether or not the key used to encrypt any data is protected with $DKVP$-shares.

Example II.

As shown in Example. I, the sensor $v = 1$ receives $W_{10}(x) = 22x^2 + 22x + 35$. Using $S'_{10}(v=1)$, v derives $\overline{DKVP_{10}(1)} = 19$. Say sensor node v holds the vulnerable key, $K_{11} = 20$. Similar to Example I, $[8-3]\overline{DKVP_{10}(1)} = 16$ is used for protecting K_{11}. After encryption, $K'_{11} = K_{11} + 16 = 36$ is replaced for K_{11} and VK-bit is set to 1.

3.2.9 Key Updating at Innocent Users

The user nodes should receive new keys after the network controller (NC) identifies a compromised user. A secure mechanism to illustrate the update of keys is described here. Suppose some user compromises has been detected, as shown in Sec. 3.2.7, NC generates a $DKVP_j(x)$ for any risky time phase j and it is also responsible to distribute the newly generated $DKVP_j(x)$ to the users.

To enable the functionality of deprivation scheme, it is important to selectively broadcast the polynomial only to innocent users. Selective broadcast prevents the compromised users from knowing the new $DKVP_j(x)$. In the literature, Liu et.al [13] have proposed a threshold based selective broadcast scheme (call S-BCast scheme).

The deprivation scheme uses a variant of $S-BCast$ $(Enh-S-B-Cast)$ scheme to satisfy the security requirement mentioned above.

(i) *Overview of S-BCast Scheme.*

Each user node u is preloaded with a secret polynomial share h_u $(=h(u))$. At some risky time phase j, let the set of compromised users be R. The network controller has to update a new polynomial $DKVP_j(x)$ only to innocent users. The following two-phase procedure is executed by the network controller to selectively broadcast $DKVP_j(x)$ to innocent users.

(1) *Construction of network-wide dissemination polynomial.*

Based on the set R $(=\{w_1, w_2, \cdots\})$, the network controller constructs a polynomial, $L(y)$, as follows.

$$L_j(y) = (y - w_1)(y - w_2)\cdots$$

Based on $L_j(y)$, $DKVP_j(x)$ polynomial, and the selective broadcast perturbing polynomial $h(y)$, the network controller constructs a $network-wide$ dissemination polynomial $H_j(x, y)$, as given below,

$$H_j(x, y) = DKVP_j(x) \times L_j(y) + h(y)$$

(2) The network controller then broadcasts polynomial $H(x, y)$ and R to all user nodes.

When the user node u receives R, it can construct the polynomial $L(y)$. Based on $H_j(x, y)$, h_u and $L(u)$, user u constructs $DKVP_j(x)$ as follows,

$$DKVP_j(x) = \frac{H_j(x, u) - h_u}{L_j(u)}$$

In case $u \in R$, then $L_j(u) = 0$, and $H_j(x, u) = h_u$. This prevents the compromised users from receiving the polynomial $DKVP_j(x)$.

The $S - BCast$ scheme is t-threshold based secure broadcast scheme, where t is the degree of the selective broadcast perturbing polynomial $h(y)$. Since, an adversary can compromise t or more user nodes, $Enh - S - BCast$ Scheme is proposed to make the system more secure.

(ii) *Description of Enh-S-BCast Scheme.*

The main goal of this scheme is to protect $h(y)$ from t or more user node compromises. This goal is achieved by preloading a perturbed share of $h(y)$ at each user node. This scheme is elaborated in detail below:

Suppose a set of compromised users R_j are detected, for any risky time phase j. To enable the innocent users to receive perturbed $DKVP_j(x)$ polynomial, NC executes the following steps -

(1) *Construction of network-wide dissemination polynomial.*

Based on $L_j(y)$, $h(y)$ and $DKVP_j(x)$, the dissemination polynomial $H_j(x, y)$ is constructed as follows,

$$H_j(x, y) = DKVP_j(x) \times L_j(y) + h_j(y) + \psi$$

where ψ is randomly picked from $\{0, \cdots, 2^{(r-1)} - 1\}$.

(2) The network controller then broadcasts polynomial $H_j(x, y)$ and R_j to all user nodes.

When a user node u receives R_j, and $H_j(x, y)$, it executes the following steps to construct $DKVP_j(x)$:

(1) *Update the set R.*

User u initially holds R, which was preloaded by the network controller (as in Sec. 3.2.6) . User updates R as follows,

$$R = R \cup R_j$$

(2) *Construction of $L_j(u)$.*

Based on R, the user u constructs $L_j(u)$ as follows,

$$L_j(y) = (y - w_1)(y - w_2) \cdots$$

$$L_j(u) = L_j(y = u)$$

(3) *Construction of $\underline{DKVP_j(x)}$.*

The user-node is pre-loaded with h'_u, where

$$h'_u = h(y) - \chi$$

where, $\chi in 0....2^{r-1} - 1$ Based on $L_j(u)$, $H_j(x, y)$, and h'_u, the user node u constructs a perturbed $DKVP$ polynomial as follows,

$$\begin{aligned} \underline{DKVP_j(x)} = & \quad \frac{H_j(x,u) - h'_u}{L_j(u)} \\ & = DKVP_j(x) + \frac{\psi + \omega}{L_j(u)} \end{aligned}$$

where,

$$\frac{\psi + \omega}{L_j(u)} \in \{\frac{0}{L_j(u)}, \cdots, \frac{2^r - 1}{L_j(u)}\}$$

In case $u \in R$, then $L_j(u) = 0$, and $H_j(x, u) = h''_u$. This keeps compromised users away from receiving the perturbed polynomial $\underline{DKVP_j(x)}$.

(4) *Store-Erase Phase.*

After obtaining $\underline{DKVP_j(x)}$, the user node erases all the received information. Each user node maintains $L_j(u)$, which helps the user in data retrieval phase (Sec. 3.2.10).

3.2.10 Data Retrieval by Innocent Users

Let us consider a user u who is authorized to query data generated in the past time phase (i.e, any time $\leq e$, for e is the current time phase). The procedure for the user to retrieve data from the sensor include following steps:

(1) *Preloading and updating Secrets.*

Before the user can access the sensor data he is authorized to access, it is preloaded with set of all $DKVP$ perturbed polynomials, i.e, $\{\underline{DKVP_1(x)}, \underline{DKVP_2(x)}, \cdots, \underline{DKVP_m(x)}\}$ (Sec. 3.2.6) at the initialization phase m. During the key update procedure, the innocent users receive set of latest $DKVP$ perturbed polynomials, i.e, $\{\underline{DKVP_{m+1}(x)}, \underline{DKVP_{m+2}(x)}, \cdots, \underline{DKVP_e(x)}\}$ (Sec. 3.2.9) .The user also holds $L_j(u)\ \forall\ j = 1, \cdots, e$. For simplicity, we presume the innocent user nodes has the vulnerable keys (preloaded or updated) used by the sensor (i.e, $K_j\ \forall\ j = 1, 2, /cdots, e$).

(2) *Data Retrieval and Decryption.*

Suppose the user u has sent out a query and received response from sensor v. The data is encrypted and can take one of the following formats:

(a) $< [Data_j],\ j,\ VD_j\text{-bit=0},\ VK_j\text{-bit=0}>$

(b) $\langle [Data_j]'_{K_j} = [Data_j]_{K_j} + \Sigma_{k=1}^{e}[l-r]\overline{DKVP_k(v)}, j, VD_j\text{-bit=1}, VK_j\text{-bit} = 0\rangle$

(c) $< [Data_j]_{K'_j} = [Data_j]_{K_j + \Sigma_{k=1}^{e}[l-r]\overline{DKVP_k(v)}},$
$j,\ VD_j\text{-bit=0},\ VK_j\text{-bit=1}>$

(d) $< [Data_j]'_{K'_j} = [Data_j]_{K_j + \Sigma_{k=1}^{m}[l-r]\overline{DKVP_k(v)}}$
$+\Sigma_{k=m+1}^{e}[l-r]\overline{DKVP_k(v)}$,
j, VD_j-bit=1, VK_j-bit=1>

u also receives $hash(Data_j)$ and $hash([l-r]\overline{DKVP_k(v)})$, where $j = 1, 2, \cdots, e$ and $\forall$ $k = 1, 2, \cdots, e$ respectively.

To identify $\overline{DKVP_k(v)}$ $\forall$ $k = 1, 2, \cdots, e$, the user first evaluates the preloaded decryption polynomial $\underline{DKVP_k(x)}$ at $x = v$ and obtains $\underline{DKVP_k(v)}$. Let us represent $[l-r]DKVP_k(v)$ as Υ. For any k and v, the $[l-r]\overline{DKVP_k(v)}$ must be one of, Υ, or $\Upsilon + 2^r$. From $DKVP_k(v)$, the user computes $[l-r]\overline{DKVP_k(v)}$, and verify the same using $hash([l-r]\overline{DKVP_k(v)})$.

Initially, the user has to derive $DKVP_k(u)$ from $\underline{DKVP_k(v)}$. From Sec. 3.2.9, we know the following,

$$\underline{DKVP_k(x)} = DKVP_k(x) + \frac{\psi + \omega}{L_k(u)}$$

where,

$$\frac{\psi + \omega}{L_k(u)} \in \{\frac{0}{L_k(u)}, \cdots, \frac{2^r - 1}{L_k(u)}\} \tag{3.1}$$

From $\underline{DKVP_k(v)}$, the user can derive $DKVP_k(u)$ as follows,

$$DKVP_k(v) = \underline{DKVP_k(v)} - \frac{\psi + \omega}{L_k(u)}$$

However, $\psi + \omega$ is unknown (since it was picked randomly). Hence, from Eq. (1), $DKVP_k(u)$ must be one of the following,

$$\underline{DKVP_k(v)} - \frac{0}{L_k(u)},$$
$$\underline{DKVP_k(v)} - \frac{1}{L_k(u)},$$
$$\cdots$$
$$\cdots$$
$$\underline{DKVP_k(v)} - \frac{2^r - 1}{L_k(u)}$$

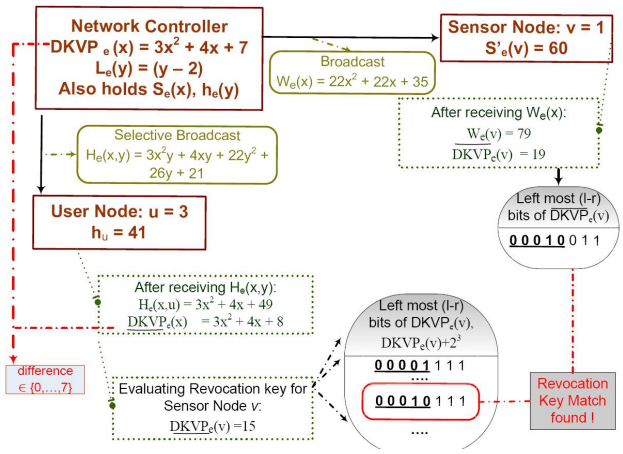

Figure 3.2 Illustration example.

