IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Cloud-based IoT: Secure and Fine-grained Self-controlled Outsourced Data Deletion

[1]Rajat Gulabrao Waghmare, [2]Prof. Sushil Venkatesh Kulkarni

[1]Student, [2]Project Guide
[1]Computer Science & Engineering Technology,
[1]M.B.E. Society's College of Engineering, Ambajogai, India

Abstract: In many IoT applications, the cost of data management and infrastructure investment has been significantly reduced thanks to the emerging cloud-based IoT paradigm, which enables IoT devices to directly upload their collected data to a distant cloud and gives data owners convenient access to those data through cloud APIs. Given that data owners don't have direct access to the outsourced data, and How to safely erase the unnecessary sensitive data saved in the cloud because the cloud server cannot always be completely trusted to avoid potential. The problem of data leakage is quite difficult. Most current solutions rely on user participation and only offer coarse-grained deletion of the cloud server, which severely limits its practicability and versatility. In this research, we offer a safe and precise self-controlled outsourced data deletion technique for cloud-based IoT, based on an upgraded policy-based puncturable encryption (P-PUN-ENC) primitive. Our system's key value is that it gives data owners a policy-based approach to precisely and permanently remove their outsourced IoT-driven data without depending on a cloud server. In order to accomplish this, we deftly take use of the logical connection between puncture policy and access policy and create a policy transform method to change the puncture process based on puncture policies into an update process of access policies. Then, we carry out the relevant key update activities using the attribute-based encryption (ABE) key delegation approach. In addition, to solve the issue of P-PUN-rising ENC's key storage and decryption costs, we suggest the outsourced key and decryption outsourcing technique with P-PUN-ENC to create the outsourced policy-based puncturable encryption (OP-PUN-ENC) primitive. A thorough analysis of the alternatives reveals that our suggested solution can more effectively satisfy the cloud-based IoT's data deletion needs. Additionally, a formal security proof and detailed simulation results indicate how reliable and effective the suggested solution is.

***Index Terms* –** AES, ECC, TF/IDF, puncturable encryption, data deletion, cloud-based IoT.

I. Introduction

The Internet of Things (IoT) has recently drawn the attention of both academia and business. IoT applications have significantly changed our way of life and advanced the creation of smart cities. Examples include environment monitoring, smart e-healthcare, and transportation. Although the enormous amounts of data produced by these devices would be very advantageous for data owners, managing them becomes a hassle, because maintaining storage systems requires a significant financial commitment and managerial overhead for data owners. The cloud computing service enables IoT devices to instantly upload their collected data to the remote cloud and allows data owners to conveniently handle such data using cloud APIs since it offers sufficient compute capability and storage space for consumers.

However, given the importance and privacy of the data gathered by IoT devices, data owners have become significantly more concerned about security as a result of the frequent instances of data leaking in the cloud. Therefore, a critical issue is how to ensure the privacy of the data that has been outsourced. This is due to the fact that the cloud cannot be completely trusted and may purposefully (for business purposes, such as data analysis) or accidentally fail to respond to the data owner's request for data erasure. To achieve effective and secure outsourcing data deletion, we used attribute-based encryption (ABE). In order to prevent the prior secret key from being utilised for decryption once the ciphertext components connected to the dummy attribute is altered, both of them included an additional dummy attribute when encrypting the data. Their systems only let the data owner delete one file at a time, and the deletion is only complete until a third party (a fog node or cloud server) updates the ciphertext. The third parties, or a quorum of key managers, are also necessary for FADE, a policy-based assured data deletion system that we have suggested, to destroy the relevant control key. To make it possible for data owners to unilaterally remove their data that has been outsourced without depending on the cloud service provider.

The data gathered by IoT devices is encrypted under a set of tags, and the owner of the data can puncture the tags using his or her secret key to prevent any ciphertext that contains the punctured tags from being decoded with the secret key. Therefore, reliable self-controlled data erasure can be accomplished right away through tag puncturing without the assistance of the cloud or communication with IoT devices. However, since only one tag is permitted in each puncture of their proposed tagbased puncturable encryption (T-PUN-ENC), only coarse-grained data erasure is possible. In this paper, we propose a secure and fine-grained self-controlled outsourced data deletion scheme in cloud-based IoT environment, based on an enhanced policy-based puncturable encryption (P-PUN-ENC) primitive, which enables the data owner to precisely and permanently delete the outsourced IoT-driven data in a policy-based way without relying on a third party. In practise, IoT devices use the linked tags to encrypt the data they collect before uploading the ciphertext to the cloud. The data owner's original secret key can decrypt all of the ciphertext. The data owner provides a fine-grained puncture (deletion) policy1 rather than a single tag when some ciphertext needs to be removed. As a result, even if the punctured key is hacked or accidentally released, the data satisfying the puncture policy can be erased at a fine-grained level and cannot be accessible by the cloud service provider or other enemies. Data that hasn't been erased can still be decrypted without any issues.

