I wrote my own encryption algorithm!

Introducing Oceantoo

- Oceantoo is a stream cipher
- It uses three 128-bit linear-feedback shift registers (LFSRs) to generate pseudo random numbers
- The LFSRs are initialised using the SHA256 of the key (password)
- The actual encryption is done using XOR

Disclaimer! Good cryptography is hard!

- "*Anyone can create an algorithm that he himself cannot break*" - Bruce Schneier.
- Good cryptography is hard, there are lots of pitfalls, etc. Oceantoo has never been scrutinized by mathematicians or cryptography experts. It should be considered purely an academic (and amateur) project.
- Oceantoo is provided "as is" and any express or implied warranties, including, but not limited to, the implied warranties of fitness for a particular purpose are disclaimed.
- YOU HAVE BEEN WARNED.

Stream Ciphers and Block Ciphers

- A stream cipher encrypts (using XOR) the next bit (or maybe byte) of the data, with no relation to what was encrypted before or after.
- A block cipher works on a block as a whole. Using a 4x4 block, the bytes can be swapped around, substitutions can be made, and XOR applied.
  - DES, IDEA, RC5, and AES are block ciphers

XOR

- Exclusive OR is true if, and only if, its arguments differ.

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

See my video “The power of XOR”

Random numbers

- For a stream cipher to work, you need a good pseudo-random sequence
- It needs to be as random as possible so that the next bit (or byte) can’t be guessed
- But it needs to be pseudo-random so that the same sequence is created every time, thus making encode and decode possible

LFSR

- A linear-feedback shift register (LFSR) is a method of manipulating a number (a register) by shifting it (moving all the bits to the left or right) and then adding a new bit in the space that was created (the feedback).
- The feedback works by taking certain bits (called taps) and combining them using XOR to generate the bit which will be inserted into the gap created by the sift.
- They are often used as pseudo-random numbers generators (as in this case).

4-bit LFSR

- If we start with `1111` (numbered left to right: 1, 2, 3, 4)
- We take the XOR of bits 3 and 4 (`1 XOR 1`) is `0`
- Shift right and add the `0` into the space `0111`
- We take the XOR of bits 3 and 4 (`1 XOR 1`) is `0`
- Shift right and add the `0` into the space `0011`

4-bit LFSR

- Current state is `0011`
- We take the XOR of bits 3 and 4 (`1 XOR 1`) is `0`
- Shift right and add the `0` into the space `0001`
- We take the XOR of bits 3 and 4 (`0 XOR 1`) is `1`
- Shift right and add the `0` into the space `1000`

4-bit LFSR

1111	1100
0111	0110
0011	1011
0001	0101
1000	1010
0100	1101
0010	1110
1001	1111

1111000100110101

1111	1100
0111	0110
0011	1011
0001	0101
1000	1010
0100	1101
0010	1110
1001	1111

Weakness

1111	1100
0111	0110
0011	1011
0001	0101
1000	1010
0100	1101
0010	1110
1001	1111

Weakness

1111	1100
0111	0110
0011	1011
0001	0101
1000	1010
0100	1101
0010	1110
1001	1111

Weakness

- Even if the taps are unknown, given a sufficient amount of data there are ways to determine ("crack") the LFSR and get its initial state and simulate the numbers it generates.
- The Berlekamp–Massey algorithm is an algorithm that will find the shortest linear feedback shift register (LFSR) for a given binary output sequence.

Solutions?

- To overcome this weakness there are various methods which can be used.
- One method is to use a multiplexed LFSR. These were first mentioned in the PhD thesis of S. M. Jennings in 1980.
- Multiplexed LFSRs and Shrinking Generators use multiple LFSRs.
  - In the case of the latter one LFSR controls how another is used.
  - When the LSB of LSFR1 is 1, then the output of the sequence is the LSB of LFSR2.
  - When the LSB LFSR1 is 0, the LSB of LFSR2 is discarded. The algorithm then loops, until LFSR1 generates a 1, and so the LSB of LFSR2 is used.
- Other variations include alternating between LFSR1 and LFSR2, etc.

