```
                for(i = 0; i < 4; i++)
                        t[i] = getSBoxValue(t[i]);
        }
        /* We XOR t with the four-byte block 16,24,32 bytes before the
        new expanded key.
        * This becomes the next four bytes in the expanded key.
        */
        for(i = 0; i < 4; i++) {
                expandedKey[currentSize]   =   expandedKey[currentSize   -
                size] ^ t[i];
                currentSize++;
                }
    }
}
```

همان طور که می بینید من هیچ گاه از حلقه های داخلی برای تکرار یک عمل استفاده نکرده ام تنها حلقه های داخلی برای گام زدن در ۴ قسمت آرایه ی موقت t هستند. من از عملگر پیمانه ای برای چک کردن اینکه آیا نیاز به اعمال عملگر هست یا نه استفاده کرده ام.

اگر (currentSize % size == 0): هرگاه n بایت از expandedKey (که n اندازه ی کلید رمزنگاری ست) را تولید کرده ایم، هسته ی گسترش کلید را یک بار اجرا می کنیم.

اگر ((currentSize % size) == 16) && size == SIZE_32) اگر در حال گسترش کلید ۳۲ بیتی هستیم و تا کنون ۱۶بایت را گسترش داده ایم، یک جانشینی اضافی S-Box اضافه خواهیم کرد. چنان که گفته شد در نسخه ی ۳۲بیتی اولین حلقه را تنها۳ بار اجرا می کنیم که ۱۲ بایت + ۴ بیات از هسته را تولید می کند.

پیاده سازی: استفاده از گسترش کلید

حالا می توانیم گسترش کلید جدیدمان را امتحان کنیم. من اندازه کلید گسترده شده را محاسبه نخواهم کرد و در عوض یک مقدار ثابت به آن خواهم داد. این کدی ست که یک کلید رمزنگاری داده شده را توسعه می دهد:

```
/* the expanded keySize */
int expandedKeySize = 176;
/* the expanded key */
unsigned char expandedKey[expandedKeySize];
/* the cipher key */
unsigned char key[16] = {0};
/* the cipher key size */
enum keySize size = SIZE_16;
int i;
expandKey(expandedKey, key, size, expandedKeySize);
printf("Expanded Key:n");
for (i = 0; i < expandedKeySize; i++)
{
      printf("%2.2x%c", expandedKey[i], (i%16) ? 'n' : ' ');
}
```

البته این کد از چند ثابت استفاده می کند که وقتی بدنه ی AES را پیاده سازی کردیم تولید خواهند شد. چند نمونه از نتایج تست:

گسترش کلید ۱۲۸ بیتی که شامل کاراکتر های Null هم می شود:

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
62 63 63 63 62 63 63 63 62 63 63 63 62 63 63 63
9b 98 98 c9 f9 fb fb aa 9b 98 98 c9 f9 fb fb aa
90 97 34 50 69 6c cf fa f2 f4 57 33 0b 0f ac 99
ee 06 da 7b 87 6a 15 81 75 9e 42 b2 7e 91 ee 2b
7f 2e 2b 88 f8 44 3e 09 8d da 7c bb f3 4b 92 90
ec 61 4b 85 14 25 75 8c 99 ff 09 37 6a b4 9b a7
21 75 17 87 35 50 62 0b ac af 6b 3c c6 1b f0 9b
0e f9 03 33 3b a9 61 38 97 06 0a 04 51 1d fa 9f
b1 d4 d8 e2 8a 7d b9 da 1d 7b b3 de 4c 66 49 41
b4 ef 5b cb 3e 92 e2 11 23 e9 51 cf 6f 8f 18 8e
```

گسترش کلید ۱۹۲ بیتی که شامل کاراکتر های تهی هم می شود:

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 62 63 63 63 62 63 63 63
62 63 63 63 62 63 63 63 62 63 63 63 62 63 63 63
9b 98 98 c9 f9 fb fb aa 9b 98 98 c9 f9 fb fb aa
9b 98 98 c9 f9 fb fb aa 90 97 34 50 69 6c cf fa
f2 f4 57 33 0b 0f ac 99 90 97 34 50 69 6c cf fa
c8 1d 19 a9 a1 71 d6 53 53 85 81 60 58 8a 2d f9
c8 1d 19 a9 a1 71 d6 53 7b eb f4 9b da 9a 22 c8
89 1f a3 a8 d1 95 8e 51 19 88 97 f8 b8 f9 41 ab
c2 68 96 f7 18 f2 b4 3f 91 ed 17 97 40 78 99 c6
59 f0 0e 3e e1 09 4f 95 83 ec bc 0f 9b 1e 08 30
0a f3 1f a7 4a 8b 86 61 13 7b 88 5f f2 72 c7 ca
43 2a c8 86 d8 34 c0 b6 d2 c7 df 11 98 4c 59 70
```