The following is a summary of our contributions:

(1) In order to accommodate expressive puncture policies such as AND, OR, and threshold gates, we introduce policy-based puncturable encryption (PPUN-ENC), an extension of tag-based puncturable encryption (T-PUN-ENC). In order to transform the puncture process based on a puncture policy into an update process of an access policy, we gently exploit the logical link between puncture policy and access policy. Then, using solely public keys and the key delegation approach, we change the decryption key in accordance with the access policy. We officially demonstrate the security of P-PUN-ENC based on the Decisional Bilinear Diffie-Hellman (DBDH) assumption to ensure the dependability of key puncturing. The proposed P-PUN-ENC is unquestionably more adaptable and practical than TPUN-ENC, and it has a larger application focus.

(2) We implement P-PUN-ENC in the cloud-based IoT environment and provide a secure and precise self-controlled data deletion method that enables the data owner to unilaterally achieve immediate and reliable erasure of the data stored in the cloud. The data owner can independently regulate his or her decryption capability via key puncturing in the suggested approach, which encrypts the data gathered by IoT devices with a set of related tags and uploads it directly to the cloud without the need for intermediate stages. As a result, accurate data deletion is accomplished without supplying IoT devices with new key materials or the involvement of the cloud server.

(3) We propose an enhanced primitive dubbed outsourced policy-based puncturable encryption (OPPUN-ENC) by fusing the key and decryption outsource technique with P-PUN-ENC to address the problem of rising key storage and decryption cost present in both T-PUN-ENC and PPUN-ENC. To save local storage space, most of the punctured secret key components in OP-PUN-ENC are blinded with a random value and outsourced to the cloud. Additionally, the outsourced key can be used by the cloud to partially decode the ciphertext without jeopardising data secrecy, which greatly lowers the cost of decryption for the data owner.

(4) We thoroughly contrast our suggested plan with a few comparable studies to show that it can better satisfy the outsourced data eradication needs in a cloud-based IoT context. In order to prove the proposed scheme's efficiency, we additionally quantitatively assess the storage and computing overheads and run extensive simulations.

II. RELATED WORK

We suggested Vanish, which would allow the data to be automatically deleted after a certain amount of time. In Vanish, the data is encrypted using a symmetric encryption method, and a sizable peer-to-peer network receives shares of the decryption key. The associated ciphertext is inaccessible after the key's shares vanish from the network. We presented a secure self-destructing strategy (SSDD) for electronic data based on the Vanish system. To make their approach resistant to attacks on the DHT network as well as traditional cryptanalysis and the brute-force attack, they spread the decryption key and the retrieved ciphertext into a distributed hash table (DHT) network.

The key-policy attribute-based encryption with time-specified attributes (KP-TSABE) method that we suggested ties each data item to a particular set of characteristics and each attribute to a certain time period is a novel approach. The sensitive data securely self-destructs at the chosen expiration period since it can no longer be decrypted. We put forth FADE, a policy-based assured data deletion strategy that relies on a group of key managers who collectively self-maintain a set of cryptographic key operations. We used ABE to accomplish both assured deletion and fine-grained data access restriction at once. Both of them added an extra dummy attribute to the ciphertext during encryption, preventing the prior secret key from being used for decryption after the ciphertext components relating to the dummy attribute had been modified. Unfortunately, only file-based deletion is enabled, and these schemes are less flexible and reliable because a third party (a fog node or cloud server) is needed.

We also discussed various researches on attribute-based encryption and forward security because puncturable encryption (T-PUN-ENC), which uses non-monotonic attribute-based encryption (NM-ABE) as its foundation, was initially proposed to address the problem of forward security.

2.1 Attribute-based encryption (ABE):

Attribute-based encryption (ABE) was first developed as a one-to-many encryption primitive. It later developed into two variants: key-policy ABE (KPABE) and ciphertext-policy ABE (CP-ABE). In KP-ABE, a collection of attributes are used to encrypt the data, and the secret key is generated using the access policy. The access policy is incorporated into the CP-ABE when in the secret key is produced using the ciphertext as the attribute group. To support the access's negative qualities first proposed non-monotonic policy. Any access policy represented by a Boolean is supported by ABE formula containing the operations AND, OR, NOT, and threshold. In ABE, delegation is envisioned. KP-ABE framework, and are then specifically described. We created tag-based puncturable encryption (T-PUN-ENC) by combining NM-ABE with the key delegation approach, but their implementation only supports a single tag as a simple kind of puncture policy.