Once the user computes $DKVP_k(u)$, it can extract $[l-r]\overline{DKVP_k(v)}$ for all $k = 1, 2, \cdots, e$. Hence the data can be decrypted.

Fig.3.2 shows an example to illustrate the entire scheme. Let the system parameters be, $l = 8, r = 3, t = 2$. Let the compromised user node be, ID $w_e = 2$ for current risky time phase e. NC constructs new DKVP polynomial $DKVP_e(x) = 3x^2 + 4x + 7$ and let the broadcast perturbing polynomial be $S_e(x) = 19x^2 + 18x + 27$, and the arbitrarily chosen random number $\alpha_e = 1$. The NC constructs $W_e(x) = 22x^2 + 22x + 35$ and broadcasts to all sensor nodes. Based on disseminated polynomial and with the preloaded secret $S'_e(v) = 60$, sensor node $v = 1$ derives $\overline{DKVP_e(v)} = W_e(v) - S'_e(v) = 19$. The left most $(l - r)$ bits of $\overline{DKVP_e(v)}$ is used to encrypt the vulnerable datas or vulnerable keys at the sensor node.

Similarly, with the selective broadcast perturbing polynomial $h(y) = 3y^2 + y + 11$, and $L_e(y) = (y-2)$, NC constructs $H_e(x, y) = DKVP_e(x) \times L_e(x) + h(x) + \psi = 3x^2y + 4xy + 22y^2 + 26y + 21$. Then, NC broadcasts $H_e(x, y)$ to all the user nodes. Although, it is broadcast to all user node, the Enh-S-BCast scheme lets only innocent user nodes to retrieve the $\underline{DKVP_e(x)}$ polynomial. The innocent user node $u = 3$, having $h_u = 41$, can compute $\underline{DKVP_e(x)} = (H_e(x, u) - h_u)/L_e(u) = 3x^2 + 4x + 8$. From the results we can see $\underline{DKVP_e(x)} - DKVP_e(x) \in \{0/L_e(u), 1/L_e(u), \cdots, (2^r - 1)/L_e(u)\} \in \{0, 1, \cdots, 7\}$.

When the user node $u = 3$, needs to decrypt data from sensor node $v = 1$, it calculates $\underline{DKVP_e(v)} = 15$. The user node then identifies, $DKVP_e(v) \in \{\{15, 14, , 8\}, DKVP_e(v) + 2^3 \in \{23, 22, , 16\}\}$. As shown in the Fig.3.2, one of the possible choices matches the left most $(l-r)$ bits of the key used to protect the vulnerable data/keys at the sensor node using which the user can decrypt the data.

3.2.11 Security Analysis

The security analysis for $DKVP$ is similar to APB. In this case, just like APB we classify 2 types of attacks:

3.2.11.1 Attacks without Collusions

In this case we consider user-nodes which are compromised and try to derive the key without colluding with other user nodes. Unlike APB, there is no key update at every interval and hence the user node need not derive any previous keys. The user node will be updated with the latest key till the time it was compromised. On being detected as a compromised node, the base station would disseminate fresh polynomial for re-encryption. The following are the ways of deriving at the key:

- One way of breaking the key is applying a brute-force attack to guess the re-encryption key; the complexity of which will be 2^{64}, since all the operations we perform are on 64-bit keys.

- Another way for the user-node to derive the key is to derive the coefficients of $DKVP(x)$ that is disseminated by the base-station. Since we use randomize function to generate coefficients, which are also 64-bit in length, the coefficients and hence the key is almost impossible to be derived.

- The compromised user node just gets $H(x, y)$ from the Base-station, on substituting the value of its own ID say u in the function, the user-node will end up with $h(u) + \chi$, which will not give the key at any point.

3.2.11.2 Attacks with Collusions

In the case of DKVP, the only thing that can be derived out of collusion is the perturbation polynomial $h(y)$. The reason being, for the user-node that is compromised to derive the master polynomial $DKVP$ it has to know the entire list of the nodes that are being revoked upto the current revocation. This is because since $l(y)$ is an addendum of the previous revocations and the user-node must know all the revocations upto that point. If it does so even then, the complexity to find out an $h(y)$ is $\Omega(2^{(r-2)*(t+1)})$. This is proved by the Theorem 1.

Theorem 1 *Let $h(y) = \sum_{j=0}^{t} B_j y^j$ of degree t. Let us assume that the adversary can collude with n nodes to obtain n shares of $h(y)$, say $h_{u_0}, h_{u_1}, ..h_{u_{n-1}}$, where $u_0, u_1..u_{n-1}$ are IDs of user-nodes. Given t and the shares the complexity to find the coefficients of $h(y)$ is $\Omega(2^{(r-2)*(t+1)})$*

Proof. To find out $h(y)$, the attacker needs to find out its $(t+1)$ the coefficients $B_0, B_1, \cdots, B_t$. Since the adversary knows n shares of $h(y)$, it can obtain the following system of linear equations:

$$h_i = \sum_{j=0}^{t} B_j (i)^j - \chi_i, \ i = 0, \cdots, n-1.$$

Here, $\chi_i \in 0, ...2^{r-1} - 1$, and thus $h_i = h(i) - \chi_i$. B_j $(0 \leq j \leq t)$ and χ_k $(1 \leq k \leq n)$ are unknowns. Therefore the total number of unknowns are $n + t + 1$, while the total number of linear equations is n. Since the number of linear equations are less than the number of unknowns, the unique solution of B_i cannot be found. The only way to solve $h(y)$ is to guess some variables B_j $(0 \leq j \leq t)$ or χ_k $(1 \leq k \leq n)$. Suppose the adversary guesses n_1 coefficients B_j and n_2 random number χ_k. n_1 and n_2 must satisfy $n + t + 1 - (n_1 + n_2) \leq n$. This is because the number of remaining unknowns cannot exceed the number of equations. Therefore, we have $n_1 + n_2 \geq t+1$. Since $\chi_k \in 0...2^{r-1} - 1$, which are shorter than the coefficients B_j. The adversary must choose to guess $t + 1$ of χ_k's. Each χ_k is picked from a set of $2^{(r-2)}$ numbers. Since there is only a unique solution, the expected time complexity to guess the right answer is $\Omega(2^{(r-2)*(t+1)})$.

CHAPTER 4. INTEGRATING SECURITY PROTOCOLS

4.1 Background

Wireless sensor network solutions should have the following features: *scalability, security, reliability, self-healing and robustness.* The main aim of this work is to implement an *Integration Suite*, to be used for the purpose of protecting sensor data. For the sake of integration we use three protocols that was mentioned earlier, namely -

- APB - Securing Data at the storage node.
- MAF - Maintaining the integrity of the data.
- $DKVP$ - Protecting data confidentiality and integrity from compromised user nodes.

As mentioned earlier there are various applications that use wireless sensor network and more importantly the distributed data storage or *in-network* storage feature. Hence the main motivation for this work derives from providing *End-to-End* security solutions for applications that use distributed data storage in WSN.

Designing a suite of features is a challenging task by itself and more so when it has to be done for applications for WSN, which have resource constrained entities. The main goals that were addressed in this work are:

- Provide an integrated security system.
- Provide an abstraction, for applications to easily include and adapt the security suite provided.

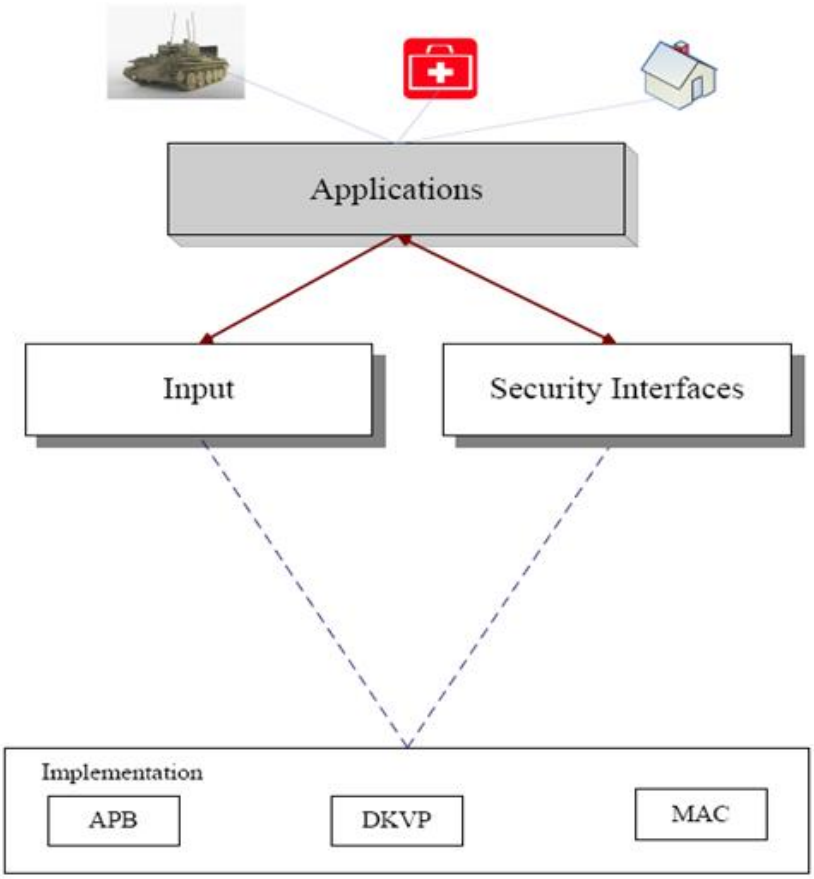

Figure 4.1 Design of the *Integration − Suite*.

- Develop and efficient system which requires as limited resources as possible.
- Make the system flexible to different application scenarios.
- Provide feasibility study before-in-hand, so that applications do not have to go through the effort of implementing the protocol to find this.
- Provide accurate implementation of the protocols that can be used in real-life implementations.

Figure 4.1 shows the main architecture that of the system as a whole. Here the applications that need to use the security suite provide the *Input* based on the requirement and based on

which the necessary interfaces are used. The interfaces need to be abstracted such that the inherent implementations of the underlying protocols are not exposed.

There has been a lot of work on providing security suites in real-life applications like Wireless Ad-Hoc networks, LAN, etc. This is a new approach to provide the same for sensor networks. This integration is an effort to make the system secure and also maintaining the robustness and scalable features of the network. In this section we describe the High-level and low-level design of the implementation.

