Maximizing Lifetime, Security and Data Aggregation using HEF Algorithm

Arpitha M[1] Shilpa.R[2] Soumya Bhusanur[3] Syeda Tameem[4]
[1,2,3,4]Student
[1,2,3,4]Department of Computer Science & Engineering
[1,2,3,4]NIE, Mysuru

Abstract— Wireless Sensor Networks have gained wide popularity in the recent years for its high-ranking applications such as remote environment monitoring, target tracking, safety-critical monitoring etc. However Wireless Sensor Networks face many constraints like low computational power, small storage, and limited energy resources. Two of the chief issues associated with Wireless Sensor Networks are network lifetime and data security which we aim to address here. In our proposed system we aim to maximize the network lifetime of Wireless Sensor Networks by choosing High Energy First (HEF) clustering algorithm as a design reference model for cluster head selection, which is to the best of our knowledge proved to be an optimal clustering policy under certain ideal conditions, together with a sleep/active algorithm to avoid any unnecessary energy dissipation by the sensor nodes. We also aim to address the issue of sensor data security by opting for Paillier homomorphism encryption, which is an end to end data security plan that can guarantee end-to-end data confidentiality with less transmission latency and computation cost contrasting with the hop by hop data security which involves more consumption of battery power due to several decryptions at every hop. In this paper we trying to maximize the lifetime of the data aggregation and security and we are trying to fix the benchmark for this[1].

Key words: Wireless Sensor Networks, Clustering, Cluster Head, Encryption, Decryption

I. INTRODUCTION

Wireless Sensor Network (WSN's) are used in many application in the area of tracking and monitoring. The sensor network's can be described as a collection of sensor nodes which co-ordinates to perform some specific action. The WSN is built of "nodes" – from a few to several hundreds or even thousands, where each node is connected to one (or sometimes several) sensors. Each such sensor network node has typically several parts :- a radio transceiver with an internal antenna or connection to an external antenna, a microcontroller, an electronic circuit for interfacing with the sensors and an energy source, usually a battery or an embedded form of energy harvesting.

The unique challenges for privacy and preservation in WSNs call for the development of effective privacy preserving techniques. For example a patient's blood pressure and sugar level are usually of critical privacy convert when monitored by a WSN which transmits the data to remote hospital or doctor office. The cost of sensor nodes is similarly variable, ranging from a few to hundreds of dollars, depending on the complexity of the individual sensor nodes. Size and cost constraints on sensor nodes result in corresponding constraints on resources such as energy, memory, computational speed and communications bandwidth[2].

The architecture of the sensor networks plays very important role in the performance of data aggregation protocols. Several data aggregation protocols have been proposed. These protocols can be classified based on network models. There are two categories : Tree based data aggregation and cluster based data aggregation.

The optimality of this algorithm also ensures to balance the energies of all the nodes within a cluster thereby reducing energy depletion by efficiently forming clusters in set-up phase. Also it is known that most of the energy consumption takes place during data aggregation, transmission and communication. During aggregation, data that is sensed by various sensor nodes are processed and encrypted for security purpose and sent to the aggregator node. It is important to note that the encryption and decryption functions do consume considerable amount of energy[3].

The work on secure data aggregation can be classified based on encryption of data at specific nodes like hop-by-hop encrypted data aggregation, end-to-end encrypted data aggregation and privacy homomorphism. In hop-by-hop encrypted data aggregation, all the intermediate sensor nodes has to decrypt the received encrypted data and apply aggregation function on it. Due to many decryptions perform by the intermediate node it's consuming more battery power and not provide end-to-end security[4].

In end-to-end encrypted data aggregation, intermediate nodes can aggregate the cipher text directly without decrypting the messages. Compared to the hop-by-hop one, it can guarantee the end-to-end data confidentiality and result in less transmission latency and computation cost. We try to reduce energy consumption further here by choosing the privacy homomorphism approach where certain aggregation functions can be calculated on the encrypted data. The data is encrypted and sent towards the base station, while sensors along the path apply the aggregation function on the encrypted data. This is much better than end-to-end approach because the base station receives the encrypted aggregate result and decrypts it essentially allowing only the BS to know the gist of the data thus providing security to the data even at the aggregator level.