گسترش کلید ۲۵۶ بیتی که شامل کاراکتر های تهی هم می شود:

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
62 63 63 63 62 63 63 63 62 63 63 63 62 63 63 63
aa fb fb fb aa fb fb fb aa fb fb fb aa fb fb fb
6f 6c 6c cf 0d 0f 0f ac 6f 6c 6c cf 0d 0f 0f ac
7d 8d 8d 6a d7 76 76 91 7d 8d 8d 6a d7 76 76 91
53 54 ed c1 5e 5b e2 6d 31 37 8e a2 3c 38 81 0e
96 8a 81 c1 41 fc f7 50 3c 71 7a 3a eb 07 0c ab
9e aa 8f 28 c0 f1 6d 45 f1 c6 e3 e7 cd fe 62 e9
2b 31 2b df 6a cd dc 8f 56 bc a6 b5 bd bb aa 1e
64 06 fd 52 a4 f7 90 17 55 31 73 f0 98 cf 11 19
6d bb a9 0b 07 76 75 84 51 ca d3 31 ec 71 79 2f
e7 b0 e8 9c 43 47 78 8b 16 76 0b 7b 8e b9 1a 62
74 ed 0b a1 73 9b 7e 25 22 51 ad 14 ce 20 d4 3b
10 f8 0a 17 53 bf 72 9c 45 c9 79 e7 cb 70 63 85
```

پیاده سازی: رمزنگاریAES

برای پیاده سازی الگوریتم رمزنگاری AES درست به همان شکل که برای کلید رمزنگاری بود، پیش خواهیم رفت. به این شکل که اول توابع کمکی و بعد حرکت به بالا به سمت حلقه ی اصلی. توابع یک حالت (state) را به عنوان پارامتر می گیرند که یک آرایه ی مربعی ۴*۴ از بایت ها است. حالت را به صورت آرایه ی ۲ بعدی در نظر نخواهیم گرفت، بلکه آرایه ای یک بعدی با طول ۱۶ می گیریم.

پیاده سازی: subBytes

در مورد این عملیات چیز زیادی برای گفتن وجود ندارد. فقط جانشینی ساده ای با S-Box است:

```
void subBytes(unsigned char *state)
{
	int i;
	/* substitute all the values from the state with the value in the
	SBox
	* using the state value as index for the SBox
	*/
	for (i = 0; i < 16; i++)
		state[i] = getSBoxValue(state[i]);
}
```

پیاده سازی: shiftRows

این تابع را به دو قسمت شکسته ایم. این امکان وجود نداشت که همه ی آن را در یک وحله انجام دهیم، چون خواندن و اشکال زدایی آن ساده تر می شد. تابع shiftRows همه ی سطرها را پیمایش می کند و بعد تابع shiftRow را با آفست مناسب صدا می کند. shiftRow هیچ کاری جز شیفت دادن یک آرایه ی ۴ بایتی با آفست داده شده انجام نمی دهد.