4.2 High-level Design

Figure. 4.2 shows the high level design for the implementation of APB (MAF and DKVP also has similar interfaces and function calls). Here above the red dotted line is the program that will be developed with respect to the application. We have considered an application where data from the detecting node is encrypted and stored in the storage node. The application developer uses our interfaces for APB and DKVP. The goal to hide the background program and the working from the application developer has been completely achieved and will be discussed in the later sections. From the figure it can be inferred that the application developer is masked from the internal working of the protocols. Most importantly, since our implementation also provides the polynomial generation based on a seed value input by the user is also masked from the user. Figure also shows the different messages that are passed from one node to another. Here we have introduced a way where the message passing and the communication happens over a particular *type* of the message data structure. (*This can be viewed like a server listening on a particular port for a particular application.*)

The implementations of the interfaces for APB, DKVP and MAF have several similar pieces of code which can be combined and presented in the form of the Common functionalities. All of the functionalities of the Common interface have not been listed here. It can be noted that the application developer does not use any of the Common interfaces as shown in the figure. One of the most important goal of this work also has been to separate the common features

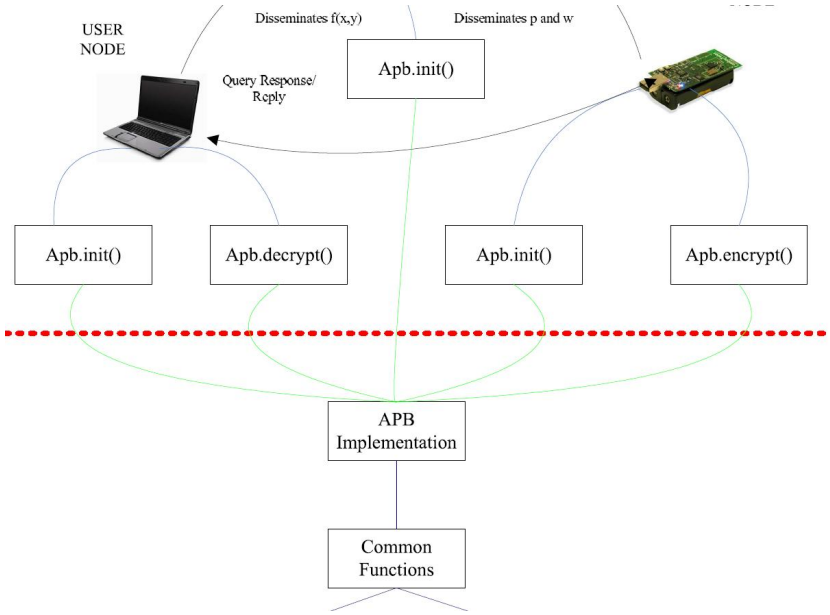

Figure 4.2 High-level Design of the *APB*.

to the protocols, as this will enable any future development of protocols. The other reason to have the Common Interfaces is that in the big picture if an application developer requires more access then all that needs to be done is access the appropriate function in any of the interfaces. It is to an extent an efficient way of modularizing.

The main components of the implementation are

- Input
- Interfaces provided to the Application Developer
- APB, DKVP and MAF
- Common Features

4.2.0.3 Input

The inputs to this implementation from the application developer are:

Security Level	Value of r	Value of l	Degree of the polynomial
1	16	64	15
2	12	56	13
3	8	48	11
4	4	40	9

Table 4.1 Security Level for APB

- Security Level: The security level is the most important input parameter. This is an input so that the various security parameters are hidden from the user. The input is pre-processed and the security parameters are set based on the security level. The Table. 4.1 shows the different values of the security level and their respective system parameters.
- Seed of the polynomials: The user inputs seeds of the various polynomials. This is done to ensure that the same coefficients are generated when this implementation is executed on different nodes.
- Type of the node: The user needs to input which node the program is being loaded into. There is only one program that runs on all the nodes. But we use *ifdef* to check if the current node is a *base station*, *storage node* or *user node*
- Interval for dissemination: The application developer needs to input the interval for sending the perturbation and the seed polynomial, call it $M\_INTERVAL$ and m_int respectively.

4.2.0.4 Interfaces

This section discusses and elaborates on the interfaces that the application developer is exposed to. The application developer uses the interfaces provided for APB, DKVP and MAF. They are as listed below

- *apb.init*: This interface initializes the polynomials that are required in the system. This is the first interface that needs to be made as part of the application. On completion

of the execution of this interface an event *initDone* is thrown. The prototype of this interface is as given below:

command result_t apb.init();

- *event apb.initDone*: This is an event that needs to be implemented by the application developer. This is introduced to prevent the call to the encrypt call before the storage node being received any appropriate polynomials. The signaling of the event is described in the later sections. The prototype of this is as given below:

  event result_t initDone();

- *apb.encrypt*: This interfaces encrypts the data using the key for the interval using the *TinySec*'s primitive interface. The prototype for this is as given below: *command uint64_t apb.encrypt(uint64_t data, int interval);*

- *apb.decrypt*: This interfaces decrypts the data using the key for the interval using the *TinySec*'s primitive interface. The prototype for this is as given below: *command uint64_t apb.decrypt(uint64_t data, int interval, uint64_t hash);*

- *apb.getKey*: This interfaces fetches the key for the interval mentioned from the shared array. As an extended goal we wish to store the keys in the flash and hence this would be read from the flash. This functionality is provided to the application developer, so as to not curtail him from just using *TinySec* for encryption and decryption. The prototype for this interface is as given below:

  command uint64_t apb.getKey(int interval);

The interfaces for DKVP are as given below:

- *dkvp.init*: This interface initializes $S(x, y)$ based on the seed that is inputted by the application developer, at the storage node. At the user node it generates $h'(y)$ and evaluates for the interval that is inputed. The interface has the following prototype:

  command result_t dkvp.init();

- *dkvp.initDone*: This event is signaled when the user node receives a revocation list or when a storage node receives a seed polynomial.

- *dkvp.Send_revok*: Once the base station has detected a compromised user node, it sends a revocation list to the user. This initiates a process at the Base station to send H to the user node and W to the storage node. This is a sequential process starting from sending the revocation list then followed by sending W to storage node and then H to the user node. On receiving the revok list and other polynomials from the base station the storage and user nodes perform the following.

  - At Storage Node: The storage node evaluates $\overline{DKVP_j(v)}$ and encrypts the data stored with this key.
  - At User Node: The user node evaluates $\underline{DKVP_k(v)}$ and stores it locally. Once the encrypted data is obtained it calls decrypt function to obtain the original data.

- *dkvp.get_encrypted_data*: This interface just fetches the data from the structure for a particular query.

- *dkvp.decrypt*: This interface is called to decrypt the data. This is an expensive operation from the basis of the protocol, as there will be $2^(r-1)$ iterations to get the correct key.

The interfaces for MAF are as given below:

- *MAF.init*: For the sake of the MAF protocol we have introduced a data structure MAF_struct. This structure will hold the sender id and the receiver id of every node. At the base station it initializes *f(x,y,z)*. At the node 2 polynomials S_i and R_i are evaluated to obtain u_s and u_r. The node is also initialized with $auth_u(y,z)$ and $verf_u(x,z)$.

- *MAF.authorize*: This interface will be called at the sender which wants to authorize the data. Here the data that needs to be sent is hashed using a hash function. We use TinySec's hash implementation. The authorize interface calls the *evaluate_MAF* function which is inline and is not exposed. The output of authorize interface is the $MAF_{u,m}(y)$.

- *MAF.send_MAF*: This interface internally calls MAF.authorize and generates the MAF. This function just sends the MAF, u_s and the message m.
- *MAF.verify*: On receiving a message of a certain type which is dedicated for this protocol the message is verified using the $verf_u(x,z)$ by substituting u_s for x and h(m) for y.

4.2.0.5 Implementation of APB

The main aim of the implementing APB is to provide the interfaces as mentioned earlier. APB has some system parameters like the degree of the polynomials and, number of storage and user nodes. The degree of the polynomial is pre-processed and derived from the security level that is set by the user. On deriving the degree of the polynomials, say let the degree of the polynomials be $degree_f$ (degree of master polynomial), and $degree_p$ (degree of the perturbed polynomial), and the values of l and r be - $value\_l$ (value of l as part of APB), $value\_r$ (value of r as part of APB). Based on this value we also derive 2 variable Q and q where $Q = 2^l \quad 1$ and $q = 2^r \quad 1$. Given below are the algorithms for each of the interfaces that APB provides. Here in our implementation we assume that the base station is resource-full and need not store the polynomials in the flash. We can also assume that the user node is not always a normal sensor mote and can be a laptop or a PDA.

- Algorithm. 1 gives the algorithm for the *apb.init*() interface.
- Algorithm. 2 gives the algorithm for the *apb.encrypt*() interface.
- Algorithm. 3 gives the algorithm for the *apb.decrypt*() interface.

4.2.0.6 Implementation of DKVP

The main aim of the implementing DKVP is to provide the interfaces as mentioned earlier. DKVP has some system parameters like the degree of the polynomials and, number of storage and user nodes. The degree of the polynomial is pre-processed and derived from the security level that is set by the user. On deriving the degree of the polynomials, say let the degree of the polynomials be $degree_dkvp$ (degree of master polynomial), and $degree_s$ (degree of the

Algorithm 1 apb.init()

if *BASE STATION* **then**
  Declare coefficents of f, p and w. Call it coeff_f, coeff_p and coeff_w
  Generate coeff_p by calling Common.generateCoefficients. Here the seed to the polynomial is as mentioned in the section earlier.
  Evaluate p(x,y,i) where i = 0 .... M_INTERVAL using coeff_p
  Start TimerSendPerturbation such that it fires every M_INTERVAL epoch times
  Start TimerSendSeedPolynomial such that it fires every epoch times
if *STORAGE NODE* **then**
  Declare coefficents of p and p'. Call it coeff_p_storage, coeff_p_prime
  Generate coeff_p_storage by calling Common.generateCoefficients. Here the seed to the polynomial is as mentioned in the section earlier.
  Evaluate p(u,y,i), where u is the ID of the storage node and i ranges from 1 .... M_INTERVAL
  Remove coeff_p_storage. Here we could allocate and de-allocate immediately
  Generate the coefficients of w, as part of the pre-loading process. All the variables are removed after pre-loading the first set of Keys in the storage node in memory say in a variable Keys[m_interval]
if *USER NODE* **then**
  Declare coefficents of f and f'. Here coefficents of f are generated only in the initial set up to preload the polynomial.
  Evaluate f(x,i) where i ranges from 1 to m_interval
Signal initDone()

Algorithm 2 apb.encrypt(uint64_t data, int interval)

Get l-r bits of the Keys[interval], call it current_key
encrypted_data = Call TinySec's Primitive.encrypt(current_key, data)
return encrypted_data

Algorithm 3 apb.decrypt(uint64_t encrypted_data, int interval, uint64_t hash)

decryption_key = evaluate f_interval(storage node id say u)
decryption_key_final = Get the appropritate key based on the hash
decrypted_data = Call TinySec's Primitive.encrypt(decryption_key_final, data)
return decrypted_data

perturbed polynomial), and the values of l and r be - *value_l* (value of l as part of APB), *value_r* (value of r as part of APB). Based on this value we also derive 2 variable Q and q where $Q = 2^l$ 1 and $q = 2^r$ 1. Given below are the algorithms for each of the interfaces that APB provides. Here in our implementation we assume that the base station is resource-full and need not store the polynomials in the flash. We can also assume that the user node is not always a normal sensor mote and can be a laptop or a PDA.

