International Journal of Electronics Communication and Computer Engineering
Volume 2, Issue 2, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Cipher for Smart Card Using Unicode, Length and Position

Natang. A. Salgia

***Abstract* — Smart cards have become a common thing today. Smart cards with various utilities are used today since they are easy to carry and relatively secured. But due to advancement of technologies, smart cards are becoming increasingly insecure. A paper from certain conference suggested a new encryption algorithm to keep the information secured in smart cards and to overcome forgery attacks. It claims that, on an average, even a supercomputer will take 10^{1084} years to decrypt, which is much higher than the previous proposals. It makes use of the length of the PIN, and the position and the Unicode value of every character in the PIN along with a secret key to create cipher text. Also, final cipher text is formed by combining two different cipher texts, one from date and time and another from PIN and secret key. And only this final cipher text is saved on the smart card making this algorithm further more secured. But few anomalies were noticed in it. Firstly, the cipher text generated using proposed formula for encryption makes securing it using position problematic. Secondly, the formula given in actual paper misses an important step, without which decryption process gives wrong output. With proposed changes these problems are eradicated and the algorithm becomes more difficult to hack.**

***Keywords* – Cryptography, Encryption, Hash Function, Smart Card.**

I. INTRODUCTION

Normal encryption using only English alphabets will have only 26! combinations and can be easily decrypted. Furthermore, even usage of any ASCII characters or special characters will have fewer combinations for brute force attack and can be decrypted within certain period of time. By using the relative frequency, it is common to decrypt. To avoid various attacks like dictionary attack, mathematical attack, timing attack etc., we need an efficient way to encrypt the PIN in smart cards [3].

II. HASH FUNCTION

Before stepping front, it is required to know about the basic working principle of hash function [2].

A. Working Principle

The input to the hash function can be of any length. But output length is fixed i.e. for hash function h(x) input of any length will give output of fixed length, say n. At times it has to undergo numerous iterations of operations to obtain the fixed length of hash value [4]. There are various steps involved to bring the hash value into reality and security.

B. Advantages

First advantage of hash function is uniformity. All outputs are of equal lengths. This leads to another advantage of compression. If our output length is fixed to 5, then for input of length 20 we will get compressed output of length 5. Second main advantage is that it is one-way. It is difficult to invert a hash.

Hash functions are collision resistant, both weak and strong. Given x and h(x), it's infeasible to find y, with $y \neq x$, such that h(y) = h(x). This shows that it is weak collision resistant. It is strong collision resistant because it's infeasible to find any x and y, with $x \neq y$, such that h(x) = h (y)[4].

III. THE PROPOSED ALGORITHM

The authors of original paper[1] put forth their effort to develop a new algorithm for Information security. It makes use of 255 characters and the number of alternate keys is relatively higher.

Unicode characters instead of English alphabets are used in it. Also, since it used the position value, the relative frequency attack is avoided. The proposed algorithm changes the position of cipher text too. Hence the name **'n' level UNICODE position character length ciphers**[1].

But it has couple of anomalies which makes implementation very difficult. These anomalies are explained along with below mentioned modified version of the original algorithm[1].

A. The Encryption Process

The final cipher text, obtained using this algorithm, is a combination of two cipher texts generated my applying two different encryption techniques. This adds to the security of this proposed algorithm.

1) Nomenclature

P = Plain Text (PIN in this case)

Date & Time = Date and time when the card was/will be issued to user.

CT_1 = Cipher Text 1

CT_2 = Cipher Text 2

CT = Final Cipher Text such that $CT = CT_1 + CT_2$

X_i = ith character of CT_2

Manuscript received October 2, 2011. This work was supported in part by the U.S. Department of Commerce under Grant BS123456 (sponsor and financial support acknowledgment goes here).

Natang. A. Salgia is a seventh semester computer engineering student at Thakur College of Engineering and Technology, Kandivali, Mumbai, India (phone: 91-9867545837; e-mail: natang.salgia@yahoo.com).

K = Secret Key
Med = Median Value(s)
n = Length of P.
$i = 0,1,2 \ldots (n-1)$

2) Encryption Steps

The encryption, $CT = E(P,K)$ using this algorithm consists of 5 steps.
The encrypted PIN is obtained through the following calculations.

Step 1:

$$CT_1 = BASE64\{\sum(month, day, year, hour, minute, seconds)\}$$

The DATE and TIME on which the user obtained the smart card is something not many would know. It is hard for the hackers to guess it.

