

IoT Devices Authenticity through IBC in Secure CoT Enabled Environment

Progress Report
In fulfillment of the requirements for the
NU 302 R&D Project
At NIIT University



Submitted by
Simran Tiwari
Shruti Rao
Rahul Nikki
Julakanti Harihanth
Jazmyn Singh

Area
NIIT University
Neemrana
Rajasthan

CERTIFICATE

This is to certify that the present research work entitled "IoT Devices Authenticity through IBC in Secure CoT Enabled Environment" being submitted to NIIT University, Neemrana, Rajasthan, in the fulfillment of the requirements for the course at NIIT University, Neemrana, embodies authentic and faithful record of original research carried out by Simran Tiwari ,Shruti Rao,Rahul Nikki ,Julakanti Harihanth ,Jazmyn Singh , student/s of B Tech (CSE) at NIIT University, Neemrana,. She /He has worked under our supervision and that the matter embodied in this project work has not been submitted, in part or full, as a project report for any course of NIIT University, Neemrana or any other university.

Name and Title of the Mentor

ARIJIT KARATI, Ph.D.

Assistant Professor,

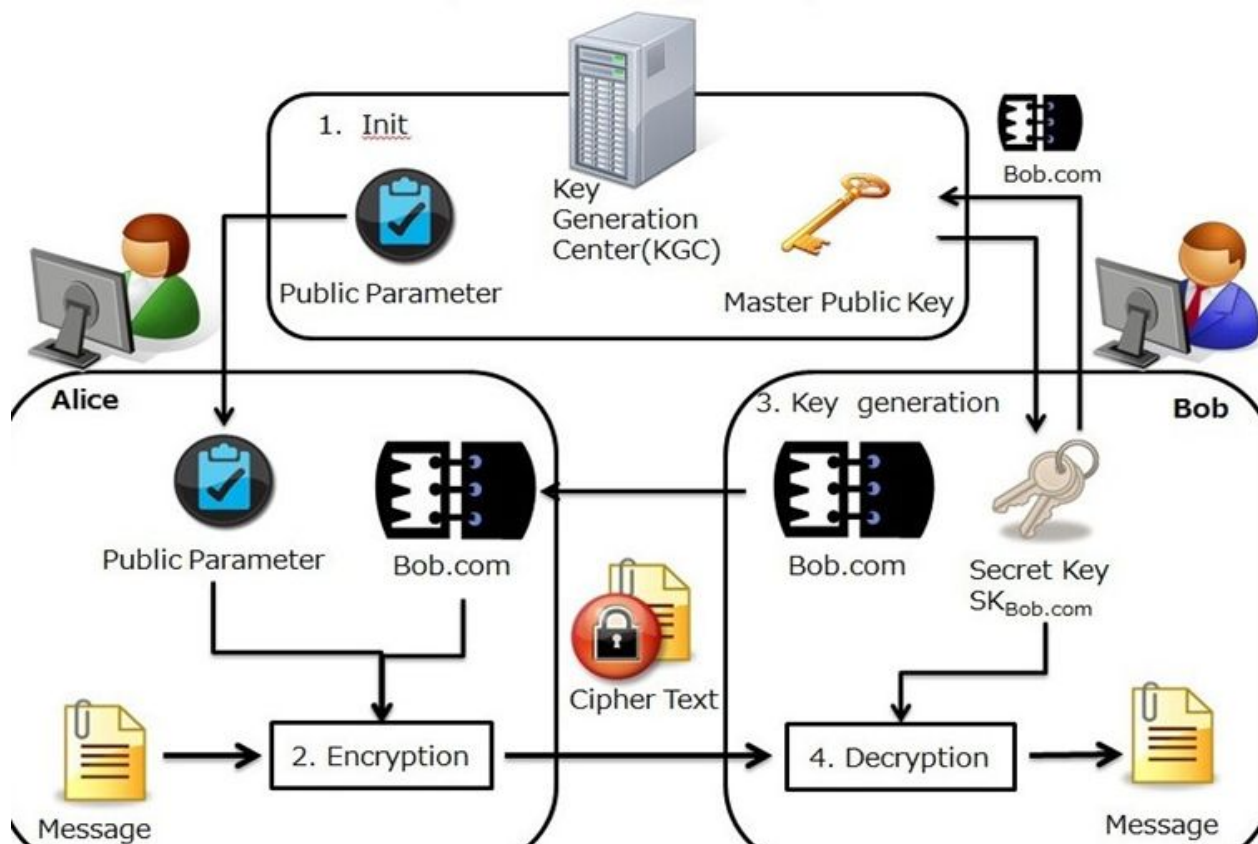
COMPUTER SCIENCE AND ENGINEERING,

TABLE OF CONTENTS

Title	Page no.
Certificate	II
List of Figures	II-IV
Rational	4-8
Literature Review	9-24
Objectives	25
Methodology	26-43
Results	44-46
Summary	47
Future Work	48
References	49

1) Rational of the work :

In large-scale Internet of Things (IoT) systems, huge volumes of data are collected from anywhere at any time, which may invade people's privacy, especially when the systems are used in medical or daily living environments. Preserving privacy is an important issue, and higher privacy demands usually tend to require weaker identity. However, previous research has indicated that strong security tends to demand strong identity, especially in authentication processes. Thus, defining a good tradeoff between privacy and security remains a challenging problem. This motivates us to develop a privacy-preserving and accountable authentication protocol for IoT end-devices



The Internet of Things (IoT) is becoming the world's largest computing platform, which collects valuable data from anywhere at any time. Intelligence can be extracted from the collected data to build a smarter world. The IoT is now changing the way we live, work, and learn. However, the IoT creates many security problems along with its benefits[3] since in IoT systems, the physical world and cyber world are more tightly coupled than before, which makes the physical world easier to attack. A malicious instruction from the cyber world may bring about serious damage in the physical world, and thus, IoT security issues should be handled effectively. However, IoT systems have many special properties such as resource-constrained end devices, high privacy requirements, heterogeneous devices and communications, and trust management. An IoT system is a typical three-layer architecture. We depict these layers as follows. The first layer is the things layer, which is used to collect data and control the physical world. The things layer in Fig. 1 is a smart home system in which some end-devices such as thermal detectors, cameras, and smoke detectors have the ability to perceive the real world and to send critical control instructions. Most end-devices in the things layer are resource constrained, having as low as 64 bytes of RAM and 2 KB of storage[6]. Thus, many conventional security mechanisms such as asymmetric key encryption-based algorithms are not suitable, since they have high resource needs including computational power and energy. Additionally, privacy issues are also important for end devices in IoT systems, because end-devices collect a huge amount of personal data, especially when they are deployed in medical or daily living environments.

Higher privacy demands usually tend to require weaker identity, since strong identity threatens the privacy of individuals. But having a strong identity is useful for protecting security, especially in authentications. So, it is difficult to select an appropriate tradeoff between security and privacy. •

The second layer is the edge layer, which is introduced to help end-devices. Edge-devices, such as smartphones, routers, and home servers, also help connect end-devices to the cloud. There are some advantages in deploying security services at the edge layer. First, edge-devices have many more resources than end devices. Additionally, end-devices can leverage these resources to offload computationally intensive tasks, such as asymmetric encryption operations and key generation. Second, edge-devices are physically close to end-devices, which can set up relatively stable relationships. Such stable relationships are very beneficial to establishing trust between them. Third, edge-devices have more information about the overall IoT system than end-devices do, and thus, they can use better optimized security management mechanisms. Finally, edge-devices can be used to protect the privacy of the Internet of Things (IoT) is becoming the world's largest computing platform[1], which collects valuable data from anywhere at any time. Intelligence can be extracted from the collected data to build a smarter world. The IoT is now changing the way we live, work, and learn. However, the IoT creates many security problems along with its benefits since in IoT systems, the physical world and cyber world are more tightly coupled than before, which makes the physical world easier to attack. A malicious instruction from the cyber world may bring about serious damage in the

