

# The ISAAC Stream Cipher

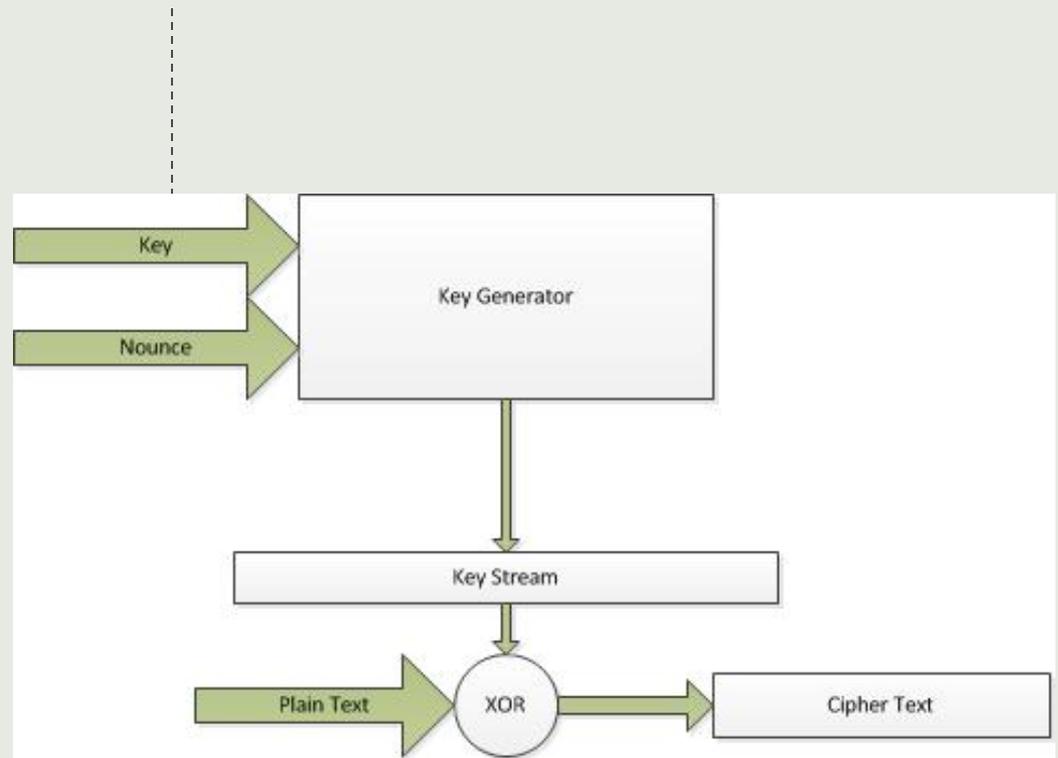
Presented to you by Team Marauder Shields

# Refresher on Stream Ciphers

-Key + Nounce =  
Keystream

-Plaintext Xor'd  
with Keystream =  
Ciphertext

-Ciphertext Xor'd  
with same  
keystream =  
Plaintext



# History

- Based off the RC4 stream cipher
- Basic structure of RC4:
  - Starts with a 256-byte array, filled with the golden ratio, then modified with bytes of the key.
  - Each keystream bit modifies the array:
    - Increments counter 1
    - Sets a second counter to be the value of the array element at the first counter
    - Sets the output index to be the sum of the array element at the first counter and the array element at the second counter.
    - Outputs the array element at the output index.

# Issues

RC4 has a few issues, namely dealing with bias.

- Some sequences are more likely to occur than others- bias.
  - This is due to initialization of RC4 to avoid short cycles: sequences that will result in the keystream repeating earlier than expected.
- First few bytes of the keystream are significantly less random and give information about the key.

# Expansion

- Robert Jenkins produced a number of ciphers attempting to expand on RC4
  - IA- gives values based on sum of values in array rather than individual values. Still prone to bias.
  - IBAA- adds an accumulator (rotating value based off array value) to deal with bias issues. Does not appear to have bias, and short cycles are significantly reduced from what would be expected in RC4
  - Finally, produced ISAAC.

# ISAAC

- Indirection, Shift, Accumulate, Add, Count
- Takes the IBAA implementation and adds a counter incremented once per call.
  - This removes any chance of short cycles, as even if the sequence would normally cycle, the counter is different and thus the sequence is different.
  - Estimated cycle length is  $2^{8287}$  calls.
- Optimized for speed
  - Unrolls function call loops to avoid additional checking
  - Changes indirection bits to save an instruction.

