online AES tool: http://aes.online-domain-tools.com/

cipher key	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	=> key length 16 characters (16bytes = 16*8bit = 128 bit key length -> AES **128**)

----

buffer number	0															
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Example String (16 characters)	B	A	S	I	C		E	X	A	M	P	L	E	!	!	!

=> a buffer of 16 characters can be easily passed to the AES128 bit encryption method encrypt(char * plainText, char * key, unsigned char * outputBuffer)

result (displayed as hex)	91	88	29	f0	84	f2	58	25	b3	6f	a4	3a	a7	58	c4	68
result (displayed as human readable)			)	ð	.	ò	X	%	³	o	¤	:	§	X	Ä	h

=> so in our case now, we call encryptString with passing the example String, this function will try to split the string into 16 character buffers to encrypt every buffer.
In fact, we only have one buffer, so in this case we will call only once encryptBuffer() and get the encrypted result back as a string

online AES tool: http://aes.online-domain-tools.com/

cipher key	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	=> key length 16 characters (16bytes = 16*8bit = 128 bit key length -> AES **128**)

---

buffer number	0															
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0-15															
Example String (multiple of 16 chars.)	B	A	S	I	C		E	X	A	M	P	L	E	!	!	!
result (displayed as hex)	91	88	29	f0	84	f2	58	25	b3	6f	a4	3a	a7	58	c4	68
result (displayed as human readable)			)	ð	.	ò	X	%	³	o	¤	:	§	X	Ä	h

buffer number	1															
index	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	16-31															
Example String (multiple of 16 chars.)	B	A	S	I	C		E	X	A	M	P	L	E	!	!	!
result (displayed as hex)	91	88	29	f0	84	f2	58	25	b3	6f	a4	3a	a7	58	c4	68
result (displayed as human readable)			)	ð	.	ò	X	%	³	o	¤	:	§	X	Ä	h

buffer number	2															
index	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	32-47															
Example String (multiple of 16 chars.)	B	A	S	I	C		E	X	A	M	P	L	E	!	!	!
result (displayed as hex)	91	88	29	f0	84	f2	58	25	b3	6f	a4	3a	a7	58	c4	68
result (displayed as human readable)			)	ð	.	ò	X	%	³	o	¤	:	§	X	Ä	h

=> now we have more than 16 characters. To handle this we split the current string into buffer parts consisting each of 16 characters.
=> each part buffer now will be encrypted with the method we used before (encrypt() or encryptBuffer()). Our result will now consists of 3 encrypted buffers, which we concat.

↑

? ? not printable in a readable format

=> now we have 3 buffers, each containing 16 characters, the important part lys now after lines 139 (Cipher.cpp):

Our string consists of 48 characters -> 48 / 16 = 3 (=> int index = plainText.length() / BUFF_SIZE)
after that we iterate through the whole string buffer by buffer and load 16 chars over and over again into
a local buffer and encrypt this one seperatly and concat the result to the output encrypted string

line 139 in Cipher.cpp: **Here we calculate the amount of needed buffers containing 16 characters**

```
for(int block=0; block < plainText.length()/BUFF_SIZE (=> 3) ; block++) {

  for(int j = block*BUFF_SIZE (=> 0,16,32); j < (block+1)*BUFF_SIZE (=> 16,32,48); j++) {
        buffer += plainText[j];
  }

  cipherTextString += encryptBuffer(const_cast<char*>(buffer.c_str()), key);
  buffer = "";
}
```

go through the whole string => buffer number 0,1,2

load new part of the string in the buffer
add each char of the string from the current index to the end of the current buffer into the buffer itself
e.g. from index 16 until 31 into buffer number 1

encrypt current buffer (0,1 or 2) and concat the result to the output encrypted string
clear the buffer

=> the result will contain in this case 3 times the same encrypted part string due to the fact that our example string consist of 3x16 same characters

online AES tool: http://aes.online-domain-tools.com/

cipher key a b c d e f g h i j k l m n o p => key length 16 characters (16bytes = 16*8bit = 128 bit key length -> AES **128**)

----

buffer number								0								
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
									0-15							
Example String (multiple of 16 chars.)	B	A	S	I	C		E	X	A	M	P	L	E	!	!	!
result (displayed as hex)	91	88	29	f0	84	f2	58	25	b3	6f	a4	3a	a7	58	c4	68
result (displayed as human readable)			)	ð	.	ò	X	%	³	o	¤	:	§	X	Ä	h

buffer number									1							
index	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
									16-31							
Example String (multiple of 16 chars.)	B	A	S	I	C		E	X	A	M	P	L	E	!	!	!
result (displayed as hex)	91	88	29	f0	84	f2	58	25	b3	6f	a4	3a	a7	58	c4	68
result (displayed as human readable)			)	ð	.	ò	X	%	³	o	¤	:	§	X	Ä	h

buffer number			2													
index	32	33	34	35	36											
			32-36			\0										
Example String (multiple of 16 chars.)	B	A	S	I	C						empty part!					
result (displayed as hex)	de	c9	e0	17	b2	83	8a	b8	63	37	58	86	aa	28	f7	14
result (displayed as human readable)	Þ	É	à	.	²	.	.	¸	c	7	X	.	ª	(	÷	.

=> now we have more than 16 characters. To handle this we split the current string into buffer parts consisting each of 16 characters and if its an odd number of characters,into one with the rest.
=> each part buffer now will be encrypted with the method we used before (encrypt() or encryptBuffer()). Our result will still consists of 3 encrypted buffers, which we concat.

↑

? ? not printable in a readable format

=> now we have 2 buffers, each containing 16 characters, and one buffer containing only 5 characters

Our string consists of 36 characters -> 36 / 16 = 2 (=> int index = plainText.length() / BUFF_SIZE) line 139 in Cipher.cpp: **Here we calculate the amount of needed buffers containing 16 characters**
encrypting the buffers containing 16 characters will be done like i explained before. The interesting part is to encrypt the rest characters in the last buffer.
First of all we have to check if we have an odd number of characters of the passed string. After that we have to calculate the index from which we have to start and to end the last buffer

```
if(  plainText.length() % BUFF_SIZE (=> 36%16 = 5) > 0  ) {

    for(int bytes_read=(index*BUFF_SIZE) (=> 32); bytes_read <= (index*BUFF_SIZE) + plainText.length()%BUFF_SIZE (=> 36); bytes_read++) {
        buffer += plainText[bytes_read];
    };

    cipherTextString += encryptBuffer(const_cast<char*>(buffer.c_str()), key);
 }
```

here we check if we have a rest e.g. 36 % 16 = 2 **R 5**
=> rest 5 > 0 !

the last buffer will start from 2*16 until 2*16 + 36%16 = 32 + 5
now we load every char from our string regarding to the index into the last buffer

finally, encrypt this last buffer and concat it to the return string

=> the result will contain in this case 2 times the same encrypted part string and a rest containing the encryption result of the last 5 characters
but still, the result string has 3x16 characters, because the encryption library will always encrypt 16 chars, if we pass less than 16 characters,
it will encrypt the characters and the basic ending character (\0) and after that random characters until 16 characters are reached.

Note: decryption works exactly the same way, we only have to finish if we find the ending character \0