IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 5, May 2016

Enhancing Authentication and Encryption using Rave and Spectra Protocols

Ritika Gupta

Department of Computer Science, Quantum School of Technology, Roorkee, India

Abstract: This paper focuses on two new technologies in the IS sector. The Rave Authentication Protocol acts as the initiator of the Spectra encryption protocol. Both these protocols work seamlessly to authenticate BOTH the sending and receiving parties, and provide a high level of encryption to the passed information. It is to be noted, at the outset, that Spectra is NOT a new keying algorithm. It uses EXISTING CIPHER ALGORITHMS (on the lines of PGP) and runs them through the Key Amplification Engine which amplifies their key strength by as much as 32 times their original key length (For example, an 8-bit key can provide security equivalent to a 256 bit encryption), using the resources of the input key length (in this case resources used would be equivalent to an 8-bit key).
Rave and Spectra provide a very dynamic authentication and encryption mechanism which generates a DIFFERENT cipher text EACH TIME using the same keys. This allows these technologies to fit optimally in circumstances which require dynamic information encryption like mobile commerce (Also owing to the speed of the overall encryption process). The technologies are also highly scalable and can accommodate keys of any length and magnify their potential accordingly.

Keywords: Authentication, Cryptanalysis, Challenge- Response, Encryption, stream ciphers, Rave, Spectra, 3-DES, Smart Cards.

INTRODUCTION

The Rave and Spectra authentication and encryption protocols, are new technologies which aim towards plugging all the major loopholes that are generally exploited by cryptanalysts for deciphering secured communications. Rave and Spectra add another dimension into the encryption process which is known as the proximity, which primarily divides to text into various mutually determined blocks of different sizes before encrypting it with a particular sub-key hence if the proximity is wrong and the key is right, the secured text will still not be comprehendible by an adversary. The version of Rave and Spectrum explained here is geared towards the wireless financial security marketplace (Ex. Mobile Commerce) but the technology can be tweaked to fit into other domains as applicable.

Rave Authentication Protocol

This version of the Rave authentication protocol follows a variation of the challenge – response model and is currently designed to work on a client – server architecture only architecture only. The major advantage of Rave over other challenge – response based technologies is that rather than sending the corresponding response, generated from the challenge, back to the server, it utilizes it to initiate and configure the Spectra encryption protocol, i.e. it acts as a "Director" which decides how all subsequent data transfers would be encrypted for the current session. Another major advantage Rave possesses over other similar technologies is that the challenge and response do not share a 1:1 relation; i.e. the challenge cannot be directly worked upon in order to generate the corresponding response or vice versa (compared to technologies where the response is actually the encrypted challenge!). Rave is designed to work best on a 2 factor authentication system where a hardware token (smart card, SIM card, cell phone, etc.) is necessary in order to commence the authentication phase. This greatly reduces the risk of software key loggers acquiring and sending the user's passwords to the hostile user. Rave also requires a lot of random number generation for its operation; hence it would be necessary to note that the purity of the generated random number would greatly affect the overall security of the technologies.

Spectra Encryption Protocol

The Spectra encryption protocol works along with Rave to encrypt information pertaining to the particular session. It is important to understand, at the outset, that Spectra is NOT a new encryption key algorithm; instead it is a better way of using pre-existing keys which are widely used. In other words, Spectra is not an encryption key algorithm like the RSA or CAST; it is actually a protocol which uses these keys differently causing the overall security of the information to improve (For example, PGP is not a key algorithm; instead it uses standard keys like RSA, CAST, etc.). Another mentionable fact is that Spectra works optimally with **stream cipher** algorithms (Ex. RC4) or block cipher algorithm which do not pad the plain text to generate the cipher text (Stream ciphers are the preferred algorithms for telecom based application security. Hence their applicability for Mobile Commerce and thin client device encryption becomes further substantiated.) Spectra could be stated as being dimensionally different from existing encryption protocols; due to the fact that it adds another dimension into the encryption process. Current encryption technologies work only on 1D (one

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 5, May 2016

dimension), in other words, the only axis within the complete encryption process is the key that is encrypting the information, because Key + Plain Text = Cipher Text. Spectra adds another dimension within the encryption process, it is known as the **proximity**. Spectra divides the complete information into blocks of various sizes (Known as proximities), determined by the reaction generated by Rave, and then uses a different key on each of those blocks (also determined by the reaction generated by Rave). Therefore, in order to decrypt the document, you would require knowledge of the key used as well as the proximity in which the key was used.

