Prime Encryption Algorithm (P.E.A)

1. Brief Introduction:

The Prime Encryption Algorithm, hence known as P.E.A is based on large primes in the range of 10000 to 11000.

Based on the combination input of Username/ID & Password provided by the user the encryption happens and the encrypted value is updated in the database. Hence even for the decryption process both the inputs are validated to obtain the decryption key and proceed with the login.

This process of key generation from Username/ID & Password, the encryption process and the decryption process will be explained in detail.

2. Algorithm Explanation:

The P.E.A is developed to tackle the problem of password encryption in the database or repository. The encryption takes place with the help of a generated key from the combination input of Username/ID & Password.

Consider the below information to explain the encryption process in detail. I have taken an email address as the Username/ID for the illustration purpose.
Username/ID: *runcy_oommen@mcafee.com*
Password: *MVS10@web*

The whole process of encryption starts by obtaining a **Key Number** in the range of 10000 to 11000. This number is obtained from the Username/ID used for login. The range mentioned above can be raised to any extend which provides that added flexibility to this algorithm. The lower range of a prime having 5 digits is chosen here for illustration. It can be remembered that, the more higher number is chosen as the base for encryption the more computing resource is needed to generate higher order primes which will impact in the speed at which the encryption is done.

The methodology used in obtaining the Key Number is explained below:

a. Obtaining the Key Number:

The logic used in obtaining the Key Number is pretty simple. A **Generic Number Table (GNT)** is created which contains a numeric value for each and every character in the keyboard. The number assigned to each of the key is just in a numerical sequence and no specific logic is followed.

It is not a hard and fast rule that the number assigned be in a numerical sequence starting from 1. It can be primes, squares, cubes or even ASCII codes for each character in the keyboard, which makes it totally flexible and help in the internationalization process.

A representation of the Generic Number Table with sequential numbering is shown below:

a	1	**A**	27	**1**	53	@	63
b	2	**B**	28	**2**	54	_	64
c	3	**C**	29	**3**	55	**.**	65
d	4	**D**	30	**4**	56	**!**	66
e	5	**E**	31	**5**	57	#	67
f	6	**F**	32	**6**	58	**$**	68
g	7	**G**	33	**7**	59	**%**	69
h	8	**H**	34	**8**	60	**^**	70
i	9	**I**	35	**9**	61	**&**	71
j	10	**J**	36	**0**	62	*****	72
k	11	**K**	37			**(**	73
l	12	**L**	38			**)**	74
m	13	**M**	39			**-**	75
n	14	**N**	40			**+**	76
o	15	**O**	41			**=**	77
p	16	**P**	42			**{**	78
q	17	**Q**	43			**}**	79
r	18	**R**	44			**[**	80
s	19	**S**	45			**]**	81
t	20	**T**	46			**|**	82
u	21	**U**	47			****	83
v	22	**V**	48			**<**	84
w	23	**W**	49			**>**	85
x	24	**X**	50			**,**	86
y	25	**Y**	51			**?**	87
z	26	**Z**	52			**/**	88
						"	89
						'	90
						~	91
						`	92

The Username/ID used here as an illustration is
Username/ID: *runcy_oommen@mcafee.com*

The numeric equivalent of this Username/ID after substitution from the GNT, known as the **User Number** is given below.
User Number: *1821143256415151313514631331655653151 3*

From this User Number the Key Number is obtained, which will be the base for encryption. As a convention here the Key Number needs to be in the range of 10000 to 11000. Remember that this range can be changed according to the encryption strength needed. Split is done for the User Number by 5 digits starting from the left to right.

The numbers are given below:
18211 Part I
43256 Part II
41515 Part III
13135 Part IV
14631 Part V
33165 Part VI
56531 Part VII

513 Ignored
The last number (513) is ignored, as it is less than 5 digits.

Step 1:
Taking Part I and Part II of the above split we notice that Part I < (lesser than) Part II
i.e. 18211 < 43256
Therefore 43256 - 18211 = 25045

Step 2:
Now taking 25045 and Part III we notice that 25045 < Part III
i.e. 25045 < 41515
Therefore 41515 - 25045 = 16470

Step 3:
Now taking 16470 and Part IV we notice that 16470 > (greater than) Part IV
i.e. 16470 > 13135
Therefore 16470 - 13135 = 3335

Step 4:
Now taking 3355 and Part V we notice that 3355 < Part V
i.e. 3335 < 14631
Therefore 14631 - 3335 = 11296

Step 5:
Now taking 11296 and Part VI we notice that 11296 < Part VI
i.e. 11296 < 33165
Therefore 33165 - 11296 = 21869

Step 6:
Now taking 21896 and Part VII we notice that 21896 < Part VII
i.e. 21869 < 56531
Therefore 56531 - 21869 = 34662

This final number also does not fall in the range of 10000 to 11000. Hence it needs to be further diminished to bring it to the desired range. As a special case, if a final value having less than 5 digits is obtained zeros are added at the end to make it into 5 digits.

