Application of Genetic Algorithm with Content Scrambling System

Dr. Pushpa R.Suri † **and Priti Puri** ††,

Department of Computer Science & Applications,Kurukshetra University,Kurukshetra,India

Summary
The paper proposed a different approach for the existing CSS algorithm (used to protect the content of DVDs from piracy and to enforce region-based viewing restrictions). CSS is based on two Linear Feedback Shift Registers (LFSRs), an example of a stream cipher and an authentication protocol. This is the application of the LFSRs with Genetic Algorithm (GA) to increase the complexity of the existing algorithm at the position of the seeds of LFSR17 and LFSR 25.GA is used in this approach to make the seed of linear feedback shift register more complex.

Key words:
Genetic, LFSR, title key, mutation, seed, sector key.

1. Introduction

Content Scrambling System (CSS)

CSS system used to play DVDs in terms of three components: the DVD itself, the DVD player that reads the disk and delivers the content, and the host (computer, host board, &c).
The DVD disk itself contains the encrypted content, as well as a hidden area. The contents of this hidden area cannot be delivered, except to an authenticated device. This hidden area contains the several pieces of information that we will soon discuss: a table of encrypted disk keys and an encrypted disk key (disk key hash). The player itself stores the player keys that are used to decrypt the disk key, the region code that identifies the region in which the player should be used, and another secret that is used for authentication with the host. The host seems to contain a secret that is used for authentication.

1.1 CSS stream cipher primitive

The CSS [3] [4] stream cipher is based on 2 LFSRs being added together to produce output bytes. There is no truncation, both LFSR are clocked 8 times for every byte output, and there are 4 ways of combining the output of the LFSRs to an output byte. These four modes are just settings on 2 inverter switches, and the modes operations are used for the following purposes.

1) Authentication to DVD drive
2) Decryption of Disk key (*DA*)
3) Decryption of Title key (*DB*)
4) Decryption of data blocks.

LFSR1: 17 bits 3 taps, and is initialized by the 2 first bytes of key, and setting the most significant bit to 1 to prevent null cycling.
LFSR2: 25 bits 4 taps, is initialized with byte 3, 4, 5 of the key shifting all but the 3 least significant bits up 1 position, and setting bit 4 to prevent null cycling.
As new bits are clocked into the LFSRs, the same bits are clocked in with reversed order to the two LFSRs output bytes. (With optional inversion of bits)
The output of LFSR1 is *O1 (1), O1 (2), O1 (3)...*
Likewise LFSR2 produces *O2 (1), O2 (2), O2 (3)...*
These two streams are combined through 8 bits addition with carry carried over to the next output. The carry bit is zero at start of stream.

$$O(i) = O1(i) + O2(i) + c \quad (1)$$

where c is carry bit from $O(i-1)$

Region Code:
Each DVD contains a region code that indicates the region of the world in which it is intended to be viewed. Each player knows the region in which it was to be sold.
If the region code of the player doesn't match the region code on the DVD, the player won't deliver the data.

Overview of Keys
Authentication Key: This "secret" is used as part of the mutual authentication process.
Session Key (Bus Key): This key is negotiated during authentication and is used to encrypt the title and disk keys before sending them over the unprotected bus.
Player Key: This key is licensed by the *DVD Copy Control Association* to the manufacturer of a DVD player. It is stored within the player. It is used to establish the trustworthiness of the player. It is used to decrypt the disk key.
Disk Key: This key is used to encrypt title key. It is decrypted using the player key.
Sector Key: Each sector has a 128-byte plain-text header. Bytes 80 - 84 of each sector's header contain an additional key used to encode the data within the sector.
Title Key: This key is XORed with a per-sector key to encrypt the data within a sector

Overview of the Process
1) Mutual Authentication
2) The host and the drive use a challenge-response system to establish their trustworthiness to each other. In the process, they negotiate a session key.

Manuscript received March 21, 2007
Manuscript revised March 25, 2007

3) Decoding disk
4) The DVD player tries each of several player keys until it can decode the disk key. The disk key is a disk-wide secret.
5) Send disk and title keys
6) The title and bus keys are sent from the player to the host. The session key is used to encrypt the title and disk keys in transit to prevent a man-in-the-middle attack.
7) The DVD player sends a sector to the host.
8) The host decodes the title key using the disk key.
9) The host decodes the sector using the title key, and a sector key in the sector's header.

