# Secure Token Generator for Accessing Application Program Interface (API)

## Implemented at Sci-Learn (e-Learning Platform) Using Pseudo Random Generator and Invertible Burg Structure Encryption Algorithm

Daniar Heri Kurniawan
Department of Informatics
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Jl. Ganesha 10 Bandung 40132, Indonesia
daniar.h.k@gmail.com

***Abstract*—in this sophisticated technology era, security is the term that got attention the most, especially for the system that is dealing with personal information. Generating a token is one of the process, which is needed in almost every system in order to give the user an additional access beyond the system's interface. The token should be secure and unique for each user to prevent from malicious user exploitation. As we concern about security breaching, Sci-Learn is a proper system to implement the secure token generator because the token is needed almost in every case. Sci-Learn is an e-Learning platform that can be accessed through www.sci-learn.com. This e-Learning platform can be differed from other e-Learning system because it combines gamification and social network features to create engaging e-Learning environment. In the existing system such as Twitter, Google maps, and Facebook, they are using additional variable such as username and password to make sure the uniqueness and the security for accessing their Application Program Interface (API). Application Program Interface enables the other system/software to use some features of the particular system. Regardless of the various implementation and the needs for accessing system's API, this paper only focus on explaining the design and implementation of secure token generator in Sci-Learn. Moreover, the token will be generated using Invertible Burg Structure (IBS) algorithm, pseudo random generator, and SHA-256 function that will provide a secure random characters for the identification purpose. For further development, the generator can be implemented to generate a CSRF (Cross Site Request Forgery) token for the client.**

***Keywords—secure token generator; invertible burg structure; sci-learn; e-Learning platform; security***

## I. INTRODUCTION (*HEADING 1*)

Secure token generator is very important in every system because it should secure enough to prevent from the brute force attract. In the Sci-Learn's API system, token is used for identification. However, the generated token should be able to be validated in the server for further security checking. The length of the token that is produced is 256 bit or 32 characters. That is related to the length of hash result. Function SHA-256 is considered to be used because it is secure and have not been broken yet. The combination of function is arranged in order to produce a secure random token. The token is also depending on user's username and password so it can be reproduced for further authentication checking.

The Invertible Burg Structure, will be called IBS from here after, is a block cipher algorithm that customized by the writer to provide the better level of security. The explanation about IBS can be found in the next section. By combining IBS, pseudo random generator and SHA-256 function, secure token will be provided to access Sci-Learn's API securely.

## II. SCI-LEARN

Sci-Learn is an e-Learning platform that is still in development stage. It aims for providing a new learning environment especially that related to gamification aspects and social network features. Beside providing a User Interface (UI), Sci-Learn also provide an API access that is used for a research on the field of multi-agent system. The purpose of the research is to develop a multi-agent system for testing proposed recommendation functions which support learners to find suitable information, knowledge, and other learners from huge information and communication in a social learning platform.

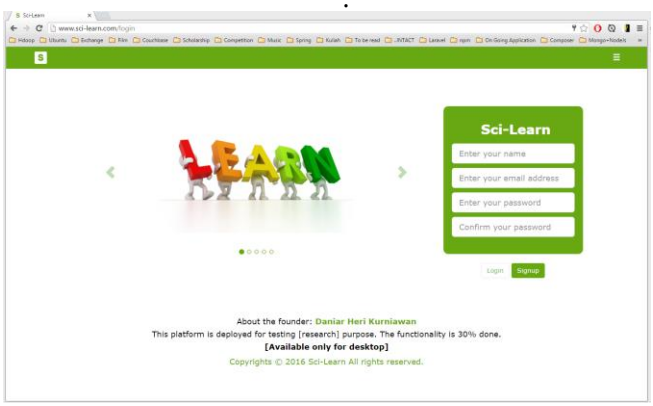


Figure II-1 Signup page of Sci-Learn

In order to confirm run ability of such functions, it is required to input a certain amount of quantity of data, but it is difficult to prepare the data to test them in advance. By using the API, the multi-agent system will simulate various behaviors of the learners by creating data for testing.

### *A. Development Approach*

Web 2.0 introduces the new way of collaboration among internet users by creating and sharing their contents. As the increasing number of information technology development in the education field, it provides collaboration and interaction space between teacher and student. The contribution of social network (Facebook, Google+, Twitter, etc) becoming very important in the educational environment because the teacher and the student can use it for discussing the course material regardless of the school/class schedules.

