Mersenne Twister Implementation and Cryptographic Insecurity: Project Documentation

Joshua Jurgensmeier (Just Me) -- CS 399: Cryptography -- Dan Kurfis

Introduction

My project is an implementation of the Mersenne Twister 19937 (MT19937) Pseudo-Random Number Generator (PRNG) algorithm, a basic exclusive-or (XOR) stream cipher using MT as the keystream generator, and a hacking tool to break the encryption with a known plaintext attack. MT19937 is a member of a family of MT PRNGs, so called for their use of Mersenne primes, specifically 2^{19937}-1 in the case of MT19937. MT19937 uses a 624-length state vector of 32-bit words, initialized by a seed word. It then uses a recurrence relation on the state to iteratively generate new pseudo-random state words. This process is known as "twisting." The state words are then multiplied by an invertible matrix, called the tempering matrix, to produce the output stream of pseudo-random 32-bit numbers. Because the tempering matrix is invertible, MT19937 – along with much of the MT family – is not suitable for cryptographic use. If an attacker "un-tempers" 624 successive outputs, then they have computed the current state of the generator, allowing all future outputs to be predicted.

I chose the following scenario to illustrate this vulnerability. A user wants to encrypt a message. MT19937 is used to generate a keystream for the message. This keystream is then XORd with the plaintext to produce the ciphertext, a basic stream cipher. However, before the user sends a plaintext to the cipher, an attacker sends a known 624 32-bit word (2,496 bytes) length plaintext to the cipher, which encrypts it. The attacker then XORs the plaintext with the ciphertext to produce the keystream. This keystream is un-tempered to compute the current state vector of the stream cipher's MT19937 instance, which the attacker copies into their own MT19937 instance. Thus, when the user encrypts their message, the attacker can independently compute the keystream, XORing it with the user's ciphertext, to compute the no-longer-secret plaintext.

Software Design

The software is written in Python 3. I chose this language because I am very familiar with it, and enjoy using it. The software design consists of a single source file (MT19937.py), three classes, corresponding to MT19937, the stream cipher, and the hacking tool, along with a main entry-point. I chose Object Oriented Programming (OOP) design over procedural, because all three major components of my project consist of closely related functionality and state. Note that an underscore before a method name is a Python idiom to indicate that it is private.

Main

- Instantiate objects: StreamCipher and MThacker
- Perform known-plaintext attack on stream cipher to capture its MT state vector: MThacker.hack_stream
- Prompt user for plaintext and encrypt it, producing the ciphertext
- Take ciphertext and decrypts it, using the previously captured MT state vector
- Display the results

MersenneTwister19937

Class constants

- MT19937 algorithm parameters from *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator* by Makoto Matsumoto and Takuji Nishimura. Published in 1998 in the ACM Transactions on Modeling and Computer Simulation journal. [Mat98]

$$
\begin{array}{ll}
\text{MT19937} & (w, n, m, r) = (32, 624, 397, 31) \\
 & \text{a} = 9908\text{B0DF} \\
 & \text{u} = 11 \\
 & \text{s} = 7,\ \text{b} = 9\text{D2C5680} \\
(135) & \text{t} = 15,\ \text{c} = \text{EFC60000} \\
 & \text{l} = 18
\end{array}
$$

None __init__(int seed)

- Set index = n + 1
- Call *_initialize_state(seed)*

None _initialize_state(int seed)

- state is initialized with n words generated from seed

None _twist()

- For i = 0—n:
    - Generate new state[i] with recurrence relation. See [Mat98] section 2.1.

$$\mathbf{x}_{k+n} := \mathbf{x}_{k+m} \oplus (\mathbf{x}_k^u | \mathbf{x}_{k+1}^l) A, \quad (k = 0, 1, \cdots).$$

int _temper()

- Perform bit-shifts on state[index] to multiply by tempering matrix and return result. See [Mat98] page 6.

$$\begin{aligned} \mathbf{y} &:= \mathbf{x} \oplus (\mathbf{x} >> u) \\ \mathbf{y} &:= \mathbf{y} \oplus ((\mathbf{y} << s) \text{ AND } \mathbf{b}) \\ \mathbf{y} &:= \mathbf{y} \oplus ((\mathbf{y} << t) \text{ AND } \mathbf{c}) \\ \mathbf{z} &:= \mathbf{y} \oplus (\mathbf{y} >> l), \end{aligned}$$

int next_word()

- If index >= n // This means we need to generate a new state
    - index = 0; Call _twist()
- Call _temper() to return the next pseudo-random word

StreamCipher

None __init__()

- Instantiate source_mt as a new MersenneTwister19937 instance, initialized with a secret seed. This will generate the keystream which will be XORd with the plaintext to

encrypt it. The keystram consists of success calls to source_mt.next_word() concatenated together, such that the Most Significant Byte of the first word will be XORd with the first character of the plaintext.