Puncturable encryption and revocation in ABE both address the problem of revocation of the ability to decode data, hence they have some similarities. The key distinction is that while puncturable encryption focuses on revocation of one's own decryption ability, ABE revocation allows the data owner to revoke the decryption ability of specific users or qualities.

2.2 Forward security:

Past sessions are shielded from potential key compromise in the future through forward security. Many of the current forward security systems rely on highly accessible network infrastructure to send out new key materials to senders or alter client interactions. We suggested a key distribution approach that doesn't need to be altered, called FS-PKE, which stands for forward safe public key encryption. Their system hasn't, however, gained much traction because, if decryption is disabled for a certain length of time, the user loses access to any messages sent earlier in that period. Users can promptly and precisely delete selected messages using this approach, which combines TPUN-ENC with a variation of FS-PKE and views tags in T-PUN-ENC as time periods. This method also achieves more useful forward-secure messaging with little overhead. With the help of a brand-new Bloom Filter Encryption (BFE) primitive, we were able to create brand-new puncture-efficient encryption techniques. In their plan, a puncturing procedure only involves a small number of extremely efficient computations and the deletion of a small portion of the secret key. The forward secure systems, however, only take into account the aspect of time slot and can only come up with limited time-based data deletion alternatives.

III. DEFINITIONS:

In this section, we formalise the cloud-based IoT system and threat models, describe the P-PUN-ENC security model and algorithm definitions, and specify the design objectives of our suggested scheme.

3.1 System and Threat Models:

The self-controlled data deletion concept in a cloud-based IoT context includes the following three parties:

- **IoT Devices (Dev):** Dev refers to Internet of Things (IoT) devices that are deployed or controlled by the data owner to gather various types of data, such as mobile phones, sensors, or wearables. Each Dev is identified by a few tags that specify its features, such as the device identity, location, and content kind. The gathered information will be directly transferred to the cloud server while being encrypted with the appropriate tags.
- **Owner of Data (DO):** DO might be a business installing sensors, a platform disseminating sensing duties, or an individual using wearable medical equipment and being the owner of IoT-driven data saved in the cloud. Through cloud APIs, DO has access to and control over the outsourced data. By actively puncturing his or her secret key, DO can destroy the unnecessary data in accordance with the puncture policies listed on the data tags. The production and distribution of the system's public keys are likewise handled by DO.
- **Cloud Server (CS):** CS offers online storage and retrieval of data for DO. All of the information gathered by Dev is encrypted, saved in CS with the related tags, and accessible to DO at any time and from any location.

Given that DO is the main beneficiary; it can be completely trusted in our system with relation to the threat model in the aforementioned system. In other words, it will faithfully follow the protocols, i.e., store and distribute data. However, it may be interested in mining any sensitive information from those saved data for their own marketing purpose. The adversaries (the cloud or outside attackers) are expected to be able to compromise the punctured key and attempt to obtain the erased data because the major goal of our technique is to remove the outsourced data. Note that we do not take into account the attacks directed towards Dev or the secrecy of the ciphertext that may be decoded with the punctured key.

3.2 Security Model for P-PUN-ENC:

For the purpose of demonstrating the security of P-PUN-ENC against a specific plaintext attack, we define the following selective-set model (s-IND-PUNCPA).

- **Init**: A series of tags T = "t1, ... , td" are committed to the challenger C by the adversary A.
- **Setup**: With the input of a security parameter and the number of tags connected to a ciphertext d, the challenger executes the KeyGen method to produce PK and SK0 and then delivers the public key PK to the adversary. The challenger additionally sets a counter at k = 0 and an empty set P to beginning state.
- **Question Phase 1:** The challenger can consistently respond to the following two types of adversarial questions.

  - **Puncture** (PP). The challenger receives a puncture policy PP from the adversary. The challenger raises k, calculates Puncture (PK, SKk1, PP) SKk, and then multiplies PP by P.

  - **Depraved** (). When this type of query is asked for the first time, the challenger responds with the most recent secret key SKk and returns for all subsequent questions. In addition, the challenger returns if the challenge set T does not fulfil any puncture policy PP in P.
- **Obstacle**. The antagonist sends the challenger two identical-sized messages, M0 and M1. With a randomly chosen bit v in the range of 0 to 1, the challenger executes the Encrypt (PK, M_v, T) procedure and provides the adversary with the resulting ciphertext CT.
- **Question Phase 2:** Repeat query phase 1 once again.
- **Guess:** The adversary generates a guess of v called v ′. It triumphs in the game if v ′ = v.