physical world, and thus, IoT security issues should be handled effectively. However, IoT systems have many special properties such as resource-constrained end devices, high privacy requirements, heterogeneous devices and communications, and trust management. , an IoT system is a typical three-layer architecture . We depict these layers as follows. The first layer is the things layer, which is used to collect data and control the physical world. The things layer in is a smart home system in which some end-devices such as thermal detectors, cameras, and smoke detectors have the ability to perceive the real world and to send critical control instructions. Most end-devices in the things layer are resource constrained, having as low as 64 bytes of RAM and 2 KB of storage. Thus, many conventional security mechanisms such as asymmetric key encryption-based algorithms are not suitable, since they have high resource needs including computational power and energy. Additionally, privacy issues are also important for end devices in IoT systems, because end-devices collect a huge amount of personal data, especially when they are deployed in medical or daily living environments. Higher privacy demands usually tend to require weaker identity, since strong identity threatens the privacy of individuals. But having a strong identity is useful for protecting security, especially in authentications. So, it is difficult to select an appropriate tradeoff between security and privacy. The second layer is the edge layer, which is introduced to help end-devices. Edge-devices, such as smartphones, routers, and home servers, also help connect end-devices to the cloud. There are some advantages in deploying security services at the edge layer. First, edge-devices have many more resources

than end devices. Additionally, end-devices can leverage these resources to offload computationally intensive tasks, such as asymmetric encryption operations key generation. Second, edge-devices are physically close to end-devices, which can set up relatively stable relationships. Such stable relationships are very beneficial to establishing trust between them. Third, edge-devices have more information about the overall IoT system than end-devices do, and thus, they can use better optimized security management mechanisms. Finally, edge-devices can be used to protect the privacy of end devices with weaker identity by using secure authentication protocols. The third layer is the cloud layer, which is resource rich, and can be used to store, process, and analyze the collected data. The edge layer usually has a high-speed connection with the cloud layer, and it is easy for the edge-devices to receive extra support from the cloud as needed.

2) Literature Review :

a)Developing A Secure Cloud Storage System for Storing IoT Data by Applying Role Based Encryption.

Author:JayantD. Bokefodea.et all[5]

Year :- 2016

Abstract:

Internet of things is one of the most emerging and popular technology, which has changed our life, by impacting different areas such as shopping, enterprise production, storage, monitoring physical devices, etc. The Internet of Things (IoT) is an atmosphere in which all physical objects, peoples or animals are having unique identity and they are able to transfer data over a network without any interaction. An improvement in electronic sensing devices and rapid growth in communication infrastructure, and their monitoring systems give very fast access to retrieve data and allow communication with physical devices.

Now days, Organizations use IoT devices to collect real time and continuous data and make better business decisions to increase customer satisfaction. But collected data need to be processed and transferred in appropriate format to store on storage system – which is triggering organizations to rethink their data storage infrastructures. An enterprise has to store data generated from the Internet of Things and this data grows

exponentially, it forces to think about cloud storage for storing IoT data. The proposed work allows the organization to store the IoT data on the cloud securely by applying different Access control policies and the cryptography concepts.

Conclusion:

This system, addresses the security issues in a cloud storage environment. The presented approach, allows an organization to upload IoT data securely in a public cloud, while organizational information stored on a private cloud. The implemented approach provides great efficiency during encryption and decryption of the message. The implemented system is useful in various commercial organizations, where data is collected from IoT devices and job functionalities are divided according to the roles played by the user in organizations. The collected data from IoT devices is securely uploaded on cloud storage by enforcing AES and RSA cryptographic techniques.

b)A Privacy-Preserving and Accountable Authentication Protocol for IoT End-Devices with Weaker Identity

Author :- Zhiwei Wang

Year :-2017

Abstract :

In large-scale Internet of Things (IoT) systems, huge volumes of data are collected from anywhere at any time, which may invade people's privacy, especially when the systems are used in medical or daily living environments. Preserving privacy is an important issue, and higher privacy demands usually tend to require weaker identity. However, previous research has indicated that strong security tends to demand strong identity, especially in authentication processes. Thus, defining a good tradeoff between privacy and security remains a challenging problem. This motivates us to develop a privacy-preserving and accountable authentication protocol for IoT end-devices with weaker identity, which integrates an adapted construction of short group signatures and Shamir's secret sharing scheme. We analyze the security properties of our protocol in the context of six typical attacks and verify the formal security using the Proverif tool. Experiments using our implementation in MacBook Pro and Intel Edison development platforms show that our authentication protocol is feasible in practice.

CONCLUSION :

In this work, we propose a novel authentication protocol for IoT end-devices with weaker identity, which can achieve a good tradeoff

between privacy and security. Our protocol integrates an adapted construction of a short group signature and Shamir's secret sharing scheme as an effective approach to achieve five goals: security, efficiency, privacy preservation, accountability and dynamic removal. We demonstrate the security of our protocol by theoretical analysis.

c) Identity-Based Authentication Scheme for the Internet of Things.

Author :- Mohd. Aazam, Pham P.Hung, E.N Huh

Year :-2015

Abstract :

Security and privacy are among the most pressing concerns that have evolved with the Internet. As networks expanded and became more open, security practices shifted to ensure protection of the ever growing Internet, its users, and data. Today, the Internet of Things (IoT) is emerging as a new type of network that connects everything to everyone, everywhere. Consequently, the margin of tolerance for security and privacy becomes narrower because a breach may lead to large-scale irreversible damage. One feature that helps alleviate the security concerns is authentication. While different authentication schemes are used in vertical network silos, a common identity and authentication scheme is needed to address the heterogeneity in IoT and to integrate the different protocols present in IoT. We propose in this paper an identity-based authentication scheme for heterogeneous IoT. The correctness of the proposed scheme is tested with the AVISPA tool and results showed that our scheme is immune to masquerade, man-in-the-middle, and replay attacks.

CONCLUSION :

Security is a crucial concern in networks. In particular, with IoT, security implications are even more pronounced since the impact of attacks is more drastic, mainly because IoT includes a wide range of applications from

simple location awareness to very critical healthcare uses. In this paper, we proposed an identification and authentication scheme for heterogeneous IoT networks based on SDN. The central SDN controller translates the different technology-specific identities from the different silos into a shared identity based on virtual IPv6 addresses and authenticates devices and

d)Reliable and Secure Traffic Exchange Approach for Internet of Things (IoT) Devices

Author:-FagenLi,YananHan,ChunhuaJin

Year:-2015

Abstract :

Different authentication mechanisms have been proposed for different fields using different protocols and encryption mechanisms. Certificates have been also widely deployed for authenticating Wireless sensor networks and IoT devices. In addition, a two-phase authentication protocol for wireless sensor networks in distributed IoT applications has been proposed in. It supports resource limitation of sensor nodes and takes into consideration network scalability and heterogeneity. This protocol also encompasses an end to end application layer authentication approach, and it depends on other lower layer security features. A secure and efficient authentication and authorization architecture for IoT-based healthcare using smart gateways has been proposed in. It mainly depends on the certificate-based DTLS handshake protocol to authenticate medical sensors. Moreover, DTLS handshakes depend on ECC, RSA and X.509 certificates. Another certificate-based authentication mechanism with the TPM is provided in . In this authentication mechanism, both direct user's request access to IoT devices and user' request tv access to IoT devices are performed through the IoT Cloud, where certificates are issued by a trusted third party, such as a Certification Authority (CA). However, the main drawbacks of the certificate-based approach using a third-party certificate authority are the high communication and computation overhead resulted from certificates and the vulnerability of the authentication

mechanism to attacks of certificates. The ID-based approach utilizes a sensor identity to complete authentication, and its identity can be hardware or software based. The enhanced mutual authentication model of IoT using RFID improves the algorithm of authentication of the challenge-response-based RFID authentication protocol for a distributed database environment, thus making it more suitable to an IoT control system environment. An ID-based multiple authentication scheme against attacks in wireless sensor networks proposes that each sensor node should be authenticated by its neighbor's ID.