- Algorithm. 4 gives the algorithm for the *dkvp.init*() interface.

Algorithm 4 dkvp.init()

1: **if** *BASE STATION* **then**
2: *Once a user node is compromised, the revocation list and new polynomials needs to be disseminated*
3: *Generate l(y) based on the node ids of the nodes compromised*
4: *Generate H(x,y) based on the coefficients of h(y), DKVP(x) and l(y)*
5: *Evaluate $W_i(x)$ based on DKVP(x) and S(x,i)*
6: *Disseminate H(x,y) to the user nodes*
7: *Disseminate $W_i(x)$ to the storage nodes*
8: **if** *STORAGE NODE* **then**
9: *Generate S(x,y) and evaluate the shares of S(u,i) for all i = 0 ... M_DKVP_INTERVAL*
10: **if** *USER NODE* **then**
11: *Generate h(y)*
12: *Evaluate DKVP_user for M_DKVP_INTERVAL intervals*
13: *Signal initDone()*

- Algorithm. 5 gives the algorithm for the *dkvp.encrypt*() interface. In DKVP for all practical purposes we assume that the user knows the purpose of using the protocol and will call encrypt separately for keys and data as per the convenience. In this regard, all the function will do is generate the key for the current interval and add it to the value of the data that needs to encrypted and store it.

- Algorithm. 6 gives the algorithm for the *dkvp.decrypt*().

Algorithm 5 dkvp.Send_revok(uint8_t *revok_list, int num_of_revok)

1: *Generate the list of nodes that are revoked*
2: *Send the revocation list to user node(s)*
3: *Generate 1-variable polynomial W_temp.*
4: *Based on W_temp generate coeff_W and hence generate DKVP.*
5: *Send the coefficients of W_temp to the storage node(s)*
6: *Based on DKVP generate H based on revocation list and h.*
7: *Send the coefficients of H to user node(s)*
8: *return SUCCESS*

Algorithm 6 dkvp.decrypt(uint64_t encrypted_data, int interval, uint64_t hash_of_key)

1: *Evaluate DKVP"(u).*
2: **for** $i = 0$ to $2^r - 1$ **do**
3: *hash_to_check = hash(DKVP"(u) - i/l(u)*
4: **if** *hash_to_check = hash_of_key* **then**
5: *decrypted_data = encrypted_data - (l-r) bits of (DKVP"(u) - i/l(u))*
6: *return decrypted_datat*

4.2.0.7 Implementation of MAF

The main aim of the implementing MAF is to provide the interfaces as mentioned earlier. MAF has some system parameters like the degree of the polynomials and, number of storage and user nodes. The degree of the polynomial is pre-processed and derived from the security level that is set by the user. On deriving the degree of the polynomials, say let the degree of the polynomials be $degree_maf$ (degree of master polynomial), and $degree_s$ (degree of the perturbation polynomial), and the values of l and r be - *value_l* (value of l as part of APB), *value_r* (value of r as part of APB). Based on this value we also derive 2 variable Q and q where $Q = 2^l$ 1 and $q = 2^r$ 1. Given below are the algorithms for each of the interfaces that APB provides. Here in our implementation we assume that the base station is resource-full and need not store the polynomials in the flash. We can also assume that the other nodes can be of any architecture normal sensor mote or PDA or node is not always a normal sensor mote and can be a laptop or a PDA.

- Algorithm. 7 gives the algorithm for the $maf.init()$ interface.

Algorithm 7 MAF.init()

1: **if** *BASE STATION* **then**
2: *Declare coefficients for f, auth,*α*,* β*,* S_i *and* R_i
3: *Call Common.generateCoefficients to generate 3 variable polynomial f*
4: *Call Common.generateCoefficients to generate a 1 variable polynomial* α
5: *Evaluate* α *for all values between 0 and* F_q *store it in* S_i
6: *Choose the* S_i *with the highest cardinality*
7: *Alot ids to the nodes based on the set* S_i *for senders*
8: *Call Common.generateCoefficients to generate a 1 variable polynomial* β
9: *Evaluate* β *for all values between 0 and* F_q *store it in* R_i
10: *Choose the* R_i *with the highest cardinality*
11: *Alot ids to the nodes based on the set* R_i *for receivers*
12: **if** *NORMAL NODE* **then**
13: *Store the coefficients of* $auth_u(y,z)$
14: *Store the coefficients of* $verf_u(x,z)$
15: *Signal initDone()*

- Algorithm 8 gives the algorithm for authorize interface.

Algorithm 8 MAF.authorize(uint64_t data)

1: *Evaluate h(m) using the TinySec hash function*
2: *Evaluate MAF(y) =* $auth_u(y, h(m)$
3: *Return MAF(y)*

- Algorithm 9 gives the algorithm for *send_MAF* interface.

Algorithm 9 MAF.send_MAF(uint64_t data)

1: *Call MAF.authorize(data)*
2: *Send (MAF(y),* u_s*, data*

- Algorithm 10 gives the algorithm for *Verify* interface.

4.2.0.8 Separting the Common functionalities

Both APB and DKVP have several common functionalities. These functionalities are separated to avoid repetitive code segments in both these implementations. The separation is

Algorithm 10 MAF.verify(uint64_t data, uint64_t MAF[], uint64_t send_id)

1: *maf_value = Evaluate MAF(u_r)*
2: *Evaluate hash of data*
3: *verf_value = Evaluate verf_u(send_id, hash(data))*
4: **if** *(verf_value - maf_value) ¡ 2^{r-1} - 1* **then**
5: *Data not tampered*

such that if any other similar protocol needs to be implemented, then these code segments can be used.

- *Common.generate_coefficients()*: This interface is used to generate the coefficients based on some seed value and the number of coefficients. Algorithm. 11 gives the algorithm for this interface.

Algorithm 11 Common.generate_coefficients(int seed, int degree, uint64_t coeff, int no_of_variables)

1: *srand(seed)*
2: **for** $i = 0$ to $no\_of\_variables * DEGREE_F + 1$ **do**
3: *Coeff[i] = rand() mod Q*
4: *return decrypted_data*

- *Common.evaluate_3_variable_poly_onevar()*: This interface is used to evaluate 3 variable polynomial. This is used mainly for the sed polynomial which is a 3 variable polynomial. Algorithm.12 gives the algorithm for this interface.

- *Common.evaluate_one_var(int value, uint64_t coeff)*

  We use Horner's method to evaluate a uni-variate polynomial. Horner's method works on the philosophy of factorizing and representation of a polynomial. Given below is the representation of the polynomial $p(x)$:

  $p(x) = a_0 + a_1 x + a_2\ x^2 + ... + a_n\ x^n$

  $p(x)$ can be represented as:

  $p(x) = a_0 + x(a_1 + x(a_2 + ... x(a_{n-1} + a_n\ x$

Algorithm 12 Common.evaluate_3_variable_poly_onevar(uint64_t *coeff, uint64_t *pow_1, uint64_t *pow_2, uint64_t *coeff_return)

```
1: count = 0
2: for i = 0 to DEGREE + 1 do
3:    for j = 0 to DEGREE + 1 do
4:       for k = 0 to DEGREE + 1 do
5:          temp = coeff[count] * pow_1[i]
6:          temp = temp mod Q
7:          temp = coeff[count] * pow_2[k]
8:          temp = temp mod Q
9:          coeff_return[j] += temp
```

Algorithm. 13 gives the algorithm for this interface.

Algorithm 13 Common.evaluate_one_var(int value, uint64_t coeff)

```
1: for i = 0 to DEGREE + 1 do
2:    temp = value * coeff[i]
3:    return_value += temp
4: return return_value
```

CHAPTER 5. SYSTEM DEPLOYMENT AND USAGE MODELS: CASE STUDIES

As part of the research and analysis, of the protocols implemented, we present analysis of the execution of the models with an illustrated example. The analysis is based on the complexity and the amount of resource that may be required to execute the protocols. These will be verified by the experimental results which will give a framework for an application developer to know the limitations of using the protocols. We also provide the environment where they may be useful with the necessary restrictions. Finally compromising on a certain parameters implies reducing the security complexity. Hence it is important to establish a tradeoff and to identify the necessary settings for every protocol to be executing to keep the system secure.

5.1 Model I : Protocols Executing Individually

The models that are presented here are on a broader perspective. These may not be direct application but are some analogy to the real system. The illustration of the protocols executing individually can be obtained from the published work and for DKVP it is mentioned above. The instances and scenarios for executing each of the protocols executing individually are as given below:

- When Data Centric storage is used and the application developer may only need to secure the data and hence only APB will be used. The application may be such that the user node cannot be compromised and is trusted. Also the only communication may be between nodes to access the data from the storage node which does not involve much of tampering of data and hence there is not compromise on the integrity of the system.

- There can be a system where there are multiple hops in the network where the data could be tampered by various entities. This system may not even use Data Centric Storage and hence the only kind of security vulnerability and hence using any other protocol may be an additional burden.

- In Data Centric storage there is a possibility where the nodes are in a secure and secluded zone. In this case the nodes that are querying can be compromised. Hence it is important to secure the system from the user nodes that can cause threat to the system. A direct example to this could be an army environment where the nodes are placed to monitor movement, but the user nodes are surveying nodes that capture the data from the field and hence it is important to see to that only legitimate user nodes get the data.

5.2 Model II: Protocols Executing Together

To continue on the analysis we provide a model where the protocols can be executed in groups and where all the protocols can be executed together as well. Here we would describe where the combination of the protocols can be used and what could be the possible overhead. Finally we give an illustration when all the 3 protocols execute together with an example.

Given below are the descriptions of the proposed combinations:

- APB and DKVP: This is the most obvious combination. These two protocols can be used seamlessly together. In fact DKVP was developed as an extension to APB. The most common application of this combination can be a traditional forest deployment. The overhead of this combination is that at the storage node to store the coefficients. Also if the interval of dissemination or if the number of user nodes compromised is high then the communication overhead also increases considerably. This will be justified in the experimental evaluations.

- APB and MAF: The main application of APB is to store data securely. Consider a forest deployment or an army deployment movement of soldiers is monitored. When soldiers

want to query regarding the data stored, there may be necessity to see to that the data will not be tampered at the intermediate hops. To ensure this the MAF is combined with the APB. The disadvantage with this is that MAF has an inherent limitation that it is not scalable to a very large network, but is definitely practical and will be shown with the experimental results.

- DKVP and MAF: DKVP will not be used without APB, but as given above there can be a hypothetical use of DKVP individually and hence in that system, it can be combined with MAF. The overhead here is that of the coefficients being stored and that for the coefficients for MAF to verify and authorize.