Disk and Player Keys each player has a small number of licensed player keys. These keys can be used to decrypt the disk key on a particular DVD. This disk key is used to decrypt title keys on the disk. Each work on the disk is encrypted with a title key. To decrypt the work, begin by decrypting the disk key. This disk key is stored on the hidden sector of the DVD along with a a table containing the disk key encrypted will each of the 409 possible player keys. It also holds the disk key encrypted with the disk key. The player decrypts the appropriate entry in the table and then verifies that it has correctly decoding the disk key, by decoding the encrypted disk key. The result should be the disk key. That is to say that the decryption of the disk key, using the disk key, should prove to be the identity function. Players have more than one player key, so if the operation fails, they try again with an alternate.

1.2 Linear Feedback Shift Registers (LFSRs) and Encryption

One technique used to encode a stream is to XOR it with a pseudo-random bit stream. If this random-looking bit stream can be regenerated by the receiver of the message, the receiver will be able to decode the message by repeating the XOR operation. The LFSR is one popular technique for generating a pseudo-random bit stream. After the LFSR is seeded with a value, it can be clocked to generate a stream of bits. LFSRs aren't truly random. They are periodic and will eventually repeat. The period also depends on the particular configuration of the LFSR. If the initial value of an LFSR is 0, it will produce only 0s; this is sometimes called *null cycling*. LFSRs are often combined through addition, multiplexers, or logic gates, to generate less predictable bit streams. An LFSR is *seeded* with an initial value. With each clock tick, certain *tapped* bits of the LFRS are evaluated by a *feedback function*. The output of this feedback function is then shifted into the register. The output of the register is the bit that is shifted out.

CSS's LFSRs: The CSS algorithm makes use of two LFSRs. The first is a 17-bit LFSR. Initially, it contains a two byte seed, with a 1 injected into the fourth bit, for a total of 17 bits. This is placed into the register to prevent null cycling. The second LFSR operates the same way, except it holds 25 bits. Unlike typical LFSR-based stream ciphers, CSS throws away the bit that is shifted out of each LFSR. Instead, it considers the output of the feedback function to be both the input to the LFSR and the output. CSS uses a 40-bit, or 5 byte key. This is explains the size of the two registers: one is seeded with the first two bytes of the key, and the other the remaining three bytes of the key.

LFSR Addition: The output from the two LFSRs is combined using 8-bit addition. After each LFSR clocks out 8 bits of output, this output is added to form an output byte. The carry out from this addition is used as the carry in for the addition yielding the next output byte. CSS actually has four different modes. Depending on the mode, the output of either or both LFSRs may be bit-wise inverted before the addition. The table below shows the inverter settings for each mode:

Table 1. Different key modes

Invert Output of LFSR?		
Mode	**LFSR-17**	**LFSR-25**
Authentication	Yes	No
Session Key	No	No
Title Key	No	Yes
Data	Yes	No

Data Encryption/Decryption: To encrypt or decrypt data, each LFSR is seeded with a portion of the title key. LFSR-17 is seeded with bytes 0 and 1 and LFSR-25 is seeded with bytes 2, 3, and 4. These bytes are seeded with a nonce, called the *sector key* that is read from each sector. The sector key is stored in bytes 80-84 of the sector. The first 128 bytes of each sector, the sector header, which includes the sector key, is plain text. The first two bytes (0 and 1) of the title key are XORed with the first two bytes of the sector (80 and 81), before seeding LFSR-17. Similarly, bytes 2-4 of the title key are XORed with bytes 82-84 of the sector, before seeding LFSR-25. A "1" is injected into each seed at bit 4, to make the seeds 17 and 25 bits, respectively. Once the LFSRs are seeded, their output can be added together, to form the pseudo-random bit stream. This bit stream is XORed with the plaintext, to generate the cipher text. Upon decoding, this operation is reversed.