Leading to the fact that, even if you have the right key but the wrong proximity, you will not be able to decrypt the information! The keys and proximities that would be under consideration by Spectra for that particular session would be decided by Rave at the time of authentication and hence the keys and proximities considered would be more or less different for each cryptographic session. As we proceed with examples of how Rave and Spectra operate, some of these, currently fuzzy, concepts will become clearer.

Rave and Spectra Symbiosis

The Rave and Spectra protocols work hand-in-hand. Rave determines how Spectra will encrypt data. The following portion of the document displays how Rave and Spectra authenticate and encrypt the communication session between the client and the server (the following examples are made to fit the mobile device and smart card transaction scenarios only; other implementations may vary) …

The Registration Process

Step 1 Registration

Before proceeding to use Rave and Spectra, a user has to be registered to the server so that he could be granted the credentials that would be needed to authenticate and encrypt all subsequent transmissions. The following steps show the registration process…

The user will be given a smart card or a mobile SIM card which will contain, within it, the following entities.

1) The Universal Identification Number (UID).
2) The personality.
3) The current mood.
4) The user's Key Chain.

The Universal Identification Number

The Universal identification number will be a unique number which would be granted to each smart card or SIM card (or any other 2^{nd} factor token, inclusive of downloadable mobile applications).

The UID is not to be confused with the credit card number; it will be, basically, used to identify the following statistics about the user.

a) His country of origin
b) His bank
c) His username.

<Country Code> **:** *<Bank code>* **:** *<UserId>*
Example: **091:12345:2133874787**

Exhibit 1: The UID Number

Personality

The personality is a reference table which will be assigned to each user upon registration. The personality resembles a conversion table which has a numeric representation of all characters from A → Z, a → z, 1 → 0, and the character period (.). It is to be observed that the allocated value should be a non – repeating number from 70 → 141. An example of one such personality is shown in Exhibit 2 below…

A B C D E F G H I J	136 140 137 88 135 99 109 116 101 124	K L M N O P Q R S T	134 79 115 138 125 130 98 132 126 108	U V W X Y Z a b c d	133 127 97 139 114 131 128 90 105 117	e f g h i j k l m n	123 89 110 93 81 82 100 80 102 111	o p q r s t u v w x	94 106 121 83 112 95 107 84 122 96	y z 1 2 3 4 5 6 7 8	113 85 119 91 118 86 103 87 120 92	9 0 .	104 129 141

Exhibit 2: The Personality Table

It is not necessary that each user will have a unique personality but the likelihood of a personality repeating itself will be after $3*10^{85}$ users (In an ideal situation)! Which is a number long enough to be considered unreachable in the foreseeable future. Taking one character to be 1 byte in size and given the fact that the numbers 79 → 141 lie within the unsigned short range, which, in turn, occupies 2 bytes of memory, the complete table would be 186 bytes in length. The personality table is an integral part of the Rave authentication process.

The Current Mood

The mood is perhaps the only dynamic entity within Rave. The Mood is, also, a small table which assists Rave in depicting the Spectra directors from the generated reactions. The Spectra directors, as explained earlier, contain the various proximities and the index numbers of the sub-keys which would be used to encrypt the communication channel. Although, at this point, it would be difficult to understand what the contents of this table mean, I would describe the structure of the table, so that you can refer to it when we get to its application.

Scenario	Product Style
00	→
01	3→
10	←2
11	←

Exhibit 3: Mood Table

The User Key Chain

The user key chain consists of the subkeys of the included key. In our example we are using an 80 bit key divided

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 5, May 2016

into 10, 8-bit subkeys.

These 10 subkeys would constitute the key chain.

Step 2 *Registration*

Once the pre-requisites have been programmed in the card chip, the card is allotted to the new user. The issuing authority will also include in the card an initial spectrum (proximity + Keys) along with the second factor device (referred to as token from now onwards). At this point the user can set the password of his choice. This password will be used by the Rave authentication protocol for creating the Spectra director which will determine how data communication between the client and the server would be encrypted for the current session. Only capital letters, small case letters, numbers and the period (.) should constitute the password (A →Z, a → z, 1 → 0, or period) [The period character is, generally, the most conveniently type-able character in cell phones.]

The registration stage is the ONLY time the password will travel through the network in an encrypted state decided by the initial spectrum present on the token when it is issued. Once the desired password reaches the server, the following steps are taken…

1) The user's corresponding Situation Table is generated.
2) The next mood table is randomly selected from the mood bank and sent to the client. A copy of the associated mood table is also stored in the server.
3) All instances of the user's password are eliminated from the server's memory.