- APB, DKVP and MAF: This is the scenario where all the 3 protocols are used together. To illustrate this we will show an example and to physically described as to what is stored where.

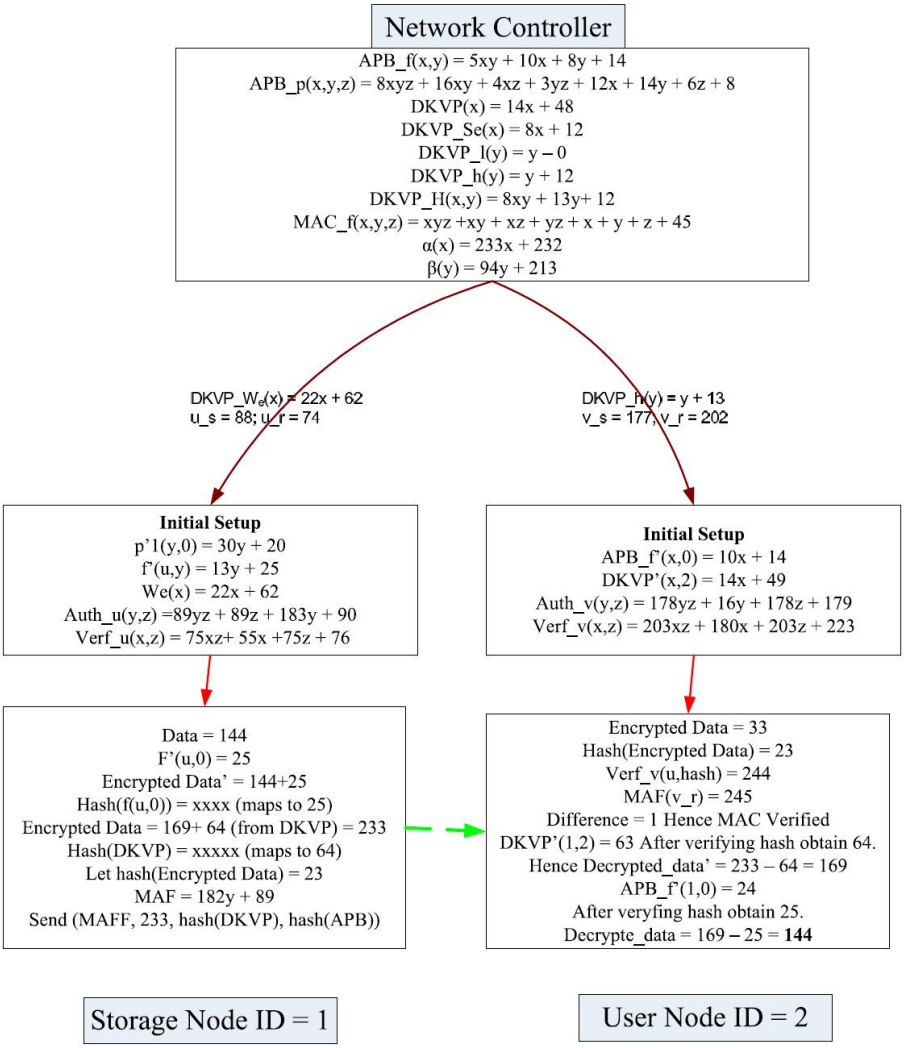

Figure 5.1 Example Illustrating the 3 protocols working together.

CHAPTER 6. IMPLEMENTATION AND RESULTS

The implementation was carried out on the TinyOS using nesC. The initial implementation was performed on TOSSIM [17]. The results were tested and evaluated using PowerTOSSIM [18] an evaluation tool for TinyOS on TOSSIM.

6.1 Programming Limitations

During the working on this study/project various programming limitations with respect to TinyOS and nesC were exposed. These were brought to the limelight the difference between practical feasibility and theoretical analysis. These limitations ranged from memory limitations to communication limitations. Listed below are few of the limitations:

- The size of code must be much less than 40KB as this would enable any other operation code to be performed easily. This limitation is countered by making the implementation efficient, by dividing the code into as many common modules so that, different protocols can use the common functionalities.

- Data has to be allocated statically since dynamic allocations are not advisable in nesC and TinyOS. Though not advisable, our implementation does use dynamic allocation, but this is done efficiently such that the dynamic allocation and de-allocation happens very quickly such that the program heap is not utilized for a long period of program execution time.

- With respect to the MAF protocol, the value of l and r cannot be very high. This is because, the functionality of generating the IDs based on $\alpha(x)$ and $\beta(y)$ is computationally very expensive, hence this is done for reduced values of l and r. From our experiments we

have learnt that maintaining the value of l to 24 and r to 19, makes the system feasible at the same time maintain the security.

6.2 Network Model Assumed

For the sake of simulation, we assumed a network of 3 nodes, namely - the Base station, a storage node and another node which is used to perform the operations of a user node. The performance of the base station is ignored as it is assumed that the base station is a powerful and resourceful entity. The performance of the storage node is treated as a normal sensor node and that of the user node to be a laptop or a PDA. Here the polynomial dissemination happens from the Base station to the storage and user nodes. The experiments were performed at a smaller scale to ensure that the results obtained were specific to operations of the protocols. For the sake of the MAF protocol, we assume both the non-*Base Station* nodes to be authorizing and verifying nodes. *nesC* on *TinyOS* are used for most of the implementation and a separate tool to generate the IDs for the MAF protocol which was written in *C*.

6.3 Results

6.3.1 Storage Overhead

Table 6.1 shows the Storage Overhead at the storage node which is a normal sensor node for the implementation of APB, $DVKVP$, MAF and *All 3 protocols* executing in *mica*2 motes. The implementation is efficient enough to consume only a small portion of the RAM and ROM. In TinyOS it can be noted that a feasible usage of ROM can go up to 40000 bytes and RAM can go upto 4000 bytes. From the table it can be noted that the value of both the ROM and RAM usage can still accommodate any other application. A lot of RAM and ROM will reduce when the storage happens on the flash which means that all the global variables will be stored on the flash rather than on RAM.

Model	ROM (bytes)	RAM (bytes)
DKVP	23098	1176
APB	27234	1188
MAF	22952	1152
All	33144	2568

Table 6.1 Storage Overhead

6.3.2 Computation Overhead

This section discusses the overhead that is incurred in our protocol implementation. Here we have evaluated APB and $DKVP$ separately and when executed together. The parameters that we have varied are degree of polynomial and the value of r. The results will be displayed as processing time for a single operation and the energy consumed for the same at the storage node. We do not display the results for energy consumed at the user node as only the storage nodes are power constrained and the main goal while implementing applications for sensor networks is to minimize the power as much as possible. This section is divided into 3 parts. The first part explains the results (*Processing Time and Energy Consumed*) obtained by varying the degrees of the polynomial in the corresponding protocols when executed individually and when executed together, followed by the *Processing Time and Energy Consumed* of APB, $DKVP$ and Both the protocols executing together while varying the value of r. It could be noted that we do not vary the value of r for the MAF protocol as this will not impact the execution of the APB protocol. However, varying the value of r does affect the execution of the C program, but we do not evaluate this as it is a onetime operation and does not repeat during the lifetime of a sensor network.

6.3.2.1 Simulation environment

The project was simulated using TOSSIM, using 3 motes, where one of them is the basestation, another mote is behaves like a storage node ($nodeID = 1$) and the last node will behave like a user node ($nodeID = 1$). We use PowerTOSSIM to evaluate the CPU Cycle count and

from that arrive at the processing time and energy consumed. The value of l is set to 64 bits. We do not consider the initialization for the evaluation as that will happen when the mote is connected to the base station and will draw energy from the system and not from its own battery. The results given below are for just one operation of encryption/decryption at the respective nodes.

The above mentioned values and setup is only for APB and $DKVP$. For the purpose of the MAF protocol, we assume the l to be 24 bits and r to be 19 bits for the reasons mentioned earlier. Also for the purpose of the MAF protocol, we assume both *nodeID = 1* and *nodeID = 2* to be sensor nodes and hence do not treat any of them to be a PDA or a Laptop, like the other protocols.

6.3.2.2 Varying the degree of polynomial

This section deals with the results of the system by varying the value of the degree of the polynomial, namely, the master polynomial in DKVP and master polynomial in APB. It can be noted that all the polynomials have the same degree and we do not consider varying degrees for different variables in a polynomial. All the graphs that will be discussed in this section have the following properties -

- X-Axis - Degree of Polynomial
- Y-Axis - Processing time for one operation.
- Value of r - 10 for APB and $DKVP$, whereas for MAF the value of r is set to 19.
- Processing speed of Laptop - 1.5 GHz
- Processing speed of PDA - 624 MHz.
- Processing speed of mica2 motes - 7.38 MHz
- Value of γ for MAF protocol is set to 5.

Here the value of r is set to 10 to maintain the complexity greater than 2^{64}.

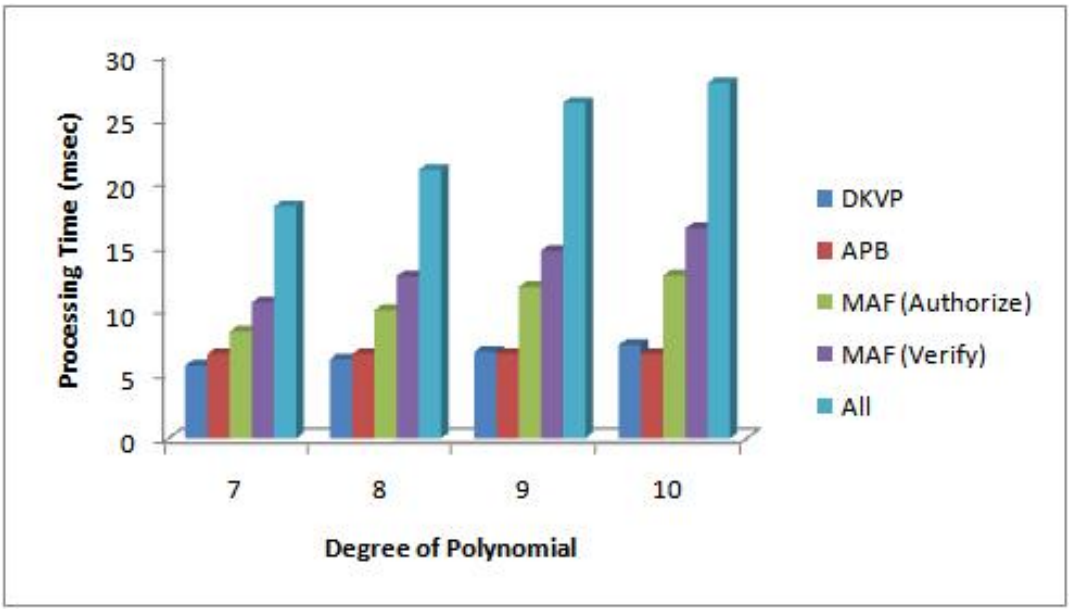

Figure 6.1 Computation Overhead at Sensor/Storage Node.

