Steven Comer
CSC 705
Algorithm 1
20 Jul 2016

Class of Algorithm: Stream Cipher

Overview: Stream ciphers are an important class of symmetric encryption algorithms. Symmetric, in this case, means that the same key is used for both encryption and decryption. Encryption aims to prevent unauthorized reading of a given message, by applying a difficult-to-reverse mathematical function to a plaintext message. In order for the message to be read, it must be decrypted, ideally by the intended recipient of the message.

For a generic stream cipher, the following type of mathematical function applies, where C is the ciphertext, P is the plaintext, and K is the keystream.

$C_i = P_i \text{ XOR } K_i$
$P_i = C_i \text{ XOR } K_i$

The mathematical operation (XOR) applied to each bit is not sophisticated. The generation and protection of the keystream is the important part of the algorithm.

Representative Algorithms in this Class:

CryptMT: uses Mersenne twister for PRNG, no known attacks
Phelix: high speed, 256-bit key, no known attacks
Salsa20: follows add-rotate-xor (ARX) paradigm, differential attacks
RC4: simple and fast, no initialization vector, known attacks
Spritz: improved, more secure version of RC4, no published attacks

Algorithm: Spritz
aka: modified RC4

Techniques: XOR stream cipher

Categories: symmetric encryption, stream cipher

Problem: Providing confidentiality for a message passed between two parties.

Applications: Meant to be a drop-in, more secure replacement for RC4 that addresses the issues of this popular algorithm. RC4 is (or can be) used in many applications including WEP, pdf documents, Microsoft Office, Skype, etc.

References:

1. https://people.csail.mit.edu/rivest/pubs/RS14.pdf
2. https://en.wikipedia.org/wiki/RC4#Spritz

3. https://www.schneier.com/blog/archives/2014/10/spritz_a_new_rc.html
4. https://en.wikipedia.org/wiki/RC4#RC4-based_protocols

Implementation details:

- **Big Idea**: XOR-based encryption is simple and fast.
- **Description**: Input message of length n and key. Generate a keystream of length n. Perform the operation “inputs XOR key” to generate the output. This can be either encryption or decryption, as the operation is symmetric.
- **Pseudo-code**: https://people.csail.mit.edu/rivest/pubs/RS14.pdf

```
INITIALIZESTATE(N)
1  i = j = k = z = a = 0
2  w = 1
3  for v = 0 to N − 1
4      S[v] = v

ABSORB(I)
1  for v = 0 to I.length − 1
2      ABSORBBYTE(I[v])

ABSORBBYTE(b)
1  ABSORBNIBBLE(LOW(b))
2  ABSORBNIBBLE(HIGH(b))

ABSORBNIBBLE(x)
1  if a = ⌊N/2⌋
2      SHUFFLE()
3  SWAP(S[a], S[⌊N/2⌋ + x])
4  a = a + 1

ABSORBSTOP()
1  if a = ⌊N/2⌋
2      SHUFFLE()
3  a = a + 1

SHUFFLE()
1  WHIP(2N)
2  CRUSH()
3  WHIP(2N)
4  CRUSH()
5  WHIP(2N)
6  a = 0

WHIP(r)
1  for v = 0 to r − 1
2      UPDATE()
3  do w = w + 1
4  until GCD(w, N) = 1

CRUSH()
1  for v = 0 to ⌊N/2⌋ − 1
2      if S[v] > S[N − 1 − v]
3          SWAP(S[v], S[N − 1 − v])

SQUEEZE(r)
1  if a > 0
2      SHUFFLE()
3  P = ARRAY.NEW(r)
4  for v = 0 to r − 1
5      P[v] = DRIP()
6  return P

DRIP()
1  if a > 0
2      SHUFFLE()
3  UPDATE()
4  return OUTPUT()

UPDATE()
1  i = i + w
2  j = k + S[j + S[i]]
3  k = i + k + S[j]
4  SWAP(S[i], S[j])

OUTPUT()
1  z = S[j + S[i + S[z + k]]]
2  return z
```

```
ENCRYPT(K, M)
1  KEYSETUP(K)
2  C = M + SQUEEZE(M.length)
3  return C

DECRYPT(K, C)
1  KEYSETUP(K)
2  M = C − SQUEEZE(M.length)
3  return M

KEYSETUP(K)
1  INITIALIZESTATE()
2  ABSORB(K)
```

- **Specific implementation**: See included file spritz.c

Correctness:

Theoretical: The argument for the correctness of this program is a proof of the exclusive or (XOR) operation.