II. SYSTEM DESIGN

Software design is a process of problem solving and planning for a software solution. After the purpose and specifications of software are determined, software developers will design or employ designers to develop a plan for a solution. It includes low-level component and algorithm implementation issues as well as the architectural view. The following includes the module for the proposed design[5].

All rights reserved by *www.ijsrd.com*

A. Module 1: Key Generation and Distribution:

1) In order to provide security to the data that is sensed by the sensor nodes in the network, the data has to be encrypted. Here, for encryption of the sensed data we have chosen a homomorphic encryption scheme called Paillier Cryptosystem.
2) Homomorphic encryption is a form of encryption that allows computations to be carried out on cipher text, thus generating an encrypted result which, when decrypted, matches the result of operations performed on the plaintext.
3) The Paillier cryptosystem, named after and invented by Pascal Paillier in 1999, is a probabilistic asymmetric algorithm for public key cryptography. The scheme is an additive homomorphic cryptosystem; this means that, given only the public-key and the encryption of m_1 and m_2, one can compute the encryption of m_1+m_2.
4) For every aggregation phase, the BS computes the public (encryption) keys (n, g) and the private (decryption) keys (μ, λ) and sends the public encryption key along with the data sensing request to the sensor nodes for data encryption.

B. Module 2: Cluster Formation and Aggregator Selection

The clustering of nodes is done by exchanging battery values with the neighboring nodes and cluster heads/aggregators are selected using the High Energy First (HEF) cluster head selection algorithm. All the nodes in the network exchange their node IPs and battery power and form clusters, the nodes within the clusters compare each other's battery values and elect the one having the highest battery power as Cluster Aggregator using HEF algorithm.

HEF algorithm:

1) The idea behind the HEF clustering algorithm is to choose the node with the highest residual energy within every cluster as the cluster head/aggregator respectively.
2) HEF selects aggregator according to the energy remaining for each sensor node by comparing the battery values of every sensor node within the cluster. The node with the highest battery value will be elected as the aggregator for that cluster for that particular round.
3) The aggregator of each cluster broadcasts its IP to the sensor nodes of its cluster announcing that it is the aggregator for the current round.
4) Each aggregator acknowledges itself by sending its IP address to the BS and registers as the cluster aggregator for the current round.
5) The aggregator forwards the time schedule sent by the BS to its cluster members for each round.

C. Module 3: Data Sensing and Forwarding

1) Each node in the cluster senses the data upon receiving data sensing request, encrypts it using the secret public key sent by the BS and then forwards the encrypted data to the aggregator node. Here we have chosen a temperature sensor LM35 to sense the surrounding temperature[6].

D. Module 4: Data Aggregation and Encryption

At the cluster head/aggregator node, data received from all cluster sensor nodes are added using homomorphic encryption scheme. Homomorphic encryption can be additive or multiplicative. We have adopted the additive scheme here. The additive homomorphic property is described below.

1) Homomorphic addition of plaintexts:The product of two cipher texts will decrypt to the sum of their corresponding plaintexts,
$$\left(E(m_1,r_1).E(m_2,r_2) \bmod n^2\right) = m_{1+}m_2 \bmod n$$
2) The product of a cipher text with a plaintext raising g will decrypt to the sum of the corresponding plaintexts,
$$\left(E(m_1,r_1).g^{m_2} \bmod n^2\right) = m_1 + m_2 \bmod n$$
3) The data thus encrypted is forwarded to the base station by all the cluster aggregator nodes.

E. Module 5: Data Decryption and Computation

1) The base station (Sink) receives the aggregate sensed data from aggregator nodes of all clusters along with the number of active nodes in a given cluster. It decrypts the received aggregate using its private keys (λ, μ).
2) The decryption is done using the following decryption rules;
   Let c be the cipher text to decrypt
   Compute the plaintext message as:
$$m = L(c^{\lambda} \bmod n^2) \cdot \mu \bmod n$$
3) The average of the decrypted aggregate data is computed by dividing the aggregate sum by the number of active nodes in a given cluster which gives the average aggregate data sensed by every cluster. The BS gets to know the result/sum of the data sent but will not come to know about the values that were added to get the result/sum thus protecting its integrity.