Storage nodes are normal sensor nodes which in our case is assumed to be *mica*2 motes. Since they are resource constrained and are the most important part of our protocol, it is necessary for us to ensure that the processing time (*CPU Usage*) is feasible and low. The results for executing the protocols on the storage node(s) is as shown in Figure 6.1. The figure is a main reflection on the feasibility and somewhat a practical proof for the theory behind the protocols. An assumption that was made during the evaluation is that - *initialization* of polynomials, etc will happen when the node is connected to the base station and hence the overhead for it can be ignored. From the figure the following can be noted:

- The values for processing time is very low in the order of *milliseconds*.
- A fact that is not noted in the figure is that the percentage of CPU used was less than 60% in the worst case scenario.
- As a proof for the protocols, as the value of the degree of the polynomial increases, the processing time increases for all the protocols, for *APB* the increase is minimal

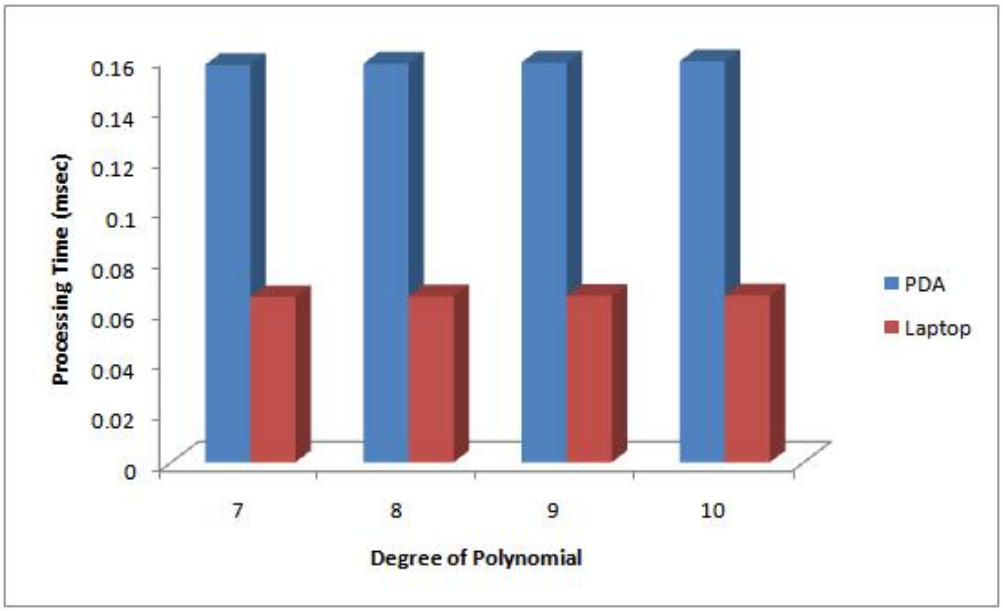

Figure 6.2 Computation Overhead at User Node for APB.

(*negligible*) as encryption just involves using the pre-computed keys for that time interval.

Figure 6.2, 6.3 and 6.4 reflects on the processing time at the User node for APB, $DKVP$ and when both these protocols are executed together. These graphs are a reflection of the time that is consumed when the querying node is a PDA or a Laptop. It can be noted that even though the operation of decrypting is very high for DKVP, the processing time is in the order of *single-digit* milliseconds, which in the case of a laptop is very low. The main purpose of this graph is to provide a guideline for the user who would be using these protocols on a PDA or a Laptop to query. It can also be noted that if the user node had to be a normal *mica*2 sensor mote, the processing time would be in the order of several seconds, which is not advisable. These figures are also a reflection of the feasibility of the protocols.

Figure 6.5 is another way to represent Figure 6.1, the only difference being here we represent the energy consumed at the sensor node rather than the processing time. The motivation behind this kind of representation is that, for several user-specific applications it would be

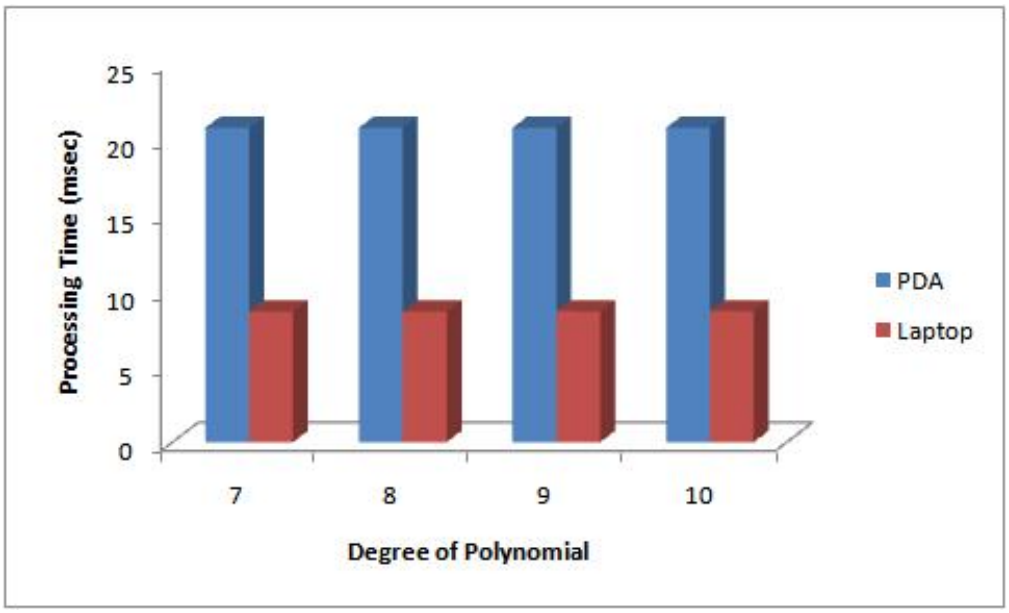

Figure 6.3 Computation Overhead at User Node for DKVP.

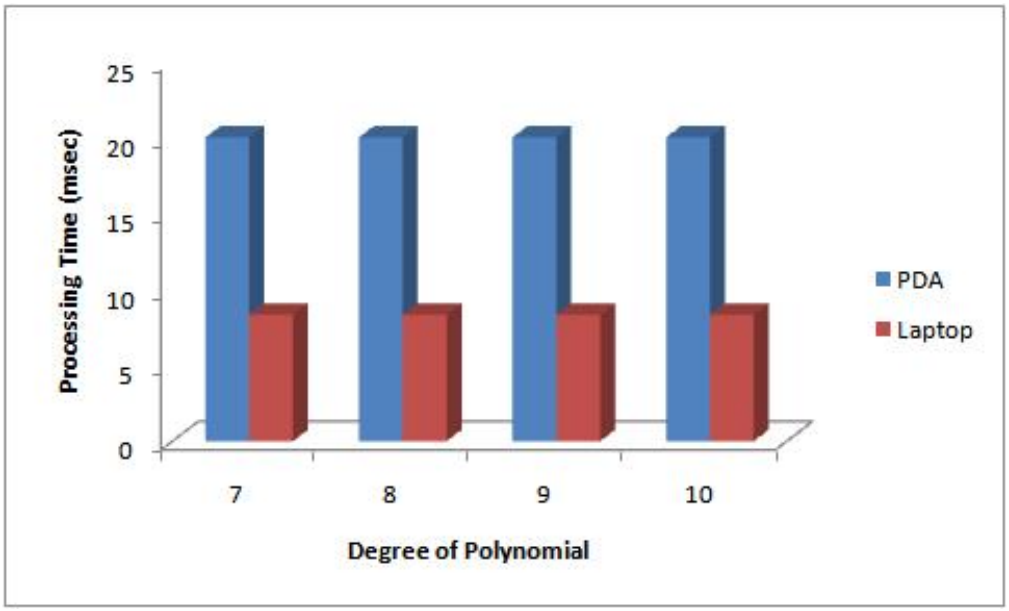

Figure 6.4 Computation Overhead at User Node for Both APB and DKVP.

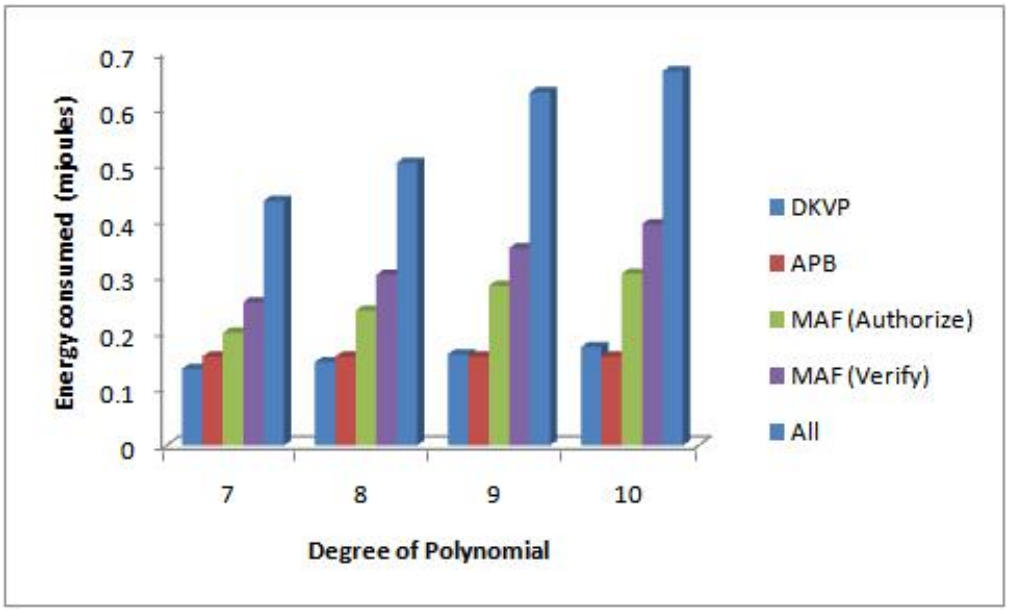

Figure 6.5 Energy consumed at Sensor/Storage Node.

easy to find the amount of energy consumed. Hence this figure will be useful, to know the feasibility of the user-specific coupled with any of our protocol implementations. Also, in the long-run this result may provide a way to know the lifetime of the nodes in the network depending on what protocol is being executed and the power supply connected to them.

6.3.2.3 Varying the value of r

This section deals with the results of the system when the value of r is varied. All the graphs that will be discussed in this section have the following properties -

- X-Axis - Value of r
- Y-Axis - Processing time for one operation.
- Degree of polynomial in all protocols - 13.
- Processing speed of Laptop - 1.5 GHz
- Processing speed of PDA - 624 MHz.

