1 RC4 Initialization

1. 16 byte Key $k[0], \ldots, k[15]$. So each $k[i]$ is an 8-bit number, hence between 0 and 255.

2. For $i = 0$ to 255
   
   (a) $S[i] := i$. $S$ is 256 bytes.
   (b) $k[i] = k[i \mod 16]$. $k$ is now 256 bytes.

3. For $i = 0$ to 255
   
   (a) $j := j + S[i] + k[i]$
   (b) Swap $S[i]$ and $S[j]$

4. $i := 0$, $j := 0$, Return ($S, i, j$).

Let's say the first three bytes of the key were
$k[0] = 3$
$k[1] = 255$
$k[2] = X$ (known)

We show that, from the first output bit after the init phase, Eve can learn $k[3]$ 5% of the time.

After the first For loop is done we have the following:

1. For all $0 \leq i \leq 255$, $S[i] = i$.

2. For all $0 \leq i \leq 255$, $k[i]$ is defined. (I don’t think we need this part.)

3. $j = 0$.

We are now in the second loop.

What happens when $i = 0$?

$i = 0$

$j := j + S[i] + k[i] = 0 + S[0] + k[0] = 0 + 0 + 3 = 3$

We swap $S[i] = S[0]$ and $S[j] = S[3]$ so now have

$S[0] = 3$
$S[3] = 0$

For all other $i$, $S[i] = i$.

What happens when $i = 1$?
\[ i = 1 \]
We swap \( S[i] = S[1] \) and \( S[j] = S[3] \) so now have
\[ S[0] = 3 \]
\[ S[1] = 0 \]
\[ S[3] = 1 \]
For all other \( i \), \( S[i] = i \).

**What happens when \( i = 2 \)?**
\[ i = 2 \]
We swap \( S[i] = S[2] \) and \( S[j] = S[X + 5] \) so now have
\[ S[0] = 3 \]
\[ S[1] = 0 \]
\[ S[2] = X + 5 \]
\[ S[3] = 1 \]
\[ S[X + 5] = 2 \]
For all other \( i \), \( S[i] = i \).

**What happens when \( i = 3 \)?**
\[ i = 3 \]
We swap \( S[i] = S[3] \) and \( S[j] = S[X + 6 + k[3]] \) so now have
\[ S[0] = 3 \]
\[ S[1] = 0 \]
\[ S[2] = X + 5 \]
\[ S[X + 5] = 2 \]
\[ S[X + 6 + k[3]] = 3 \]
For all other \( i \), \( S[i] = i \).

**What happens when \( i \geq 4 \)?**
When \( i \geq 4 \) we will be swapping \( S[i] \) with \( S[j] \). Note that if in the next 252 iterations \( j \neq 0, 1, 3 \) then the values above for \( S[0], S[1], S[3] \) will stay the same.
Assuming \( j \) is uniform the prob that \( j \neq 0, 1, 3 \) is
\[ \left( \frac{253}{256} \right)^{252} = 0.05. \]
So 5% of the time \( j \neq 0, 1, 3 \). This may seem small but its not.
SO, 5% of the time we have:
\[ S[0] = 3 \]
\[ S[1] = 0 \]
\[ S[3] = X + 6 + k[3] \] (NOTE - we know \( X \))
2 GetBits

1. Input \((S, i, j)\) (The \((S, i, j)\) are from init, so \(i = j = 0\).

2. \(i := i + 1\)

3. \(j := j + S[i]\)

4. Swap \(S[i]\) and \(S[j]\).

5. \(t := S[i] + S[j]\)

6. \(y := S[t]\)

7. Return\((S, i, j), y\)

Let's say the \(S\) is as at the end of the last section so we have
\(S[0] = 3\)
\(S[1] = 0\)
\(S[3] = X + 6 + k[3]\) (NOTE - we know \(X\)

Then in the first iteration of GetBits the following happens:
\(i := i + 1, \) so \(i = 0 + 1 = 1\)
\(j := j + S[i], \) so \(j = 0 + S[0] = 0\)
Swap \(S[0]\) and \(S[1]\)
\(t = S[0] + S[1] = 3\)
\(y := S[t] = S[3] = X + 6 + k[3]\).
SO, when see first output byte you have a good notion of what \(k[3]\) is.

3 But its only 5%. So What

Assume that the IV is prepended to the key (A terrible idea! This writeup is why its a terrible idea!). Also assume that the IV is 3 bytes long. So Alice and Bob are using

\[ IV[0]IV[1]IV[2]k[0]\]

But effectively we know the first three bytes of the key but not the fourth one. They will use the key for a long time and constantly change IV’s. Some of the IV’s (like \((3, 255, X)\)) lead to a small prob of getting what we are now calling \(k[0]\).

For each init vector that Eve sees she does the following:

1. See if that init vector leads to knowing \(k[0]\) with prob more than uniform.

2. If so then record what \(k[0]\) might be using the methods above.

After a while she will have A LOT of data. The real \(k[0]\) will be obvious after enough data.