The Situation Table

The situation table helps generate the challenge (henceforth known as the situation) which would be sent to the user to initiate the authentication process. The situation table consists of three columns (See figure below)

Serial Number	Character Set	Sum Of Products (SOPs)

Exhibit 4: The Rave Situation Table Schema

Serial Number: This column will contain the serial number of the corresponding row containing the character set and the SOP.

Character Set: This column will contain the position numbers of the password's characters which have to be extracted for inspection. The character set would contain 10 numbers within 1 → l, where l is the length of the password (In case of mobile commerce implementation we can limit it to 10. The longer the password, the more secure the system becomes!), distributed in random order with a single such number not repeating more than twice. It is also to be understood that all the numbers from 1 to 10 may or may not be accommodated on a single character set. A character set will resemble the example given below.

2537116890 (VALID)
6487215999 (INVALID)
1234567890 (VALID BUT UNLIKELY)

Sum of Products (SOP): The SOP column will contain the sum of the products of the five pairs of numeric equivalents of the password's characters considered in the corresponding character set. Supposing n' is the numeric equivalent of the character (obtained from the personality table) present at the n^{th} position of the password. Then taking the VALID character set example displayed in the "Character Set" column (2537116890), the SOP will be denoted by…

$$(2'*5') + (3'*7') + (1'*1') + (6'*8') + (9'*0').$$

Consider the following example. Suppose the user's desired password is "Im Lovin. IT". If we divide this password into individual characters and look up the corresponding conversion in the personality table, we will obtain the following…

Character	I	m	L	o	v	i	n	.
Position	1	2	3	4	5	6	7	8
Conversion	101	102	79	94	84	81	111	141

Exhibit 5: Sample Password Conversion Chart

Therefore, for the character set 2537116890 the considered characters would be "m, v, L, n, I, I, i, ., I, T" the corresponding SOP would be calculated by:-

$$(102*84) + (79*111) + (101*101) + (81*141) + (101*108)$$
or,
$$(8568) + (8769) + (10201) + (11421) + (10908)$$

Hence the SOP will be **49867**

For added security, the SOP column is the only column that is going to be stored in an encrypted form within the server. Here we can use any conventional symmetric key mechanism for encryption rather than Spectra. To further enforce security, the server can throw away the personality of the user once the user registers, but doing so will introduce some overhead in case the user wishes to change his password at a later date.

Generating this table every time a user registers would pose a large overhead on the registration server. In order to increase efficiency, a pre- generated pool of numerous such tables would be available within the registration server. Whenever a user registers, one table from the pool will be taken and allocated to the user. The only calculation needed at such a scenario would be for generating the SOPs.

Once the table generation concludes, a new mood will be sent over to the client. The new mood is decrypted and stored inside the token. All instances of the password are eliminated from the server.

DOI 10.17148/IJARCCE.2016.55129

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 5, May 2016

This concludes the registration session.

The Login Process

Once the registration process concludes, the user is ready to make transactions or communicate to the server using Rave and Spectra. During the explanation of the login stage, many of the concepts discussed during the registration process would get clarified. The login process proceeds over the following steps.

Step 1 *Server Side*

The user initiates the authentication process by sending a login request to the server (in some M- Commerce modules the login request may be initiated by the server). The user's username (see exhibit 1) is sent to the server along with the request. The server fetches the corresponding situation table for the user.

Step 2 *Server Side*

The server generates 6 random numbers between 1 and n (where n is the serial number of the last situation of the table. The number of entries in the situation table can be defined on implementation.). The server will then fetch the character sets of the situation table corresponding to the randomly generated serial numbers. At this moment the server will have 6 character sets. In order to decide the spectrum (the entity which determines where the proximities lie and which keys are going to be used to encrypt them), we require four Critical Mass Character Sets (CMCS). Hence the CMCS pairs are going to be fetched; in totality, 12 character sets (6 CMCS) are going to be used to create 3 spectrums. The CMCS are fetched by finding the match within the Personality Table by scanning DOWNWARDS from the generated random serial number, in case the randomly generated serial number is the last in the personality table, then the counter will loop back to the beginning. The order in which the CMCS are fetched and later sent to the client is extremely important. For implementation purposes, it is recommended that the CMCS are grouped by the order they were fetched. For example if the random numbers that were generated were 23,50,11,31,35,6; then the CMCS that were created were:

23 → 2511748940 & 4473660051
50 → 2283950284 & 5147663490
11 → 5748396758 & 6837198600
31 → 1758299657 & 6748210799
35 → 1966092852 & 5637212883
06 → 7281096837 & 3893567244