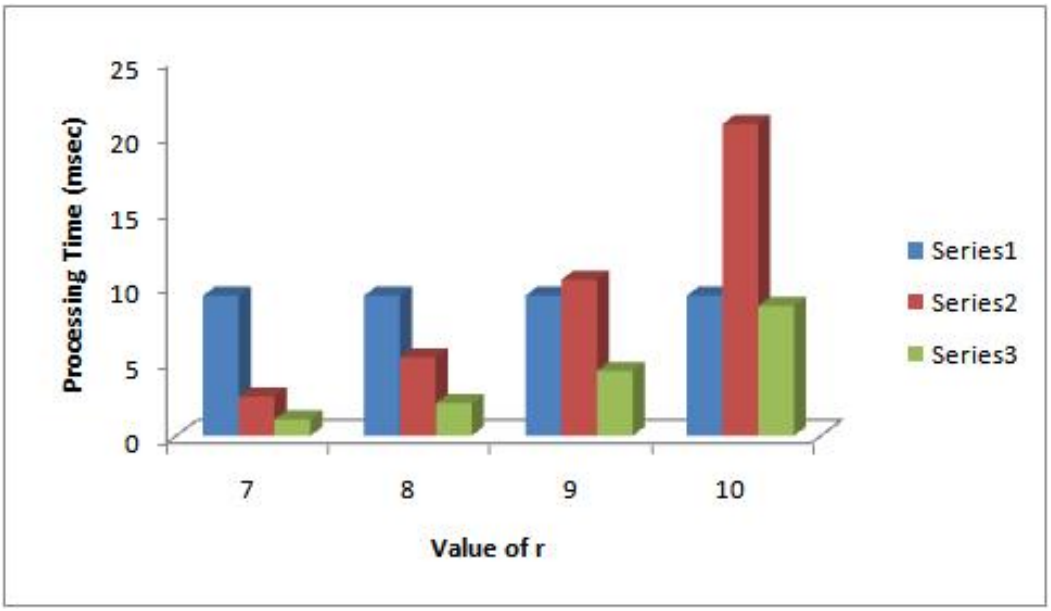

Figure 6.6 Computation Overhead of DKVP (Degree of Polynomial = 13).

The value of the degree of the polynomial is chosen at 13 to maintain the complexity greater than 2^{64}.

Figure 6.6 gives the overhead of DKVP when the value of r is varied. It can be noted from the figure that as the value of r increases the processing time required at the storage node remains the same. This is because all the storage node does is encrypts the data/key by adding the new key and evaluate the hash using TinySec's [11] encrypt function. This will not be affected by the value of r and hence irrespective of the value of r the overhead is the same. But at the user node the processing time increases as the value of r increases. This is because, as mentioned earlier the decryption has to be done $2^r - 1$ times to verify the key. Since the user node is considered to be powerful in the form of Laptop or PDA it is affordable to perform this operation.

Figure 6.7 gives the overhead of APB when the value of r is varied. Here the values are almost the same for all values of r. This is because no operation depends on the value of r. The processing time is for one encryption at the storage node and one decryption at the user node. The value is very low at the user node as the processor speed of the user node is very high. It

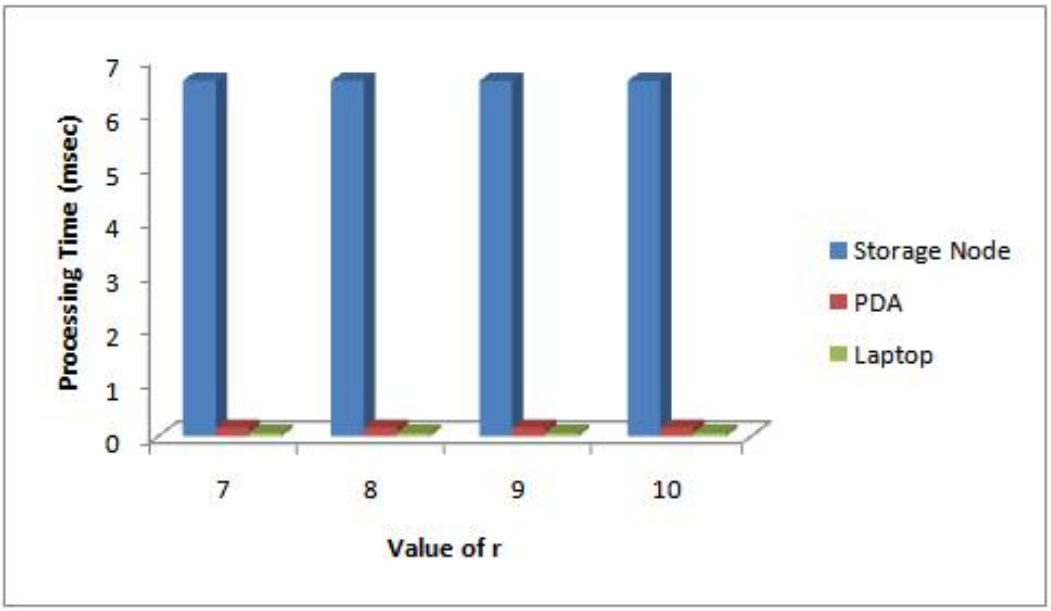

Figure 6.7 Computation Overhead of APB (Degree of Polynomial = 13).

can be noted that the time taken is very low either which ways on which ever architecture.

Figure 6.8 gives the overhead of APB and DKVP are executed simultaneously, when the value of r is varied. It can be noted that the processing time is pretty much the addendum of the times noted in APB and DKVP individually, but not exactly the sum as there can be a lot of shared operation time and the CPU cycle count will not work in the sequence of one protocol after another.

6.3.2.4 Analysis of Computation Overhead

From the various representations of the results the following can be inferred:

- All the implementations are feasible on the corresponding hardware architectures mentioned earlier.
- Results prove the theoretical description of the protocol.
- Various representation of the system provide a set of guidelines for user-specific applications to adopt our implementation.

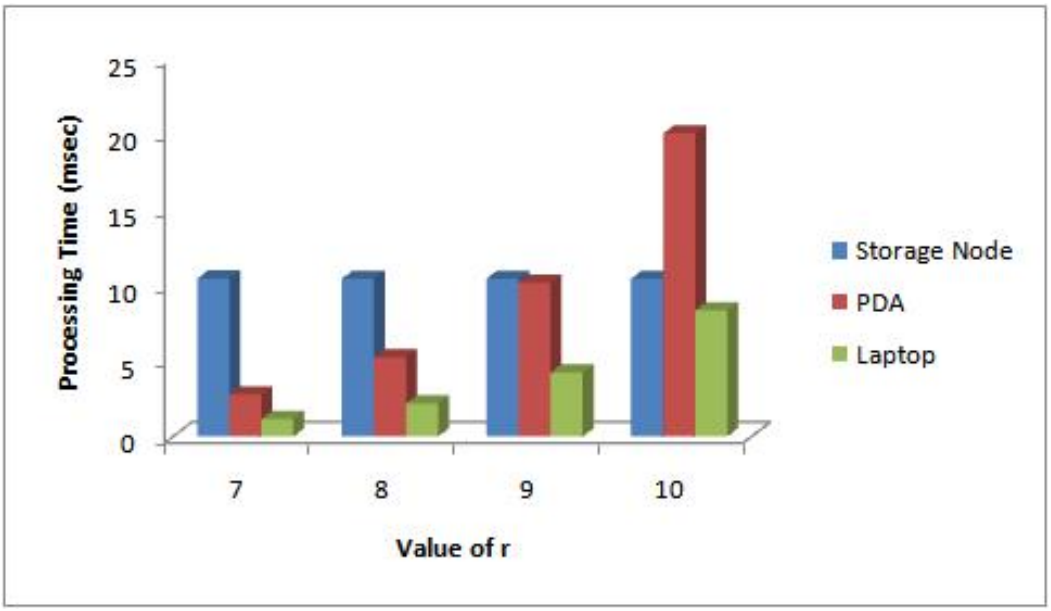

Figure 6.8 Computation Overhead of Both (Degree of Polynomial = 13).

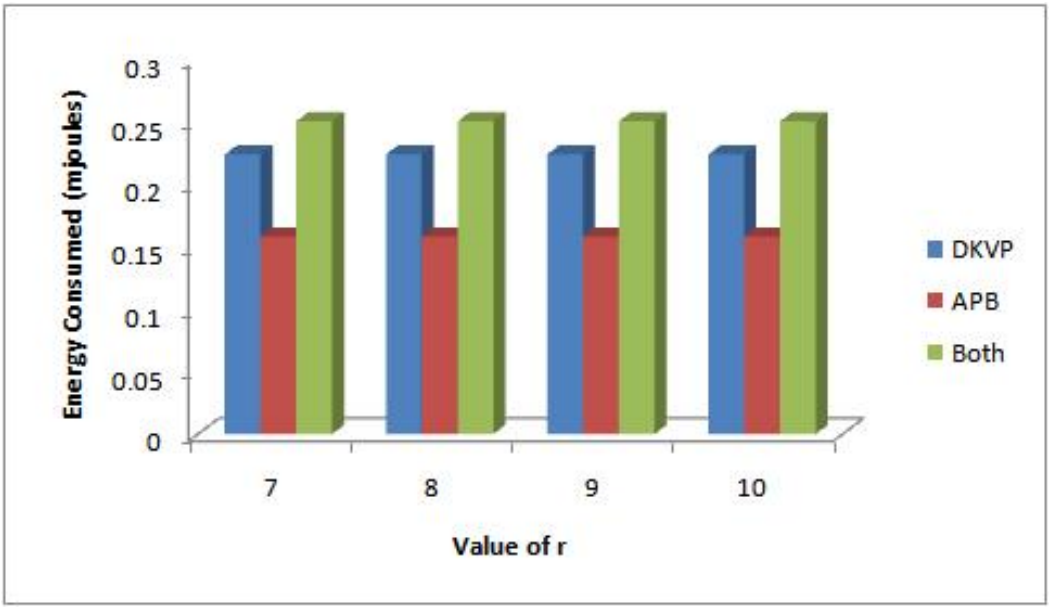

Figure 6.9 Energy Consumed at the Storage node.

- Experiment results provide a feasibility study on the parameters that can be used for various protocols.

6.3.3 Communication Overhead

6.3.3.1 Simulation Environment

The simulation was performed on TOSSIM and the evaluation was based on the Power-TOSSIM. Here we fix the value of r all through this section to 10 for APB and $DKVP$, but for the implementation with the MAF protocol we use r as 19. We perform the experiments varying the degree of the polynomial. To evaluate the communication overhead these were the individual setup of the protocols:

- For APB there was a single query and a single dissemination of the coefficients of the seed polynomial.
- For $DKVP$ there was a single revocation and a single query.
- For MAF there was just a single data that was authorized and that was verified.