There are a lot of teacher that are using e-Learning system / Learning Management System (LMS) for storing course's material and social network for discussion room. Besides, the implementation of Game Based Learning (GBL) is also increasing. However, the most famous concept in the field of game that widely used in educational system is gamification. Gamification is not like GBL because gamification is only applying the theory or concept of game without needs any game to be played.

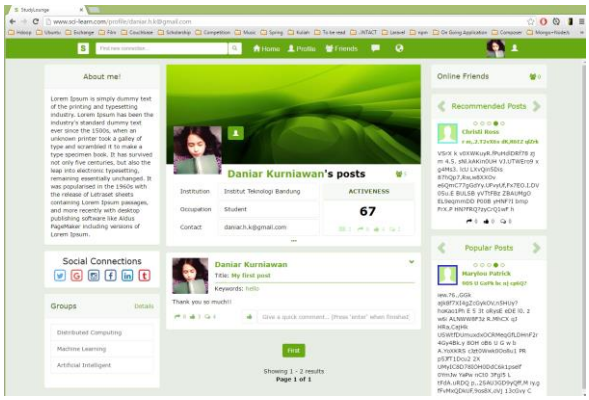


Figure II-2 Profile page of Sci-Learn

According to gamification concept and social network features, Sci-Learn is developed to provide an e-Learning platform that can accommodate the teacher's and student's needs. The existing system such as Edx, Coursera, Moodle, and Khan Academy does not support social media features. Therefore Sci-Learn will give a new user experience and can be integrated to other existing e-Learning system to support scalability and compatibility. The detail integration will be described in API section.

### *B. Functionality*

Sci-Learn is not only implementing e-Learning functionality, but also the social network features and gamification approach to attract user engagement. Below are the full functionality that will be implemented.

- Share content
- Create content
- Like, comment, edit, and delete content
- Profile page
- Publication review
- Online course review
- Online class
- Create series content
- Chatting
- Collect assignment
- Online Test
- Popular and recommended post

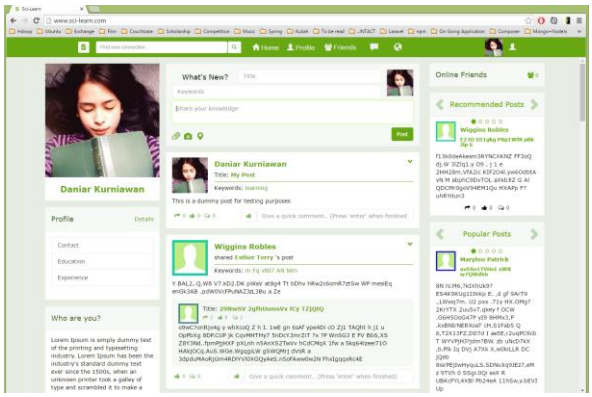


Figure II-3 Timeline Page of Sci-Learn

### *C. Sci-Learn Application Program Interface (API)*

An API expresses a software component in terms of its operations, inputs, outputs, and underlying types, defining functionalities that are independent of the respective implementations, which allows definitions and implementations to vary without compromising the interface. A good API makes it easier to develop a program by providing all the building blocks, which are then put together by the programmer. In this system, the API will provide every aspect of Sci-Learn features. However, the user should do a specific configuration in order to have a full access to the API.

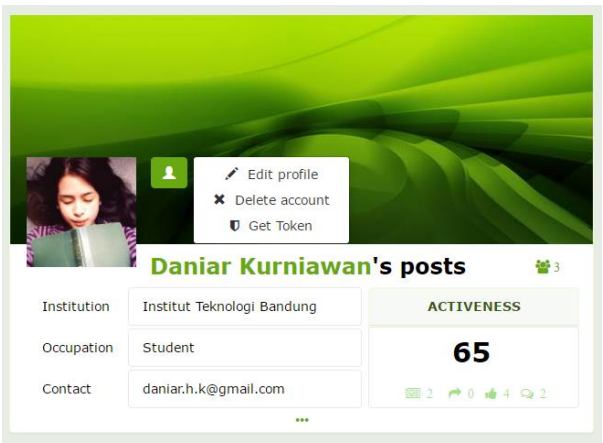


Figure II-4 Option menu to generate API Token

## III. INVERTIBLE BURG STRUCTURE

Invertible Burg Structure (IBS) is a block cipher encryption algorithm with the block length is 16 Bytes. The key length is start from 8 Bytes to 32 Bytes. The length of the key is up to the user, IBS only use the hash of the key to create the subkey. However, the main key also used in some particular process that will not be covered in this paper. The main key will be used to generate subkey with total 12 subkeys. The number of round in IBS is 4 round. Each round consists of 3 process that will be described in the encryption stage section.