Security-related intuition the s-IND-PUN-CPA model includes new oracles for Puncture and Corrupt in addition to the in distinguishability definition for selective-set public key encryption. To ensure that the punctured secret key cannot be used to decrypt the ciphertext fulfilling PP, the Puncture oracle updates the current secret key depending on PP, and the Corrupt oracle provides the opponent with the most recent state of the secret key. Unless the challenge set T fulfils at least one of the puncture policies in P, the adversary cannot corrupt the punctured secret key. The opponent cannot easily decipher the challenge ciphertext if it has not been punctured thanks to this constraint in the corrupt queries.

3.3 Design Goals:

In order to enable the data owner to securely and autonomously erase the sensitive IoT-driven data stored in the cloud in a policy-based manner, we plan to build a secure and fine-grained data deletion scheme for cloud-based IoT. Therefore, even if the data owner's secret key is disclosed, the erased data cannot be recovered. More specifically, our plan should accomplish the following objectives.

- **Information security:** Without DO's private key, it is impossible to access the data kept in CS. Even with the secret key revealed by the perforation, the erased data can no longer be accessed by anyone.
- **Fine-grained data deletion:** In order to provide flexible and exact data deletion, fine-grained policy-based data deletion should be allowed. In particular, an expressive Boolean formula including AND, OR, and threshold operations should be used to indicate the puncture policy.
- **Efficiency:** The system should operate with little compute and storage overhead, especially for Dev and DO.

3.4 System Operations:

The four system actions in our approach are System Initialization by DO, Data Encryption by Dev, Data Deletion by DO, and Data Decryption by DO. These operations correspond to the four algorithms in P-PUN-ENC.

- **Initialization of the system.** In this step, DO first specify the number of tags d that will be used to describe the IoT devices as well as a system security parameter. The KeyGen (, d) procedure is then used to produce the system's public key PK and its first secret key SK0. The IoT devices will be deployed by DO, who will give them a set of tags T = t1,..., td and implant the matching public key components in them.
- **Data Encryption:** Dev will routinely transfer the data M they have gathered to CS in order to conserve local storage resources. Dev uses the Encrypt (PK, M, T) algorithm to create the ciphertext CT and uploads CT to CS in order to safeguard the confidentiality of the data. Every data ciphertext gathered by IoT devices is stored in CS, and each ciphertext is identified by a set of tags.
- **Data removal:** In order to remove specific types of ciphertext in CS, DO must first create a deletion policy, called PPk that identifies the ciphertext that is to be destroyed. In order to update its existing secret key SKk1 to the newly punctured secret key SKk, it then invokes the Puncture (PK, SKk1, and PPk) procedure. As a result, it can be said that the method is successful if the ciphertext saved in CS with tags satisfying PPk cannot be decrypted with SKk.
- **Decryption of data:** DO have access to that un-deleted info still. DO can get the plain data by invoking the Decrypt (CT, SKk) algorithm after receiving the unpunctuated ciphertext CT.

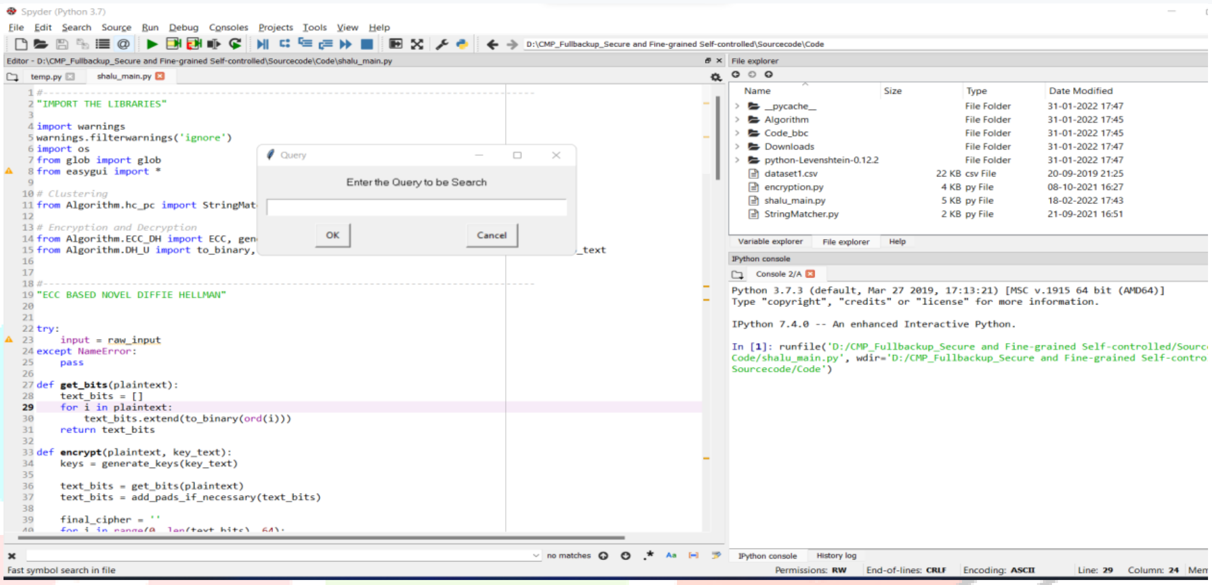
3.5 Debate of property:

