

## Re: Cipher "arikahs"

---

*Source:* <http://www.derkeiler.com/Newsgroups/sci.crypt/2006-07/msg00597.html>

---

- *From:* jt64@xxxxxxxx
  - *Date:* 17 Jul 2006 02:50:33 -0700
- 

However one *could* and maybe *should* use the keyexpander also within cipher for every 3-10kb because it will reinitialise the internal state of the cipher dependant upon current blockkey.

Any attacker currently analysing stream will lose track of analyse, because the internal state changed and he will have to start over again.

JT

jt64@xxxxxxxx skrev:

A cipher built from my automatic keyrescheduler.  
The cipher totally byte scalable, due built around permutations of bytes. That swap  
Assume original key 256 bit

PSEUDOCODE *\*arikahs\**

```
[init]->
invkey=reorder_bits(key)
savestate_key=key^invkey
permute(key) // ** bitwise see permutation code below
permute(invkey) // ** bitwise
```

```
[encrypt while more plaintext blocks]
blockkey=key^invkey^savestatekey
savestatkey=key^savestatkey
keyexpander(blockkey) /* bitwise see expander code below
permute(key) // ** bitwise
permute(invkey) // ** bitwise
ciphertext=plaintext^blockkey
```

---

The PRNG below, you would have to adjust it to swap indexed bits from key instead  
of the actual permutation values from the expander which is done below.

```
/*PERMUTE*/
```

Re: Cipher "arikahs"

```
ordered=key
reversed=invkey

int permute()
{
int savestate[256];
int slot_1,slot_2,slot_3,slot_4;
int i;

/*THE PERMUTATION ARRAY FILLED WITH INTEGER/VALUES FROM 0-255*/
for(i=0;i<256;i++)
{
if(i==256){
/*PERMUTATION WRAPAROUND CASE*/
slot_1 = ordered[0];
slot_2 = ordered[slot_1];
ordered[slot_1] = ordered[i];
ordered[i] = slot_2;
slot_3 = reversed[0];
slot_4 = reversed[slot_3];
reversed[slot_3] = reversed[i];
reversed[i] = slot_4;
} else {
/*OTHERWISE*/
slot_1 = ordered[i+1];
slot_2 = ordered[slot_1];
ordered[slot_1] = ordered[i];
ordered[i] = slot_2;
slot_3 = reversed[i+1];
slot_4 = reversed[slot_3];
reversed[slot_3] = reversed[i];
reversed[i] = slot_4;
}
}

return 0;

}

/**KEYXPAND*/

int keyexpander(int key[], int keysize)
{
int serie[256];
int i=0,j=0,k=0;
int stop=0;
int found=0;
```

Re: Cipher "arikahs"

```
for(i=0;i<256;i++)
{
stop=0;
while(!stop)
{
found = 0;
for(k=0;k<i;k++)
{
if(key[j]==serie[k]){ found=1;}
}
if(!found)
{
serie[i]=key[j];stop=1;
} else
{
key[j] = key[j]+1;
}
if(key[j] > 255) {
key[j]=0;
}
}
j++;
if(j > keysize-1) {j=0;}
}

for(k=0;k<42;k++)
{
for(i=0;i<255;i++)
{
int slot_1,slot_2;
slot_1 = serie[i+1];
slot_2 = serie[slot_1];
serie[slot_1] = serie[i];
serie[i] = slot_2;

}
}

for(i=0;i<256;i++)
{
ordered[i]=serie[i];
}
return 0;
}

int keyexpander(int key[], int keysize) //works on bytlevel
{
int serie[256];
```

Re: Cipher "arikahs"

```
int i=0,j=0,k=0;
int stop=0;
int found=0;

for(i=0;i<256;i++)
{
stop=0;
while(!stop)
{
found = 0;
for(k=0;k<i;k++)
{
if(key[j]==serie[k]){ found=1;}
}
if(!found)
{
serie[i]=key[j];stop=1;
} else
{
key[j] = key[j]+1;
}
if(key[j] > 255) {
key[j]=0;
}
}
j++;
if(j > keysize-1) {j=0;}
}

for(k=0;k<42;k++)
{
for(i=0;i<255;i++)
{
int slot_1,slot_2;
slot_1 = serie[i+1];
slot_2 = serie[slot_1];
serie[slot_1] = serie[i];
serie[i] = slot_2;

}
}

for(i=0;i<256;i++)
{
ordered[i]=serie[i];
}
return 0;
}
```