```
void shiftRows(unsigned char *state)
{
	int i;
	/* iterate over the 4 rows and call shiftRow() with that row */
	for (i = 0; i < 4; i++)
		shiftRow(state+i*4, i);
}
void shiftRow(unsigned char *state, unsigned char nbr)
{
	int i, j;
	unsigned char tmp;
	/* each iteration shifts the row to the left by 1 */
	for (i = 0; i < nbr; i++)
	{
		tmp = state[0];
		for (j = 0; j < 3; j++)
		state[j] = state[j+1];
		state[3] = tmp;
	}
}
```

این پیاده سازی کمترین کارایی را دارد اما از همه ساده تر است.یک بهبود ساده می تواند این باشد که از آنجایی که اولین سطر شیفت داده نمی شود، پیمایش را به جای ۰ از ۱ شروع کنیم.

پیاده سازی: addRoundKey

این قسمتی ست که در برگیرنده ی roundkey است که در هر گام تولید می کنیم. صرفا هر بایت کلید را با حالت XOR می کنیم:

```
void addRoundKey(unsigned char *state, unsigned char *roundKey)
{
	int i;
	for (i = 0; i < 16; i++)
		state[i] = state[i] ^ roundKey[i] ;
}
```

پیاده سازی: mixColumns

احتمالا mixColumns سخت ترین مورد بین این ۴ عملیات است. این عمل دربردارنده ی جمع و ضرب galois و پردازش ستون ها به جای سطرهاست (که خوب نیت چون ما یک آرایه خطی داریم که نمایشگر سطر هاست و نه ستون ها).

اول یک تابع نیاز داریم که اعداد را در میدان گالویس ضرب کند. از تابع حاضر و ارائه شده توسط Sam Trenholme استفاده شده است:

```
unsigned char galois_multiplication(unsigned char a, unsigned char b)
{
	unsigned char p = 0;
	unsigned char counter;
	unsigned char hi_bit_set;
	for(counter = 0; counter < 8; counter++) {
		if((b & 1) == 1)
			p ^= a;
		hi_bit_set = (a & 0x80);
```

```
            a <<= 1;
            if(hi_bit_set == 0x80)
                  a ^= 0x1b;
            b >>= 1;
      }
      return p;

}
```

دوباره تابع را به دو قسمت شکسته ایم که قسمت اول یک ستون ایجاد می کند و بعد mixColumns را صدا می زند که بعدا ضرب ماتریسی اعمال می گرد.

```
void mixColumns(unsigned char *state)
{
      int i, j;
      unsigned char column[4];
      /* iterate over the 4 columns */
      for (i = 0; i < 4; i++)
      {
            /* construct one column by iterating over the 4 rows */
            for (j = 0; j < 4; j++)
            {
                  column[j] = state[(j*4)+i];
            }
            /* apply the mixColumn on one column */
            mixColumn(column);
            /* put the values back into the state */
            for (j = 0; j < 4; j++)
            {
                  state[(j*4)+i] = column[j];
            }
      }
}
```

mixColumns فقط یک ضرب گالویسی ستون های ماتریس ۴*۴ گفته شده در تئوری ست. چون بقیه مربوط به یک عمل XOR است و تابع ضرب را داریم، پیاده سازی تقریبا ساده است:

```
void mixColumn(unsigned char *column)
{
      unsigned char cpy[4];
      int i;
      for(i = 0; i < 4; i++)
      {
            cpy[i] = column[i];
      }
      column[0] = galois_multiplication(cpy[0],2) ^
      galois_multiplication(cpy[3],1) ^
      galois_multiplication(cpy[2],1) ^
      galois_multiplication(cpy[1],3);
      column[1] = galois_multiplication(cpy[1],2) ^
      galois_multiplication(cpy[0],1) ^
      galois_multiplication(cpy[3],1) ^
      galois_multiplication(cpy[2],3);
      column[2] = galois_multiplication(cpy[2],2) ^
      galois_multiplication(cpy[1],1) ^
      galois_multiplication(cpy[0],1) ^
      galois_multiplication(cpy[3],3);
      column[3] = galois_multiplication(cpy[3],2) ^
      galois_multiplication(cpy[2],1) ^
      galois_multiplication(cpy[1],1) ^
      galois_multiplication(cpy[0],3);
}
```

این تابع هم می توانست بهینه شود (مثلا استفاده از memcpy به جای حلقه) اما برای خواناتر شدن این کار انجام نشد.

پیاده سازی: حلقه ی AES

همان طور که در تئوری دیدید، یک حلقه ی AES هیچ کاری جز اعمال متوالی آن چهار عمل بر روی حالت انجام نمی دهد.