Key Encryption: CSS goes through an additional two-step mangling operation in the case of keys, including the title key and session key. Each column represents one byte of the key. L_k is the output of the encryption step for the byte represented by the column. The output of the first stage of the mangling function feeds the input of the second stage. The first and second stage are otherwise identical, except for the fact that the output of the 4th byte of the second stage does not wrap around and feed the XOR in the first column.

Mutual Authentication: Before the DVD player will begin to send data over the bus to the host, it first goes through a form of weak mutual authentication with the host. In the process, it negotiates a key for use in encrypting the data in transit over the bus. This encryption is necessary because it would otherwise be possible to snoop the plaintext data right off of the bus, rendering the prior encryption virtually useless. The key that is negotiated is known as the *session key* or *bus key*.

The negotiation begins when the host requests an *Authentication Grant ID (AGID)* from the drive. This ID is much like a session ID or a thread ID. The next is the host generates an arbitrary stream of bytes called a *nonce* or *challenge* and sends it to the drive. The drive then encrypts this stream of bytes and sends them back to the host. The host then decrypts the byte stream and ensures that it is correct. It assumes that the drive is authentic, because it knew the correct secret and algorithm to encode the nonce. The host performs exactly the same operation. It generates a nonce, encrypts it, and sends it to the host. The host in turn encrypts the nonce and sends it back to the drive. The drive then decrypts the nonce and makes sure that it is in fact correct. At this point, both the host and the drive trust each other. Both the host and the drive now know each other's nonces. Each then takes the two nonces, combines them, and encrypts them as described earlier. The result is the bus key, i.e. session key. This key is used to encrypt all data sent between the drive and the player. Since only the player and the host know the nonces which change every session, only the player and the host can generate the key needed to decrypt the data. During encryption and decryption, a different substitution table is used to perform the initial substitution for each of the keys.

2. Purposed approach:

The above approach of CSS using LFSR 17 and LFSR 25 and for making the seed to initiate the LFSR's using the XORing of title key and sector key as

LFSR 17' seed is XOR of 0 and 1 byte of sector key and title key.

LFSR 25' seed is XOR of 2, 3 and 4 byte of sector key and title key.

Now here in this approach, we are introducing genetic algorithm to increase the complexity of this method. With the help of genetic algorithm we apply the operations of GA as selection, crossover and mutation on the bytes.

We can also reduce the brute force attack with this approach because intruder not only need to find the sector and title key but also required the mutation and crossover positions of the seeds of LFSRs.

Genetic algorithm[1][2] is a search algorithm based on the mechanics of natural selection and natural genetics [1] using the Darwinian principle of reproduction and survival of the fittest with genetic operations. GA pioneered by John Holland as a modification of evolutionary programming [1] in 60's. His idea was to construct a search algorithm modeled on the concepts of natural selection in the biological sciences. The process begins by constructing a random population of possible solutions. This population is used to create a new generation of possible solutions which is then used to create another generation of solutions, and so on. The best elements of the current generation are used to create the next generation. It is hoped that the new generation will contain "better" solutions than the previous generation.

Evaluation: The fitness value represented by an agent's genes is calculated for all agents in the population. A relatively high fitness value is desirable to ensure survival in the selection phase.

Selection: The selection process is based on probability. This phase has an element of randomness just like the survival of organisms in nature. The probability for selection is based on the agent's fitness value relative to the rest of the population (survival of the fittest).Then a random number generator is used to select agents for its cross over phase.

For LFSR 17,

After the XORing of 0 and 1 byte of sector key and title key, apply operation of GA as selection (select the XORing of 0 and 1 byte output) then do crossover and mutation on these bytes as

Sector key	SK1=1100110010101011
Title key	TK1=1001001101011001

(2)

Cross over: The process of combining the genes of one agent with those of another to create offspring (that inherit traits of both parents) is crossover. The crossover rate is the odds of an agent being selected for the crossover operation. The agents that are not selected will not have their genes changed before proceeding to the mutation phase. Those that are chosen will be paired with a mate who is another agent that was also selected for crossover. From each pair, two offspring will be created that will replace their parents. If the length of the each string is r, then a random number between 1 and r is selected, say s. The mating process is one of swapping bits s + 1 to r of the first parent with bits s + 1 to r of the second parent.