III. IMPLEMENTATION

After the design part we are trying to implement the above modules based on the requirements provided. The implementation is shown below.

A. Network Sockets

A network socket is an endpoint of a connection across a computer network. Today, most communication between computers is based on the Internet Protocol; therefore most network sockets are Internet sockets. More precisely, a socket is a handle (abstract reference) that a local program can pass to the networking application programming interface (API) to use the connection, for example "send this data on this socket". Sockets are internally often simply integers, which identify which connection to use.

A socket API is an application programming interface (API), usually provided by the operating system, that allows application programs to control and use network sockets. Internet socket APIs are usually based on the Berkeley sockets standard. In the Berkeley sockets standard, sockets are a form of file descriptor (a file handle), due to the Unix philosophy that "everything is a file", and the analogies between sockets and files: you can read, write, open, and close both. In practice the differences mean the analogy is strained, and one instead use different interfaces

All rights reserved by *www.ijsrd.com*

(send and receive) on a socket. In inter-process communication, each end will generally have its own socket, but these may use different APIs: they are abstracted by the network protocol.

A socket address is the combination of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particular extension. Sockets need not have an address (for example for only sending data), but if a program binds a socket to an address, the socket can be used to receive data sent to that address. Based on this address, internet sockets deliver incoming data packets to the appropriate application process or thread[3].

B. User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.

UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network protocol. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.

With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths.

UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level.

Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.[1] If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

C. Serialization

Serialization is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.

– Uses for Serialization

Serialization allows the developer to save the state of an object and recreate it as needed, providing storage of objects as well as data exchange. Through serialization, a developer can perform actions like sending the object to a remote application by means of a Web Service, passing an object from one domain to another, passing an object through a firewall as an XML string, or maintaining security or user-specific information across applications.

– Making an Object Serializable

To serialize an object, you need the object to be serialized, a stream to contain the serialized object, and a Formatter. System. Runtime. Serialization contains the classes necessary for serializing and deserializing objects.

Apply the Serializable Attribute attribute to a type to indicate that instances of this type can be serialized. A Serialization Exception exception is thrown if you attempt to serialize but the type does not have the Serializable Attribute attribute.

If you do not want a field within your class to be serializable, apply the Non Serialized Attribute attribute. If a field of a serializable type contains a pointer, a handle, or some other data structure that is specific to a particular environment, and the field cannot be meaningfully reconstituted in a different environment, then you may want to make it non serializable. If a serialized class contains references to objects of other classes that are marked Serializable Attribute, those objects will also be serialized.

D. Transmission Control Protocol(TCP)

The Transmission Control Protocol (TCP) is a core protocol of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network. Major Internet applications such as the World Wide Web, email, remote administration and file transfer rely on TCP. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.

The Transmission Control Protocol provides a communication service at an intermediate level between an application program and the Internet Protocol. It provides host-to-host connectivity at the Transport Layer of the Internet model. An application does not need to know the particular mechanisms for sending data via a link to another host, such as the required packet fragmentation on the transmission medium. At the transport layer, the protocol handles all handshaking and transmission details and presents an abstraction of the network connection to the application.

At the lower levels of the protocol stack, due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets may be lost, duplicated, or delivered out of order. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data, and even helps minimize network congestion to reduce the occurrence of the other problems. If the data still remains undelivered, its source is notified of this failure. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the receiving application. Thus, TCP abstracts the application's communication from the underlying networking details.

TCP is utilized extensively by many popular applications carried on the Internet, including the World Wide Web (WWW), E-mail, File Transfer Protocol, Secure Shell, peer-to-peer file sharing, and many streaming media applications.