- **Security**: CS is unable to recover the data using simply the random secret keys. because the exponent of Z′i contains a random number zi. Furthermore, based on P-PUN-security, ENC's even in the case that CS has access to DO's local secret key, the deleted ciphertext—that is, the ciphertext that satisfies the puncture polices—cannot be retrieved. Thus, our revised technique may ensure data confidentiality and precise data deletion.
- **Efficiency:** In our revised plan, CS is contracted to handle the secret key elements associated to the puncture policies. Only the secret key components for each puncture policy need to be stored by DO. As a result, the cost of storing DO is greatly decreased, and each puncture will only result in a small rise. Given that CS has access to the secret random keys, it can accomplish the majority of pairing operations for the lowest possible computing cost. DO just needs to do some exponentiation operations on the pairing results acquired from CS and pair the key components connected to t0 with the ciphertext components. Given that pairing is the most costly process, the calculation cost for DO is significantly lower and increases with the number of punctures at a very slow rate.

IV. RESULTS:

On a laptop running Ubuntu 20.04 and Python 3.8 and equipped with an Intel Core i5-7600U CPU operating at 2.80GHz and 16GB of RAM, we replicate our system. To implement the cryptographic operations, the Charm-crypto framework (v0.5) is used in conjunction with the OpenSSL library (v1.0.2) and PBC library (v0.5.14). The number of tags d connected to the ciphertext ranges from 1 to 10, and the group operations are based on the SS512 elliptic curve. The average run time over 40 simulations is the outcome.

We demonstrate how long it takes for the KeyGen and Encrypt algorithms to run. The more tags d there is in the ciphertext, the longer KeyGen and Encrypt take to complete. Keep in mind that the encryption operations with 10 tags can be successfully finished in 20ms. The Decode method in P-PUN-ENC and the FinalDecrypt algorithm in OP-PUN-ENC execution times are shown in Figure, which also displays the calculation costs for DO to decrypt the ciphertext in the two schemes. Each puncture policy has a specified number of tags (n) from 1 to 4, and a range of punctures (k) from 0 to 3, where k = 0 decrypts using the initial secret key.


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\CMP_Fullbackup_Secure and Fine-grained Self-controlled\Sourcecode\Code
Editor - D:\CMP_Fullbackup_Secure and Fine-grained Self-controlled\Sourcecode\Code\shalu_main.py
temp.py
shalu_main.py
Query
Enter the Query to be Search
OK
Cancel
File explorer
Variable explorer
Help
IPython console
Console 2/A
History log
Fast symbol search in file
no matches
Permissions: RW
End-of-lines: CRLF
Encoding: ASCII
Line: 29
Column: 24

```
#-----------------------------------------------------------------------
"IMPORT THE LIBRARIES"

import warnings
warnings.filterwarnings('ignore')
import os
from glob import glob
from easygui import *

# Clustering
from Algorithm.hc_pc import StringMat
 
# Encryption and Decryption
from Algorithm.ECC_DH import ECC, gen
from Algorithm.DH_U import to_binary,                                  _text


#-----------------------------------------------------------------------
"ECC BASED NOVEL DIFFIE HELLMAN"


try:
    input = raw_input
except NameError:
    pass

def get_bits(plaintext):
    text_bits = []
    for i in plaintext:
        text_bits.extend(to_binary(ord(i)))
    return text_bits

def encrypt(plaintext, key_text):
    keys = generate_keys(key_text)

    text_bits = get_bits(plaintext)
    text_bits = add_pads_if_necessary(text_bits)

    final_cipher = ''
    for i in range(0, len(text_bits), 64):
```