```
void aes_round(unsigned char *state, unsigned char *roundKey)
{
	subBytes(state);
	shiftRows(state);
	mixColumns(state);
	addRoundKey(state, roundKey);
}
```

پیاده سازی: بدنه ی اصلی AES

حال که همه ی توابع کوچک را داریم، حلقه ی اصلی واقعا ساده خواهد بود.تمام کاری که باید بکنیم گرفتن حالت، expandedKey و تعداد دورها به عنوان پارامتر است و اعمال ۴ عملیات، یکی پس از دیگری. تابع کوچک creatRoundKey برای کپی کردن ۱۶ بایت بعدی از expandedKey به roundKey با استفاده از ترتیب نگارنده ی خاص استفاده می شود.

```
void createRoundKey(unsigned char *expandedKey, unsigned char *roundKey)
{
	int i,j;
	/* iterate over the columns */
	for (i = 0; i < 4; i++)
	{
		/* iterate over the rows */
		for (j = 0; j < 4; j++)
		roundKey[(i+(j*4))] = expandedKey[(i*4)+j];
	}
}
void aes_main(unsigned char *state, unsigned char *expandedKey, int
nbrRounds)
{
	int i = 0;
	unsigned char roundKey[16];
	createRoundKey(expandedKey, roundKey);
	addRoundKey(state, roundKey);
	for (i = 1; i < nbrRounds; i++) {
		createRoundKey(expandedKey + 16*i, roundKey);
		aes_round(state, roundKey);
	}
	createRoundKey(expandedKey + 16*nbrRounds, roundKey);
	subBytes(state);
	shiftRows(state);
	addRoundKey(state, roundKey);
}
```

پیاده سازی: رمزنگاری AES

نهایتا تمام کاری که باید بکنیم این است که همه چیز را در کنار هم قرار دهیم. پارامتر های ما متن اصلی ورودی، کلیدی به طول keySize و خروجی ست. ابتدا تعداد دورها را بر اساس keySize محاسبه می کنیم و بعد باید ورودی ۱۶بایتی متن اصلی را به ترتیب صحیح به یک حالت ۴*۴ بایتی نگاشت کنیم، کلید را با زمانبند گسترش دهیم، حالت را با بدنه ی اصلی

AES رمز کنیم و نهایتا حالت را دوباره به ترتیب صحیح از نگاشت خارج کنیم (unmap) تا ۱۶ بایت رمز شده ی خروجی را به دست آوریم. به نظر پیچیده می آید اما خواهید دید که این طور نیست:

```
char aes_encrypt(unsigned char *input,
                 unsigned char *output,
                 unsigned char *key,
                 enum keySize size)
{
      /* the expanded keySize */
      int expandedKeySize;
      /* the number of rounds */
      int nbrRounds;
      /* the expanded key */
      unsigned char *expandedKey;
      /* the 128 bit block to encode */
      unsigned char block[16];
      int i,j;
      /* set the number of rounds */
      switch (size)
      {
            case SIZE_16:
                  nbrRounds = 10;
                  break;
            case SIZE_24:
                  nbrRounds = 12;
                  break;
            case SIZE_32:
                  nbrRounds = 14;
                  break;
            default:
                  return UNKNOWN_KEYSIZE;
                  break;
      }
      expandedKeySize = (16*(nbrRounds+1));
      if ((expandedKey = malloc(expandedKeySize * sizeof(char))) == NULL)
      {
            return MEMORY_ALLOCATION_PROBLEM;
      }
      /* Set the block values, for the block:
      * a0,0 a0,1 a0,2 a0,3
      * a1,0 a1,1 a1,2 a1,3
      * a2,0 a2,1 a2,2 a2,3
      * a3,0 a3,1 a3,2 a3,3
      * the mapping order is a0,0 a1,0 a2,0 a3,0 a0,1 a1,1 ... a2,3 a3,3
      */
      /* iterate over the columns */
      for (i = 0; i < 4; i++)
      {
            /* iterate over the rows */
            for (j = 0; j < 4; j++)
            block[(i+(j*4))] = input[(i*4)+j];
      }
      /* expand the key into an 176, 208, 240 bytes key */
      expandKey(expandedKey, key, size, expandedKeySize);
      /* encrypt the block using the expandedKey */
      aes_main(block, expandedKey, nbrRounds);
      /* unmap the block again into the output */
      for (i = 0; i < 4; i++)
      {
```