Critical Mass Character Sets

Recollecting from the previous section where we were discussing what the character sets will consist of, it was said that it was highly unlikely that a character set will consider all the characters of the password. In this situation Rave will be extremely prone to password guessing attacks. Let us re- examine the character set "2537116890". If we notice carefully, the character at position "4" is not considered. So if the original password is "ImLovin.IT" and we replace the character at 4th position which is the character "o" with any other character, say, "p" then the server will still grant access because the character "o" was not being considered by this character set! To overcome this problem the concept of "Critical Mass Character Sets (CMCS)" was introduced. Critical Mass Character Sets are a pair of character sets which, when combined, consider ALL characters of the password. In the example stated above the CMCS would consist of the following character set pair.

2537116890 ← The Number "4" is not a part of this character set.
1563884932 ← The Number "4" is a part of this character set.

Following are other examples of CMCS…

6448277662 & 5320118499 (Characters at position 1,3,5,9,0 were not considered in initial set.) 5382967385 & 5724760173 (Characters at position 1,4,0 were not considered in initial set.)

Step 3 *Server Side*

Once the CMCS are extracted, the server will have to create the spectrums in order to initiate the communication session with the client. The spectrums are created by multiplying the SOPs of the corresponding CMCS according to the current mood. Consider the above given character sets, their corresponding SOPs (according to the personality provided in exhibit 2) are demonstrated below.

Character Set	*SOP*
2511748940	53596
4473660051	44314
2283950284	46397
5147663490	43813
5748396758	51392
6837198600	53476
1758299657	50862
6748210799	54736
1966092852	50620
5637212883	51396
7281096837	56661
3893567244	46080

Exhibit 6: POS of fetched CMCS

In order to calculate the spectrum, we would have to understand what the mood does. Let's take, for example, the first CMCS (2511748940 & 4473660051); whose respective SOPs are, 53596 & 44314. To obtain the spectrum we will have to multiply these SOPs together according to the mood. The mood provides the method by which the SOPs would be multiplied by referring to the Least Significant Digit (LSD) of the individual SOPs. For 53596 the LSD is "6" and for 44314 the LSD is "4". Refer to exhibit 3, the first column denotes the scenario of the LSDs of the SOPs corresponding to the CMCS. An

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 5, May 2016

"Even" LSD is denoted by a "0" and an "Odd" LSD is denoted by a "1". Hence if the LSDs of both the SOPs are even then they are denoted by "00" and if they are both odd then they are denoted by "11" in case any one is odd then they are denoted by a "10" or "01" accordingly. The second column of the mood denotes the Product Style. The product style is the arrangement of the second SOP before they are multiplied together. The mood is designated by a number followed by a forward or backward arrow. The number represents the numeric position to the LEFT of the LSD. In case there is no number mentioned then transformation of the second SOP is initiated by the LSD itself. This is further clarified by this example.

Let's take the first CMCS from the table in exhibit 6 viz. 2511748940 & 4473660051; their respective SOPs being 53596 & 44314. As mentioned earlier, their corresponding LSDs are "6" and "4" respectively; hence the scenario becomes "00". The product style corresponding to this scenario is given as a forward arrow without any numeric value preceding it or "→" hence the second SOP (44314) is re-arranged by rotating clockwise from the LSD, post transformation, the new value becomes "44431". Taking the second character set (7281096837 & 3893567244) along with their corresponding SOPs (56661 & 46080) and observing their LSDs, the scenario becomes "10", referring to exhibit 6 the suggested product style is "←2" hence the second SOP is rendered "06408". Further examples of this concept are displayed in exhibit 7 below.

Scenario	Example SOPs	Style	New Operands
00	34522, 23450	→	34522 * 02345
01	51338, 32879	3→	51338 * 28793
10	44691, 32456	←2	44691 * 42365
11	27877, 43719	←	27877 * 91734

Exhibit 7: Mood Explanation

Step 4 *Server Side*

The above obtained operands are multiplied together to derive the product. Two such products make a Spectrum.

Spectrum

The spectrum is the most important concept in Spectra. The Spectrum denotes the proximities the plain text is divided into, as well as, the keys which are going to be used on the proximities to encrypt the plain text. Let's refer back to exhibit 6 which displays a list of CMCSs we are frequently using in our examples. Let's take the 1st and the 2nd CMCSs (2511748940 & 4473660051, 2283950284 & 5147663490) and their respective SOPs are 53596 & 44314, 46397 & 43813. And hence their products become, (53596 * 44431 = 2381323876, 46397 * 31834 = 1477002098). The first obtained product (2381323876) denotes the proximities of the plaintext and the second obtained product (1477002098) denotes the serial number of the 10 keys which will encrypt the corresponding proximity of the plaintext. To further clarify, let's take the following example.