# Program Elements

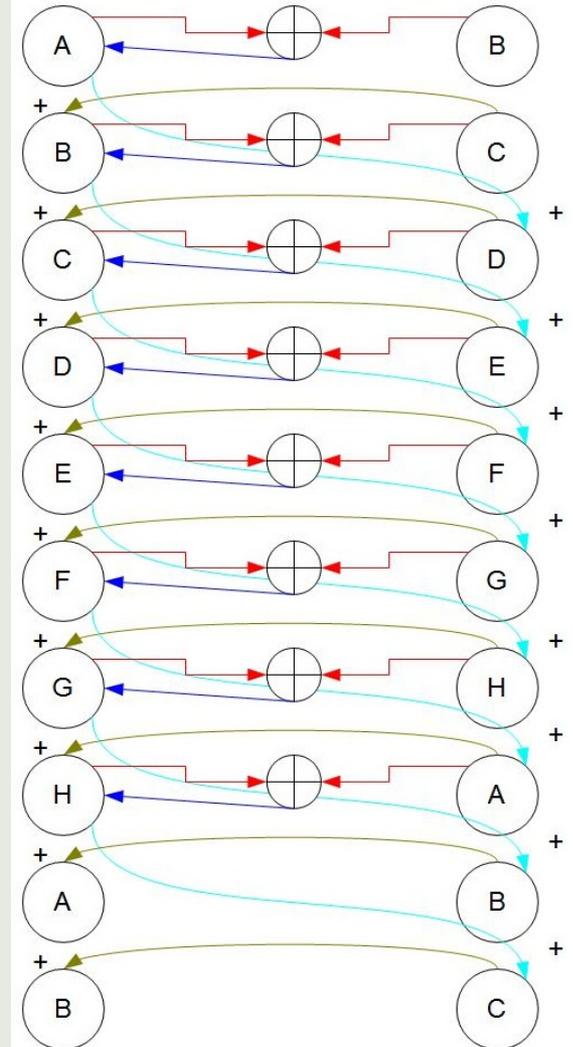
## Notes:

- No standardized initialization routine: author of cipher is less confident in initialization.
  - In the author's code, eight integers are used to fill the array: these integers have elements of the key added to them before being permuted and XORed.
  - Note that this particular initialization routine requires the key be the same length as the output, so KEYSIZE bits of keystream will be output on each function call.

# Program elements- mix() function

Used with eight integers that will contain traces of the key: designed to ensure array elements will not reflect key.

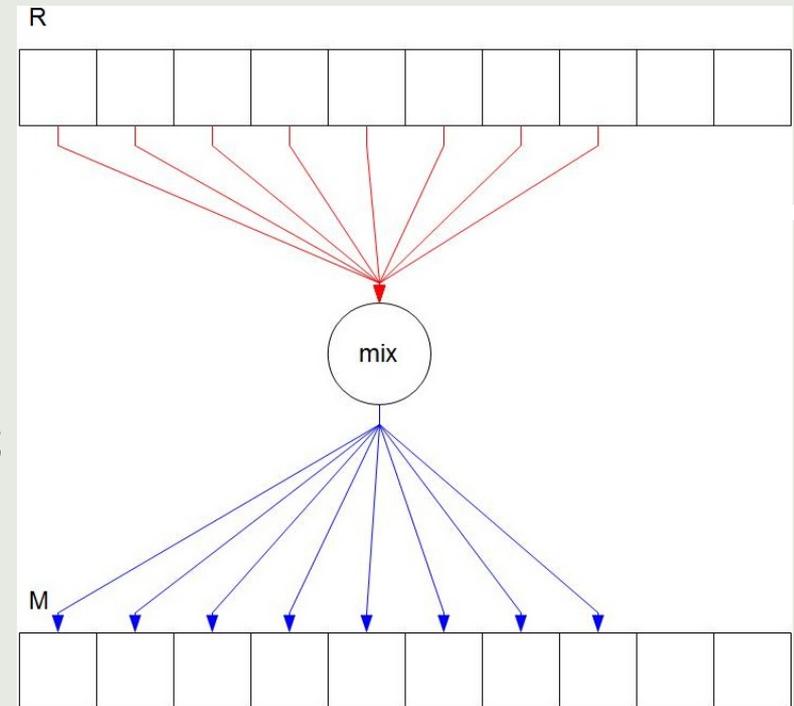
```
mix(a,b,c,d,e,f,g,h)
{
  a^=b<<11; d+=a; b+=c;
  b^=c>>2; e+=b; c+=d;
  c^=d<<8; f+=c; d+=e;
  d^=e>>16; g+=d; e+=f;
  e^=f<<10; h+=e; f+=g;
  f^=g>>4; a+=f; g+=h;
  g^=h<<8; b+=g; h+=a;
  h^=a>>9; c+=h; a+=b;
}
```