$plaintext_i$ XOR key_i = $ciphertext_i$

$ciphertext_i$ XOR key_i = $plaintext_i$

a	b	a XOR b
0	0	0
0	1	1
1	0	1
0	0	0

Empirical: We wrote a test program to check for correctness by comparing the input plaintext to the decrypted ciphertext. This test was performed on 1,000 input cases and passed all cases.

```
// correctness check
for(i=0; i<text_len; i++) {
    if(plaintext[i] != decryptedtext[i]) {
        printf("Error: Messages do not match\n");
        return 0;
    }
}
```

Figure 1: Excerpt from test.c

Performance:

Theoretical: The theoretical performance of this algorithm is O(n), as it is a symmetric encryption algorithm. Again, this means that one operation is performed per byte of input.

Empirical: We wrote a Python script to test the performance of the Spritz algorithm with 1,000 data points (100, 200, …, 100000).

```
#!/usr/bin/python

import os
import time

"""
    Author: Steven Comer
    Description: Spritz empirical testing
    Updated: 20 Jul 16
"""

def main():
    d = {}
    for i in os.listdir("data"):
        start = time.time()
        os.system("./test < data/" + i)
        stop = time.time()
        d[int(i[3:])] = round(stop-start,5)
    for j in sorted(d):
        print str(j) + "," + str(d[j])

if __name__ == "__main__":
    main()
```

Figure 2: test.py

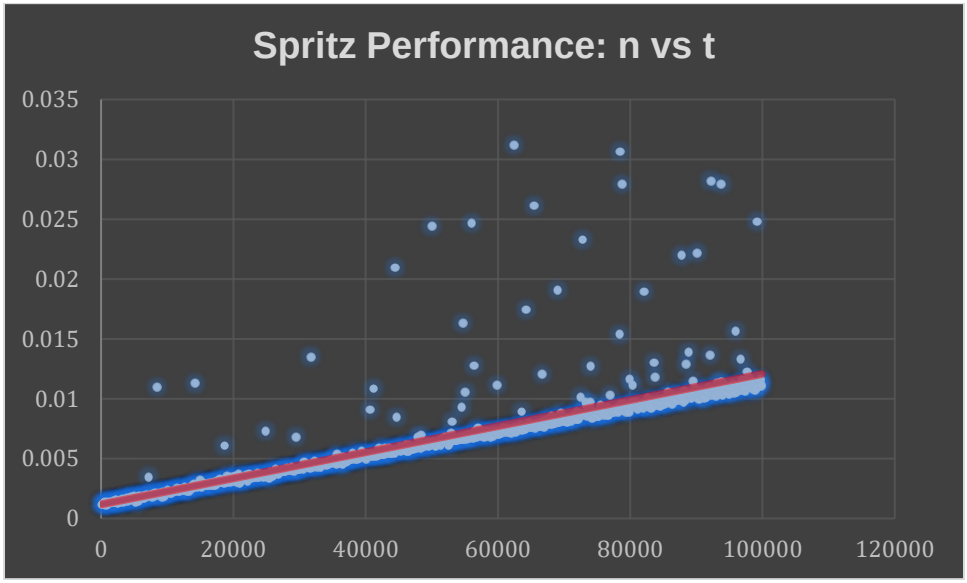

Figure 3: Scatter plot of performance data (blue) with best fit (red)

Anecdotes: The RC4 algorithm should not be used in any scenario where information confidentiality is important.

History: Ron Rivest (RSA) designed RC4 in 1987 and it was first leaked in 1994. The algorithm's key scheduling is inherently weak, in that the beginning of the output reveals information about the key. This led to many successful attacks on

the algorithm. Rivest published Spritz to address the key scheduling issues. With a non-optimized implementation, Spritz is approximately 3x slower than RC4.

Variations: Spritz is a variation of the popular stream cipher RC4.

Alternatives: An alternative to a stream cipher is a block cipher. These tend to perform more complicated mathematical operations on each block. A stream cipher is a block cipher with a block size that is equal to the size of the message.

Conclusion: The use of spritz has not been as widespread as RC4 and as such, we were not able to find any published attacks against it. It addresses the key generation issues of RC4. Stream ciphers and, more generally, symmetric encryption are very widespread and have many important applications. For this reason, it is very necessary for a computer scientist to understand how they work.