VANDER: Efficient Cooperative Watchdog Monitoring for Lossy Wireless Network Coding

Berlin Sharmila M, Mrs.R.Ranjinadevi

Abstract— Network coding achieves multicast network capacity by allowing intermediate nodes to mix information in packets. However, due to the mixing operation, network coding is vulnerable to pollution attacks. Different from conventional cryptographic solutions, watchdog-based solutions, which have been recently proposed for network coding, rely on trusted nodes to monitor and verify behaviours of transmission nodes. However, the impact of lossy wireless communications has not been considered. Since false alarm and misdetection depend on the loss rate, these false results can happen with high probability when packet loss happens. In this paper, we propose VANDER, which is a novel cooperative watchdog scheme in heterogeneous wireless networks, where multiple watchdogs collaborate and efficiently detect pollution attacks in a lossy wireless environment. The novelty of our approach is that, when lossy overhearing happens, watchdogs work cooperatively to share the packet information, where no extra overhead is introduced to normal transmission nodes, and rather than re transmitting all lost packets among watchdogs, watchdogs use randomly generated Vander mode hashes to detect corrupted packets. Moreover, VANDER is capable of detecting successive colluded adversaries. In addition to the low false alarm and misdetection probabilities, VANDER also achieves low computational complexity and communication overhead. Numerical experiments are provided to support the theoretical analysis of VANDER.

1 INTRODUCTION

1.1 MESH TOPOLOGY

DEFINITION:

In a mesh network topology, each of the network node, computer and other devices, are interconnected with one another. Every node not only sends its own signals but also relays data from other nodes. In fact a true mesh topology is the one where every node is connected to every other node in the network. This type of topology is very expensive as there are many redundant connections, thus it is not mostly used in computer networks. It is commonly used in wireless networks. Flooding or routing technique is used in mesh topology.

1.1.1 Advantage:

i) Data can be transmitted from different devices simultaneously. This topology can withstand high traffic.

ii) Even if one of the components fails there is always an alternative present. So data transfer doesn't get affected.

iii) Expansion and modification in topology can be done without disrupting other nodes.

1.1.2 Disadvantage:

- There are high chances of redundancy in many of the network connections.
- Overall cost of this network is way too high as compared to other network topologies.
- Set-up and maintenance of this topology is very difficult. Even administration of the network is tough.

1.2 Wireless Mesh Network:

Advantage:

Wireless mesh networks provide a number of advantages. In a mesh network, nodes connect wirelessly to transmit information. The nodes are less expensive to install than wired routers required for typical wireless networks. The more nodes there are, the larger and faster the wireless area becomes. Wireless mesh networks are more convenient in areas that have obstacles that can cause problems with "line of sight" wireless transmission. More nodes in large, busy places means more chances to find a clear wireless signal. Mesh networks are self-configuring, self-healing and are very easy to install. In addition, they are much faster because information can travel between nodes without being sent back to a central server.

1.2.1 Project Overview

Since their introduction, wireless mesh networks (WMNs) have attracted a lot of interest from both the international research community and industries. Such an interest from industries is due to the possibility to cover metropolitan areas without a wired infrastructure, which makes WMNs a cost-effective solution to implement, for instance, wireless ISPs. Researchers, instead, have been attracted by the challenging issues related to the configuration and management of WMNs. One of such issues is the assignment of channels to radios in case mesh routers are equipped with multiple radios. The multiradio configuration is becoming increasingly common, as routers may exploit the availability of multiple radios to simultaneously transmit and/or receive on different channels. Consequently, it is possible to reduce the interference and increase the throughput by carefully planning the assignment of channels to radios.

The assignment of channels is not independent of the routing problem. Indeed, nodes using the same channel in a neighbourhood have to share the channel capacity, and hence the amount of bandwidth available on a link depends on how many nodes are using the same channel in the neighbourhood. Then, the way channels are assigned affects the amount of bandwidth available on links, and hence the channel assignment problem must be jointly studied with the routing problem. Therefore, the proposals that recently appeared in the literature addressing such joint problem solve the channel assignment problem and the routing problem separately. Enforcing the new assignment will thus require changing the channels assigned to several radios. Switching channel on a radio breaks the network connectivity for a much longer time than that required by the radio hardware to shift to the new frequency. Thus, it is clear that the more radios switch channel, the higher the impact on the network performance. For this reason, we present a simple heuristic that takes the current channel assignment into account and aims to adjust at most a configurable number of channels to cope with a variation in the set of flow rates in the best manner possible. Our channel reassignment algorithm, instead, starts from one such solution and makes some adjustments to find a better solution.

Another contribution of this paper is minimization of the maximum total utilization over all the collision domains as the objective of our channel reassignment algorithm. Also, the aforementioned analysis enables us to find a reference value for the maximum total utilization that ensures that the network is actually able to carry the offered traffic load. Thus, the condition that the maximum total utilization exceeds such reference value can be used as an indication that a channel reassignment is needed. We applied such criterion in real traffic traces, which showed that reassigning channels by using our heuristic allows a remarkable throughput increase with respect to the strategy of leaving the channel assignment unchanged.

II .LITERATURE REVIEW

2.1 CDA: Concealed Data Aggregation for Reverse Multicast Traffic in Wireless Sensor Networks

Description:

Wireless sensor networks (WSN) are a particular class of ad hoc networks that attract more and more attention both in academia and industry. The sensor nodes themselves are preferably cost-cheap and tiny consisting of a) application

specific sensors, b) a wireless transceiver, c) a simple processor, and d) an energy unit which may be battery or solar driven.
In particular we cannot assume a sensor node to comprise a tamper-resistant unit. Such sensor nodes are envisioned to be spread out over a geographical area to form in an indeed self organizing manner a multihop network.

2.1.1 De-Merits:

Analysis in most scenarios presumes computation of an optimum, e.g., the minimum or maximum, the computation of the average, or the detection of movement pattern. Precomputation of these operations may be either fulfilled at a central point or by the network itself. The latter is beneficial in order to reduce the amount of data to be transmitted over the wireless connection. Since the energy consumption increases linearly with the amount of transmitted data, an aggregation approach helps increasing the WSN's overall lifetime. Another way to save energy is to only maintain a connected backbone for forwarding traffic, whereas nodes that perform no forwarding task persist in idle mode until they are re-activated.

2.1.2 Merits:

Apply a particular class of encryption transformation and exemplarily discuss the approach on the basis of two aggregation functions. Use actual implementation to show that the approach is feasible and flexible and frequently even more energy efficient than hop-by-hop encryption.