The authentication process is performed at the sink that maintains a binding list of nodes in authentication neighbors' ID. A dynamic ID-based authentication scheme for M2M communication of healthcare systems uses IoT device ID as a secret key in the system to mutually authenticate each other where the probabilistic key management framework is being applied to assign key information to each node. Yet, the main drawbacks of the ID-based authentication are the restrictions of the RFID support device and most of ID-based authentication depends on hashing for encrypting data which is vulnerable to different types of attacks. The group-based authentication approach mainly depends on a group communication model, Threshold Cryptography-based Group Authentication (TCGA) scheme for the Internet of Things (IoT). Other authentication mechanisms mainly depend on application layer protocols and encryption protocols to provide a reliable authentication mechanism. Examples of these are ECC with HTTP used in [10], and AES with CoAP been used in [11], where 128-bit is used. OAuth 2.0 protocol was used in [12], where Security manager using OAuth 2.0 protocol was also built as a third party. Thus, the HTTP protocol adds extra communication overhead than CoAP, however AES require long keys to provides reliable and secure communication.

CONCLUSION :

The main goal of this paper is provide a reliable and secure authentication mechanism for IoT networks. IoT concepts, architecture and protocols have been investigated and reviewed. The restrictions or limitations of the IoT device make the traditional network mechanism authentication unreliable where high communication and computation overhead is required.

Therefore, they proposed a mechanism that utilizes shorter keys and higher security of ECC for the authentication mechanism over CoAP protocol which is a lightweight application Protocol. One of the interesting aspects for future work is the group based authentication, where a group of IoT devices can authenticate itself as a single unit with control machines. Inter communication between IoT devices can be done to minimize the communication overload with the control machine. This enhanced authentication mechanism can fit the requirement of high density IoT device networks where reduction of authentication cost is required.

e)A Lightweight Encryption Algorithm for Secure Internet of Things

Author:-Sheetal Kalra, Sandeep K. Sood

Year:-2015

Abstract :

The Internet of Things (IoT) being a promising technology of the future is expected to connect billions of devices. The increased number of communication is expected to generate mountains of data and the security of data can be a threat. The devices in the architecture are essentially smaller in size and low powered. Conventional encryption algorithms are generally computationally expensive due to their complexity and requires many rounds to encrypt, essentially wasting the constrained energy of the gadgets. Less complex algorithm, however, may compromise the desired integrity. In this paper we propose a lightweight encryption algorithm named as Secure IoT (SIT). It is a 64-bit block cipher and requires 64-bit key to encrypt the data. The architecture of the algorithm is a mixture of

feistel and a uniform substitution-permutation network. Simulations result shows the algorithm provides substantial security in just five encryption rounds. The hardware implementation of the algorithm is done on a low cost 8-bit micro-controller and the results of code size, memory utilization and encryption/decryption execution cycles are compared with benchmark encryption algorithms.

CONCLUSION:

In the near future Internet of Things will be an essential element of our daily lives. Numerous energy constrained devices and sensors will continuously be communicating with each other the security of which must not be compromised. For this purpose a lightweight security algorithm is proposed in this paper name SIT. The implementation show promising results making the algorithm a suitable candidate to be adopted in IoT applications. In the near future we are interested in the detail performance evaluation

f)Improving the Security of Internet of Things Using Encryption Algorithms

Author:- Amirhossein Safi

Year:-2016

Abstract:-

Internet of things (IOT) is a kind of advanced information technology which has drawn societies' attention. Sensors and stimulators are usually recognized as smart devices of our environment. Simultaneously, IOT security brings up new issues. Internet connection and possibility of interaction with smart devices cause those devices to involve more in human life. Therefore, safety is a fundamental requirement in designing IOT. IOT has three remarkable features: overall perception, reliable transmission, and intelligent processing. Because of IOT span, security of conveying data is an essential factor for system security. Hybrid encryption technique is a new model that can be used in IOT. This type of encryption

generates strong security and low computation. In this paper, we have proposed a hybrid encryption algorithm which has been conducted in order to reduce safety risks and enhancing encryption speed and less computational complexity. The purpose of this hybrid algorithm is information integrity, confidentiality, non repudiation in data exchange for IOT. Eventually, the suggested encryption algorithm has been simulated by MATLAB software, and its speed and safety efficiency were evaluated in comparison with conventional encryption algorithm.

CONCLUSION:

In this paper, we have discussed IOT, introduction and its usage, models, methods and security frameworks and also encryption algorithm in connection with IOT that researchers have studied before. Also, we have studied the suggested method in hybrid encryption algorithm used in IOT. We have provided a suggested method that can improve IOT by hybrid encryption algorithm.

HAN algorithm is considered as a suggested method that is a combination of AES symmetric encryption algorithm and NTRU asymmetric encryption algorithm for IOT improvement. This algorithm has high speed to create a key, encryption and decryption, and acceptable security in IOT. Safety of this algorithm is due to multinomial usage in encryption, decryption and digital signature to achieve a correct message. This algorithm uses less memory because of less fiscal complexity. This algorithm makes available the encryption in IOT with deduced attacks and improved security.

g)Securing Communications in the Internet of Things using ID-based Cryptography and Modern Elliptic Curves

Author:-Tobias Markmann

Year:-2017

Abstract:

The Internet of Things (IoT) is a fast evolving technology with many devices already available to end-users. However, the security of the IoT is less evolved. Current approaches either employ proprietary protocols, or use standard Internet protocols including the existing heavyweight and complex public-key infrastructure. In this thesis, we propose a security architecture built on identity-based cryptography that embeds cryptographic information in IPv6 addresses, allowing end-to-end authentication of federated IoT networks. Our prototype using twisted Edwards curves allows deployment constrained IoT devices. Our evaluation shows performance characteristics suitable for the scale and diversity of the IoT.

Conclusion:

They had presented a new system architecture for federated end-to-end authentication in the Internet of Things (IoT). It is built from identity-based signature (IBS), which allows to eliminate public key management and key distribution overhead known from classic public-key cryptography (PKC), leading to a smaller size of authenticated messages. With the proposed system, independently administered IoT networks in the IPv6 Internet can exchange identity-based cryptography (IBC) authenticated messages.

These messages can be verified by a federated authentication process which is secured by embedding cryptographic information in the IPv6 network prefix, an idea similar to Cryptographically Generated Addresses (CGAs) and Crypto-based Identifiers (CBIDs).

The proposed architecture includes means to deal with compromised devices in the network and provides a mitigation strategy using revocation combined with a secure rollover of the trusted authority (TA). Our system implementation uses an IBS based on modern elliptic curve cryptography (ECC), specifically twisted Edwards curves. This allows the usually computationally complex IBS to be used on constrained IoT devices which have limited physical resources, e.g. memory, processing speed and energy.

We further discussed the interoperability of our proposed architecture and protocol with existing standards of the Internet and the IoT, specifically standard IPv6 addressing & 6LoWPAN. The proposed system architecture and procedures are proven to be secure under a specified attacker model. Our implementation and practical evaluation on a constrained IoT device, i.e. the SAM R21 controller, demonstrates the implementability of our system architecture and its suitability for the constraints of the IoT. Utilizing additional resources on border gateways to support the authentication process, enables good scalability regarding the number of participating IoT networks and devices for the federated end-to-end authentication architecture.

h) Mutual Authentication Scheme in Secure Internet of Things Technology for Comfortable Lifestyle.

Author:- Namje Park¹ and Namhi Kang²

Year:- 2014

Abstract:

The Internet of Things (IoT), which can be regarded as an enhanced version of machine-to-machine communication technology, was proposed to realize intelligent thing-to-thing communications by utilizing the Internet connectivity. In the IoT, “things” are generally heterogeneous and resource constrained. In addition, such things are connected to each other over low-power and lossy networks. In this paper, we propose an inter-device authentication and session-key distribution system for devices with only

encryption modules. In the proposed system, unlike existing sensor-network environments where the key distribution center distributes the key, each sensor node is involved with the generation of session keys. In addition, in proposed scheme, the performance is improved so that the authenticated device can calculate the session key in advance. The proposed mutual authentication and session-key distribution system can withstand replay attacks, man-in-the-middle attacks, and wiretapped secret-key attacks.