```
            /* iterate over the rows */
            for (j = 0; j < 4; j++)
            output[(i*4)+j] = block[(i+(j*4))];
      }
      return 0;
}
```

در کد بالا UNKNOWN_KEYSIZE و MEMORY_ALLOCATION_PROBLEM ماکرو هایی هستند برای برخی از کد خطا های از پیش تعریف شده که می توان برای چک کردن اینکه آیا همه چیز صحیح است یا نه استفاده کرد. کد نباید خیلی پیچیده باشد و توضیحات برای درک آن کافی به نظر می رسد.

رمزگشایی AES

اگر تا الان موفق شده اید همه چیز را درک و پیاده سازی کنید، با رمزگشایی هم مشکلی نخواهید داشت. درواقع همه چیز را در رمزنگاری معکوس می کنیم و عملیات را برعکس انجام می دهیم.

از آن جا که زمانبند کلید به همان شکل باقی می ماند، تنها عملیاتی که باید پیاده سازی کرد subBytes معکوس، shifRows و mixColumns است اما addRowKey به همان شکل باقی می ماند. غیر از mixColumns معکوس، بقیه ساده هستند و کد آن ها را در زیر آورده ام:

```
void invSubBytes(unsigned char *state)
{
      int i;
      /* substitute all the values from the state with the value in the
      SBox
      * using the state value as index for the SBox
      */
      for (i = 0; i < 16; i++)
            state[i] = getSBoxInvert(state[i]);
      }
      void invShiftRows(unsigned char *state)
      {
            int i;
            /* iterate over the 4 rows and call invShiftRow() with that row
            */
            for (i = 0; i < 4; i++)
            invShiftRow(state+i*4, i);
}
void invShiftRow(unsigned char *state, unsigned char nbr)
{
      int i, j;
      unsigned char tmp;
      /* each iteration shifts the row to the right by 1 */
      for (i = 0; i < nbr; i++)
      {
            tmp = state[3];
            for (j = 3; j > 0; j--)
            state[j] = state[j-1];
            state[0] = tmp;
      }
}
```

همان طور که می بینید این دو بسیار شبیه همتای رمزنگاری خود هستند غیر از اینکه اینجا چرخش به راست از و از S-Box وارون استفاده شده است.

برای mixColumns معکوس، تفاوت در ماتریس ضربی ست که این می باشد:

```
14 11 13  9
 9 14 11 13
13  9 14 11
11 13  9 14
```

همان طور که می بینید تمام کاری که باید انجام دهید تغییر دادن مقادیر در صدا زدن ضرب گالویس با مقادیری از ماتریس بالاست که حاصل در زیر آمده است:

```
void invMixColumns(unsigned char *state)
{
	int i, j;
	unsigned char column[4];
	/* iterate over the 4 columns */
	for (i = 0; i < 4; i++)
	{
		/* construct one column by iterating over the 4 rows */
		for (j = 0; j < 4; j++)
		{
			column[j] = state[(j*4)+i];
		}
		/* apply the invMixColumn on one column */
		invMixColumn(column);
		/* put the values back into the state */
		for (j = 0; j < 4; j++)
		{
			state[(j*4)+i] = column[j];
		}
	}
}
void invMixColumn(unsigned char *column)
{
	unsigned char cpy[4];
	int i;
	for(i = 0; i < 4; i++)
	{
		cpy[i] = column[i];
	}
	column[0] = galois_multiplication(cpy[0],14) ^
	galois_multiplication(cpy[3],9) ^
	galois_multiplication(cpy[2],13) ^
	galois_multiplication(cpy[1],11);
	column[1] = galois_multiplication(cpy[1],14) ^
	galois_multiplication(cpy[0],9) ^
	galois_multiplication(cpy[3],13) ^
	galois_multiplication(cpy[2],11);
	column[2] = galois_multiplication(cpy[2],14) ^
	galois_multiplication(cpy[1],9) ^
	galois_multiplication(cpy[0],13) ^
	galois_multiplication(cpy[3],11);
	column[3] = galois_multiplication(cpy[3],14) ^
	galois_multiplication(cpy[2],9) ^
	galois_multiplication(cpy[1],13) ^
	galois_multiplication(cpy[0],11);
}
```

لطفا توجه کنید که می شد از تکرار کد زیادی دوری کرد اگر یک پارامتر دیگر به mixColumns عادی اضافه می کردیم که برای تعیین ماترسی رمزنگاری یا رمزگشایی استفاده شود.

حال که همه ی عملیات را برعکس کرده ایم می توان مشخص کرد که یک حلقه ی AES معکوس چگونه خواهد بود