Step 2:

$$Unicode(X_i) = (i + Unicode(P_i) + Unicode(K_i) + n) mod\ 256$$

Store 'n' value in highly secured database to be used during decryption.
This is where proposed algorithm[1] went wrong. As per it,

$$X_i = (i + Unicode(P_i) + Unicode(K_i) + n) mod\ 256$$

Thus value of X_i varies from 0 to 255. Now if the cipher text is stored in this form it would be difficult rather impossible to decrypt it when needed. Consider $X_0 = 101, X_1 = 5, X_2 = 23$. Thus, intermediate $CT_2 = 101523$. Now while decrypting it to authenticate a user how would the system know whether its 101, 5, 23 or 10, 15, 23 or 101,52,3?
Also, it goes completely against the property of hash function of getting output always of fixed length.
My proposal is that these values should be used as Unicode values and CT_2should be stored in character form. Even for that a work around is required because not all Unicode values have character representation[5] e.g. decimal values from 0 to 32. For this, next 33 decimals after 255 can be used. And hence we add an intermediate step:

If $Unicode(Xi) < 33$
Then $Unicode(Xi) += 256$

This is only if Decimal code points are used. Decimal code points 127-129, 141-144, 149, 157, 158 and 160, also, don't have character representation. So they are replaced by 289-299, in order. Numbers in condition and action depends on Unicode type used. This adds a bit more security to algorithm because it takes away uniformity of encryption without making it difficult for authorized system to authenticate. No complete uniformity means difficult to use relative attack on this algorithm.

Repeat this step n time to get X_0 to X_{n-1}.

Step 3:

$Med = X_{(n-1)/2}$ When n is odd.

$Med = X_{n/2}, X_{(\frac{n}{2})+1}$ When n is even.

As we can see, center value(s) of X is obtained and saved as Med.
Step 4:

$$CT_2 = \{[X_0, X_1 \ldots X_{n-1}] - Med\}, Med$$

Now, we remove the centre value(s) from X and append it to X to obtain CT_2. This again reduces chances of being decrypted correctly using relative frequency because the PIN obtained would not be in proper order.

Step 5:

$$CT = CT_1 + CT_2$$

This CT should be loaded into the smart card. While CT_1 and CT_2 are stored in secure database.

3) Explanation

Initially the Date is retrieved from system date and the sum of date, month and year is calculated. Further, the Time value is also retrieved and hour, minutes and seconds are summed up. Then sum up both date and time. Now the obtained value is subjected to BASE 64 computation.
Original algorithm states that the value n should be taken from the result of above calculations. This is of no use as the summation always gives a 4 digit number and its BASE64 will always give an 8 character output. So the algorithm doesn't really need to rely on this calculation for the value of n. It's redundant.
Second part of the cipher text is obtained by subjecting Unicode values of P and K to above mentioned formula. Median of this cipher text is calculated and appended to cipher text after removing it from its position. This algorithm will take less time for encryption and takes more time for hackers to decrypt.

B. The Decryption Process

When a person enters his/her PIN to login, first the CT_1part is obtained from the smart card and compared with date and time in database. Only if it matches, PIN will be checked. This helps prevent intermediate attack.
Also the PIN is never sent across the network, the encrypted PIN is decrypted on single system and checked against the PIN entered by user, thus enhancing efficiency of the algorithm.
The decryption algorithm performs the reverse operation of encryption such that

$P = D(K, CT)$.

1) Decryption Steps

Step 1:

International Journal of Electronics Communication and Computer Engineering
Volume 2, Issue 2, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Obtain n from the database through a highly secure channel. Using this separate CT_1 and CT_2 from CT stored in the smart card.

Step 2:

$BASE64^{-1}(CT_1) = \sum(month, day, year, hour, minute, seconds)$

$BASE64^{-1}(CT_1)$ should be equal to the sum of date and time stored in the database. If not, stop the process.

Step 3:

$\{[X_0, X_1 \ldots X_{n-1}, Med] - Med\}$

$X = \{[X_0, X_1 \ldots Med \ldots X_{n-1}]\}$

Place last or last two characters of CT_2 at exact centre if n is odd or even, respectively.

Step 4:

Restore original values for the 11 Unicode values of characters, if any, which were replaced since they don't have character representation. For e.g. if decimal code point was used replace characters representing values 289-299 with corresponding value from 127-129, 141-144, 149, 157, 158 and 160. We need not have a reverse process for

If $Unicode(Xi) < 33$
Then $Unicode(Xi) += 256$

Because the mod256 function nullifies the effect of change of values. Further reducing the vulnerability of the algorithm to relative attacks.