Now getting back to the example, 34662 need to be diminished by 20000 resulting in 14662. This number also does not fall in the desired range and hence it is diminished again by 4000 resulting in 10662. This falls in the desired range level of 10000 to 11000

But the number obtained (10662) is not a prime number and the next prime number after this is 10663.

Thus **10663** is the Key Number.

b. Creating the Table of Primes (ToP):

Just like the GNT created for the Username/ID used for deriving at the Key Number, **ToP** is created based on the obtained Key Number.

The difference between these two tables lies in the fact that the Generic Number Table is just a sequential numbering (or whatever applied logic) of all the characters in the keyboard (as illustrated for this example) and does not change the ToP is created dynamically based on the derived Key Number.

The Table of Primes created in this case with Key Number being 10663 is shown below:

a	10663	**A**	10909	**1**	11171	**@**	11273
b	10667	**B**	10937	**2**	11173	**_**	11279
c	10687	**C**	10939	**3**	11177	**.**	11287
d	10691	**D**	10949	**4**	11197	**!**	11299
e	10709	**E**	10957	**5**	11213	**#**	11311
f	10711	**F**	10973	**6**	11239	**$**	11317
g	10723	**G**	10979	**7**	11243	**%**	11321
h	10729	**H**	10987	**8**	11251	**^**	11329
i	10733	**I**	10993	**9**	11257	**&**	11351
j	10739	**J**	11003	**0**	11261	*****	11353
k	10753	**K**	11027			**(**	11369
l	10771	**L**	11047			**)**	11383
m	10781	**M**	11057			**-**	11393
n	10789	**N**	11059			**+**	11399
o	10799	**O**	11069			**=**	11411
p	10831	**P**	11071			**{**	11423
q	10837	**Q**	11083			**}**	11437
r	10847	**R**	11087			**[**	11443
s	10853	**S**	11093			**]**	11447
t	10859	**T**	11113			**|**	11467
u	10861	**U**	11117			****	11471
v	10867	**V**	11119			**<**	11483
w	10883	**W**	11131			**>**	11489
x	10889	**X**	11149			**,**	11491
y	10891	**Y**	11159			**?**	11497
z	10903	**Z**	11161			**/**	11503
						"	11519
						'	11527
						~	11549
						`	11551

The Password used here as an illustration is
Password: *MVS10@web*

The **Encrypted Password** obtained from ToP is given below
Encrypted Password: *11057 11119 11093 11171 11261 11273 10883 10709 10667*

N.B: (The *SPACE* given between the numbers in the Encrypted Password is just for easy identification and illustration purpose only)

c. Determining the Number of Iterations:

The **Number of Iterations** referred here is the number of times the specified encryption technique is applied for password encryption. The default minimum number of iterations to be performed on a password as per this algorithm is set as 5.

The greater the Number of Iterations, more secure is the password with a direct effect on the computing power needed for encryption/decryption.

The Number of Iterations is derived from the password. The password used here as an illustration is
Password: *MVS10@web*

The numeric equivalent of this Password substituted from the GNT (refer above), known as the **Password Number** is given below
Password Number: *394845362632352*

Password Number is split by 2 digits starting from left to right.

N.B: Remember that it can be split into 3 or even more depending on the robustness required by encryption. This is the level of iteration followed for encryption and hence it is totally customizable depending on the need.

The numbers are given below:
39 Part I
48 Part II
45 Part III
36 Part IV
26 Part V
32 Part VI
35 Part VII

2 Ignored
The last number (2) is ignored, as it is less than 2 digits.

Step 1:
Taking Part I and Part II of the above split we notice that Part I < (lesser than) Part II
i.e. 39 < 48
Therefore 48 - 39 = 9

Step 2:
Taking 9 and Part III of the above split we notice that 9 < Part III
i.e. 9 < 45
Therefore 45 - 9 = 36

Step 3:
Taking 36 and Part IV of the above split we notice that 36 == (equal to) Part IV
i.e. 36 == 36
Therefore 36

Step 4:
Taking 36 and Part V of the above split we notice that 36 > (greater than) Part V
i.e. 36 > 26
Therefore 36 - 26 = 10

Step 5:
Taking 10 and Part VI of the above split we notice that 10 < Part VI
i.e. 10 < 32
Therefore 32 - 10 = 22

Step 6:
Taking 22 and Part VII of the above split we notice that 22 < Part VII
i.e. 22 < 35
Therefore 35 - 22 = 13

Hence 13 is taken as the **Number of Iterations**.