2.1.3 Algorithm Used:

- Privacy Homomorphism
- Aggregation

2.2 A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack

Description:

While there are several provably secure encryption schemes in the literature, they are all quite impractical. Also, there are several practical cryptosystems that have been proposed, but none of them has been proven secure under standard intractability assumptions. The significance of our contribution is that it provides a scheme that is provably secure and practical at the same time. There appears to be no other encryption scheme in the literature that enjoys both of these properties simultaneously.

2.2.1 Demerits:

All of the previously known schemes provably secure under standard intractability assumptions are completely impractical (albeit polynomial time), as they rely on general and expensive constructions for non-interactive zero knowledge proofs. This includes non-standard schemes like Rackoff and Simon's as well.
Practical Schemes. Damgard proposed a practical scheme that he conjectured to be secure against lunch-time attacks; however, this scheme is not known to be provably secure, and is in fact demonstrably insecure against adaptive chosen cipher text attack.

2.2.2 Merits:

Present and analyze a new public key cryptosystem that is provably secure against adaptive chosen cipher text attack. The scheme is quite practical, requiring just a few exponentiations over a group. Moreover, the proof of security relies only on

a standard intractability assumption, namely, the hardness of the Diffie Hellman decision problem in the underlying group.

2.3 Classify Encrypted Data in Wireless Sensor Networks

Description:

While sensors are scattered around some area, collecting useful information from those sensors becomes difficult since there is generally no fixed plotted network covering the area. Data collected by individual sensor has to be passed to other sensors. And one by one, until finally it is passed down to its destination. In actual deployment, there are also some more powerful sensors that can act as the fusion points to aggregate the collected data. The In-network processing [6], [8], [9] can effectively reduce large amount of duplicated data in sensor network. The processed data are then sent out to the base station where events can be properly processed. Fig. 1 depicts a typical sensor network.

2.3.1 Demerits:

End-to-end security mechanisms like SSL [1], which are popular on Internet, may seriously limit the capability of network processing that is the most critical function in sensor network. Since supporting In-network processing can significantly improve the performance of extremely resource-constraint sensor networks featuring many-to-one traffic pattern. It is an open problem of how to protect the traffics and to support In-network processing at the same time.

2.3.2 Merits:

This paper tackles the problem by proposing a model of categorizing encrypted messages in wireless sensor networks. A classifier, an intermediate sensor node in our setting, is embedded with a set of searching keywords in encrypted format. Upon receiving an encrypted message, it matches the message with the keywords and then processes the message based on certain policies such as forwarding the original message to the next hop, updating it and forwarding or simply dropping it on detecting duplicates. The messages are encrypted before being sent out and decrypted only at its destination. Although the intermediate classifiers can categorize the messages, they learn nothing about the encrypted messages except several encrypted keywords, even the statistic information. The presented scheme is efficient, flexible and resource saving. The performance analysis shows that the computational cost and communication cost are minimized.

2.3.3 Algorithm Used:

- Diffie Helman Algorithm
- Aggregation

2.4 Public Key Based Cryptoschemes for Data Concealment in Wireless Sensor Networks

Description:

Wireless sensor networks (WSNs) are becoming increasingly popular in many spheres of life. Application domains include monitoring of the environment (such as temperature, humidity, entity movement and seismic activity) as well as numerous other ecological, law enforcement and military settings. Due to the limited amount of power a sensor is deployed with, the technique of data aggregation is commonly employed in an effort to minimize the amount of data that needs to be transmitted. It is a method which consists in condensing the sensed values according to a specific application and does not aim at reconstructing all the values at the receiver. By using data aggregation and in-network processing, the network can converge to a single result, such as, for example, he average, variance, minimum or maximum of the sensed values. This method can also be applied in certain points of the network, that store the values sensed over a region and

time. Since applications often do not require every individual aggregated value, we can also aggregate for long-term storage, thus minimizing the amount of storage space required.

2.4.1 Demerits:

Secure aggregation is the absence of solutions involving public-key encryption schemes. However, public-key based solutions provide a higher level of system security, since nodes would not be equipped with private keys, which would limit the advantage gained by an attacker compromising some of the nodes. It is therefore important to note why public-key encryption schemes have not yet been deployed. The reason is two-fold: (1) they are deemed too costly for computationally weak devices and (2) the expansion in bit size during the transformation of plaintext to ciphertext introduces costly communication overhead, which directly translates to a faster depletion of the sensor's energy.

2.4.2 Merits:

Data privacy issues between readers, aggregators and the sensors themselves. We aim at providing end-to-end encryption of sensors' measurements as they are aggregated in the network via hop-by-hop transmission.We attempt to achieve the highest level of security possible, while taking power consumption and platform specific limitations as the major metric for the overall system. We outline a list of desired criteria that reflect the security and system requirements, and contrast a set of candidate solutions. We take the step towards the application of additive asymmetric privacy homomorphisms as a feasible solution to the problem of end-to-end aggregated encryption in some classes of WSNs.

2.4.3 Algorithm Used:

- Additively Homomorphic Encryption:
- Elliptic Curve Naccache-Stern (EC-NS) Encryption

2.5 E_cient Aggregation of encrypted data in Wireless Sensor Networks

Description:

Wireless sensor networks (WSNs) are becoming increasingly popular in many spheres of life. Application domains include monitoring of the environment (such
As temperature, humidity and seismic activity) as well as numerous other ecological, law enforcement and military settings. Regardless of the application, most WSNs have two notable properties in common: (1) the network's overall goal is typically to reach a collective conclusion regarding the outside environment, which requires detection and coordination at the sensor level, and (2) WSNs act under severe technological constraints: individual sensors have severely limited computation, communication and power (battery) resources and need to operate in settings with great spatial and temporal variability.

2.5.1 Demerits:

Although simple and well-understood, aggregation becomes problematic if end-to-end privacy between sensors and the sink is required. If we assume that all sensors are trusted, sensors could encrypt data on a hop-by-hop basis. For an intermediate sensor (i.e., one that receives and forwards data), this would entail: 1) sharing a key each neighbouring sensor, 2) for each downstream1 neighbour, decrypting the received encrypted value, 3) aggregating all received values, and 4) encrypting the result for transmission upstream. Though viable, this approach is fairly expensive and complicated. The former because of having to decrypt each received value before aggregation and the latter due to the overhead imposed by key management.

2.5.2 Merits:

Focus on efficient, bandwidth-conserving privacy in WSNs. More specifically, to blend inexpensive encryption techniques with simple aggregation methods to achieve very efficient aggregation of encrypted data. To assess the practicality of proposed techniques, we evaluate them and present very encouraging results which clearly demonstrate appreciable bandwidth conservation and small overhead stemming from both encryption and aggregation operations.