Step 5:

$Unicode(P_i) = (Unicode(X_i) - Unicode(K_i) - n - i) mod 256$

Original algorithm doesn't have 'i' in above formula because of which correct answer can never be obtained.
Repeat this step n-1 times. By getting corresponding characters from every repetition we obtain PIN in plain text. After comparing it with the PIN user entered access can be denied or granted.

Here is an example of encryption and decryption of CT_2 using actual algorithm and using algorithm with changes proposed in this paper:

PIN: egpin
KEY: egkey

Encryption:
Using original algorithm:
(Ignoring that the output for every character of PIN is of different length)

X: 207212226214240
CT_2: 207212262142402

Using this algorithm:
X: ÏÔâÖð
CT_2: ÏÔÖðâ

Decryption:
Using original algorithm:
CT_2: 207212262142402
X: 207212226214240
PIN: Can't be calculated because the system doesn't know whether its 20, 72,122…or 207, 21,222…and so on.

Using this algorithm:
CT_2: ÏÔÖðâ
X: ÏÔâÖð
PIN: egpin

As it is evident, this algorithm preserves the advantages of hash function[4] unlike original algorithm. But more importantly it is feasible and can be implemented that too effectively.

Step 6 (optional):
Receiving time - Sending time ≤ ΔT
Where ΔT is the legal time interval due to transmission delay, if not, then, transaction is not allowed to that user

C. Advantages

1. Every character of PIN is encrypted with different value of i (its position) thus variation in hash function.
2. Use of 256 different characters increases the number of combinations making it difficult to hack.
3. Even the relative frequency of occurrence of data will not help the cryptanalysts because the position of the plain text is used to convert into cipher text.
4. Availability of different Unicode options and unavailability of characters for certain values makes this algorithm less uniform and hence less prone to attacks.

IV. DEPLOYMENT IN SMART CARD

Once we generate a cipher text we need to insert it in smart card. Magnetic swipe cards have a magnetic strip at it back where data is stored. Magnetic swipe card writer is required to write data in it. A special will have to be made which can interact with this peripheral and writes our encrypted plain text. Currently, this algorithm has been implemented in JAVA swing. But to access smart card module it need to be implemented such that it is possible to interact with serial ports of a computer. This can be best done using JAVA libraries like javax.comm[6] or javax.smartcardio.
It should also be noted that anybody with smart card reader can access data and can also modify it using smart card write but this is illegal.

V. CONCLUSION

The explanation above shows that the originally proposed algorithm is not feasible to implement and with recommended changes it could provide effective service level security. Furthermore it is quite simple and suitable for high speed encryption applications. Non-uniformity is the backbone of its level of security.

Thus this cipher provides us with highly reliable, secure and fast option to store information in smart cards to tackle increasing forgery attacks.

VI. FUTURE ENHANCEMENT

As of now only one level of encryption is used. In future we can make this algorithm more secured by implementing two levels of encryption. This can be achieved by using secret key for level one encryption and using one more key as public key for level-two encryption. This public key can be stored on smart card itself as cipher text obtained using this algorithm is quite small.

REFERENCES

[1] L.M.Palanivelu, Latha Karthigaa.M, Balachandar.N, Karthi.M " 'n' - LEVEL UNICODE POSITION CHARACTER LENGTH CIPHERS FOR SECURING SMART CARDS", in the "Second International Conference on Computing, Communication and Networking Technologies (ICCCNT)" July 2010. Print ISBN: 978-1-4244-6591-0.
[2] Specifications for secure hash standard: http://www.csrc.nist.gov/publications/
[3] "Cryptography and network security – Principles and practices" by William Stallings, Third edition.
[4] "Information Security – Principles and practices" by Mark Stamp.
[5] http://www.rishida.net/tools/conversion/
By Richard Ishida, Internationalization Activity Lead at the W3C (World Wide Web Consortium) and contributes to the Unicode Editorial Committee.
[6] Chapter 11, "Java Cookbook" by Ian Darwin (August 2011). Chapter available online at :
http://java.sun.com/developer/Books/javaprogramming/cookbook/11.pdf

AUTHOR'S PROFILE

Natang A Salgia
born in Mumbai, India on 12th August 1990 is a student at Thakur College of Engineering and Technology, Mumbai. He is in final year of computer engineering and will be granted the degree of Bachelor of Engineering in July, 2012.