```
void aes_invRound(unsigned char *state, unsigned char *roundKey)
{
	invShiftRows(state);
```

```
	invSubBytes(state);
	addRoundKey(state, roundKey);
	invMixColumns(state);
}
```

نهایتا باید همه چیز را در یک الگوریتم معکوس شده کنار هم بگذاریم. توجه کنید که از گسترش کلید به صورت وارون استفاده کرده ایم که از ۱۶ بایت آخر استفاده می کند و به سمت آغاز می رود.

```
void aes_invMain(unsigned char *state, unsigned char *expandedKey, int
nbrRounds)
{
	int i = 0;
	unsigned char roundKey[16];
	createRoundKey(expandedKey + 16*nbrRounds, roundKey);
	addRoundKey(state, roundKey);
	for (i = nbrRounds-1; i > 0; i--)
	{
		createRoundKey(expandedKey + 16*i, roundKey);
		aes_invRound(state, roundKey);
	}
	createRoundKey(expandedKey, roundKey);
	invShiftRows(state);
	invSubBytes(state);
	addRoundKey(state, roundKey);
}
```

با اینکه مطمئنم چیزی که در ادامه خواهد آمد را خود قادر به ایجاد هستید، می خواهم الگوریتم رمزگشایی را هم اینجا قرار دهم که بسیار شبیه رمزنگاری ست، غیر از آنکه تابع اصلی معکوس شده را فراخوانی می کند.

```
char aes_decrypt(unsigned char *input,
		 unsigned char *output,
		 unsigned char *key,
		 enum keySize size)
{
	/* the expanded keySize */
	int expandedKeySize;
	/* the number of rounds */
	int nbrRounds;
	/* the expanded key */
	unsigned char *expandedKey;
	/* the 128 bit block to decode */
	unsigned char block[16];
	int i,j;
	/* set the number of rounds */
	switch (size)
	{
		case SIZE_16:
			nbrRounds = 10;
			break;
		case SIZE_24:
			nbrRounds = 12;
			break;
		case SIZE_32:
			nbrRounds = 14;
			break;
		default:
			return UNKNOWN_KEYSIZE;
			break;
	}
	expandedKeySize = (16*(nbrRounds+1));
	if ((expandedKey = malloc(expandedKeySize * sizeof(char))) == NULL)
	{
		return MEMORY_ALLOCATION_PROBLEM;
```

```
	}
	/* Set the block values, for the block:
	* a0,0 a0,1 a0,2 a0,3
	* a1,0 a1,1 a1,2 a1,3
	* a2,0 a2,1 a2,2 a2,3
	* a3,0 a3,1 a3,2 a3,3
	* the mapping order is a0,0 a1,0 a2,0 a3,0 a0,1 a1,1 ... a2,3 a3,3
	*/
	/* iterate over the columns */
	for (i = 0; i < 4; i++)
	{
		/* iterate over the rows */
		for (j = 0; j < 4; j++)
		block[(i+(j*4))] = input[(i*4)+j];
	}
	/* expand the key into an 176, 208, 240 bytes key */
	expandKey(expandedKey, key, size, expandedKeySize);
	/* decrypt the block using the expandedKey */
	aes_invMain(block, expandedKey, nbrRounds);
	/* unmap the block again into the output */
	for (i = 0; i < 4; i++)
	{
		/* iterate over the rows */
		for (j = 0; j < 4; j++)
		output[(i*4)+j] = block[(i+(j*4))];
	}
}
```

این انتهای پیاده سازی AES است. تمام چیزی که باقی می ماند این است که تابع AES را درون یک block cipher modes of operation استفاده کنیم تا بتوان پیغام هایی از هر اندازه را رمزنگاری و رمزگشایی کرد.