2.5.3 Algorithm Used:

- AGG protocol
- HBH (hop-by-hop encryption and aggregation)

III.EXISTING SYSTEM

- A single corrupted packet from the adversarial node can potentially pollute all the information reaching the destination.
- The information-theoretic approaches in and use end-to-end error correction codes to decode source messages.
- Cryptographic primitives are used to enable honest network nodes to detect and drop corrupted packets. However, these approaches usually have high implementation complexity.
- The broadcast nature of the wireless medium has been utilized to design schemes against pollution attacks.
- The idea of maximum distance separable (MDS) codes to deal with watchdog lossy overhearing in multihop routing networks.
- First polynomial-time network coding schemes that tightly achieve the error correction rate bounds.

Disadvantages:

- Forge aggregated results
- Doesn't provide multiple application aggregate process.
- Doesn't count aggregate message count

IV.PROPOSED SYSTEM

- Lossy overhearing leads to the false alarm and the misdetection, as well as their consequences in wireless network coding with watchdog monitoring.
- To achieve both computation and communication efficiencies in preventing pollution attacks of lossy wireless network coding transmissions.
- A novel watchdog cooperative scheme that leverages the property of the Vandermonde1 matrix to detect pollution attacks through cooperation.
- Finite-field linear operations at the watchdog are sufficient to detect pollution attacks with the effective security guarantee.
- We analyze the communication overhead and the computational complexity of VANDER.
- Lossy overhearing, VANDER can achieve both low false alarm probability and low misdetection probability, which decreases *exponentially* with the collaborative communication overhead.
- The tradeoffs between the overhead and the misdetection probability, as well as the false alarm probability.
- We conduct extensive numerical simulations to evaluate the performance of VANDER and compare it with other schemes, where VANDER significantly reduces approximately 80% of the communication overhead.
- The verification process is at least six to eight times faster than that of typical cryptographic schemes in average.

Advantages:

- WSNs, constructing PH via ECC is more efficient.
- Provide temporal key and maintain a security

- Reduce communication count

V. BACKGROUND & RELATED WORKS

SOFTWARE DESCRIPTION

4.1 net framework overview

Microsoft's new software development platform, .NET Framework, is the first Microsoft development environment designed from the ground up for Internet development. Although .NET is not meant to be used exclusively for Internet development, its innovations were driven by the limitations of current Internet development tools and technology.
The basis of this new development platform consists of three primary components or layers: the common language runtime, the .NET Framework base classes, and the user and program interfaces, as demonstrated in
Figure 4.1

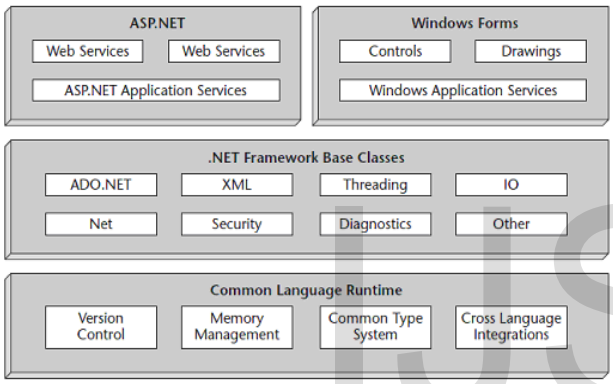

FIG 4.1 COMPONENTS OF .NET FRAMEWORK

The foundation of the .NET Framework is the common language runtime. Its principal purpose is to load, execute, and manage code that has been compiled to Microsoft's new intermediate byte code format called Intermediate Language (IL). Several languages, notably Microsoft's Visual Basic .NET and C# .NET (pronounced "C sharp"), have compilers supporting this format, and many more are currently in development. It is important to note that the IL code is not interpreted. The common language runtime uses just in time compilers to compile the IL code to native binary code before execution.
Other significant features of the common language runtime include the following:

a. Version control
b. Memory management
c. Cross language integration
d. Common data type system

The .NET Framework software developer's kit (SDK) not only provides several runtime hosts but also supports the development of third party runtime hosts. For example, ASP.NET hosts the runtime to provide a scalable, server side environment for managed code. ASP.NET works directly with the runtime to enable .asp pages and Web services.
The final layer of the .NET Framework consists of the user and program Interfaces. The two components of this layer are ASP.NET Application Services and Windows Application Services. The cornerstone of the ASP.NET Application Services is, of course, ASP.NET, which in turn supports the new Web services and Web Forms technologies that are discussed later. The Windows Application Services component supports traditional Windows programming applications through Windows Forms.

Features of the Common Language Runtime

The common language runtime provides an execution environment that manages the set of code targeted to the .NET Framework. Code management can include memory management, thread management, security management, code verification, and compilation of the code. All managed components must first be assigned a level of trust. The level or degree of trust can vary on the basis of the following origins:

- Local computer
- Internet connection
- Enterprise local area network (LAN) connection

The common language runtime also uses a common type system (CTS) to enforce code robustness and language interoperability.

The common language runtime also should increase application performance. This may be accomplished in two ways: just in time (JIT) compliers and server side applications. The JIT compilers are used to compile all managed code to native machine binary code at execution.

Server side applications can house application logic and business rules on .NET Enterprise Servers such as Internet Information Server (IIS) and SQL Server.

4.2 NET Framework Class Library

The .NET Framework class library is a collection of reusable classes, or types, that tightly integrate with the common language runtime. .NET applications benefit from using and extending or inheriting the functionality from the classes and types available in the class library. The class library is very hierarchical and well organized, as shown in Figure 4.2. It starts with the most generic classes and continues to build down to classes with very specific and precise functionality. Although this library is extensive, its organization makes it easy to learn and use. In an age of ever growing technology it is refreshing to see a new technology and a new architecture that promise a reduced learning curve.

This model also makes it easy for third party components to be integrated easily with the existing class library.

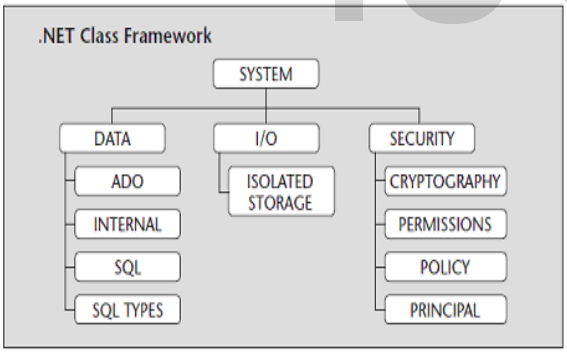

FIG 4.2 UNIFIED PROGRAMMING MODEL

As expected in an object oriented class library, the .NET Framework classes enable developers to accomplish rapidly a wide range of common programming tasks, including things such as string management, file access, and database connectivity. Also, several classes facilitate highly specialized and custom development environments. These classes make the application development environment very flexible. The following types of applications are readily supported through the .NET Framework:

- ASP.NET applications
- Console applications
- Scripted applications
- Windows applications (Windows Forms)
- Web services

For example, the Windows Forms classes are used to create Windows graphical user interface (GUI) applications, which often are referred to as standalone applications. This is facilitated through a series of reusable graphical interface classes.