By observing the generated Spectrum (2381323876, 1477002098) the encryption/decryption would be brought about in the following manner…

1) The token will form 10 blocks in its memory (Each block should preferably be a two dimensional array containing one column and rows adjustable to the length of the information that needs to be encrypted. Each field should accommodate 10 characters.)
2) The arrays will be populated according to the proximities decided by the current spectrum…

First 2 bytes will be stored in Block 1 row 1; Next 3 bytes will be stored in Block 2 row 1; Next 8 bytes will be stored in Block 3 row 1; Next 1 byte will be stored in Block 4 row 1; Next 3 bytes will be stored in Block 5 row 1; Next 2 bytes will be stored in Block 6 row 1; Next 3 bytes will be stored in Block 7 row 1; Next 8 bytes will be stored in Block 8 row 1; Next 7 bytes will be stored in Block 9 row 1; Next 6 bytes will be stored in Block 10 row 1;

Moving cyclically,
The next 2 bytes will be stored in Block 1 row 2; Next 3 bytes will be stored in Block 2 row 2; Next stored in Block 10 row 2;

1) This cycle will continue till the EOM (End of Message) is reached. In case the number of characters left in the message is lesser than the number of characters needed to be considered according to the proximity, the remaining characters are stored in the designated block regardless of the length.
2) The characters of each block are concatenated together and then encrypted by the corresponding key. Considering the Spectrum stated above, the divisions would 8 bytes will be stored in Block 3 row 2; Next 1 byte will be stored in Block 4 row 2; Next 3 bytes will be stored in Block 5 row 2; Next 2 bytes will be stored in Block 6 row 2; Next 3 bytes will be stored in Block 7 row 2; Next 8 bytes will be stored in Block 8 row 2; Next 7 bytes will be stored in Block 9 row 2; Next 6 bytes will be be (A plus (+) symbol representations concatenation).

(B1 [Row 1] +B1 [Row 2] + B1 [Row 3] +
B1 [Row 4]…) is encrypted by key # 1
(B2 [Row 1] +B2 [Row 2] + B2 [Row 3] +
B2 [Row 4]…) is encrypted by key # 4 (B3 [Row 1] +B3 [Row 2] +B3 [Row 3]
+B3 [Row 4]…) is encrypted by key # 7 (B4 [Row 1] +B4 [Row 2] +B4 [Row 3]
+B4 [Row 4]…) is encrypted by key # 7 (B5 [Row 1] +B5 [Row 2] +B5 [Row 3]
+B5 [Row 4]…) is encrypted by key # 0 (B6 [Row 1] +B6 [Row 2] +B6 [Row 3]
+B6 [Row 4]…) is encrypted by key # 0 (B7 [Row 1] +B7 [Row 2] +B7 [Row 3]
+B7 [Row 4]…) is encrypted by key # 2 (B8 [Row 1] +B8 [Row 2] +B8 [Row 3]
+B8 [Row 4]…) is encrypted by key # 0 (B9 [Row 1] +B9 [Row 2] +B9 [Row 3]
+B9 [Row 4]…) is encrypted by key # 9 (B10 [Row 1] +B10 [Row 2] +B10 [Row 3]
+B10 [Row 4]…) is encrypted by key # 8

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 5, May 2016

3) The individual cipher texts are then divided into their various proximities and stored back in same rows they were initially concatenated from.
4) Finally, the complete cipher texts are concatenated to form a single stream of cipher text. In other words, … B1[Row 1] + B2 [Row 1] + B3 [Row 1] + … + B10[Row 1] + B1[Row 2] + B2 [Row 2] +… + B10[Row 4]
5) This cipher text is encrypted 2 more times (in our examples we are generating 3 spectrums hence the plain text will undergo 3 passes from the engine to render the final cipher).

Step 5 *Server Side*

The CMCS sent to the client would hence be:
25117489404473660051228395028451476634905 7
48396758683719860017582996576748210799196 6
09285256372128837281096837389356724 4
(Total size = 120 bytes, in characters).

The CMCS transmitted to the client for authentication are collectively known as a "Situation" This ends the server side process. Once the server side processes are understood then the client side is easy to understand because most of the processes are the same.