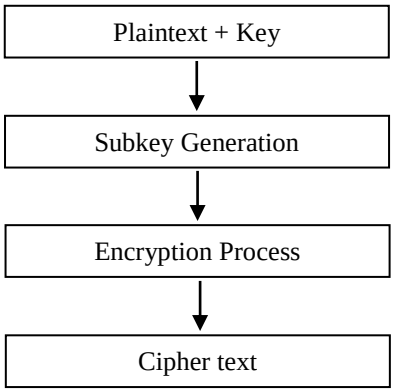


Figure III-1 IBS Encryption Schema

### A. *Subkey Generation*

There are total 12 subkeys which are consist of 2 types (5 Byte and 4 Byte). Below is the subkey generation algorithm. The number of round is four. Each round will use one round function that consists of two 5-Bytes subkey and one 4-Bytes subkey. The subkey are generated based on the SHA-256 of main key. The hash is used to create a 12 different subkey by doing 4 rounds processing.

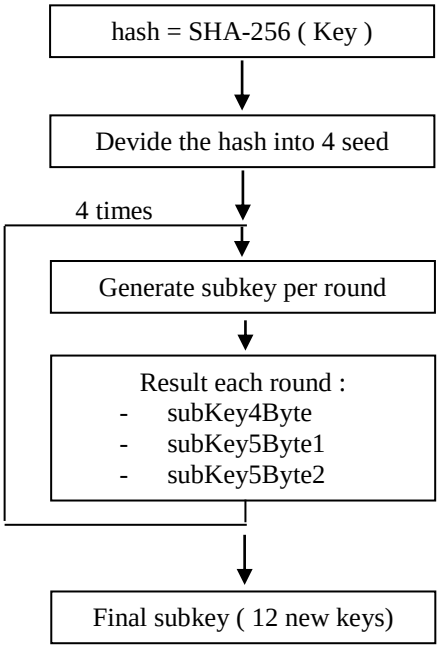


Figure III-2 IBS Subkey Generation Schema

```
public class SubKey {
   private ArrayList<Integer> hashOfKey;
   public ArrayList<RoundKey> arrayRoundKey;
   private int roundNumber;

   public SubKey(int roundNumber){
     this.roundNumber = roundNumber;
     hashOfKey = new ArrayList<Integer>();
     arrayRoundKey = new ArrayList<>();
    }

   public void generateSubKey(ArrayList<Integer> mainKey){
     int firstSeed = getFirstSeed(mainKey);
     int secondSeed = getSecondSeed(mainKey);
     hashOfKey=commonOperation.getHash(mainKey.toString());
     hashOfKey = shuffleArrayList(firstSeed,secondSeed,
                 hashOfKey);
     for (int i = 0; i < roundNumber; i++) {
       RoundKey roundKey = new RoundKey();
       roundKey.generateRoundKey(getBlockHash(i), mainKey);
                          arrayRoundKey.add(roundKey);
         }
     }

     private ArrayList<Integer> getBlockHash(int i) {
       int firstIdx = i*8;
       return new ArrayList<Integer>(hashOfKey.subList(
            firstIdx, firstIdx+8 ));
     }

     private ArrayList<Integer> shuffleArrayList(int
       firstSeed, int secondSeed, ArrayList<Integer>
       hashOfMsg2) {
       Collections.shuffle(hashOfMsg2, new
          Random(Long.valueOf(firstSeed)));
       Collections.shuffle(hashOfMsg2, new
          Random(Long.valueOf(secondSeed)));
       return hashOfMsg2;
     }

     private int getFirstSeed(ArrayList<Integer> mainKey) {
       ArrayList<Integer> arraySeed = commonOperation.XOR(
         new ArrayList<Integer>(mainKey.subList(0, 3)),
         new ArrayList<Integer>(mainKey.subList(3, 6)));
       int result= Integer.valueOf(arraySeed.get(0)
            +""+arraySeed.get(1)+ ""+arraySeed.get(2));
       return result;
     }

     private int getSecondSeed(ArrayList<Integer>mainKey) {
       Integer seed = commonOperation.XOR(
       (mainKey.get(5)), (mainKey.get(6)));
       int result= Integer.valueOf(seed+""+mainKey.get(7));
       return result;
     }
}
```