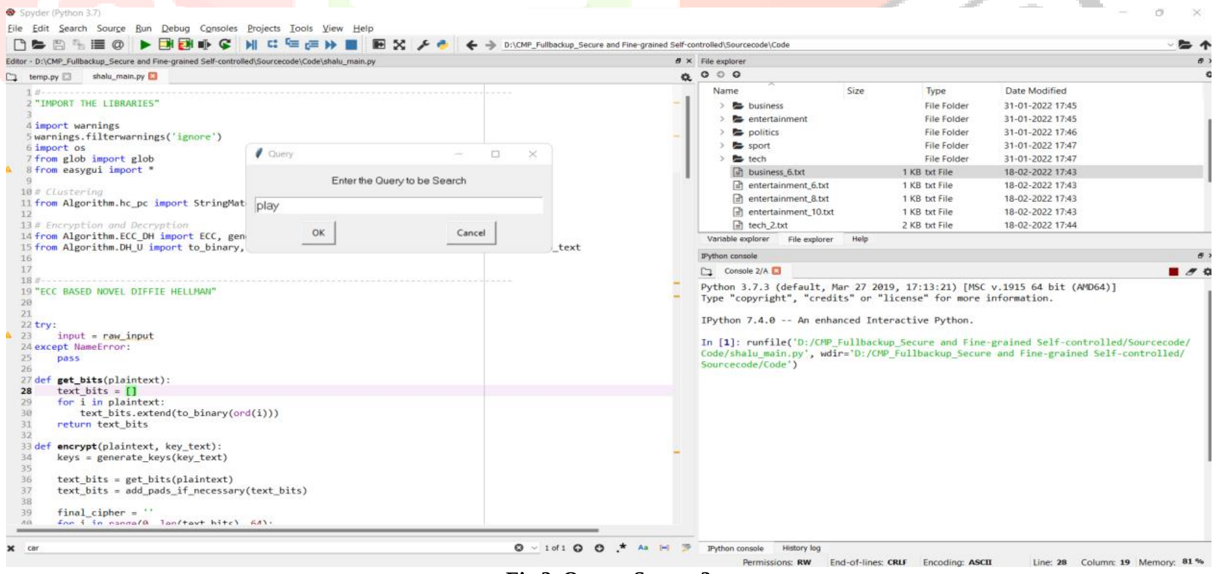
```
Name                       Size     Type          Date Modified
__pycache__                         File Folder   31-01-2022 17:47
Algorithm                           File Folder   31-01-2022 17:45
Code_bbc                            File Folder   31-01-2022 17:45
Downloads                           File Folder   31-01-2022 17:47
python-Levenshtein-0.12.2           File Folder   31-01-2022 17:47
dataset1.csv               22 KB    csv File      20-09-2019 21:25
encryption.py              4 KB     py File       08-10-2021 16:27
shalu_main.py              5 KB     py File       18-02-2022 17:43
StringMatcher.py           2 KB     py File       21-09-2021 16:51
```

```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.4.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/CMP_Fullbackup_Secure and Fine-grained Self-controlled/Sourc
Code/shalu_main.py', wdir='D:/CMP_Fullbackup_Secure and Fine-grained Self-contro
Sourcecode/Code')
```

Fig.1. Outpur screen 1


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\CMP_Fullbackup_Secure and Fine-grained Self-controlled\Sourcecode\Code
Editor - D:\CMP_Fullbackup_Secure and Fine-grained Self-controlled\Sourcecode\Code\shalu_main.py
temp.py
shalu_main.py
Query
Enter the Query to be Search
play
OK
Cancel
File explorer
Variable explorer
Help
IPython console
Console 2/A
History log
car
1 of 1
Permissions: RW
End-of-lines: CRLF
Encoding: ASCII
Line: 28
Column: 19
Memory: 81 %

```
#-----------------------------------------------------------------------
"IMPORT THE LIBRARIES"

import warnings
warnings.filterwarnings('ignore')
import os
from glob import glob
from easygui import *

# Clustering
from Algorithm.hc_pc import StringMat

# Encryption and Decryption
from Algorithm.ECC_DH import ECC, gen
from Algorithm.DH_U import to_binary,                                  _text


#-----------------------------------------------------------------------
"ECC BASED NOVEL DIFFIE HELLMAN"


try:
    input = raw_input
except NameError:
    pass

def get_bits(plaintext):
    text_bits = []
    for i in plaintext:
        text_bits.extend(to_binary(ord(i)))
    return text_bits

def encrypt(plaintext, key_text):
    keys = generate_keys(key_text)

    text_bits = get_bits(plaintext)
    text_bits = add_pads_if_necessary(text_bits)

    final_cipher = ''
    for i in range(0, len(text_bits), 64):
```