Step 6 *Server Side*

The CMCS sent to the client would hence be:
25117489404473660051228395028451476634905 7
48396758683719860017582996576748210799196 6
09285256372128837281096837389356724 4
(Total size = 120 bytes, in characters).

The CMCS transmitted to the client for authentication are collectively known as a "Situation" This ends the server side process. Once the server side processes are understood then the client side is easy to understand because most of the processes are the same.
This ends the server side process. Once the server side processes are understood then the client side is easy to understand because most of the processes are the same.

Step 1 *Server Side*

The CMCS are received and the Spectrums are calculated exactly the way they were calculated by the server.

Step 2 *Server Side*

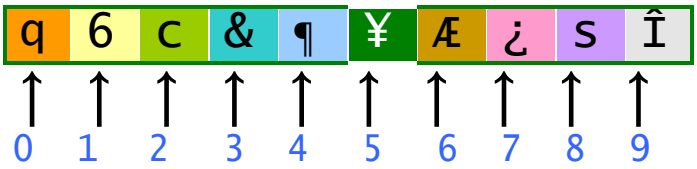
The entire information to be communicated will be encrypted by Spectra. In the manner demonstrated below… (A space is denoted by <SP> and all non-printable characters are denoted by <SC> (or in white font over black background)).

<BOM>-----<SP>TransBegin<SP>-----

MerCode:0910067576\\Crncy:USD\\Amt:25.99\\Name:Harko<SP>Robroch\\CC#MC-5472-7769-2448-7563\\Exp:10/31/2020

Exhibit 8: Example Transaction (Actual Implementation may vary)

Assuming that the 10, 8-bit keys are given as under…



The following couple of pages demonstrate the encryption process followed by Spectra for encrypting the above given plaintext…

Loop 1:

CMCS:	2511748940	4473660051	2283950284	5147663490
SOPs:	53596	44314	46397	43813
Spectrum:	**Proximity**		**Keys**	
	2381323876		1477002098	

(Note: The total size of the array stored in the token's memory would be 400 bytes.)

--	---	<SP>TransBe	g	in<SP>	--	---	MerCode:	0910067	576\\C
rn	cy:	USD\\Amt	:	25.	99	\\N	ame:Hark	o<SP>Robro	ch\\CC
#M	C-5	472-7769	-	244	8-	756	3\\Exp:1	0/31/20	20----
-<SP>	Tra	nsEnd<SP>--	-	--					
↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
Key 1	**Key 4**	**Key 7**	**Key 7**	**Key 0**	**Key 0**	**Key 2**	**Key 0**	**Key 9**	**Key 8**

Exhibit 9: Plain Text divided into proximities and acted upon by keys. (Pass 1)

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 5, May 2016

This cycle will run 3 times for the 3 Spectrums created The numbers of spectrums created can be defined upon implementation, the more the spectrums, the higher the security!)

How the text will be further encrypted is depicted by the two tables below…
Resulting Cipher Text from Loop 1:

aB–„B²< £7óZõ%µaBE ¼µ<SC>M*à#&Ÿ0aó]Ïu ßrðZ<SC>-M‡åc¤û–(avó[ºi-PðÏféjxÇ]íØA‡Õý…Ú²^ ÜÂ·¬<SC>N<SC>
Zƒ#ð÷„A rüq‘¼·j‰ô’tôÅÉÂ À·z‰}<SC>ÉÊú kÜ7n…"‹<SC>äÊ {pÍ<SC>K*0

The thus obtained cipher text will be divided according to the proximities of the next spectrum and then encrypted by the keys suggested by the next spectrum.

Loop 2:
(The actual "Key Half" of the Spectrum was calculated to be 3330087726, but since we are using stream ciphers, a character encrypted by the same key twice, decrypts it7, hence in case any key index matches with the prior Spectrum's key index, the latter's key index is incremented by "1". In the above example, the 5th place index was zero for both the Key halves of the spectrum, hence the latter's clashing key index was incremented by "1" making it 0 + 1 = 1.)

aB–	„B²	<£	µ7	óZõ%µaB	E¹÷	µ	M*	à#&Ÿ0aó]Ïu	ß	rðZ
-M<	‡å	c¤û–(		avó[ºi–	Pðï	f	éjx	Ç]íØA‡Õý…	Ú	²^ Ü
Â·¬	N	Zƒ#ð÷		„A rüq‘¼	·j‰	ô	’tô	ÅÉÂ À·z‰}	É	Êú
kÜ7	n…"	‹] äÊ		{pÍ	K*0					