Conclusion:

The IoT, which can be regarded as an enhanced version of M2M communication technology, was proposed to enable intelligent thing-to-thing communications by utilizing Internet connectivity. In the IoT, “things” are generally heterogeneous & resource constrained. In addition, such things are connected with each other over LLNs. In this paper, we focused only on inter-device authentication and the session-key distribution system for devices with only encryption modules. In the proposed scheme, unlike existing sensor-network environments where the key distribution center distributes the key, each sensor node is involved with the generation of session keys. In the scheme, the performance is improved so that the authenticated device can calculate the session key in advance. The proposed mutual authentication and session key distribution system can withstand replay attacks, man-in-the-middle attacks and wiretapped secret-key attacks. However, the proposed system assumes that the devices involved securely share k_{IR} . If k_{IR} is compromised, there is a limit to the security that can be guaranteed by the proposed system. Like many existing proposed systems using preset secret keys, in this paper, the proposed system assumes secure channels and storage methods. However, further research is required to realize the secure sharing of k_{IR} .

3) Objectives :

IoT technology is experiencing a significant growth in consumer and business environments. The importance of device identification in IoT is growing as more and more devices in IoT are connecting to the network intermittently and are required to communicate securely with other devices as well as the backend infrastructure.

Our Main objective of this project is to propose a privacy-preserving and accountable authentication protocol for IoT end-devices. Also analysis of security properties of our protocol in the context of typical attacks and verification of formal security. Securing authenticity of IOT devices through identity based cryptography(IBC) in Cloud of things(CoT) environment.

We all agree on the fact that authentication plays an important role in securing access for IoT devices. But, the authentication methods which are used in today's IT may not work for all IoT devices classes. Securing class 1, class 2, class 3 IOT devices.

Implementation a method that enables IoT devices to communicate among themselves inside the network. Integration of adapted construction of short group signatures and shamir's algorithm. Analysis of security properties of our protocol in the context of typical attacks and verification of formal security.

4) Methodology :

Elliptic Curve :

Elliptic-curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. ECC requires smaller keys compared to non-ECC cryptography (based on plain Galois fields) to provide equivalent security.

Elliptic curves are applicable for key agreement, digital signatures, pseudo-random generators and other tasks. Indirectly, they can be used for encryption by combining the key agreement with a symmetric encryption scheme. They are also used in several integer factorization algorithms based on elliptic curves that have applications in cryptography, such as Lenstra elliptic-curve factorization.

The use of elliptic curves in cryptography was suggested independently by **Neal Koblitz** and **Victor S. Miller** in 1985. Elliptic curve cryptography algorithms entered wide use in 2004 to 2005.

In public key cryptography each user or the device taking part in the communication generally have a pair of keys, a public key and a private key, and a set of operations associated with the keys to do the cryptographic operations. Only the particular user knows the private key whereas the public key is distributed to all users taking part in the communication.

The public key is a point on the curve and the private key is a random number. The public key is obtained by multiplying the private key with a generator point G in the curve.

The mathematical operations of ECC is defined over the elliptic curve $y^2 = x^3 + ax + b$, where $4a^3 + 27b^2 \neq 0$. Each value of the 'a' and 'b' gives a different elliptic curve.

One main advantage of ECC is its small key size. A 160-bit key in ECC is considered to be as secured as 1024-bit key in RSA.

RSA Algorithm :

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. The other key must be kept private. It is based on the fact that finding the factors of an integer is hard (the factoring problem).

RSA stands for **Ron Rivest, Adi Shamir and Leonard Adleman**, who first publicly described it in 1978. A user of RSA creates and then publishes the product of two large prime numbers, along with an auxiliary value, as their public key. The prime factors must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime factors can feasibly decode the message.

RSA involves a public key and private key. The public key can be known to everyone; it is used to encrypt messages. Messages encrypted using the public key can only be decrypted with the private key. The keys for the RSA algorithm are generated the following way:

1. Choose two different large random prime numbers p and q .
2. Calculate $n = pq$, Here n is the modulus for the public key and the private keys .
3. Calculate the totient: $\Phi(n) = (p-1)(q-1)$.

4. Choose an integer e such that $1 < e < \Phi(n)$ and e is coprime to $\Phi(n)$ ie: e and $\Phi(n)$ share no factors other than 1;
 $\gcd(e, \Phi(n)) = 1$. e is released as the public key exponent.
5. Compute d to satisfy the congruence relation $de \equiv 1 \pmod{\Phi(n)}$
 ie: $de \equiv 1 + k\Phi(n)$ for some integer k . d is kept as the private key exponent.

Encrypting Messages:

1. Alice gives her public key (n & e) to bob and keeps her private key secret. Bob wants to send message M to Alice.
2. First he turns **M** into a number m smaller than n by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c corresponding to:

$$c = m^e \bmod n$$

3. This can be done quickly using the method of exponentiation by squaring. Bob then sends c to Alice.

Decrypting Messages:

1. Alice can recover m from c by using her private key d in the following procedure:

$$m = c^d \bmod n.$$

2. Given m , she can recover the original distinct prime numbers, applying the Chinese remainder theorem to these two congruences yields

$$(m)^{ed} \equiv m \pmod{pq}.$$

3. Thus,

$$c^d = m \pmod{n}.$$

Signing messages:

Suppose Alice uses Bob's public key to send him an encrypted message. In the message, she can claim to be Alice but Bob has no way of verifying that the message was actually from Alice since anyone can use Bob's public key to send him encrypted messages. So, in order to verify the origin of a message, RSA can also be used to sign a message.

Suppose Alice wishes to send a signed message to Bob. She produces a hash value of the message, raises it to the power of $d \bmod n$ (just like when decrypting a message), and attaches it as a "signature" to the message.

When Bob receives the signed message, he raises the signature to the power of $e \bmod n$ (just like encrypting a message), and compares the resulting hash value with the message's actual hash value. If the two agree, he knows that the author of the message was in possession of Alice's secret key, and that the message has not been tampered with since.

Note that secure padding schemes such as RSA-PSS are as essential for the security of message signing as they are for message encryption, and that the same key should never be used for both encryption and signing purposes.

Program for Encryption and Decryption of data:

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.math.BigInteger;
import java.security.Key;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.KeySpec;
import java.security.spec.RSAPrivateKeySpec;
import java.security.spec.RSAPublicKeySpec;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
class Test {
    public static void main(String[] args) throws Exception {
```

```

        generateKeys();
        rsaEncrypt("C://Users//Harihanth Julakanti//Desktop//R&D//link.txt",
        "C://Users//Harihanth Julakanti//Desktop//R&D//encrypted.txt");

rsaDecrypt("C://Users//HarihanthJulakanti//Desktop//R&D//encrypted.txt",
"C://Users//Harihanth Julakanti//Desktop//R&D//decrypted.txt");
    }

    public static void generateKeys() throws Exception {
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(512);
        KeyPair kp = kpg.genKeyPair();
        PublicKey publicKey = kp.getPublic();
        PrivateKey privateKey = kp.getPrivate();
        System.out.println("keys created");
        KeyFactory fact = KeyFactory.getInstance("RSA");
        RSAPublicKeySpec pub = fact.getKeySpec(publicKey,
        RSAPublicKeySpec.class);
        RSAPrivateKeySpec priv =
fact.getKeySpec(privateKey, RSAPrivateKeySpec.class);
        saveToFile("public.key", pub.getModulus(),
        pub.getPublicExponent());
        saveToFile("private.key", priv.getModulus(),
        priv.getPrivateExponent());
        System.out.println("keys saved");
    }

    public static void saveToFile(String fileName, BigInteger mod,

```

```

        BigInteger exp) throws IOException {
    ObjectOutputStream fileOut = new ObjectOutputStream(
        new BufferedOutputStream(new
FileOutputStream(fileName)));
    try {
        fileOut.writeObject(mod);
        fileOut.writeObject(exp);
    } catch (Exception e) {
        throw new IOException("Unexpected error");
    } finally {
        fileOut.close();
        System.out.println("Closed writing file.");
    }
}

// Return the saved key
static Key readKeyFromFile(String keyFileName) throws IOException
{
    InputStream in = new FileInputStream(keyFileName);
    ObjectInputStream oin = new ObjectInputStream(new
BufferedInputStream(
        in));
    try {
        BigInteger m = (BigInteger) oin.readObject();
        BigInteger e = (BigInteger) oin.readObject();
        KeyFactory fact = KeyFactory.getInstance("RSA");
        if (keyFileName.startsWith("public"))

```