All rights reserved by *www.ijsrd.com*

E. Remoting

.NET Remoting is a Microsoft application programming interface (API) for interprocess communication released in 2002 with the 1.0 version of .NET Framework. It is one in a series of Microsoft technologies that began in 1990 with the first version of Object Linking and Embedding (OLE) for 16-bit Windows. Intermediate steps in the development of these technologies were Component Object Model (COM) released in 1993 and updated in 1995 as COM-95, Distributed Component Object Model (DCOM), released in 1997 (and renamed Active X), and COM+ with its Microsoft Transaction Server (MTS), released in 2000. It is now superseded by Windows Communication Foundation (WCF), which is part of the .NET Framework 3.0.

Like its family members and similar technologies such as Common Object Request Broker Architecture (CORBA) and Java's remote method invocation (RMI), .NET Remoting is complex, yet its essence is straightforward. With the assistance of operating system and network agents, a client process sends a message to a server process and receives a reply.

F. Encryption

In cryptography, encryption is the process of encoding messages or information in such a way that only authorized parties can read it. Encryption does not of itself prevent interception, but denies the message content to the interceptor. In an encryption scheme, the intended communication information or message, referred to as plaintext, is encrypted using an encryption algorithm, generating ciphertext that can only be read if decrypted.
For technical reasons, an encryption scheme usually uses a pseudo-random encryption key generated by an algorithm. It is in principle possible to decrypt the message without possessing the key, but, for a well-designed encryption scheme, large computational resources and skill are required. An authorized recipient can easily decrypt the message with the key provided by the originator to recipients, but not to unauthorized interceptors.

The purpose of encryption is to ensure that only somebody who is authorized to access data (e.g. a text message or a file), will be able to read it, using the decryption key. Somebody who is not authorized can be excluded, because he or she does not have the required key, without which it is impossible to read the encrypted information.

1) Types of Encryption

a) Symmetric key encryption

In symmetric-key schemes, the encryption and decryption keys are the same. Communicating parties must have the same key before they can achieve secure communication.

Fig. 1: Symmetric Key

G. Public Key Encryption

Illustration of how encryption is used within servers Public key encryption.

In public-key encryption schemes, the encryption key is published for anyone to use and encrypt messages. However, only the receiving party has access to the decryption key that enables messages to be read. Public-key encryption was first described in a secret document in 1973; before then all encryption schemes were symmetric-key (also called private-key). A publicly available public key encryption application called Pretty Good Privacy (PGP) was written in 1991 by Phil Zimmermann, and distributed free of charge with source code; it was purchased by Symantec in 2010 and is regularly updated.

a) Uses of encryption

Encryption has long been used by military and governments to facilitate secret communication. It is now commonly used in protecting information within many kinds of civilian systems. For example, the Computer Security Institute reported that in 2007, 71% of companies surveyed utilized encryption for some of their data in transit, and 53% utilized encryption for some of their data in storage. Encryption can be used to protect data "at rest", such as information stored on computers and storage devices (e.g. USB flash drives). In recent years there have been numerous reports of confidential data such as customers' personal records being exposed through loss or theft of laptops or backup drives. Encrypting such files at rest helps protect them should physical security measures fail. Digital rights management systems, which prevent unauthorized use or reproduction of copyrighted material and protect software against reverse engineering (see also copy protection), is another somewhat different example of using encryption on data at rest.

Encryption is also used to protect data in transit, for example data being transferred via networks (e.g. the Internet, e-commerce), mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. There have been numerous reports of data in transit being intercepted in recent years. Data should also be encrypted when transmitted across networks in order to protect against eavesdropping of network traffic by unauthorized users.

b) Additive Homomorphism

In abstract algebra, a homomorphism is a structure-preserving map between two algebraic structures (such as groups, rings, or vector spaces). The word homomorphism comes from the ancient Greek language: homos meaning "same" and morphe meaning "form" or "shape". Isomorphisms, automorphisms, and endomorphisms are special types of homomorphisms.

A homomorphism is a map that preserves selected structure between two algebraic structures, with the structure to be preserved being given by the naming of the homomorphism.