Alternatively, in developing a Web based application, the HTML and Web Forms classes facilitate its rapid development. Either way the underlying framework provides the flexibility for feature rich applications regardless of the choice of application environment.

Client Application Development

Client applications represent the most common application environment in use today. These are the applications that one would invoke by opening some type of form and initiating an action. Examples of client applications range from word processing applications to a customized accounting package.

The traditional Windows programming environment has been replaced in .NET by the Windows Forms control. The managed Windows Forms control allows the application to be deployed over the Internet, and the user can view the application as a Web page. In the past developers created such applications by using C or C++ in conjunction with the Microsoft Foundation Classes (MFC) or with a rapid application development (RAD) environment such as Microsoft Visual Basic. The .NET Framework incorporates aspects of the earlier products into a single, consistent development environment. The goal of the single environment is to simplify the development of client applications.

Each object and Web server may have security rights different from those of another server. All security rights are evaluated even though the objects are used within a single application. Windows Forms applications still have some advantages over Web based applications. Windows Forms applications have a level of trust that is already assumed. This allows binary and natively executing code to interact with some of the resources on the local machine. This is used to perform the necessary GUI elements of the application.

About C# Programming

C# is Microsoft's programming language for its new .NET development environment. Microsoft's goal with C# is to provide a simple, modern, object oriented .NET programming language that is Internet centric.

Although .NET code can be written in many languages, C# is the only language designed specifically for the .NET platform and for that reason may become the language of choice for this environment.

C# may be deceptively simple. Although it has only about 80 keywords and a dozen intrinsic data types, it is highly expressive. It includes support for all types of modern component based, object oriented development.

C#, like C++ and Java, owes its origins to the C programming language. For that reason, C++ and Java developers will notice a striking similarity to those languages and enjoy an easy to learn and familiar programming environment. Specifically, C# is an elegant language object oriented language that:

- Readily supports the definition of and working with classes and their properties and methods.
- Supports the implementation of interfaces, or contracts, to implement functionality. While a class can only inherit, or extend, from a single class, it can implement multiple interfaces.
- Supports struts, which are lightweight custom types that require fewer resources than does a more complex, robust type defined as a class.
- The common language runtime now features a garbage collector, a process responsible for removing objects from memory that no longer have references to them. The developer is freed from the tedious and often fruitless activity of trying to assure that all objects have been destroyed.

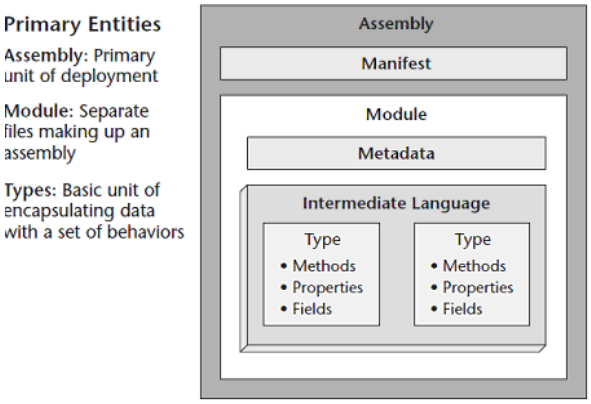

FIG 4.2(a) ANATOMY OF .NET APPLICATIONS

4.3 ASP.NET:

ASP.NET is a Web application framework developed and marketed by Microsoft to allow programmers to build dynamic Web sites, Web applications and Web services.

It was first released in January 2002 with version 1.0 of the .NET Framework, and is the successor to Microsoft's Active Server Pages (ASP) technology. ASP.NET is built on the Common Language Runtime (CLR), allowing programmers to write ASP.NET code using any supported .NET language. The ASP.NET SOAP extension framework allows ASP.NET components to process SOAP messages.

ASP.NET Web pages, known officially as Web Forms, are the main building block for application development. Web forms are contained in files with an ".aspx" extension; these files typically contain static (X)HTML markup, as well as markup defining server-side Web Controls and User Controls where the developers place all the required static and dynamic content for the Web page.

Additionally, dynamic code which runs on the server can be placed in a page within a block <% -- dynamic code -- %>, which is similar to other Web development technologies such as PHP, JSP, and ASP. With ASP.NET Framework 2.0, Microsoft introduced a new code-behind model which allows static text to remain on the .aspx page, while dynamic code remains in an .aspx.vb or .aspx.cs or .aspx.fs file (depending on the programming language used).

Table Design:

Hopnodeinfo:

Name	Type	Null
hopid	varchar(100)	Yes
hopip	varchar(100)	Yes
port	varchar(20)	Yes

Nodeinfo:

Name	Type	Null
nodeid	varchar(100)	Yes
nodeip	varchar(100)	Yes
nodeport	varchar(100)	Yes
nodejoingroup	varchar(100)	Yes

Node Power Table:

Name	Type	Null
nodeid	varchar(100)	Yes
nodepowervalue	int(20)	Yes

TESTING:

UNIT TESTING

Unit testing comprises the set of test performed by an individual programmer prior to integration of the unit in to a larger system. A program unit is usually small enough that the programmer who developed it can test it in great details and certainly is greater detail than will be possible when the unit is integrated in to an evolving software product.

INTEGRATION TESTING:

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.
Test Results: All the test cases mentioned above passed successfully. No defects encountered.

TEST RESULT:

All testing process successfully verified and gets best result in output; Entire project has been tested in three type of testing process.

VI. SYSTEM DESIGN

System design is the process of defining the architecture, components, modules, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering. If the broader topic of product development blends the perspective of marketing, design, and manufacturing into a single approach to product development, then design is the act of taking the marketing information and creating the design of the product to be

manufactured. System design is therefore the process of defining and developing systems to satisfy specified requirements of the user.