```

        return fact.generatePublic(new RSAPublicKeySpec(m, e));
    else
        return fact.generatePrivate(new RSAPrivateKeySpec(m, e));
    } catch (Exception e) {
        throw new RuntimeException("Spurious serialisation error", e);
    } finally {
        oin.close();
        System.out.println("Closed reading file.");
    }
}

// Use this PublicKey object to initialize a Cipher and encrypt some
data

public static void rsaEncrypt(String file_loc, String file_des)
    throws Exception {
    byte[] data = new byte[32];
    int i;
    System.out.println("start encryption");
    Key pubKey = readKeyFromFile("public.key");
    Cipher cipher = Cipher.getInstance("RSA");
    cipher.init(Cipher.ENCRYPT_MODE, pubKey);
    FileInputStream fileIn = new FileInputStream(file_loc);
    FileOutputStream fileOut = new FileOutputStream(file_des);
    CipherOutputStream cipherOut = new CipherOutputStream(fileOut,
        cipher);
    // Read in the data from the file and encrypt it

```

```

        while ((i = fileIn.read(data)) != -1) {
            cipherOut.write(data, 0, i);
        }
        // Close the encrypted file
        cipherOut.close();
        fileIn.close();
        System.out.println("encrypted file created");
    }
    // Use this PublicKey object to initialize a Cipher and decrypt some
data
    public static void rsaDecrypt(String file_loc, String file_des)
        throws Exception {
        byte[] data = new byte[32];
        int i;
        System.out.println("start decryption");
        Key priKey = readKeyFromFile("private.key");
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, priKey);
        FileInputStream fileIn = new FileInputStream(file_loc);
        CipherInputStream cipherIn = new CipherInputStream(fileIn,
cipher);
        FileOutputStream fileOut = new FileOutputStream(file_des);
        // Write data to new file
        while ((i = cipherIn.read()) != -1) {
            fileOut.write(i);

```

```

    }
    // Close the file
    fileIn.close();
    cipherIn.close();
    fileOut.close();
    System.out.println("decrypted file created");
}
}

```

Shamir's Secret Sharing :

Shamir's Secret Sharing is an algorithm in cryptography created by Adi Shamir. It is a form of secret sharing, where a secret is divided into parts, giving each participant its own unique part, where some of the parts or all of them are needed in order to reconstruct the secret.

The essential idea of Adi Shamir's threshold scheme is that 2 points are sufficient to define a line, 3 points are sufficient to define a parabola, 4 points to define a cubic curve and so forth. That is, it takes k points to define a polynomial of degree $k-1$.

Suppose we want to use a (k,n) threshold scheme to share our secret S , without loss of generality assumed to be an element in a finite field F of size P where $0 < k \leq P$; $S < P$ and P is a prime number.

Choose at random $k-1$ positive integers a_1, \dots, a_{k-1} with $a_i < P$, and let $a_0 = S$. Build the polynomial

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots$$

$$+a_{k-1}x^{k-1}, \\$$

$f(x)=a_0+a_1x+a_2x^2+a_3x^3+\cdots+a_{k-1}x^{k-1}$. Let us construct any n points out of it, for instance set $i=1, \dots, n$ to retrieve $f(i)$. Every participant is given a point (a non-zero integer input to the polynomial, and the corresponding integer output) along with the prime which defines the finite field to use. Given any subset of k of these pairs, we can find the coefficients of the polynomial using interpolation. The secret is the constant term a_0 . a_0 is required to reconstruct the secret. The essential idea of Adi Shamir's threshold scheme is that 2 points are sufficient to define a line, 3 points are sufficient to define a parabola, 4 points to define a cubic curve and so forth. That is, it takes k points to define a polynomial of degree $k-1$.

Suppose we want to use a (k, n) threshold scheme to share our secret S , without loss of generality assumed to be an element in a finite field F of size P where $0 < k \leq n < P$; $S < P$ and P is a prime number.

Choose at random $k-1$ positive integers a_1, \dots, a_{k-1} with $a_i < P$, and let $a_0 = S$.

Build the polynomial

$$f(x)=a_0+a_1x+a_2x^2+a_3x^3+\cdots+a_{k-1}x^{k-1}, \\$$

$f(x)=a_0+a_1x+a_2x^2+a_3x^3+\cdots+a_{k-1}x^{k-1}$. Let us construct any n points out of it, for instance set $i=1, \dots, n$ to retrieve $f(i)$. Every participant is given a point (a non-zero integer input to the polynomial, and the

corresponding integer output) along with the prime which defines the finite field to use. Given any subset of $\{k\}$ of these pairs, we can find the coefficients of the polynomial using interpolation. The secret is the constant term a_0 .

Algorithm Code :

```
import java.math.BigInteger;
import java.util.Random;

public final class Shamir {

    public final class SecretShare {
        public SecretShare(final int num, final BigInteger share) {
            this.num = num;
            this.share = share;
        }

        public int getNum() {
            return num;
        }

        public BigInteger getShare() {
            return share;
        }

        @Override
        public String toString() {
            return "SecretShare [num=" + num + ", share=" + share + "]";
        }
    }

    private final int num;
```

```

        private final BigInteger share;
    }

    public Shamir(final int k, final int n) {
        this.k = k;
        this.n = n;

        random = new Random();
    }

    public SecretShare[] split(final BigInteger secret) {
        final int modLength = secret.bitLength() + 1;

        prime = new BigInteger(modLength, CERTAINTY, random);
        final BigInteger[] coeff = new BigInteger[k - 1];

        System.out.println("Prime Number: " + prime);

        for (int i = 0; i < k - 1; i++) {
            coeff[i] = randomZp(prime);
            System.out.println("a" + (i + 1) + ": " + coeff[i]);
        }

        final SecretShare[] shares = new SecretShare[n];
        for (int i = 1; i <= n; i++) {
            BigInteger accum = secret;

            for (int j = 1; j < k; j++) {
                final BigInteger t1 =
                    BigInteger.valueOf(i).modPow(BigInteger.valueOf(j), prime);
                final BigInteger t2 = coeff[j - 1].multiply(t1).mod(prime);

                accum = accum.add(t2).mod(prime);
            }
        }
    }

```

```

    }
    shares[i - 1] = new SecretShare(i - 1, accum);
    System.out.println("Share " + shares[i - 1]);
}

return shares;
}

public BigInteger getPrime() {
    return prime;
}

public BigInteger combine(final SecretShare[] shares, final BigInteger
primeNum) {
    BigInteger accum = BigInteger.ZERO;
    for (int i = 0; i < k; i++) {
        BigInteger num = BigInteger.ONE;
        BigInteger den = BigInteger.ONE;

        for (int j = 0; j < k; j++) {
            if (i != j) {
                num = num.multiply(BigInteger.valueOf(-j -
1)).mod(primeNum);
                den = den.multiply(BigInteger.valueOf(i - j)).mod(primeNum);
            }
        }

        System.out.println("den: " + den + ", num: " + num + ", inv: " +
den.modInverse(primeNum));
        final BigInteger value = shares[i].getShare();

        final BigInteger tmp =
value.multiply(num).multiply(den.modInverse(primeNum)).mod(primeNum);

```

```

        accum = accum.add(primeNum).add(tmp).mod(primeNum);

        System.out.println("value: " + value + ", tmp: " + tmp + ", accum: " +
accum);
    }

    System.out.println("The secret is: " + accum);

    return accum;
}

private BigInteger randomZp(final BigInteger p) {
    while (true) {
        final BigInteger r = new BigInteger(p.bitLength(), random);
        if (r.compareTo(BigInteger.ZERO) > 0 && r.compareTo(p) < 0) {
            return r;
        }
    }
}

private BigInteger prime;

private final int k;
private final int n;
private final Random random;

private static final int CERTAINTY = 50;

public static void main(final String[] args) {
    final Shamir shamir = new Shamir(11, 20);

    final BigInteger secret = new
BigInteger("1234567890123456789012345678901234567890");

```