Example:

SK1 and TK1, each having 16 bits; the number 9th is chosen for crossover and denotes the split position of bytes is given below:

SK1 Bytes:	110011001-0101011
TK1 Bytes:	100100110-1011001
SK1'sbytes:	110011001/1011001

Left side from SK1 and right side from TK1

TK1's bytes 100100110/0101011

Left side from TK1 and right side from SK1.
We have

SK1'=1100110011011001
TK1'=1001001100101011 (3)

Now for LFSR25,
Three bytes of sector key

SK2=111100111100010110011000 (4)

Three bytes of title key

TK2=110011011100010010011100 (5)

Now applying crossover on SK2 and TK2 at 12th position.

SK2 =111100111100-010110011000
TK2=110011011100-010010011100

SK2'sbytes: 111100111100/010010011100

Left side from SK2 and right side from TK2

TK2's bytes 110011011100/010110011000

Left side from TK2 and right side from SK2.

We have

SK2'=111100111100010010011100
TK2'=110011011100010110011000 (6)

Mutation: After crossover, each individual has a small chance of mutation. The purpose of the mutation operator is to simulate the effect of transcription errors that can happen with a very low probability when a chromosome is mutated [2]. Just as in nature, some agents will have random mutation occur in their genes. The mutation rate specifies the odd that a given gene in an agent will be mutated. If a gene is selected from mutation then its value will be changed. In the case of bit representation, the gene will simply be flipped, that is a one changed to a zero or a zero is changed to a one
Then apply mutation on SK1' and TK1' on mutating the bit positioned at 5th and 11th as making bits 0 to 1 and 1 to 0.

SK1"= 1100010011111001
TK1"= 1001101100001011 (7)

Now we can XOR this SK1" and TK1". Then add one bit of carry to result which make it 17 bits seed for LFSR17.

We have

SK2'=111100111100010010011100
TK2'=110011011100010110011000

Then apply mutation on SK2' and TK2' on mutating the bit positioned at 11th and 19th as making bits 0 to 1 and 1 to 0.

SK2''=111100111110010010111100

TK2''=110011011110010110111000 (8)

Now we can XOR this SK2" and TK2". Then add one bit of carry to result which make it 25 bits seed for LFSR 25.

At the time of decryption, first ask for the mutation places of SK1" and TK1" for LFSR-17, SK2" and TK2" for LFSR-25. After mutating the positions, we get SK1'and TK1' for LFSR-17, SK2' and TK2' for LFSR-25.
Now the second step is to find the crossover positions for SK1'and TK1' for LFSR-17, SK2' and TK2' for LFSR-25.
Then, we get SK1 and TK1 for LFSR-17, SK2 and TK2 for LFSR-25.
These values will be seeded in LFSRs and the remaining process of CSS will be continued.

Conclusion

This paper utilizes the concept of GA to increase the complexity of existing CSS algorithm. CSS algorithm is used to protect the content of DVDs from piracy and to enforce region-based viewing restrictions. CSS is based on two Linear Feedback Shift Registers.
This approach has used GA for the encryption/decryption purposes of cryptography where earlier GA has used for the cryptanalysis purpose.
Existing CSS algorithm is weak at the seed creation level of LFSRs i.e. intruder can find the keys if seeds known to him. So that we have increased the complexity for the seed of LFSRs by applying GA on it. Another benefit of this approach is the avoidance of brute force attack up to some extent.
Genetic algorithm is introduced on data before making it seeds for LFSRs. Man in the middle attack is also not effective due to genetic algorithm.

The complexity of CSS approach can be increased double as position of crossover and mutation of sector key and title key are not known. This purposed approach has been given mathematically and shown in the figure.1also.

References

[1] Goldberg, David. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading MA: Addison-Wesley.
[2] Tomassini, M. (1999). Parallel and Distributed Evolutionary Algorithms: A Review. in K. Miettinen, M. Makel¨a, P. Neittaanmaki and J. Periaux (Eds.), *Evolutionary*

Algorithms in Engineering and Computer Science (pp. 113 - 133). Chichester: J. Wiley and Sons.
[3]Fawcus, D. and Roberts, Mark, css-auth package, December, 1999.
[4]Stevenson, Frank A., "Cryptanalysis of Content Scrambling System", 8 Nov. 1999, as updated 13 Nov. 1999.