FISH and PIKE

- FISH (FIbonacci SHrinking) is an encryption algorithm published by Siemens in 1993. It uses two LFSRs.
- Ross Anderson showed that FISH can be broken with just a few thousand bits of known plaintext (see his paper: On Fibonacci Keystream Generators).
- In the same paper Anderson suggested PIKE. Anderson notes that "had the control bits been the carry bits rather than the least significant bits, then our attack would have been much harder".
- Therefore Oceantoo uses the carry bits of an LFSR to shrink the sequence.
- PIKE also used XOR for the LSBs of each LFSR, as does Oceantoo.

Statistical Weaknessof Multiplexed Sequences

- In the paper "Statistical Weaknessof Multiplexed Sequences" Jovan Dj. Golic et al showed that there are some inherent weaknesses in multiplexed LFSRs.
- The paper also suggested some remedies.
  - To decimate (shrink) the output of the LFSR (similar to the shrinking generator)
  - To XOR the output of the two LFSRs (like PIKE).

Oceantoo's LFSRs

- Oceantoo uses three 128-bit LFSRs
- With the taps of 128, 127, 126, 121.
- Such an LFSR is of maximum cycle.
- It will cycle through 340,282,366,920,938,463,463,374,607,431,768,211,455 unique numbers, with no repeats!!!

Oceantoo's LFSRs

- LFSR3 is used to decimate the output.
  - If the carry bit of LFSR3 is 1 then the LSB of LFSR1 is ignored for 1 bit, while the LSB of LFSR2 is ignored for 2 bits.
  - However the algorithm doesn't loop back around.
- The output of the random number sequence generator is LSB LFSR1 `XOR` LSB LFSR2.
  - Using `XOR` makes it hard to reconstruct the LFSRs

Using XOR on the LFSRs

LFSR1 `11001110010100010111110110`

LFSR2 `10011011001111011010111111`

XOR `01010101011011001101010001`

The XOR’ed result is not directly related to either of the LFSRs, making reconstruction hard.

Seed

- Each LFSR needs an initial state, in fact 128-bits of initial state.
- Oceantoo initializes the LFSRs by taking a plaintext password and generating a SHA256 hash for the string.
- The first 128 bits of that hash are used to initialise an LFSR, called the LFSR Captain.
- The captain is then used as a standard LFSR to generate 3x128bits of data, that are used to initialize LFSR1-3.
    - LFSR Captain is then discarded.

One-Time Pad

- Oceantoo also has the option to move along its sequence by *n* bytes before starting the encode or decode.
  - This means that to encode a file you need to supply a password and an optional offset.
  - To decode you need the same password and the same offset, otherwise decode will fail.
- The use of the offset has the potential to use Oceantoo as a one-time pad (OTP).
  - Every new message that is encrypted needs to start from a new offset, which should be +1 longer than the length of the previously encoded message.
  - One of the conditions for successful OTP is that the key (password + offset) must never be reused in whole or in part.

Performance

- One of the advantages of a LFSR is that it is easy to implement in hardware.
- The nature of the shift register setup allows for high performance encryption.
- The addition of shrinking and the use of multiple LFSRs means Oceantoo is not as efficient as other encryption mechanisms.

Source Code

- You can find the source code (written in C) in my GitHub repository:
  - https://github.com/garyexplains/oceantoo

Example usage

- `oceantoo -p mypassword1 secret_designs.doc secret_designs.doc.oct`
  - This will encode the Word document secret_designs.doc with the key *mypassword1* and write the output to `secret_designs.doc.oct'
- `oceantoo -p mypassword1 secret_designs.doc.oct secret_designs.doc`
  - Is the reverse (i.e. decode) of above.
- `oceantoo -p mypassword1 -n 187172 secret_designs.doc secret_designs.doc.oct`
  - This will encode the Word document secret_designs.doc with the key *mypassword1* and offset of 187172 and write the output to `secret_designs.doc.oct'
- `oceantoo -p mypassword1 -n 9854213 -r -l 50`
  - Will print 50 random bytes (0-255) from the Oceantoo's LFSRs based on the password and offset.