```

        final SecretShare[] shares = shamir.split(secret);
        final BigInteger prime = shamir.getPrime();

        final Shamir shamir2 = new Shamir(11, 20);
        final BigInteger result = shamir2.combine(shares, prime);

    }
}; import java.math.BigInteger;
import java.util.Random;

public final class Shamir {

    public final class SecretShare {
        public SecretShare(final int num, final BigInteger share) {
            this.num = num;
            this.share = share;
        }

        public int getNum() {
            return num;
        }

        public BigInteger getShare() {
            return share;
        }

        @Override
        public String toString() {
            return "SecretShare [num=" + num + ", share=" + share + "]";
        }

        private final int num;
        private final BigInteger share;
    }
}

```

```

    }

    public Shamir(final int k, final int n) {
        this.k = k;
        this.n = n;

        random = new Random();
    }

    public SecretShare[] split(final BigInteger secret) {
        final int modLength = secret.bitLength() + 1;

        prime = new BigInteger(modLength, CERTAINTY, random);
        final BigInteger[] coeff = new BigInteger[k - 1];

        System.out.println("Prime Number: " + prime);

        for (int i = 0; i < k - 1; i++) {
            coeff[i] = randomZp(prime);
            System.out.println("a" + (i + 1) + ": " + coeff[i]);
        }

        final SecretShare[] shares = new SecretShare[n];
        for (int i = 1; i <= n; i++) {
            BigInteger accum = secret;

            for (int j = 1; j < k; j++) {
                final BigInteger t1 =
                    BigInteger.valueOf(i).modPow(BigInteger.valueOf(j), prime);
                final BigInteger t2 = coeff[j - 1].multiply(t1).mod(prime);

                accum = accum.add(t2).mod(prime);
            }
        }
    }

```

```

        shares[i - 1] = new SecretShare(i - 1, accum);
        System.out.println("Share " + shares[i - 1]);
    }

    return shares;
}

public BigInteger getPrime() {
    return prime;
}

public BigInteger combine(final SecretShare[] shares, final BigInteger
primeNum) {
    BigInteger accum = BigInteger.ZERO;
    for (int i = 0; i < k; i++) {
        BigInteger num = BigInteger.ONE;
        BigInteger den = BigInteger.ONE;

        for (int j = 0; j < k; j++) {
            if (i != j) {
                num = num.multiply(BigInteger.valueOf(-j -
1)).mod(primeNum);
                den = den.multiply(BigInteger.valueOf(i - j)).mod(primeNum);
            }
        }

        System.out.println("den: " + den + ", num: " + den + ", inv: " +
den.modInverse(primeNum));
        final BigInteger value = shares[i].getShare();

        final BigInteger tmp =
value.multiply(num).multiply(den.modInverse(primeNum)).mod(primeNum);
        accum = accum.add(primeNum).add(tmp).mod(primeNum);
    }
}

```

```
        System.out.println("value: " + value + ", tmp: " + tmp + ", accum: " +  
accum);  
    }
```

```
        System.out.println("The secret is: " + accum);
```

```
    return accum;  
}
```

```
private BigInteger randomZp(final BigInteger p) {  
    while (true) {  
        final BigInteger r = new BigInteger(p.bitLength(), random);  
        if (r.compareTo(BigInteger.ZERO) > 0 && r.compareTo(p) < 0) {  
            return r;  
        }  
    }  
}
```

```
private BigInteger prime;
```

```
private final int k;  
private final int n;  
private final Random random;
```

```
private static final int CERTAINTY = 50;
```

```
public static void main(final String[] args) {  
    final Shamir shamir = new Shamir(11, 20);
```

```
        final BigInteger secret = new  
BigInteger("12345678901234567890123456789012345678901234567890");  
        final SecretShare[] shares = shamir.split(secret);
```



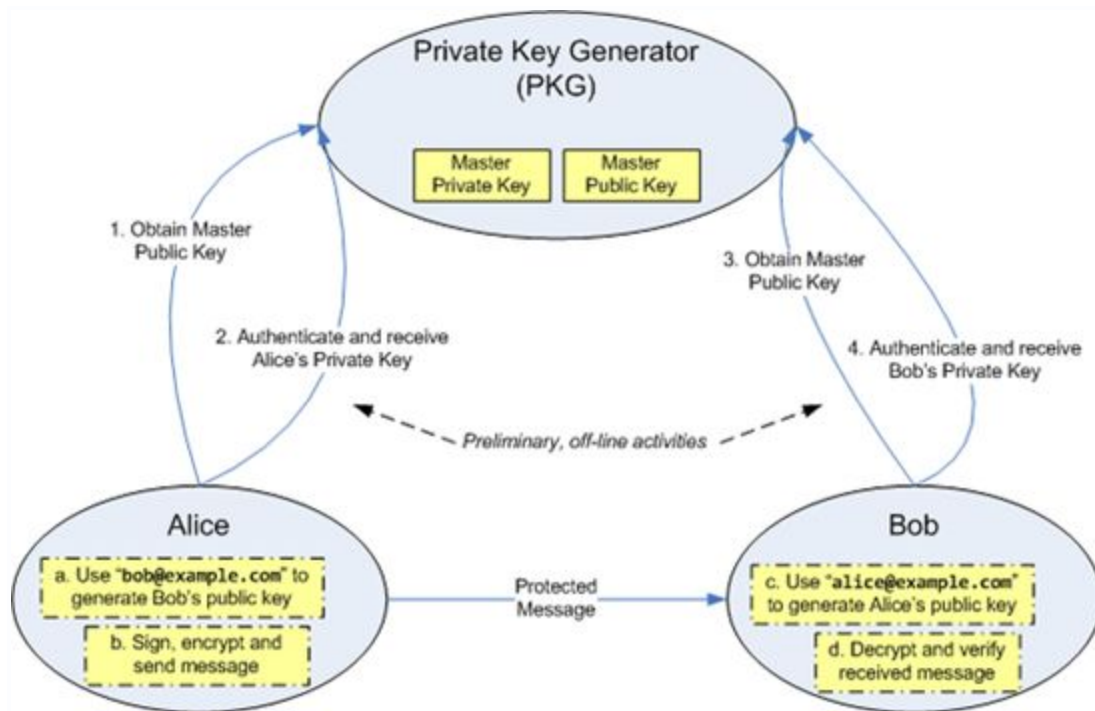
```

final BigInteger prime = shamir.getPrime();

final Shamir shamir2 = new Shamir(11, 20);
final BigInteger result = shamir2.combine(shares, prime);

}
}

```



DHT11 Sensor :

A humidity sensor senses, measures and regularly reports the relative humidity in the air. It measures both moisture and air temperature. Relative humidity, expressed as a percent, is the ratio of actual moisture in the air to the highest amount of moisture air at that temperature can hold. The warmer the air is, the more moisture it can hold, so relative humidity changes with fluctuations in temperature.

Raspberry Pi 3 :

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. The original model became far more popular than anticipated, selling outside its target market for uses such as robotics. It does not include peripherals (such as keyboards, mice and cases). However, some accessories have been included in several official and unofficial bundles.

According to the Raspberry Pi Foundation, over 5 million Raspberry Pis were sold by February 2015, making it the best-selling British computer. By November 2016 they had sold 11 million units, and 12.5m by March 2017, making it the third best-selling "general purpose computer". In July 2017, sales reached nearly 15 million. In March 2018, sales reached 19 million. They are made in a Sony factory in Pencoed, Wales.

Connecting DHT11 sensor with Raspberry Pi 3 :

Hardware and softwares required:

1. Raspberry Pi 3.
2. DHT11 Sensor.
3. Raspbian.

Hardware Connections:

1. VCC of DHT11 -> 5v Pin of Raspberry Pi 3.
2. GND of DHT11 -> GND Pin of Raspberry Pi 3.
3. Signal pin of DHT11 -> GPIO 4 Pin of Raspberry Pi3.

Program for Humidity & Temperature Sensor (DHT11 Sensor):