NB: If the Number of Iterations falls in the range of 1-4 it is taken as 5 as per the rule of default minimum number of iterations needed.

d. Creating the Prime Number Reference Table (PNRT):

The PNRT is nothing but which has the corresponding prime number for a specified number in the number line. This table will be looked into after the Number of Iterations is obtained as explained above; to check the number of increments that needs to be done for each iteration encrypting the password. This will be explained with detailed relevant example further below.

An example of a PNRT is given below which gives the corresponding prime number in the number line:

1	2	**11**	31
2	3	**12**	37
3	5	**13**	41
4	7	**14**	43
5	11	**15**	47
6	13	**16**	53
7	17	**17**	59
8	19	**18**	61
9	23	**19**	67
10	29	**20**	71

This table is interpreted as the 1st prime number in the number line being 2, 2nd prime number being 3, 3rd being 5 and so on…

So according to the scenario, Number of Iterations being 13 will correspond to 41 increments in encrypting the password. This will be called as the **Prime Increments**

For easy illustration here let's take the least Number of Iterations (i.e. 5), which corresponds to 11 as the Prime Increments (obtained from the above Prime Number Reference Table)

e. Encryption Logic explained:

Before explaining further about how the encryption logic is followed lets recollect the data available:
Username/ID: *runcy_oommen@mcafee.com*
Password: *MVS10@web*

From this the following information for encryption is obtained:
Key Number: *10627*
Number of Iterations: *13* (But for demonstration purposes here *5 is taken*)

Now getting back the encrypted string for the password is obtained from the Table of Primes:
11057 11119 11093 11171 11261 11273 10883 10709 10667 (The *SPACE* given between the numbers is just for easy identification)

As stated above Number of Iteration taken for this example is 5. Obtaining the next corresponding prime number depending on the frequency in the PNRT does the encryption for each iteration.

The position in which this increment is done again depends on whether the Number of Increments is odd/even. If the Number of Iterations is odd as the case here (5), the first increment will start from the odd block of the encrypted string (i.e. position 1, 3, 5, 7, 9 and so on). A block is nothing but a 5 digit number read from left to the right in the encrypted string. Here this corresponds to the numbers 11057, 11093, 11261, 10883, and 10667 in the encrypted string for the first iteration

Iteration 1 will be based on the 2nd consecutive prime number for each number of the odd block in the encrypted string (since 2 being the first reference entry in the PNRT). The numbers in the even block will not be incremented but just written in reverse.

11057 11119 11093 11171 11261 11273 10883 10709 10667 (The Original Encrypted string)

Iteration 1 with Prime Increment 2:
11069 91111 11117 17111 11279 37211 10891 90701 10691 (The numbers in the odd block—denoted by blue color incremented by 2 prime numbers while those in the even block—denoted by red written in reverse)

Iteration 2 with Prime Increment 3:
96011 11159 71111 11197 97211 11299 19801 10729 19601 (The numbers in the odd block—denoted by blue color written in reverse while those in the even block—denoted by red color incremented by 3 prime numbers reading from the right hand side)

Iteration 3 with Prime Increment 5:
11113 95111 11161 79111 11321 99211 10949 92701 10733 (The numbers in the odd block—denoted by blue color incremented by 5 prime numbers reading from the right hand side while those in the even block—denoted by red written in reverse)

Iteration 4 with Prime Increment 7:
31111 11239 16111 11273 12311 11369 94901 10799 33701 (The numbers in the odd block—denoted by blue color written in reverse while those in the even block—denoted by red color incremented by 7 prime numbers reading from the right hand side)

Iteration 5 with Prime Increment 11:
11213 93211 11273 37211 11443 96311 11069 99701 10859 (The numbers in the odd block—denoted by blue color incremented by 11 prime numbers reading from the right hand side while those in the even block—denoted by red written in reverse)

The last iterated encrypted string with Iteration 5 and Prime Increment 11 is given below:
11213 93211 11273 37211 11443 96311 11069 99701 10859

N.B: Again the *SPACE* in between the numbers is just for easy identification purpose.

f. Embedding the number of iterations in the encryption string:

As per the above encryption method the least default Number of Iteration (5) was performed. This Iteration step (i.e. 5) needs to be embedded along with the final encryption string for decryption. This is explained below.

The final encrypted string is:
11213 93211 11273 37211 11443 96311 11069 99701 10859

The default minimum no of iterations defined for this algorithm as 5. Hence it is considered as a 'factor' for embedding in the encrypted string.
So considering the no of iteration step as a factor of 5 the Factor Number for embedding in the encrypted string is arrived as given below. Consider Factor Number as nothing but the Quotient arrived at when the Number of Iteration gets divided by 5.
Iteration 5 = Factor Number 1
Iteration 10 = Factor Number 2
Iteration 15 = Factor Number 3 and so on…

Now in this scenario the Number of Iteration is taken as 5, hence the Factor Number as 1. This number (1) is embedded into the encrypted string.