Key 3 Key 3 Key 3 Key 0 Key1 Key 8 Key 7 Key Key 2 Key 6

Exhibit 10: Cipher Text from Pass 1 divided into proximities and acted upon by keys. (Pass 2)

Resulting Cipher Text from Loop 2:

sà'-ày<µ·Á·¿5ÊLŽX"ç(ç(ë˜„ûrê÷ù_%B(ù,{ögOùfVÍ|ª˜ž‡¾@“ éq ¹6Ã,‡¹ #<|<xÜ™øZj¨Á?_Òv<Op“Ø¡¨‡õ
õÖ6gÃéŠþF!k– XaÉ2#KÂv¿çk©<õq-I™áéÍç ^›A[ÂT Æ'Ë“†¹lÂV4U:

Pass 3:

CMCS:	1966092852	5637212883	7281096837	3893567244
SOPs:	50620	51396	56661	46080
Spectrum:	**Proximity**		**Keys**	
	3297336180		464183688	0

<SC>sà' éq aÉ2	–à #K	y<µ·Á·¿5Ê 6Ã,‡¹ #< Âv¿çk©<õq	LŽX"ç(ç | <xÜ™øZ –I™áéÍç	(ë˜ j¨Á ^›	„ûr ?_Ò A[Â	ê÷ù_%B v< Op“Ø T Æ'Ë“	(¡ †	ù,{ögOùf ¨‡ õ õÖ6 ¹lÂV4U:	VÍ|ª˜ž‡ ¾@“ gÃéŠþF!k—X

Key 4 Key 6 Key 4 Key 1 Key 8 Key 3 Key 6 Key 8 Key 8 Key 0

Exhibit 11: Cipher Text from Pass 2 divided into proximities and acted up by keys. (Pass 3)

Step 3	Client Side

The obtained individual ciphers are recombined and sent to the receiving entity:

<BOM>
ÉIHƒ_¸&n¾ h²[¥!m‡õ ú□Â®^–
YÂÿHZ3● SÂ Ç½ð1š(P·¢CÃ]Ù[·È"¶ˆ®°brÏµ|Æª™¾þ|ik•"]lþ¶ÿß©·ßlG|}äé<tWõÍCpz>¶–
Tlfãô lî]µZar¹®û€?…·{BŒ–âxøÌhÚ[àÆØ¬ø ¡ óõv^\<aó
<EOM>

Exhibit 12: Final Cipher Text sent back to receiving entity

DOI 10.17148/IJARCCE.2016.55129

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 5, May 2016

Step 6 *Server Side*

Upon receiving the cipher text from the client, decryption will happen in the reverse order of encryption, for example, Pass 3 of encryption will become Pass 1 of decryption and so on.

Step 7 *Server Side*

Once the server successfully decrypts the cipher text obtained by the user, it will perform the necessary actions to complete the requested transaction. Before the current transaction session is closed the server will send the user another, randomly chosen mood (from the mood bank), encrypted by the currently used spectrums.

The thus obtained plain text will authenticate the user. Once the identity is confirmed, the acquiring agency will perform the necessary transactions the way the credit card company currently does.

In case the produced plain-text does not match the required format and syntax, or, let's just say the server cannot successfully decrypt the cipher-text to its corresponding plaintext, then it is obvious that there is a mismatch between the spectrums generated by the server (used for decrypting) and the ones the client generated to encrypt the plaintext. A mismatch in the spectrum implies that the user is not authentic. A mismatch could result from any one of more of the following…

- Wrong Token.
- Wrong Password
- Wrong Mood

In case of a mismatch the server can give the client 3 more tries, the CMCS sent to the user will remain within the client token till a successful transaction is performed or if the server decides to scrap the bogus session altogether. It is up to the implementer to decide how many tries are to be granted to the user before the user's account is frozen. In an event of an account freezing, the user can walk to the issuing bank's branch to "Unfreeze" the account. The account is unfrozen by granting a new mood/personality or change of password (Implementer's decision).

Step 4 *Client Side*

The client will receive the mood sent by the server, decrypt it and store it. The next session's spectrums would be hence dependant on the newly allocated mood.

Scenario	Style
00	4→
01	←2
10	→
11	←1

Exhibit 13: Example of the new mood assigned to the client.

This concludes the current transaction. The session will close and the currently used CMCS as well as Spectrums would be deleted from the token's memory. In order to increase security, we can keep the *situation* from the current transaction which will be used to create the required spectrums to help decrypt the next set of situation that we will receive for the next transaction. This will ensure that the situation travels the wire in an encrypted form, causing the hostile user to face another step of inconvenience in trying to acquire the situation off the wire.