# Program Elements- Initialization 1

Loads eight elements of the key into integers, runs the mix() function to randomize them, then loads them into eight elements of the array. Repeats until key is exhausted.

```
for (i=0; i<RANDSIZ; i+=8)
{
  a+=r[i ]; b+=r[i+1]; c+=r[i+2]; d+=r[i+3];
  e+=r[i+4]; f+=r[i+5]; g+=r[i+6];
  h+=r[i+7];
  mix(a,b,c,d,e,f,g,h);
  m[i ]=a; m[i+1]=b; m[i+2]=c; m[i+3]=d;
  m[i+4]=e; m[i+5]=f; m[i+6]=g; m[i+7]=h;
}
```

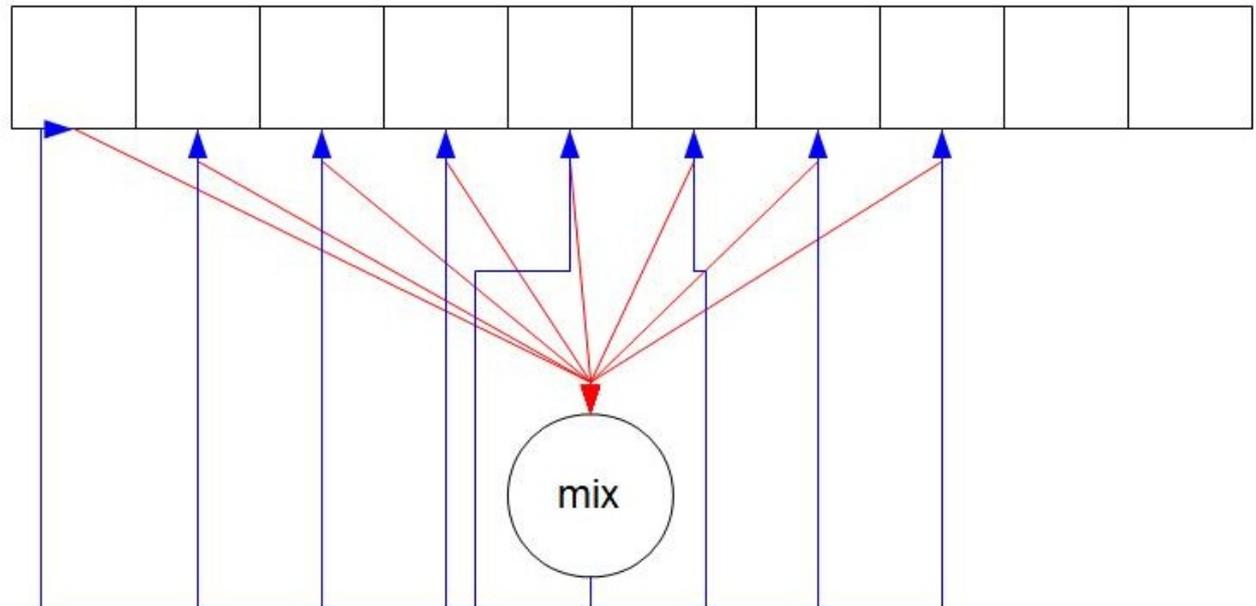


# Program Elements- Initialization 2

Routine then runs a second pass to mix more thoroughly, loading elements of the array instead of elements of the key into the integers this time.

```
for (i=0; i<RANDSIZ; i+=8)
{
  a+=m[i ]; b+=m[i+1];
  c+=m[i+2]; d+=m[i+3];
  e+=m[i+4]; f+=m[i+5];
  g+=m[i+6]; h+=m[i+7];
  mix(a,b,c,d,e,f,g,h);
  m[i ]=a; m[i+1]=b;
  m[i+2]=c; m[i+3]=d;
  m[i+4]=e; m[i+5]=f;
  m[i+6]=g; m[i+7]=h;
}
```