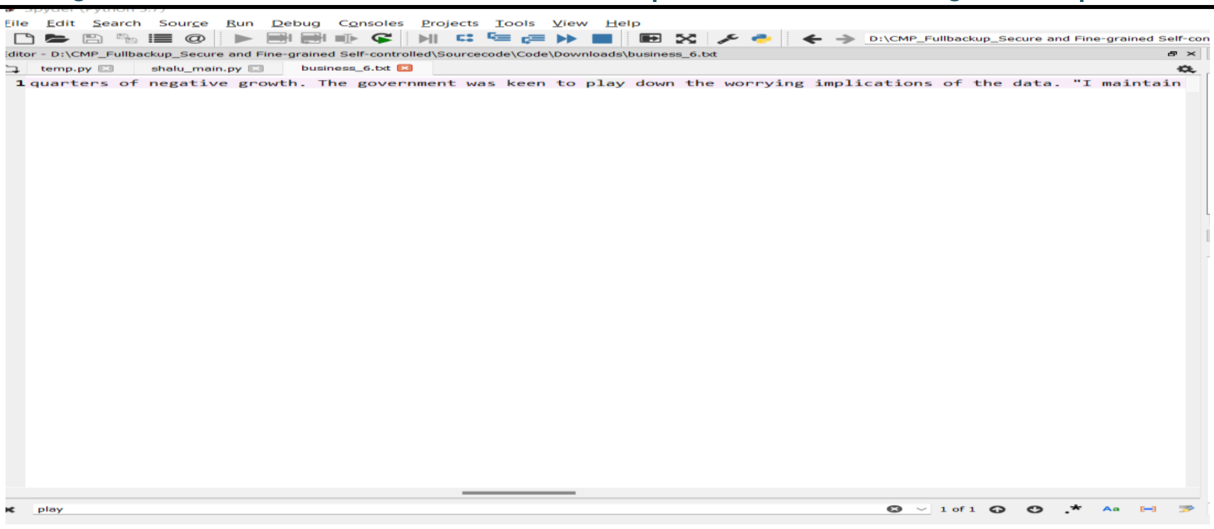
```
Name                       Size     Type          Date Modified
business                            File Folder   31-01-2022 17:45
entertainment                       File Folder   31-01-2022 17:45
politics                            File Folder   31-01-2022 17:46
sport                               File Folder   31-01-2022 17:47
tech                                File Folder   31-01-2022 17:47
business_6.txt             1 KB     txt File      18-02-2022 17:43
entertainment_6.txt        1 KB     txt File      18-02-2022 17:43
entertainment_8.txt        1 KB     txt File      18-02-2022 17:43
entertainment_10.txt       1 KB     txt File      18-02-2022 17:43
tech_2.txt                 2 KB     txt File      18-02-2022 17:44
```

```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.4.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/CMP_Fullbackup_Secure and Fine-grained Self-controlled/Sourcecode/
Code/shalu_main.py', wdir='D:/CMP_Fullbackup_Secure and Fine-grained Self-controlled/
Sourcecode/Code')
```

Fig.2. Output Screen 2



File Edit Search Source Run Debug Consoles Projects Tools View Help

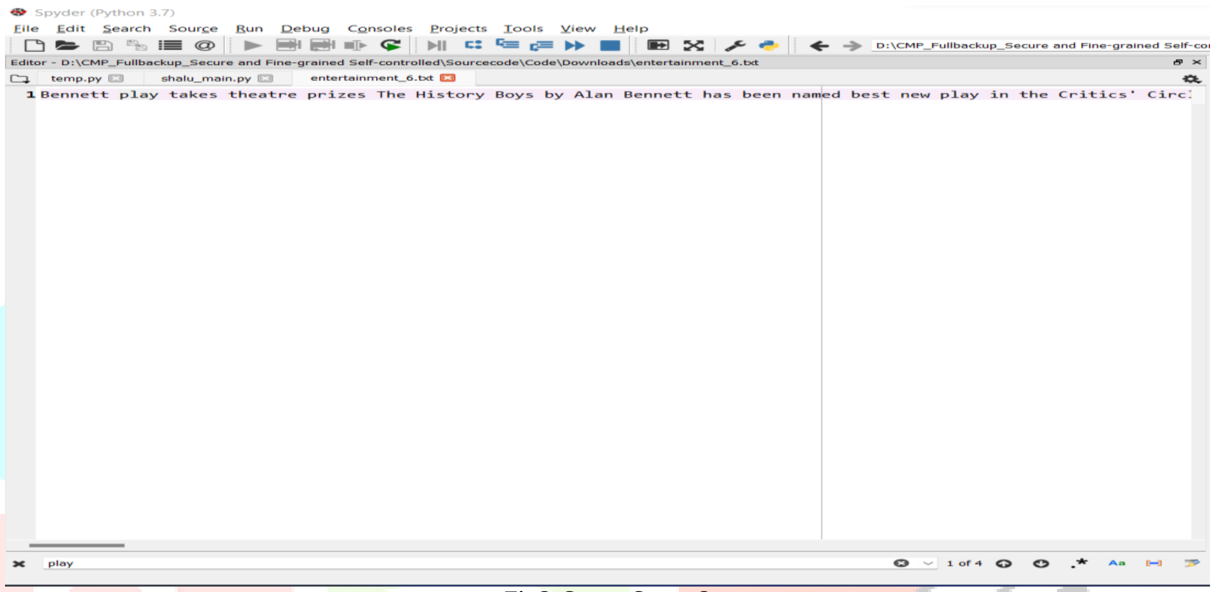
Editor - D:\CMP_Fullbackup_Secure and Fine-grained Self-controlled\Sourcecode\Code\Downloads\business_6.txt