```
import sys
import RPi.GPIO as GPIO
from time import sleep
import Adafruit_DHT
import urllib2

def getSensorData():
    RH, T = Adafruit_DHT.read_retry(Adafruit_DHT.DHT11, 23)
    # return dict
    return (str(RH), str(T))

# main() function
def main():
    # use sys.argv if needed
    #if len(sys.argv) < 2:
    #    print('Usage: python tstest.py PRIVATE_KEY')
    #    exit(0)
    print 'starting...'

    sys.argv[1]

    while True:
        try:
            RH, T = getSensorData()
            #f = urllib2.urlopen(baseURL +
                                #+"&field1=%s&field2=%s" % (RH, T))
            print f.read()
            f.close()
            sleep(5)
        except:
            print 'exiting.'
```

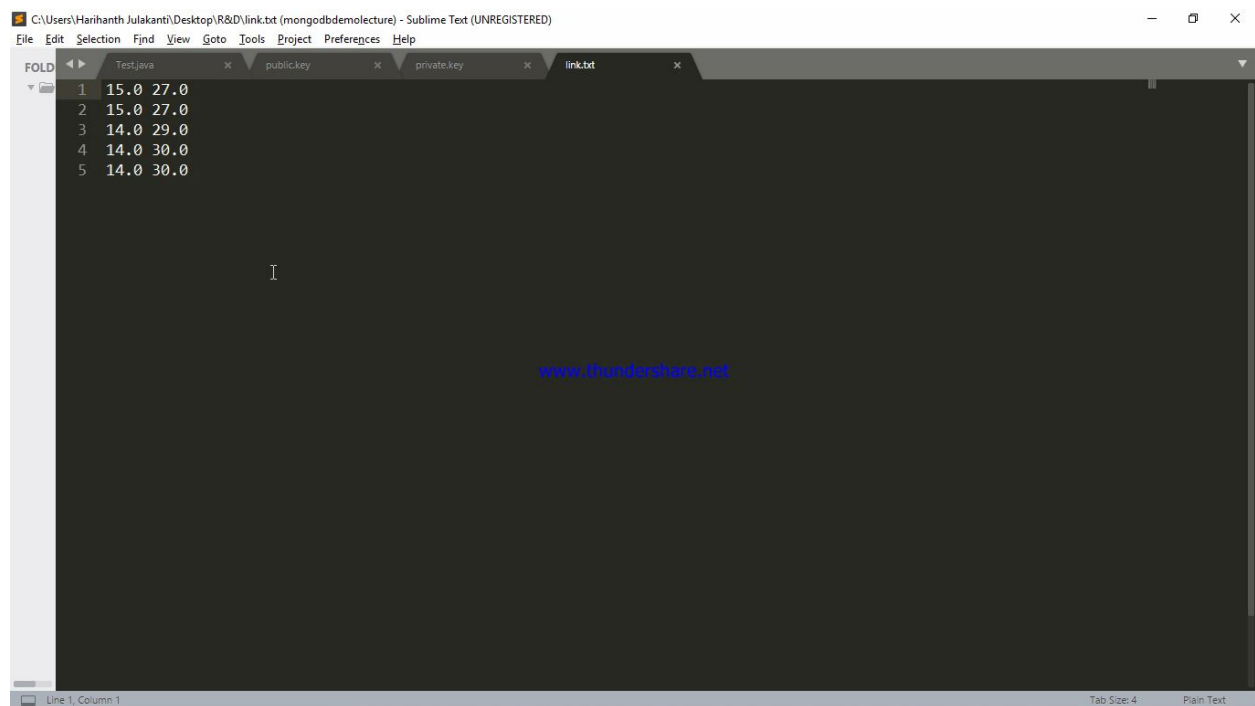
break

call main

if __name__ == '__main__':

5) Results of the work completed :

Link.txt



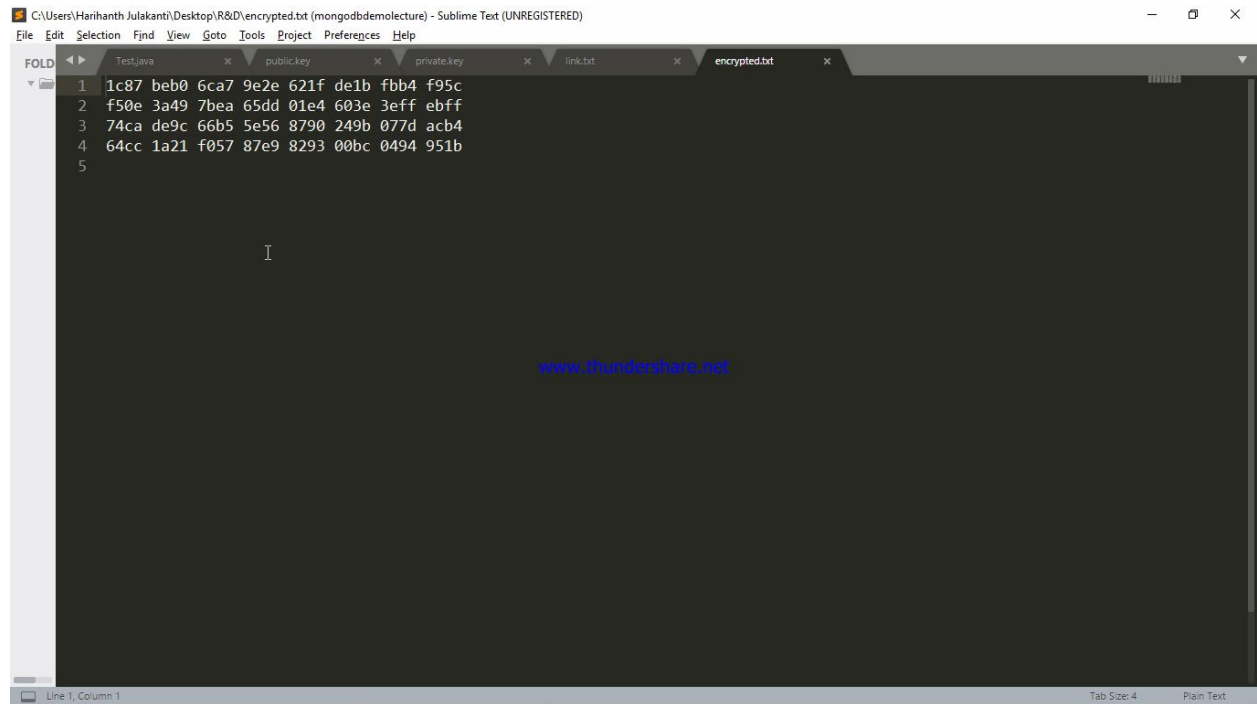
Publickey.key