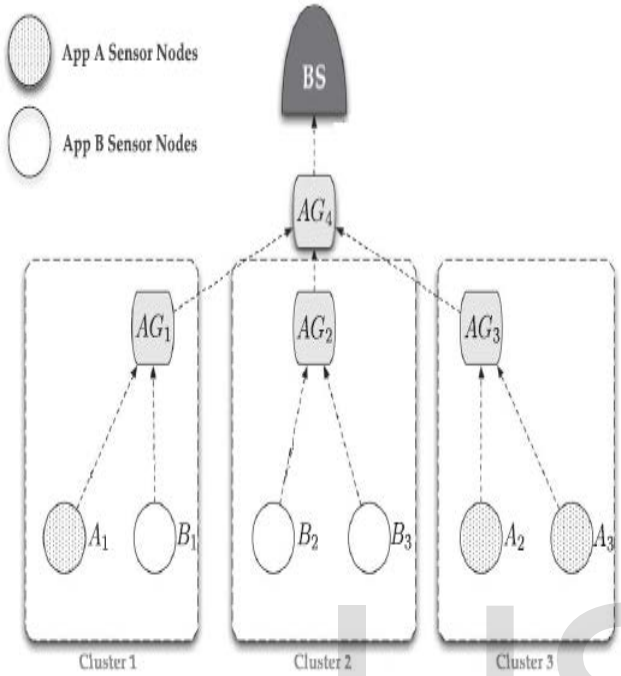

FIG 5 System Design

5.1 MODULES

List of Modules:

- Network Evaluation Module
- VANDER Construction Module
- Key Distribution Module
- Data Aggregation Module
- Secure Counting Module

5.1.1 Network Evaluation Module

This module to construct the all node for wireless setup and also SN collect information from deployed environments and forward the information back to base station (BS) via multihop transmission based on a tree or a cluster topology. The accumulated transmission carries large energy cost for intermediate nodes.

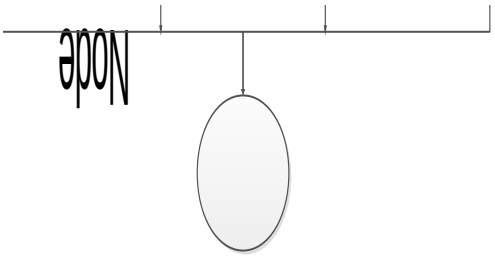

FIG 5.1.1 Network Evaluation Module

5.1.2 VANDER Construction Module

BGN is implemented by using two points of different orders so that the effect of one point can be removed by multiplying the aggregated cipher text with the order of the point, and then the scalar of the other point can be obtained. Based on the same logic of BGN, VANDER is designed by using multiple points, each of which has different order. The security of VANDER and BGN are based on the hardness assumption of subgroup decision problem, whereas VANDER requires more precise secure analysis for parameter selections.

The scalars of the first two points carry the aggregated messages in G_A and G_B, respectively, and the scalar of the third point carries randomness for security. As shown in the DEC functions, by multiplying the aggregated cipher text with q2 q3 (i.e., the SK in GA), the scalar of the point P carrying the aggregated message in G_A can be obtained. Similarly, by multiplying the aggregated cipher text with q1q3 (i.e., the S_K in G_B), the scalar of the point Q carrying the aggregated message in G_B can be obtained. In this way, the encryptions of messages of two groups can be aggregated to a single cipher text, but the aggregated message of each group can be obtained by decrypting the cipher text with the corresponding S_K.

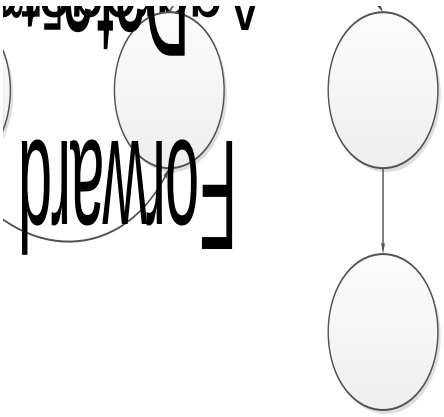

FIG 5.1.2 VANDER Construction Module

5.1.3 Key Distribution Module

Briefly address how to deliver the group public keys to SNs securely. There are two main approaches Key pre-distribution. The locations of deployed SNs, preload necessary keys and functions into SNs and AGs so that they can work correctly after being spread out over a geographical region.

FIG 5.1.3 KEY Construction Module

5.1.4 Data Aggregation Module

VANDER to the conventional aggregation model can mitigate the impact from compromising attacks. All SNs are in the same application, e.g., fire alarm, but they can be arranged into two groups through VANDER construction. Each group could be assigned a distinct group public key. Once an adversary compromised a SN in group A it only reveals PK_A, not PK_B. Since the adversary can only forge messages in group A, not group B, the SNs in group B can still communicate safely. The ideal case is that VANDER assigns every node for its own group, resulting in the strongest security VANDER ever offered.

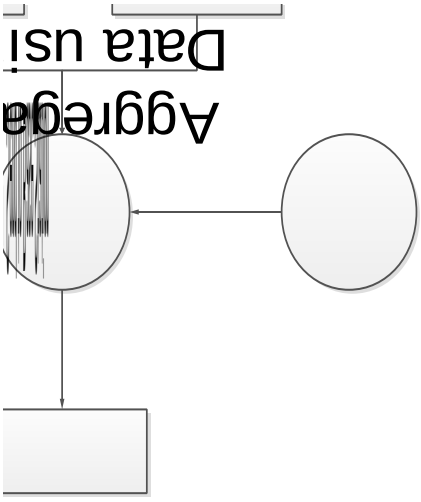

FIG 5.1.4 DATA Aggregation Module

5.1.5 Secure Counting Module

This module adopt VANDER (k = 2) scheme to provide secure counting for single application case, i.e., the BS exactly knows how many sensed readings are aggregated while it receives the final result. The BS obtains the aggregated result M and its count. If a malicious AG launches unauthorized aggregations, such as repeated or selective aggregation value would be changed to a bigger or smaller value than the reference count (e.g., the number of deployed sensors).

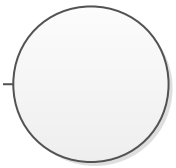

FIG 5.1.5 Secure Counting Module

5.2 ALGORITHM

1 Homomorphic Public Encryption
2 VANDER novel watchdog cooperative
3 Aggregation with Secure Counting

5.2.1 Homomorphic Public Encryption:

Homomorphic encryption (PH) is an encryption scheme with homomorphic property. The homomorphic property implies that algebraic operations on plaintexts can be executed by manipulating the corresponding ciphertexts.
PH schemes are classified to symmetric cryptosystem when the encryption and decryption keys are identical, or asymmetric cryptosystem (also called public key cryptosystem) when the two keys are different.

5.2.2 VANDER:

Assume that all SNs are divided into two groups, G_A and G_B. VANDER contains four procedures: Key generation, encryption, aggregation, and decryption, VANDER (k = 2) is implemented by using three points P,Q, and H whose orders are q1, q2, and q3, respectively.