temp.py shalu_main.py business_6.txt

```
quarters of negative growth. The government was keen to play down the worrying implications of the data. "I maintain
```

play 1 of 1

Fig.3. Output Screen 3



Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Editor - D:\CMP_Fullbackup_Secure and Fine-grained Self-controlled\Sourcecode\Code\Downloads\entertainment_6.txt

temp.py shalu_main.py entertainment_6.txt

```
Bennett play takes theatre prizes The History Boys by Alan Bennett has been named best new play in the Critics' Circ
```

play 1 of 4

Fig.3. Output Screen 3

V. CONCLUSION

In this research, we offer a safe and granular self-controlled outsourced data deletion scheme for cloud-based IoT. This scheme enables data owners too accurately and permanently erases the IoT-driven data stored in cloud in a policy-based manner without relying on a third party. In order to do this, we have presented algorithm constructs for tag-based puncturable encryption's extension, policy-based puncturable encryption (P-PUN-ENC) (T-PUN-ENC). To be more precise, we developed a policy transform method based on the logical connection between puncture policy and access policy, and we used the key delegation mechanism in ABE to carry out the key update activities. In addition, we have suggested outsourced policy-based puncturable encryption (OP-PUNENC) primitive by integrating key and decryption outsource technique with P-PUN-ENC to address the issue of rising key storage and decryption cost existing in T-PUN-ENC and P-PUN-ENC. For the purpose of proving that our suggested approach can more effectively meet the data deletion needs in a cloud-based IoT environment, we have formally confirmed the security features and thoroughly compared it with various relevant studies. The effectiveness of our system in terms of compute and storage overheads has been demonstrated by comprehensive numerical analysis and experiment simulations. Future work will focus on more specialised security concerns in the IoT environment and attempt to find a compromise between secure deletion and flexible data exchange.

ACKNOWLEDGMENT

I pay thanks to our project guide Prof. Sushil Venkatesh Kulkarni for assistance and guidance especially related to technicalities and also who encouraged and motivated us.

REFERENCES

[1] Secure and Fine-grained Self-controlled Outsourced Data Deletion in Cloud-based IoT, Jialu Hao, Student Member, IEEE, Jian Liu, Wei Wu, Fengyi Tang, and Ming Xian, JOURNAL OF LATEX CLASS FILES, VOL. 14, NO. 8, AUGUST 2015.

[2] C. Stergiou, K. Psannis, A. Plageras, Y. Ishibashi, and B.-G. Kim, "Algorithms for efficient digital media transmission over iot and cloud networking," J. Multimed. Inform. Syst., vol. 5, no. 1, pp. 1–10, 2018.

[3] R. Lu, "A new communication-efficient privacy-preserving range query scheme in fog-enhanced iot," IEEE Internet Things J., 2019.

[4] A. P. Plageras, K. E. Psannis, C. Stergiou, H. Wang, and B. Gupta, "Efficient iot-based sensor big data collection-processing and analysis in smart buildings," Future Gener. Comput. Syst., vol. 82, pp. 349 – 357, 2018.

[5] K. Zhang, J. Ni, K. Yang, X. Liang, J. Ren, and X. Shen, "Security and privacy in smart city applications: Challenges and solutions," IEEE Commun. Mag., vol. 55, no. 1, 2017.

[6] "Popular internet of things forecast of 50 billion devices by 2020 is outdated," Available at: https://spectrum.ieee.org/techtalk/telecom/internet/popular-internet-of-things-forecast-of-50- billion-devices-by-2020-is-outdated, accessed May 1, 2019.

[7] K. E. Psannis, C. Stergiou, and B. B. Gupta, "Advanced mediabased smart big data on intelligent cloud systems," IEEE T. Sustain. Comput., vol. 4, no. 1, pp. 77–87, 2019.

[8] X. Yang, R. Lu, J. Shao, X. Tang, and H. Yang, "An efficient and privacy-preserving disease risk prediction scheme for ehealthcare," IEEE Internet Things J., 2019.

[9] V. Odelu, S. Saha, R. Prasath, L. Sadineni, M. Conti, and M. Jo, "Efficient privacy preserving device authentication in wbans for industrial e-health applications," Comput. Secur., vol. 83, pp. 300 – 312, 2019

[10] S. Islam, M. Ouedraogo, C. Kalloniatis, H. Mouratidis, and S. Gritzalis, "Assurance of security and privacy requirements for cloud deployment models," IEEE T. Cloud Comput., vol. 6, no. 2, pp. 387–400, 2018.