Figure III-3 IBS Subkey Algorithm (JAVA)

### B. *Encryption and Decryption Stage*

In this section, the process that will be explained only for the encryption process because the decryption is obviously depicted from the reverse version of encryption process. In the this process, there are 3 round or iteration done by reversing the direction of encryption. In the first round, plaintext will be encrypted from the first index. In the second round, the cipher text from first round will be re-encrypted from the last bit. Finally in the last round, the ciphertext from the second encryption is got encrypted for the third time, but the encryption is started from the first index. For each round, subkey that have been used are exactly similar. Moreover, for each round there are 12 subkey that used independently. The bigger picture will be explained through the image below.

```
public ArrayList<Integer>
   startEncryptionModeCBC(ArrayList<Integer> plainText){
     plainText = commonOperation.adjustSizeOfPlaintext(
                    plainText, blockSize);
     ArrayList<Integer> result = (ArrayList<Integer>)
                                    plainText.clone();

     /*algorithm started*/
     for (int j = 0; j < 3; j++) {
         SubKey subKey = new SubKey(roundNumber);
                          subKey.generateSubKey(mainKey);
                          subKey.print();
         for (int i = 0; i < roundNumber; i++) {
            result = encrypt(subKey.arrayRoundKey.get(i),
                     result);
         }
          Collections.reverse(result);
     }
   return result;
}
```

Figure III-4 IBS Encryption Schema

```
/*To encrypt*/
public static ArrayList<Integer> blockE(RoundKey roundKey,
                    ArrayList<Integer> blockPlainText){
  Encryption encryption = new Encryption();
  blockPlainText = encryption.firstSubtitutionEnc(
                       roundKey.key4Bytes, blockPlainText);
  for (int i = 0; i < 5; i++) {
     blockPlainText = encryption.chainingOperation(
            roundKey.key5Bytes2.get(i),blockPlainText);
  }
  blockPlainText = encryption.secondSubtitutionEnc(
            roundKey.key5Bytes1, blockPlainText);
  blockPlainText = encryption.sBoxEnc("daniar",
            blockPlainText);
  return blockPlainText;
}
```

Figure III-5 IBS Encryption Schema for Each Round

### C. *Experiment Result*

The IBS algorithm is tested using one block plaintext (16 characters) to check its randomness after got changed one bit in any position. This algorithm supports padding, so if the length of plaintext is less than 16 characters, it will give padding in the plain text. The padding will be removed for further decryption process.

| **Plaintext 1** | **Plain Text 2** |
|---|---|
| Daniar Heri K. | Dbniar Heri K. |
| **Ciphertext 1** | **Ciphertext 2** |
| 9699 072a c835 e7e2 af8a fb80 8a36 540e | a9a4 c44e 657c cdd5 b097 6bef afac 9e1c |

Figure III-6 Example IDS Encryption 1

| **Plaintext 1** | **Plain Text 2** |
|---|---|
| Daniar Heri K. | Daniar Heri L. |
| **Ciphertext 1** | **Ciphertext 2** |
| 9699 072a c835 e7e2 af8a fb80 8a36 540e | a98b 066b e1aa 16ba 9ae8 0e1a 58bb 2243 |

Figure III-7 Example IDS Encryption 2

Both of the example above show that the ciphertext for the slightly different plaintext is 100% different. We can generalize this result that a single bit of change can effect another bit almost perfect. Testing IBS using the longer plaintext, the result showed that the average percentage of similarity is lower than 0.5 %. For further testing, the source code can be downloaded at the link below:

https://github.com/daniarherikurniawan/Invertible-Burg-Structure-Block-Cipher

## IV. Pseudo Random Generator

A pseudorandom number generator (PRNG), also known as a deterministic random bit generator DRBG, is an algorithm for generating a sequence of numbers that approximates the properties of random numbers. The sequence is not truly random in that it is completely determined by a relatively small set of initial values, called the PRNG's state, which includes a truly random seed.