5.2.3 Aggregation with Secure Counting

The main weakness of asymmetric CDA schemes is that an AG can manipulate aggregated results without encryption capability. An AG is able to increase the value of aggregated result by aggregating the same cipher text of sensed reading repeatedly, or decrease the value by selective aggregation. Since the BS does not know the exact number of cipher texts aggregated repeated or selective aggregation may happen

Expected Outcome:

This project implemented for .NET application and language C#. The VANDER are all built on elliptic curves, encryption and aggregation are based on two kinds of operations, point addition and point scalar multiplication. In elliptic curve arithmetic, two basic operations are point doubling and adding.

These inputs are given sensed data, these data only sense the system based sensors like Desktop capture and Folder Sensing software has been implement for this application.

Outputs are secure data access for base station and securely counting the how many data has been aggregated.

5.3 UML DIAGRAMS

5.3.1 USE CASE DIAGRAM

A use case is a set of scenarios that describing an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors. An actor is represents a user or another system that will interact with the system modelled. A use case is an external view of the system that represents some action the user might perform in order to complete a task.

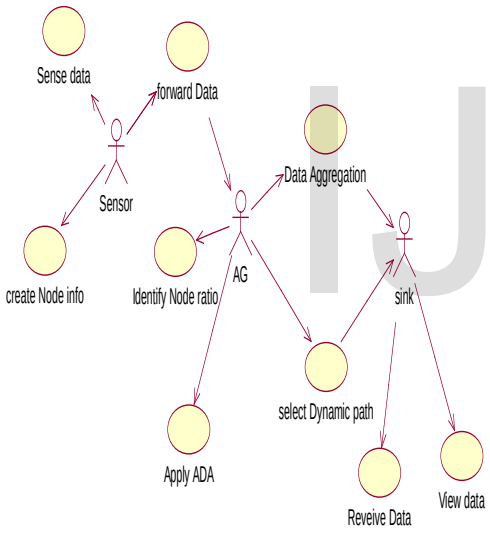

FIG 5.3.1 Use case diagram

5.3.2 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflow of components in a system. An activity diagram shows the overall flow of control.

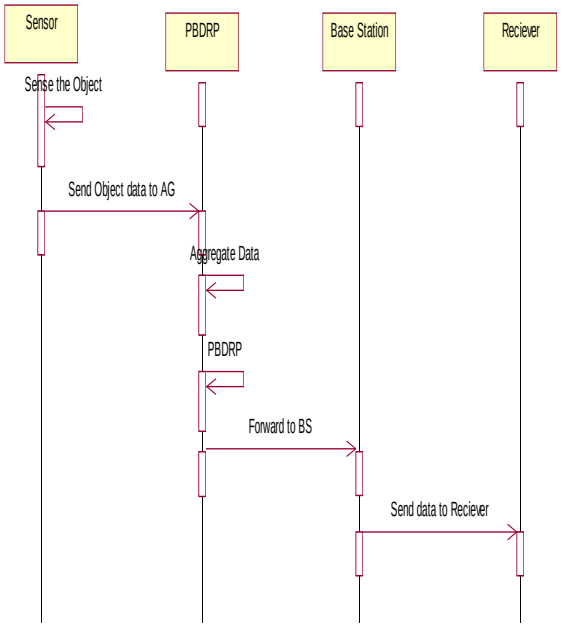

FIG 5.3.3 Sequence diagram

5.3.4 Collaboration Diagram:

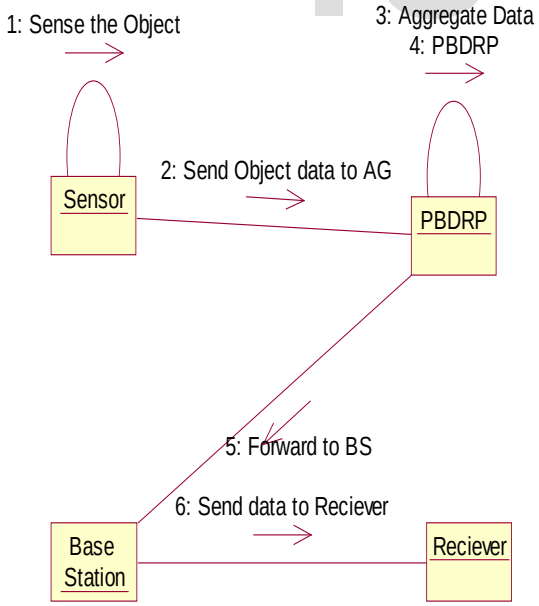

FIG 5.3.4 Collaboration diagram

5.3.5 Class Diagram:

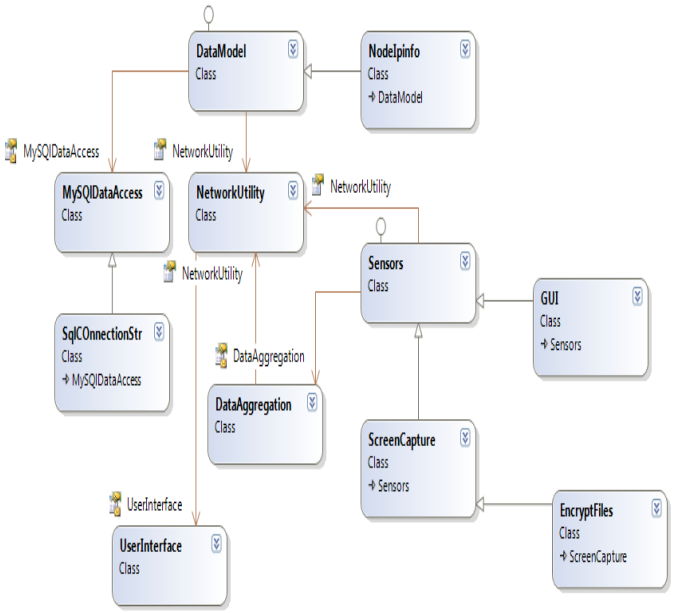

FIG 5.3.5 Class diagram

5.4 SOURCE CODE:

Code:
BS:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using System.Net.Sockets;
using System.Runtime.Serialization.Formatters.Binary;
using System.Net;
using System.Threading;
using System.Reflection;
using SensorNode.Classes;
using CDAMA.Classes;
using System.IO;



namespace CDAMA
{

   public partial class Form1 : Form
```

```
{
    private static readonly CaDiffieHellman _caClient = new CaDiffieHellman();
    private static readonly CaDiffieHellman _caServer = new CaDiffieHellman();
    DataTable dtA = new DataTable("MemberDetails");
    DataTable securecount = new DataTable("SC");
    private int transmissioncount = 0;
    public Form1()
    {
        dtA.Columns.Add(new DataColumn("Ip", typeof(string)));
        dtA.Columns.Add(new DataColumn("Port", typeof(int)));
        //dtA.Columns.Add(new DataColumn("Key",typeof(string)));
        dtA.Columns.Add(new DataColumn("Groupname", typeof(string)));
        securecount.Columns.Add(new DataColumn("Tr Count", typeof(int)));
        securecount.Columns.Add(new DataColumn("CH Id", typeof(string)));
        securecount.Columns.Add(new DataColumn("Secure Count", typeof(int)));
        InitializeComponent();
        radgridsecurecount.DataSource = securecount;