Particular definitions of homomorphism include the following:

a) A semi group homomorphism is a map that preserves an associative binary operation.
b) A monoid homomorphism is a semi group homomorphism that maps the identity element to the identity of the co domain.
c) A group homomorphism is a homomorphism that preserves the group structure. It may equivalently be defined as a semi group homomorphism between groups.

All rights reserved by *www.ijsrd.com*

d) A ring homomorphism is a homomorphism that preserves the ring structure. Whether the multiplicative identity is to be preserved depends upon the definition of ring in use.
e) A linear map is a homomorphism that preserves the vector space structure, namely the abelian group structure and scalar multiplication. The scalar type must further be specified to specify the homomorphism, e.g. every R-linear map is a Z-linear map, but not vice versa.
f) A module homomorphism is a map that preserves module structures.
g) An algebra homomorphism is a homomorphism that preserves the algebra structure.
h) A functor is a homomorphism between two categories.

The Paillier cryptosystem, named after and invented by Pascal Paillier in 1999, is a probabilistic asymmetric algorithm for public key cryptography. The problem of computing n-th residue classes is believed to be computationally difficult. The scheme is an additive homomorphic cryptosystem; this means that, given only the public-key and the encryption of m1 and m2, one can compute the encryption of m1+m2

IV. APPLICATIONS

A. Electronic Voting

Semantic security is not the only consideration. There are situations under which malleability may be desirable. The above homomorphic properties can be utilized by secure electronic voting systems. Consider a simple binary ("for" or "against") vote. Let m voters cast a vote of either 1 (for) or 0 (against). Each voter encrypts their choice before casting their vote. The election official takes the product of the m encrypted votes and then decrypts the result and obtains the value n, which is the sum of all the votes. The election official then knows that n people voted for and m-n people voted against. The role of the random r ensures that two equivalent votes will encrypt to the same value only with negligible likelihood, hence ensuring voter privacy.

B. Electronic Cash

Another feature named in paper is the notion of self-blinding. This is the ability to change one ciphertext into another without changing the content of its decryption. This has application to the development of ecash, an effort originally spearheaded by David Chaum. Imagine paying for an item online without the vendor needing to know your credit card number, and hence your identity. The goal in both electronic cash and electronic voting, is to ensure the e-coin (likewise e-vote) is valid, while at the same time not disclosing the identity.

V. CONCLUSION

We try to achieve maximum network lifetime by reducing energy consumption at the cluster formation phase and also at the data communication phase with an added property of security to the transmitted data. A set of end-to-end encrypted data aggregation protocols were proposed by and in order to overcome the drawbacks of the hop by-hop encrypted data aggregation. Paillier Homomorphism (PH) is an encryption transformation that allows direct computation on encrypted data.

REFERENCES

[1] D. J. Baker and A. Ephremides, "The architectural organization of a mobile radio network via a distributed algorithm," in IEEE Transactions on Communications COM-29(11), 1981.
[2] G. V. Crosby, N. Pissinou, and J. Gadze, "A framework for trust-based cluster head election in wireless sensor networks," in DSSNS 2006 : Second IEEE Workshop on Dependability and Security in Sensor Networks and Systems : Proceedings, Columbia, Maryland, April 24-28, 2006, sponsored by IEEE, NASA, IEEE Computer Society.
[3] L.-C. Wang, C.-W. Wang, and C.-M. Liu, "Adaptive contention window-based cluster head election algorithms for wireless sensor networks," in VTC-2005-Fall. 2005 IEEE 62nd, September 2005, vol.3.
[4] S. Sivavakeesar and G. Pavlou, "Associativity-based Stable cluster formation in mobile ad hoc networks," in Proceedings of IEEE Conference on Consumer Communications and Networking Conference (CCNC2005), January 2005, IEEE.
[5] E. Chu, T. Mine, and M. Amamiya, "A data gathering mechanism based on clustering and in-network processing routing algorithm: CIPRA," in The Third International Conference on Mobile Computing and Ubiquitous Networking, ICMU, 2006.
[6] W. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "Energyefficient communication protocol for wireless microsensor networks," in System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on Publication Date: 4-7 Jan. 2000, vol. 2.

All rights reserved by *www.ijsrd.com*