CONCLUSION

- Rave and Spectra enhance the authentication and encryption processes that are currently followed by many of the major authentication and encryption schemes. The key features of Rave are mentioned below… The challenge and response do not share a 1:1 relationship.
- The response to the challenge is never sent back to the server.
- There is no direct relationship between the Password and the underlying keys, unlike current smart card / token based technologies where the PIN or password merely unlocks the underlying private key.

Salient feature of Spectra are as follows…

- Works on two dimensional encryption architecture by the introduction of the proximity. Spectra, being two dimensional protects the user even in case his keys are compromised (See Appendix A for further details)
- Spectra can also fit optimally in scenarios involving electronic payments like Electronic or mobile based commerce by providing security even in case the card credentials are stolen.
- Spectra allows user sharing the same symmetric keys to still safely exchange exclusive information with each other without the other members of the group accessing it.
- The cipher text produced by Spectra will be different for each session. Unlike other technologies, if we take a plain text **'P'** and encrypt it with Key **'K'**, we will ALWAYS get cipher text **'C'** which is NOT TRUE in case of Spectra. The cipher text for each session will be rendered differently.

Rave and Spectra are extremely flexible in their design. They can work with block ciphers as well as stream ciphers based on either PKI or SKI. Their passes can be increased or decreased as per the security requirements of the implementer. For example, if required, we can save the previous session's situation which can be used to encrypt/decrypt the next session's situation for added security. Rave and Spectra also increase the brute force cycles of when executed on a key of a particular size. Various implementations of Spectra have a different security enforcement level in order to suite its application domain. For example in our WiFi implementation, we

DOI 10.17148/IJARCCE.2016.55129

actually initiate Spectra by encrypting a randomly generated string of 256+n bytes before each transmission. Here the additional number 'n' is actually the sum of the LSDs of the proximity half and the keys half of all the spectrums that are generated for the complete Spectra procedure. For more clarification refer to the example given below...

Consider the various spectrums from page 8 - 10 of this white paper...

Here, the proximity and keys halves of the 3 spectrums for the 3 passes of encryption/decryption processes are:-

2381323876; 1477002098:: LSDs = 6&8 :: 6+8 = 14
3358313024;3330187726 :: LSDs = 4&6 :: 4+6 = 10
3297336180;4641836880 :: LSDs = 0&0 :: 0+0 = 00

Hence 'n' = 14 + 10 + 00 = 24 Therefore, the initial stream which is encrypted by Spectra BEFORE the actual information to be encrypted will be 256 + 24 = 280 bytes long. Once this "Dummy" stream is encrypted it is discarded and NOT sent to the receiving entity, but the LFSR state of RC4 encryption engine is maintained for the complete session, making RC4's pseudo random generator near pure and very difficult to predict. Doing so prevents Spectra from being exploited by the WEP attacks as narrated by Adi Shamir in one of his latest papers exploiting the weaknesses of the 802.11 security protocols widely used in WEP ("Weaknesses in the Key Scheduling Algorithm of RC4", Proceedings of Selected Areas in Cryptography 2001, SAC'01, LNCS vol. 2259, pp. 1-24, Springer-Verlag, 2001.) The next cryptographic session would require the recreating of a new "Dummy" string by the method mentioned above BEFORE encrypting the actual information. This eliminates the necessity of the client and server being in sync. before every cryptographic session.

An added feature using the above mentioned technique is that the malicious user would also have to figure out the value of 'n' BEFORE he can try his luck in finding the keys and the proximities thus increasing the security manifold.

REFERENCES

[1] P. Rogaway and D. Coppersmith, “A Software-Optimized Encryption Algorithm”, Proceedings of the 1993 Cambridge Security Workshop, Springer-Verlag, 1994.
[2] B. Schneier, Applied Cryptography, Second Edition, John Wiley & Sons, 1996.
[3] S. Vaudenay, “Statistical Cryptanalysis of Block Ciphers -_2 Cryptanalysis”, 1995.
[4] R. Ferreira, “The Practical Application of State of the Art Security in Real Environments”
[5] Petros Mol and Stefano Tessaro,“Secret-Key Authentication Beyond the Challenge Response Paradigm: Defination al Issues and New Protocols”
[6] Chris Mitchell, “Limitations of Challenge Response Entity Authentication”
[7] Network Security and Cryptography.

DOI 10.17148/IJARCCE.2016.55129