Array



# Program Elements- Generation

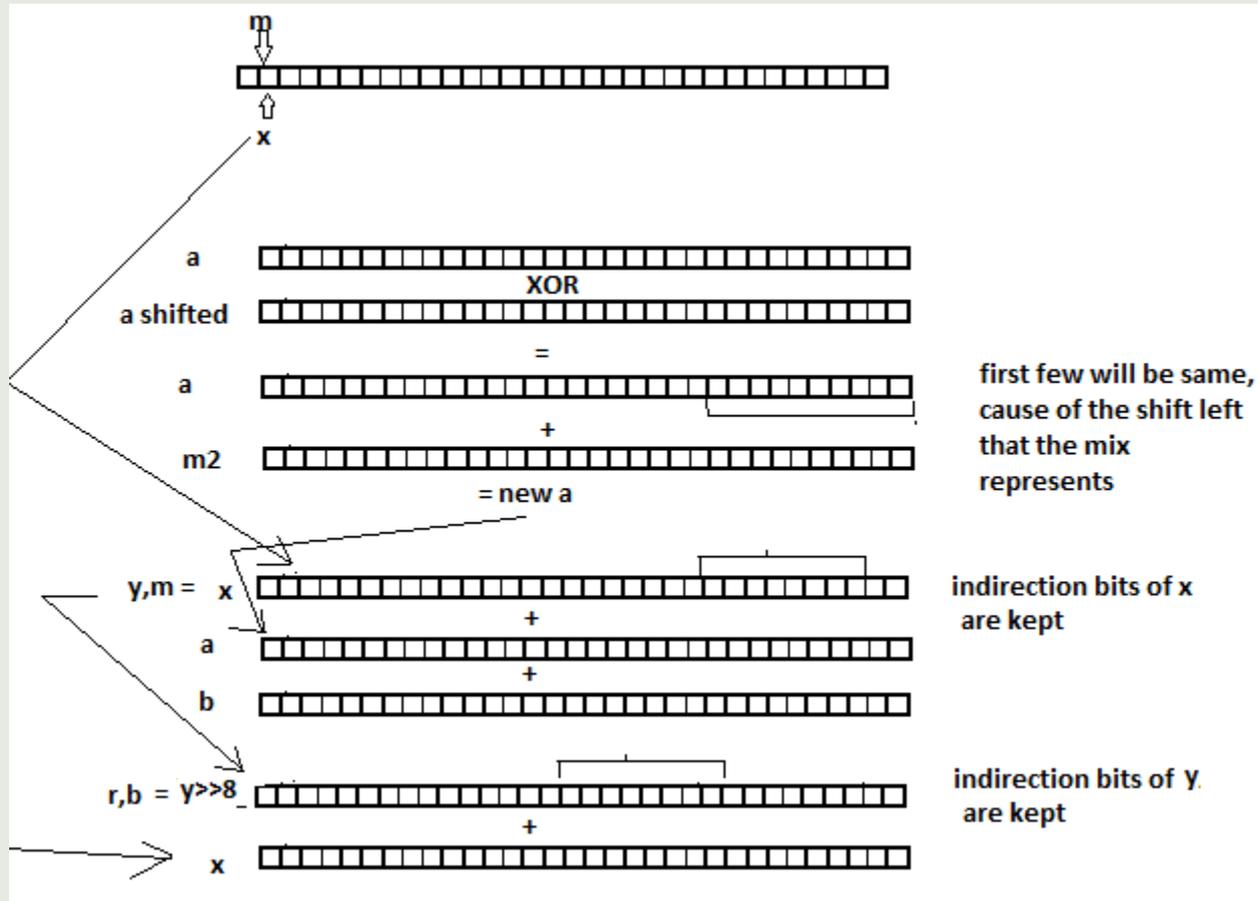
The rngstep function is the main function for the the ISAAC key generator.

```
rngstep(mix,a,b,mm,m,m2,r,x){  
  x = *m;  
  a = (a^(mix)) + *(m2++);  
  *(m++) = y = ind(mm,x) + a + b;  
  *(r++) = b = ind(mm,y>>8) + x;  
}
```

What this essentially does is

- stores the current memory into a register
- set the new value of accumulator
- set the next bit of memory to the addition of the 2-9 bits of x or the current memory with the accumulator and previous result
- Lastly the results array is incremented and set to the addition of x and the 10-17 bits of y bit shifted right by 8