The PRNG that is used is the default random generator implemented in Java JDK. It was used because the performance and the randomness are already tested and recommended by a lot of java developers. The pseudorandom generator is used to shuffle the array of integer in the subkey generation process. The seed for the PRNG is part of main key's hash result.

## V. Secure Token Generator

There are three step of generating API token in Sci-Learn. The first is the client request a token generation. Then the server will take user's salt and user's password from the data base to generate the token. Server will not save the token in the server, because it will be changed dynamically for security purpose. Finally the token will be sent to user. Whenever the user using token to access the API, server will check it by re-compute the token as mentioned in the first step. The step by step process is depicted in the picture below:

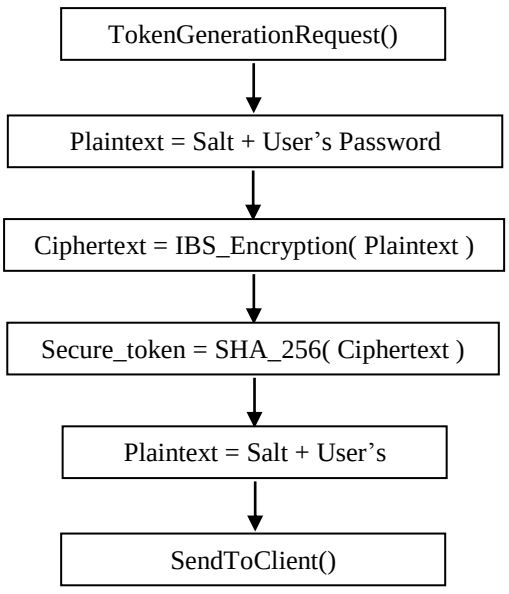


Figure V-1 Token Generation Schema

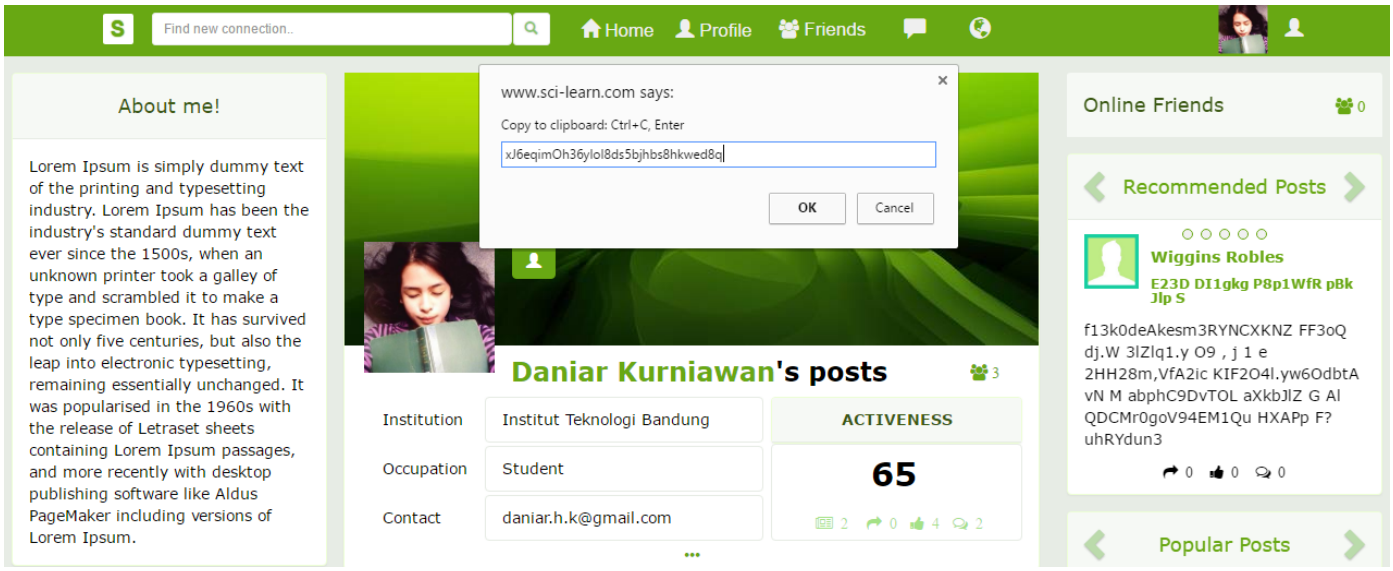


Figure V-2 Token is Successfully Generated

Beside of the main application (Sci-Learn), the writer created another system called multi agent system that can simulate the use of Sci-Learn API. Each of the token will represent a user and the multi-agent will do some actions based on the behavior that are defined as shown in the Figure V-3.