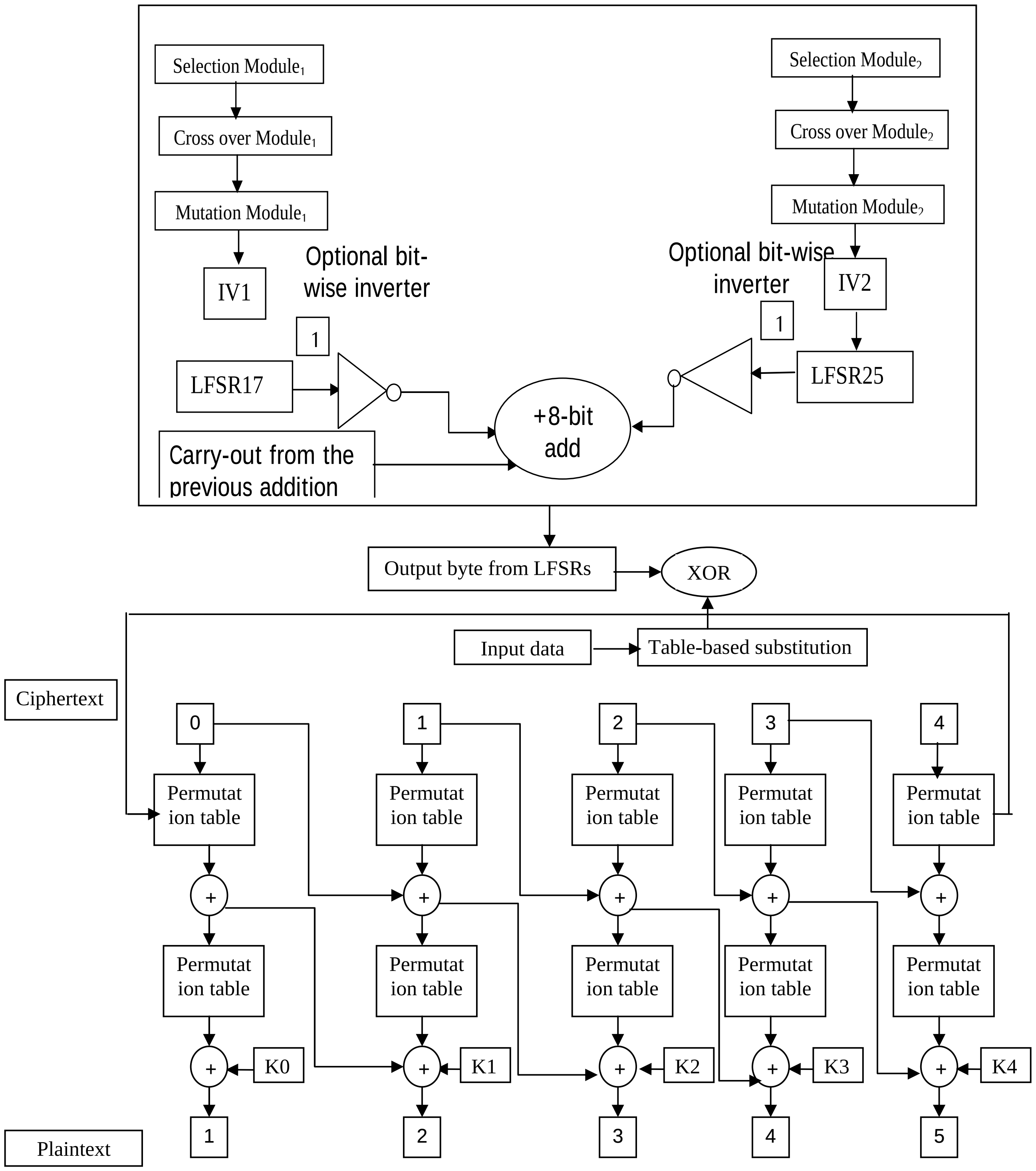

Fig.1 CSS approach with Genetic Algorithm