6.3.3.2 Varying Degree of Polynomial

Figure 6.10 gives the communication overhead for the protocols. This is a representation of the time that was used only for communication, which was extracted out of the values for *AMStandard* module. This is a reflection on the network usage and these values will vary as the size of the network increases. These values can surely be used to understand the basic feasibility of the protocol from the perspective of the network usage.

Figure 6.11, represents the energy consumed for the sending and receiving of messages which is again denoted by *AMStandard* module. The main reason for this representation is as mentioned earlier.

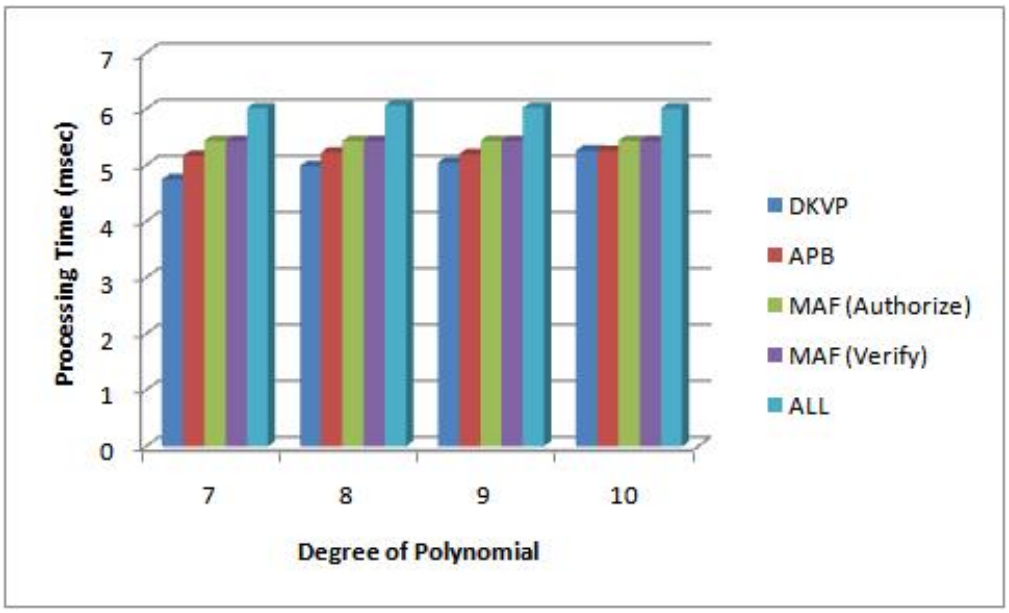

Figure 6.10 Processing Time for Communication.

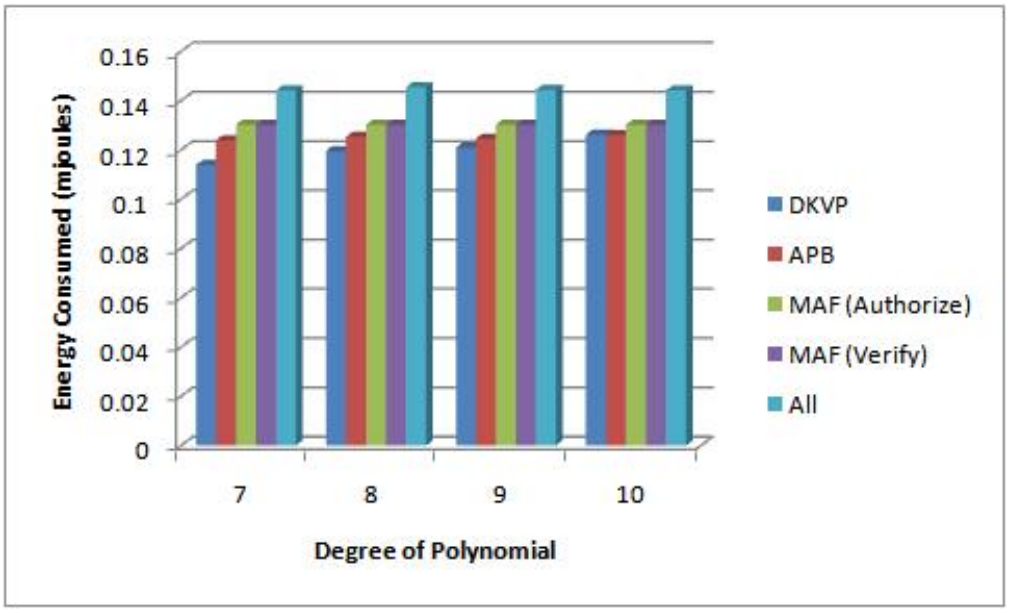

Figure 6.11 Energy Consumed for Communication.

Operation	Time Taken (msec)
APB Encrypt	15
APB Decrypt	50
DKVP Encrypt	15
DKVP Decrypt	468.75
MAF Authorize	28.25
MAF Verify	40.625

Table 6.2 Time taken for operations

6.4 Implementation on live network

We deployed the implementation in real motes to evaluate the time taken for each operation. Table 6.2 shows the time taken for each operation.

The values obtained here are consistent with the theoretical description of the protocols.

CHAPTER 7. CONCLUSIONS AND FUTURE WORK

In this thesis, we have have proposed $DKVP$, to protect data confidentiality and integrity in the presence of User-node compromise. This scheme achieves the goal of protecting the data from unauthorized user node. From the theoritical analysis it can be noted that the protocol is very secure and experimental results conducted on TOSSIM show that the protocol is very efficient for sensor network applications.

We also implemented and integrated three security protocols - APB, MAF and $DKVP$, to provide data confidentiality and integrity for secure data management. The integration suite consists of effective and friendly interfaces for application developers. It also provides example programs to demonstrate the integration. We conducted experiments on TOSSIM and telosb motes and the results show that such a type of integration is affordable.

As a part of future work, we would like to implement 128-bit keys and coefficients. This is to make the system more secure and make it more difficult for the adversary to compromise the data. Currently the implementation module for the base station, is written for sensor motes, that is, the sensors will be connected to a laptop or a work station, but the computations will be performed on the sensor motes. Although this is a working architecture, this should be changed, such that the base station performs the computation, etc and the sensor mote is only used for the communication.

Currently we have implemented 3 protocols as mentioned earlier. We would like the basic framework and infrastructure to be used to implement several other security protocols. In this work, the security levels and parameters are not decided based on any experimental results, which should be altered such that, extensive experiments should lead to a detailed analysis do decide on the security parameters. Finally, for all our experiments, we have assumed a small

scale of sensor motes. As part of future analysis we would like to implement a larger scale network which uses this work.

BIBLIOGRAPHY

[1] Ian F. Akhildiz, et. al. *A Survey on Sensor Networks, IEEE Communications, August 2002, pp. 102-114.*

[2] Tanveer Zia and Albert Zomaya *Security Issues in Wireless Sensor Networks, International Conference on Systems and Networks Communication (ICSNC'06).*

[3] Carlo Blundo et. al. *Perfectly-Secure Key Distribution for Dynamic Conferences, Lecture Notes in Computer Science, vol. 740, pp. 471486, 1993.*

[4] Sylvia Ratnasamy et, al. *Data Centric Storage in Sensornets, SIGCOMM, February, 2002.*

[5] Wensheng Zhang, Nalin Subramanian, and Guiling Wang *Lightweight and Compromise-Resilient Message Authentication in Sensor Networks, IEEE INFOCOM, April, 2008.*

[6] Wensheng Zhang, Minh Tran, Sencun Zhu, and Guohong Cao *A Compromise-Resilient Scheme for Pairwise Key Establishment in Dynamic Sensor Networks (A Random Perturbation-Based Scheme for Pairwise Key Establishment in Sensor Networks), ACM MobiHoc 2007, September 9-14, Montreal, QC, Candada.*

[7] Nalin Subramanian, Chanjun Yang, and Wensheng Zhang *Securing Distributed Data Storage and Retrieval in Sensor Networks, IEEE International Conference on Pervasive Computing and Communications (PerCom) March, 2007.*

[8] Claude Crepeau and Carlton R. Davis *A Certificate Revocation Scheme for Wireless Ad hoc Networks, Proceedings of the First ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'03), pages 54-61, October 2003.*

[9] Sandeep S. Kulkarni, Bezawada Bruhadeshwar *Rekeying and Storage Cost for Multiple User Revocation, The 12th Annual Network and Distributed System Security Symposium San Diego, California 3-4 February 2005.*

[10] Yong Wang; Ramamurthy, B. Xukai Zou *KeyRev: An Efficient Key Revocation Scheme for Wireless Sensor Networks, Communications, 2007. ICC '07. IEEE International Conference on 24-28 June 2007 Page(s):1260 - 1265.*

[11] Chris Karlof, Naveen Sastry, and David Wagner *TinySec: A Link Layer Security Architecture for Wireless Sensor Networks, Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004). November 2004.*

[12] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler *The nesC Language: A Holistic Approach to Network Embedded Systems, Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI).*

[13] D. Liu, P. Ning, and K. Sun *Efficient Self-Healing Group Key Distribution with Revocation Capability, In Proc. of the 10th ACM Conference on Computer and Communications Security (CCS 03), October, 2003.*

[14] Jun ANZAI et. al. *A Distributed User Revocation Scheme for Ad-Hoc Networks, IEICE Transactions 88-B(9): 3635-3642 (2005).*

[15] Arno Wacker, Mirko Knoll, Timo Heiber and Kurt Rothermel *A New Approach for Establishing Pairwise Keys for Securing Wireless Sensor Networks, Proceedings of the 3rd international conference on Embedded networked sensor systems 2005, San Diego, California, USA November 02 - 04, 2005.*

[16] Cynthia Kuo et, al. *Message-In-a-Bottle: User-Friendly and Secure Key Deployment for Sensor Nodes, Proceedings of the 5th international conference on Embedded networked sensor systems 2007, Sydney, Australia, November 06 - 09, 2007.*

[17] Philip Levis, Nelson Lee, Matt Welsh, and David Culler *TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications, SenSys 2003.*

[18] Victor Shnayder, Mark Hempstead, Borrong Chen, Geoff Werner Allen, and Matt Welsh *Simulating the Power Consumption of LargeScale Sensor Network Applications, IEEE INFOCOM, April, 2008.*

[19] F. Ye, H. Luo, S. Lu, and L. Zhang *Statistical En-route Filtering of Injected False Data in Sensor Networks, IEEE INFOCOM, March, 2004.*

[20] S. Zhu, S. Setia, S. Jajodia, and P. Ning *An Interleaved Hop-by-Hop Authentication Scheme for Filtering False Data in Sensor Networks, IEEE Symposium on Security and Privacy, 2004.*

[21] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar *Spins: security protocols for sensor netowrks, Proceedings of ACM Mobile Computing and Networking (Mobicom01), 2001, pp. 189199.*

[22] X. Li, Y. Kim, R. Govindan, W. Hong *Multi-dimensionalrange queries in sensor networks, Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys), pp. 6375, 2003.*