    }
    public void loadimage()
    {
        byte[] byt = null;
        FileStream fs = new FileStream(Application.StartupPath + "\\a.jpg", FileMode.Open, FileAccess.Read);
        byt = new byte[fs.Length];
        fs.Read(byt, 0, byt.Length);
        fs.Close();
        fs.Dispose();
        fs = null;
        pictureBox1.Image = ByteToImage(byt);

    }
    public Bitmap ByteToImage(byte[] blob)
    {
        MemoryStream mStream = new MemoryStream();
        byte[] pData = blob;
        mStream.Write(pData, 0, Convert.ToInt32(pData.Length));
        Bitmap bm = new Bitmap(mStream, false);
        mStream.Dispose();
        return bm;

    }
    public void mod_Receive()
    {

        //Console.WriteLine("Enter Stable Node Name Here");
        Object stablenodename = "BS1";
```

```
TcpListener tcp = new TcpListener(IPAddress.Any, 5555);
//Console.WriteLine("Server started..");
tcp.Start();
Socket s = tcp.AcceptSocket();
NetworkStream ns = new NetworkStream(s);
BinaryFormatter bf = new BinaryFormatter();
Object verify = bf.Deserialize(ns);
Control.CheckForIllegalCrossThreadCalls = false;
if (verify.ToString() == "Upload")
{
}
else if (verify.ToString() == "SecureCount")
{

   transmissioncount = transmissioncount + 1;
   Object securecountobj = bf.Deserialize(ns);
   Object chid = bf.Deserialize(ns);
   DataRow dr = securecount.NewRow();
   dr["Tr Count"] = transmissioncount;
   dr["CH Id"] = Convert.ToString(chid);
   dr["Secure Count"] = Convert.ToInt32(securecountobj);
   securecount.Rows.Add(dr);


}
else if (verify.ToString() == "BS")
{
   string nodeid = (string)bf.Deserialize(ns);
   string group = (string)bf.Deserialize(ns);
   byte[] screencapture = (byte[])bf.Deserialize(ns);
   string filecapture = (string)bf.Deserialize(ns);
   string aggdata = (string)bf.Deserialize(ns);
   Control.CheckForIllegalCrossThreadCalls = false;
   lblsensorid.Text = nodeid;
   if (group == "Group1")
   {

      lblgroupname.Text = "Group1";
      string text = EncryptFiles.Decrypt(filecapture, lblpka.Text);
      byte[] original = EncryptFiles.Decrypt(screencapture, lblpka.Text);
      FileStream fs = new FileStream(Application.StartupPath + "\\a.jpg", FileMode.Create, FileAccess.Write);
      fs.Write(original, 0, original.Length);
      fs.Close();
      fs.Dispose();

      string[] rich = text.Split(',');
      for (int i = 0; i < rich.Length - 1; i++)
```

```
        {
            richTextBox1.AppendText(rich[0].ToString());
        }
    }
    else
    {
        lblgroupname.Text = "Group2";
        string text = EncryptFiles.Decrypt(filecapture, lblbkb.Text);
        byte[] original = EncryptFiles.Decrypt(screencapture, lblbkb.Text);
        FileStream fs = new FileStream(Application.StartupPath + "\\a.jpg", FileMode.Create, FileAccess.Write);
        fs.Write(original, 0, original.Length);
        fs.Close();
        fs.Dispose();

        string[] rich = text.Split(',');
        for (int i = 0; i < rich.Length - 1; i++)
        {
            richTextBox1.AppendText(rich[0].ToString());
        }
    }

}
else
{
    Object o = bf.Deserialize(ns);
    System.Collections.ArrayList sensorinfo = (System.Collections.ArrayList)o;
    if (sensorinfo[0].ToString() == "Group1")
    {
        dtA.Rows.Add(sensorinfo[2].ToString(), Convert.ToInt32(sensorinfo[3].ToString()), "Group1");
        group1.AppendText(sensorinfo[1].ToString() + ":" + sensorinfo[2].ToString() + "\n");
        bf.Serialize(ns, "True");
    }
    else
    {
        dtA.Rows.Add(sensorinfo[2].ToString(), Convert.ToInt32(sensorinfo[3].ToString()), "Group2");
        group2.AppendText(sensorinfo[1].ToString() + ":" + sensorinfo[2].ToString() + "\n");
        bf.Serialize(ns, "True");
    }
}

tcp.Stop();
s.Close();
if (s.Connected == false)
{
    mod_Receive();
}
```

```
}
public string setGroup
{
   get { return group1.Text; }
   set { group1.Text += value; }
}
public void loadGroup1()
{
   group1.Text += acc.group1;
   MessageBox.Show(acc.group1);
   //acc.b();
}
private void Form1_Load(object sender, EventArgs e)
{
   //Classes.Receiver rc = new Classes.Receiver();
   new Thread((mod_Receive)).Start();
}

private void btnkeyGA_Click(object sender, EventArgs e)
{
   CentralAuthority.CreatePrimeTable();
   _caClient.GenerateParameters(_caServer.Prime, _caServer.Generator);
   lblpka.Text = NetworkUtility.Base64Encode(KeyAssign.publickey);
   lblska.Text = NetworkUtility.Base64Encode(KeyAssign.privatekey);
   NetworkUtility.pka = lblpka.Text;
}

private void btnkeyGB_Click_1(object sender, EventArgs e)
{
   CentralAuthority.CreatePrimeTable();
   _caClient.GenerateParameters(_caServer.Prime, _caServer.Generator);
   lblbkb.Text = NetworkUtility.Base64Encode(KeyAssign.publickey);
   lblskb.Text = NetworkUtility.Base64Encode(KeyAssign.privatekey);
   NetworkUtility.pkb = lblbkb.Text;
}

private void btndistributekey_Click_1(object sender, EventArgs e)
{
   if (!string.IsNullOrEmpty(NetworkUtility.pka) && !string.IsNullOrEmpty(NetworkUtility.pkb))
   {