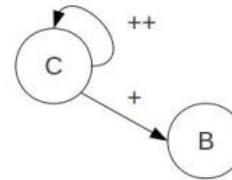
# Program Elements- Generation



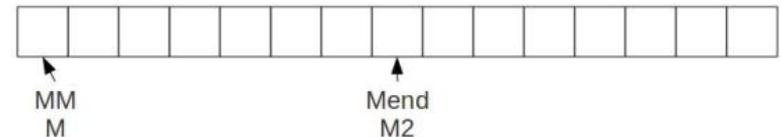
# Program Elements: Main Loop 1

```
b = ctx->randb + (++ctx->randc);
for (m = mm, mend = m2 = m+(RANDSIZ/2);
m < mend; )
{
    rngstep( a << 13, a, b, mm, m, m2, r, x);
    rngstep( a >> 6, a, b, mm, m, m2, r, x);
    rngstep( a << 2, a, b, mm, m, m2, r, x);
    rngstep( a >> 16, a, b, mm, m, m2, r, x);
}
```

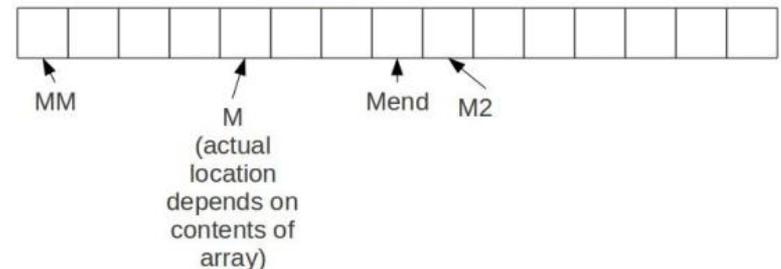
Adds the counter to element B, then calls `rngstep()` function four times with different bitshifts of A for the mix.



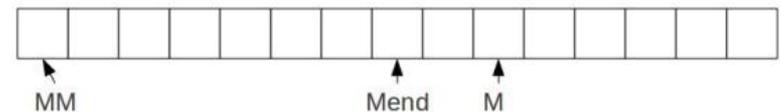
Start:



After rngstep, M moved, M2 moved, MM and Mend const.



End: M moved past Mend

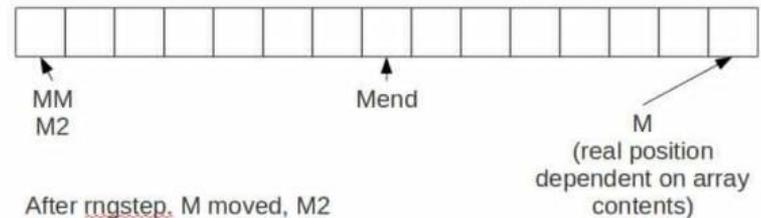


# Program Elements: Main Loop 2

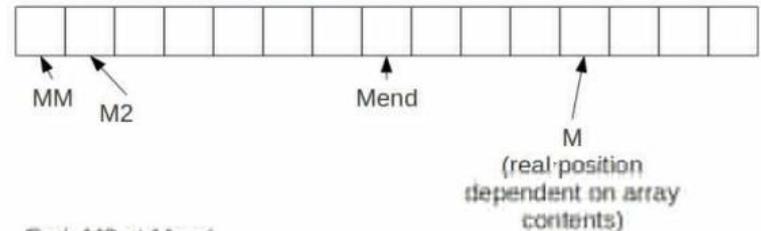
Second loop just iterates with M2 going from first element to mend, calling rngstep four times each iteration.

Designed to ensure that m2 is at each array index for at least one rngstep.

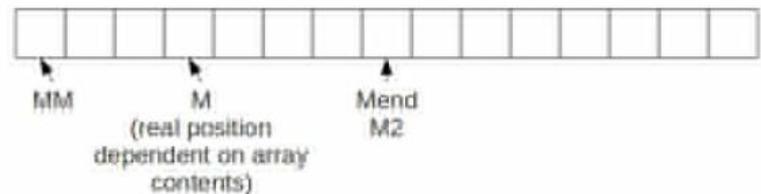
Start:



After rngstep, M moved, M2 moved, MM and Mend const.



End: M2 at Mend



# Resources

-Specification - <http://www.burtleburtle.net/bob/rand/isaac.html>

Questions?