```
C:\Users\Hanihanth Julakanth\Desktop\R&D\publickey (mongodbdemolecture) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLD
1 |aced 0005 7372 0014 6a61 7661 2e6d 6174
2 |682e 4269 6749 6e74 6567 6572 8cfc 9f1f
3 |a93b fb1d 0300 0649 0008 6269 7443 6f75
4 |6e74 4900 0962 6974 4c65 6e67 7468 4900
5 |1366 6972 7374 4e6f 6e7a 6572 6f42 7974
6 |654e 756d 4900 0c6c 6f77 6573 7453 6574
7 |4269 7449 0006 7369 676e 756d 5b00 096d
8 |6167 6e69 7475 6465 7400 025b 4278 7200
9 |106a 6176 612e 6c61 6e67 2e4e 756d 6265
10|7286 ac95 1d0b 94e0 8b02 0000 7870 ffff
11|ffff ffff ffff ffff fffe ffff fffe 0000
12|0001 7572 0002 5b42 acf3 17f8 0608 54e0
13|0200 0078 7000 0000 4089 ed4f ff11 3006
14|adf1 78fa 8931 aeef 44ee 0891 4a25 f984
15|2c1e 289a 0b68 59b5 a093 79c2 ae57 7bde
16|fc41 6a3c 2718 cd5f 4b5e 3c32 a7d5 39e1
17|dd4c d313 9020 5239 cd78 7371 007e 0000
18|ffff ffff ffff ffff ffff fffe ffff fffe
19|0000 0001 7571 007e 0004 0000 0003 0100
20|0178
www.thundershare.net
Line 1, Column 1 Tab Size: 4 Plain Text
```

Privatekey.key

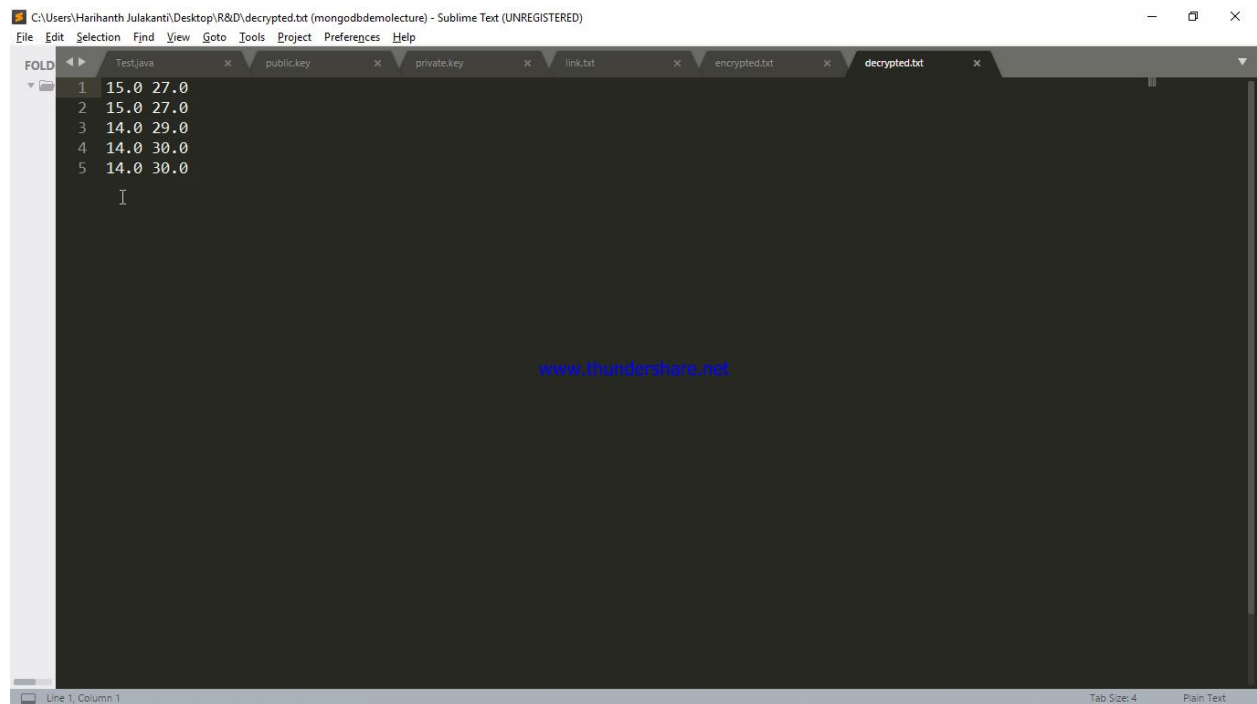
```
C:\Users\Hanihanth Julakanth\Desktop\R&D\privatekey (mongodbdemolecture) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLD
1 |aced 0005 7372 0014 6a61 7661 2e6d 6174
2 |682e 4269 6749 6e74 6567 6572 8cfc 9f1f
3 |a93b fb1d 0300 0649 0008 6269 7443 6f75
4 |6e74 4900 0962 6974 4c65 6e67 7468 4900
5 |1366 6972 7374 4e6f 6e7a 6572 6f42 7974
6 |654e 756d 4900 0c6c 6f77 6573 7453 6574
7 |4269 7449 0006 7369 676e 756d 5b00 096d
8 |6167 6e69 7475 6465 7400 025b 4278 7200
9 |106a 6176 612e 6c61 6e67 2e4e 756d 6265
10|7286 ac95 1d0b 94e0 8b02 0000 7870 ffff
11|ffff ffff ffff ffff fffe ffff fffe 0000
12|0001 7572 0002 5b42 acf3 17f8 0608 54e0
13|0200 0078 7000 0000 4089 ed4f ff11 3006
14|adf1 78fa 8931 aeef 44ee 0891 4a25 f984
15|2c1e 289a 0b68 59b5 a093 79c2 ae57 7bde
16|fc41 6a3c 2718 cd5f 4b5e 3c32 a7d5 39e1
17|dd4c d313 9020 5239 cd78 7371 007e 0000
18|ffff ffff ffff ffff ffff fffe ffff fffe
19|0000 0001 7571 007e 0004 0000 0040 3df5
20|5cfa 37be 5b41 c837 9054 10fd 847e 726f
21|66d4 e83a 2e25 a366 99cc 8613 c682 b178
22|fcc5 c783 d4b4 c10b 8429 9dad eb74 2fce
23|e06c 1a2d 9598 957d a545 c489 5401 78
www.thundershare.net
Line 1, Column 1 Tab Size: 4 Plain Text
```

Encrypted.txt



```
C:\Users\Hanihanth Julakanth\Desktop\R&D\encrypted.txt (mongodbdemolecture) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLD
1 1c87 beb0 6ca7 9e2e 621f de1b fbb4 f95c
2 f50e 3a49 7bea 65dd 01e4 603e 3eff ebff
3 74ca de9c 66b5 5e56 8790 249b 077d acb4
4 64cc 1a21 f057 87e9 8293 00bc 0494 951b
5
www.thundershare.net
Line 1, Column 1 Tab Size: 4 Plain Text
```

Decrypted.txt



```
C:\Users\Hanihanth Julakanth\Desktop\R&D\decrypted.txt (mongodbdemolecture) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLD
1 15.0 27.0
2 15.0 27.0
3 14.0 29.0
4 14.0 30.0
5 14.0 30.0
www.thundershare.net
Line 1, Column 1 Tab Size: 4 Plain Text
```

6) Summary:

As our Project is to authenticate IoT devices through IBC in large-scale Internet of Things (IoT) systems, huge volumes of data are collected from anywhere at any time, which may invade people's privacy, therefore we have developed a method of using IBE in SSL to eliminate the need for certificates. We have provided proof-of-concept code to set up the PKG, for the server to connect to the PKG and request its private key, and for the client to encrypt with the server's public key and connect to the server. While SSL is traditionally performed implicitly in modern browsers, we have chosen to write demonstration programs to show a proof of concept. If SSL used the IBE system instead, there would only be the need for one certificate: the master certificate embedded in web browsers for the PKG. The browser would then encrypt data with the URL of the website (e.g. "amazon.com") and send the encrypted text to the server. The server would get the encrypted message from the client and decrypt it with its private key that it has received from the PKG. There is no need for the server to have its own certificate: it merely needs the private key generated from the PKG to decrypt any message sent to it.

7) Future Work

As we have discussed in above report we had done the implication of encryption and decryption (using RSA) which is for **“The Thing Layer to The Edge Layer”ie IoT devices** which is successfully being implemented now the left part is to connect from **Edge layer to Cloud Layer ie CoT.**

- For that we will be designing the gateway for the above example where the database will be stored in cloud directly.
- It can be easily attacked so we will be using Digital signatures for the authentication part.
- As a whole it will be connected through a front end site where we can see the graph and the cipher text as an administrator.

8) Reference

- M. C. Domingo, “An overview of the internet of things for people with disabilities,” Journal of Network and Computer
- NIST, “Report on lightweight cryptography,”
http://csrc.nist.gov/publications/drafts/nistir-8114/nistir_8114_draft.pdf, August 2016
- H. Tsai, C. Chang, and K. Chan, “Roaming across wireless local area networks using SIM-based authentication protocol,” Comput. Standards Interfaces, vol. 31, no. 2, pp. 381-389, Feb.2009.
- researchgate.net Conference Paper.December 2013 DOI: 10.1109/INDCON.2013.6725878
- L. Atzori, A.L.; Morabito, G. The internet of things: A survey. Computer networks 2010,54, 2787-2805

- researchgate.net Conference Paper.December 2013 DOI:
10.1109/INDCON.2013.6725878