String encrypt(String plaintext)

- ciphertext = plaintext XOR keystream
- Return ciphertext

MThacker

None __init__()

- Instantiate hkd_mt as a new MersenneTwister19937 instance, initialized with an arbitrary seed. This will be used to create a copy of the stream cipher's MT instance.

int _untemper(int word)

- Perfom inverse temper bit-shift operations on word, and return.

None hack_stream(StreamCipher stream_cipher)

- Use known plaintext and untempering to duplicate state of stream_cipher's MT
- payload = arbitrary n length String
- ciphertext = stream_cipher.encrypt(payload)
- keystream = payload XOR ciphertext
- For i = 0—n:
    - hkd_mt.state[i] = MThacker._untemper(keystream[i])

String decrypt(String ciphertext)

- plaintext = keystream from hkd_mt XOR ciphertext
- Return plaintext

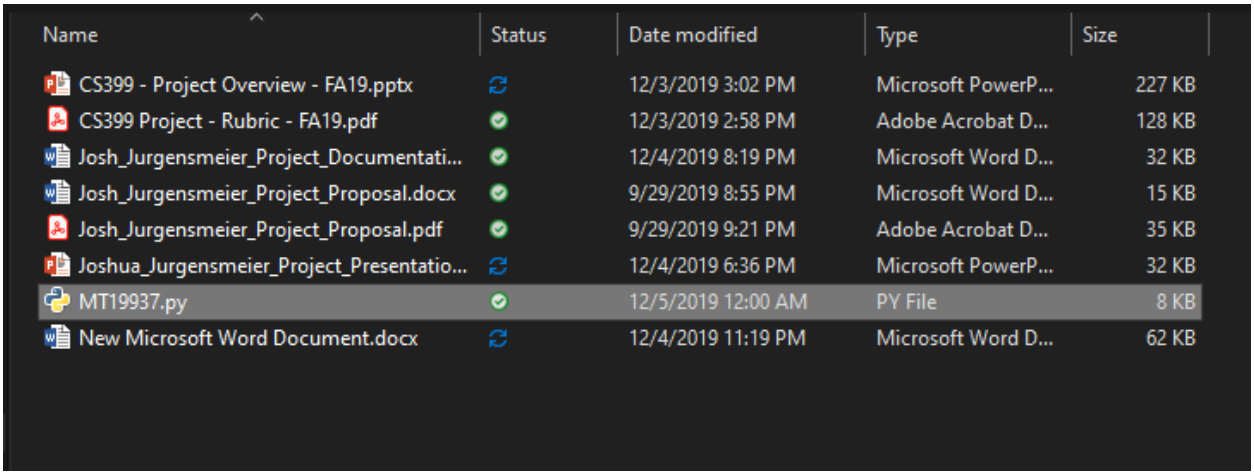
Software Operation and Example Output

- If necessary install Python 3
- Run MT19937.py with Python 3

- On command line (with Python added to the path)

```
C:\Users\ComradSSD\OneDrive\Mount Mercy Classes\Cryptography\Project>python MT19937.py
Enter plaintext message to be encrypted:
```

- From Windows Explorer, simply doubleclick on the program file (with Python set as default .py extension application).



- Respond to the prompt by entering a desired plaintext to be encrypted.

```
C:\Users\ComradSSD\OneDrive\Mount Mercy Classes\Cryptography\Project>python MT19937.py
Enter plaintext message to be encrypted: hellomynameisjosh

Plaintext:
 hellomynameisjosh

Ciphertext:
 b'\xc2\x93@\x1f\xc2\x9f\xc2\xbc\xc2\xa3\xc3\x98\xc3\xbfJQR \x0bx3\xc3\xb9\xc2\xa2'

Hacked Plaintext:
 hellomynameisjosh


C:\Users\ComradSSD\OneDrive\Mount Mercy Classes\Cryptography\Project>
```

- Observe the plaintext, the ciphertext (encrypted with StreamCipher), and the hacked plaintext (decrypted by MThacker). Note: As only about half of all possible bytes are valid ASCII characters and not all bytes are valid UTF-8 characters, about half of all ciphertext will not be displayed correctly.