With no further complications, this Factor Number is just appended at the right end of the string. So the final encrypted string after embedding the Number of Iterations will be as given below:
*11213 93211 11273 37211 11443 96311 11069 99701 10859***1** (Notice 1 added at the end of the string which is the Factor Number)

In the database the above-encrypted string (without the SPACES in between of-course) will be stored instead of just plain text password and if anyone happens to get this information has no value without obtaining the decryption key. Thus the customer information is now stored in a secured manner with no fear of data loss even if the database is leaked out or exposed.

Special Case Illustration:
Now let's take a case in which the number of iterations is not a multiple of 5, take it to be 14. In this case the Factor Number is 2 (i.e. 14 divided by 5 gives Quotient as 2 and Remainder as 4). Factor Number is embedded as explained above into the encrypted string (let's take the same encrypted string for easy illustration) and the remainder (i.e. 4) is appended to the end of the encrypted string.

So the final encrypted sting after embedding the Number of Iterations (including the Factor Number and remainder) will look like as given below:
*11213 93211 11273 37211 11443 96311 11069 99701 108591***4** (notice 4 added at the end of the string which is the remainder)

g. Decryption Process:

Now in the database the encrypted string will look like as given below:
11213 93211 11273 37211 11443 96311 11069 99701 108591 (without the SPACES in between)
NB: This encrypted string is the original one and not the one shown for Special Case Illustration

The length of the encrypted string is checked, which in this case is 46. This is a not a multiple of 5 (as the encryption depends on 5 digit prime numbers), and the remainder obtained is 1, which concludes that there is just a Factor Number at

the end with no further remainders for the Factor Number (as the special case illustration). This Factor Number is removed first from the encrypted string and the decryption process begins.

The Factor Number is extracted out from the right side of the string and multiplied with 5. Since it is 1 in this case, the Number of Iterations is 5 (i.e. 1*5).

So the modified final encrypted string after the removal of the Factor Number will look as given below:
11213 93211 11273 37211 11443 96311 11069 99701 10859

The actual decryption process will start from now. During the decryption process de-iteration 1 will point you to the 5th prime number in PNRT (i.e. 11), de-iteration 2 will actually point you to the 4th prime number in PNRT (i.e. 7) and de-iteration 3 will actually point you to the 3rd prime number in PNRT (i.e. 5) and so on. This is done to calculate the Prime Decrements needed for every de-iteration.

Now let's have a look at the actual decryption process in detail.

11213 11239 11273 11273 11443 11369 11069 10799 10859
The above string is obtained by writing the reversed string properly (given in green) in the even blocks. Consider it as a thumb rule that even if the Number of Iterations performed during encryption is even or odd, the block where the encrypted string is written as reversed is always even.
De-iteration 5 with Prime Decrement 11:
11113 11239 11161 11273 11321 11369 10949 10799 10733 (The numbers in blue were decremented by 11 prime numbers from the previous)

De-iteration 4 with Prime Decrement 7:
11113 11159 11161 11197 11321 11299 10949 10729 10733 (The numbers in blue were decremented by 7 prime numbers from the previous)

De-iteration 3 with Prime Decrement 5:
11069 11159 11117 11197 11279 11299 10891 10729 10691 (The numbers in blue were decremented by 5 prime numbers from the previous)

De-iteration 2 with Prime Decrement 3:
11069 11119 11117 11171 11279 11273 10891 10709 10691 (The numbers in blue were decremented by 3 prime numbers from the previous)

De-iteration 1 with Prime Decrement 2:
11057 11119 11093 11171 11261 11273 10883 10709 10667 (The numbers in blue were decremented by 2 prime numbers from the previous)

The final decrypted string with just the initial prime numbers is given below:
11057 11119 11093 11171 11261 11273 10883 10709 10667

Now the user would have definitely given the Username/ID for logging in. From this the Key Number is arrived at (explained above in part a.). From this Key Number the Table of Primes is created and do character substitution for each prime numbers to the above-obtained decrypted string.

Thus the real password after making the substitution is:
MVS10@web

This decrypted string is compared with the user-entered password and the login page and is validated.

3. Advantages:

- A totally customizable algorithm from head to toe. This adds to the flexibility and more importantly the robustness of the algorithm. The logic followed implemented by P.E.A in first program is totally different from the second one.
- An illustration is just done here for the encryption of password, but this can be extended to encrypt any data before it is stored to a backend/ repository.
- The speed of encryption can be boosted to a great extend if a table is already created containing the primes in the desired range. Suppose as in this example illustration the Key Number has lies within 10000 to 11000. Hence a table can be already created which has all the primes from 10000 to the maximum 99000 and then it is just a matter of referencing this table rather than creating ToP on the fly.
- The P.E.A can be easily made customizable as per internationalization standards with the ASCII value assigned to each character in the keyboard rather than a numerical sequence.