      NetworkUtility.membersTB = dtA;
      NetworkUtility.send();
   }
   else
```

```
        {
            MessageBox.Show("First generate a Group Key and then Distribute!");
        }
    }

    private void radTitleBar1_Click(object sender, EventArgs e)
    {

    }

    private void button1_Click(object sender, EventArgs e)
    {
        loadimage();
    }

  }
}
```


Sensing:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;

namespace CDAMA.Classes
{
[Serializable()]
  public class sensors:IComparable<sensors>
   {
      private int _SNId;
      private string _GroupId;
      private IPAddress ip;
      private int _port_number;
      public sensors(int id, string groupname, IPAddress ip, int port)
      {
          this._SNId = id;
          this._GroupId = groupname;
          this.ip = ip;
          this._port_number = port;
      }
      public int getID()
      {
          return this._SNId;
```

```
}

public void setID(int id)
{
   this._SNId = id;
}
public string getGroupname()
{
   return this._GroupId;
}
public void setGroupname(string groupname)
{
   this._GroupId = groupname;
}
public void setIpaddress(IPAddress ipaddr)
{
   this.ip = ipaddr;
}
public IPAddress getIpaddr()
{
   return this.ip;
}
public void setPort(int portNo)
{
   this._port_number = portNo;
}
public int getPort()
{
   return this._port_number;
}
public override bool Equals(object p)
{
   sensors p1 = (sensors)p;
   if (p1._SNId != this._SNId)
   {
      return false;
   }
   else
   {
      return true;
   }
}


public override string ToString()
{
```

```
        return string.Format(this._SNId + ":" + this._GroupId + ":" + this.ip + ":" + this._port_number);
    }

    public SensorNode.Classes.DataModel getDataModel()
    {
        return new SensorNode.Classes.DataModel(this._SNId, this._GroupId, NetworkUtility.getLocalHost(),
"sensorinfo", this);
    }
    #region IComparable<Sensors> Members

    public int CompareTo(sensors other)
    {
        throw new NotImplementedException();
    }

    #endregion
  }
}
```


Packet Sending Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Data;
using System.Net.Sockets;
using System.Runtime.Serialization.Formatters.Binary;
using System.Threading;

namespace CDAMA.Classes
{
  public class NetworkUtility
  {
    public static string pka { set; get; }
    public static string pkb { set; get; }
    public static DataTable membersTB { get; set; }
    //public static DataTable groupB { get; set; }
     public static IPAddress getLocalHost()
     {
        IPHostEntry host;
        IPAddress localIP = null;
        host = Dns.GetHostEntry(Dns.GetHostName());
        foreach (IPAddress ip in host.AddressList)
        {
          if (ip.AddressFamily.ToString() == "InterNetwork")
```

```
            {
                localIP = ip;
            }
        }
        return localIP;
       }
    public static string Base64Encode(string plainText)
    {
        var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(plainText);
        return System.Convert.ToBase64String(plainTextBytes);
    }

    public static void send()
    {
        int rows= membersTB.Rows.Count;
        for (int i = 0; i < rows; i++)
        {
            //Object stablenode = null;
            TcpClient tcp = new TcpClient(membersTB.Rows[i][0].ToString(),Convert.ToInt32(membersTB.Rows[i][1].ToString()));
            NetworkStream ns = tcp.GetStream();
            BinaryFormatter bf = new BinaryFormatter();
            if (membersTB.Rows[i][2].ToString() == "Group1")
            {
                bf.Serialize(ns, "key");
                bf.Serialize(ns, pka);
            }
            else
            {
                bf.Serialize(ns, "key");
                bf.Serialize(ns, pkb);
            }
            ns.Close();
            tcp.Close();
            Thread.Sleep(200);
        }



    }
  }
}
```

VII. CONCLUSION

We have investigated pollution attacks of lossy network coding in heterogeneous wireless networks. In particular, helper nodes in heterogeneous wireless networks played the role of watchdogs that can overhear and monitor packet transmissions from the same wireless node. We proposed a novel watchdog cooperative monitoring scheme, i.e.,

VANDER, to address pollution attacks in the lossy wireless environment. In VANDER, low overhead packets that contained random Vandermonde hashes were introduced to share packet information and check the validity of packets when lossy overhearing happened at watchdogs. As each column subspace of a Vandermonde matrix could be calculated efficiently, only a small overhead was introduced among watchdogs, and no extra overhead was introduced to normal transmission nodes due to the cooperation. Moreover, even successive colluded adversarial nodes could be detected. We demonstrated that VANDER had provably high security guarantee, low computational complexity (linear respect to the block length n), and a constant upper bounded communication overhead when $2n$ was no more than the field size (whose typical value is 232).

VIII. REFERENCE

- Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks," IEEE Comm. Magazine, vol. 40, no. 8, pp. 102-114, Aug. 2002.
- R. Min and A. Chandrakasan, "Energy-Efficient Communication for Ad-Hoc Wireless Sensor Networks," Proc. Conf. Record of the 35th Asilomar Conf. Signals, Systems and Computers, vol. 1, 2001.
- B. Przydatek, D. Song, and A. Perrig, "SIA: Secure Information Aggregation in Sensor Networks," Proc. First Int'l Conf. Embedded Networked Sensor Systems, pp. 255-265, 2003.
- A. Perrig, J. Stankovic, and D. Wagner, "Security in Wireless Sensor Networks," Comm. ACM, vol. 47, no. 6, pp. 53-57, June 2004.
- L. Hu and D. Evans, "Secure Aggregation for Wireless Networks," Proc. Symp. Applications and the Internet Workshops, pp. 384-391, 2003.
- H. Cam, S. O ¨ zdemir, P. Nair, D. Muthuavinashiappan, and H.O. Sanli, "Energy-Efficient Secure Pattern Based Data Aggregation for Wireless Sensor Networks," Computer Comm., vol. 29, no. 4, pp. 446-455, 2006.
- H. Sanli, S. Ozdemir, and H. Cam, "SRDA: Secure Reference-based Data Aggregation Protocol for Wireless Sensor Networks," Proc. IEEE 60th Vehicular Technology Conf. (VTC 04-Fall), vol. 7, 2004.
- Y. Wu, D. Ma, T. Li, and R.H. Deng, "Classify Encrypted Data in Wireless Sensor Networks," Proc. IEEE 60th Vehicular Technology Conf., pp. 3236-3239, 2004.
- D. Westhoff, J. Girao, and M. Acharya, "Concealed Data Aggregation for Reverse Multicast Traffic in Sensor Networks: Encryption, Key Distribution, and Routing Adaptation," IEEE Trans. Mobile Computing, vol. 5, no. 10, pp. 1417-1431, Oct. 2006.