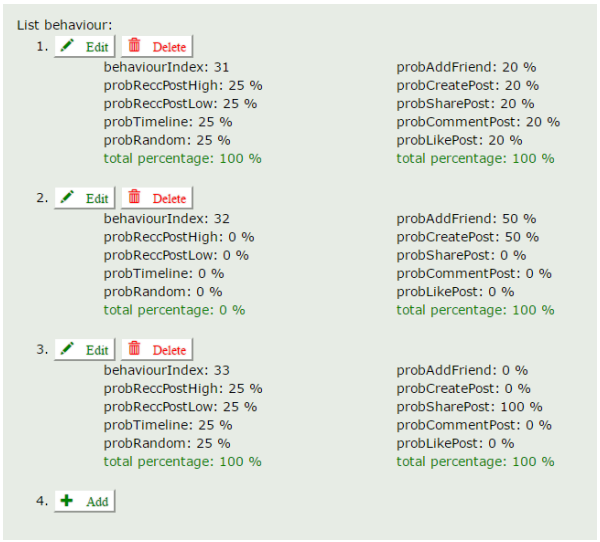


Figure V-3 Agent System That Will Simulate User's Behaviour through Sci-Learn API Access

### A. *SHA-256*

The SHA-256 compression function operates on a 512-bit message block and a 256-bit intermediate hash value. It is essentially a 256-bit block cipher algorithm which encrypts the intermediate hash value using the message block as key. Hence there are two main components to describe: (1) the SHA-256 compression function, and (2) the SHA-256 message schedule.

One of the drawbacks with SHA-2 is that there are some older applications and operating systems that do not support it. Compatibility problems are the main reason why SHA-2 algorithms have not been adopted more rapidly. Windows XP Service Pack 2 or lower does not support the use of SHA-2. The use of SHA-2 on websites may pose a problem if the end user has an older operating system.

### B. *Salt*

Salt is random data that is used as an additional input to a one-way function that "hashes" a password or passphrase. The primary function of salts is to defend against dictionary attacks versus a list of password hashes and against pre-computed rainbow table attacks.

## VI. Result and Analysis

In this paper, there is a new block cipher encryption algorithm that is introduced, called Invertible Burg Structure Algorithm. Besides, the algorithm is combined with secure hash function, SHA-256, to generate secure API token. The token is implemented in e-Learning system that also developed by the writer called Sci-Learn. The result of this result is very satisfying because the schema that is proposed is worked well in Sci-Learn platform. The experiment result of IBS algorithm also shows that the substitution and transposition worked well for creating truly different chipertext for slightly different plaintext (differed by one bit).

For the future work, the secure token generation schema could be implemented for preventing surface attack on the website by providing CSRF token for the client. Moreover, the pseudo random generator still could be improved by referring to different pseudorandom generator in widely used library such as google v8 engine pseudorandom generator function.

## Acknowledgment *(Heading 5)*

Daniar Heri Kurniawan, as the author of this paper, want to express his deepest gratitude to Dr. Ir. Rinaldi Munir, M.T. as the lecturers of Cryptography Course, ITB 2016. Special thanks to all of my family, my friends in Informatics 2012, and other people that give any form of support to me to finish this paper.

## REFERENCES


[1] FIPS 2012. Secure Hash Standard (SHS). Technical Report FIPS PUB 180-4. Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD.

[2] A Model for Determining Information Release By – Andrei Sabelfeld and Andrew C. Myers, Software Security: theories and system 2nd Mext_NSF-JSPS international symposium, ISSS-2003.

[3] https://cseweb.ucsd.edu/~mihir/papers/gb.pdf. Access at: 5/17/2016