The Unbreakable One-Time Pad Cipher

Frank Rubin

Excerpted from

SECRET KEY CRYPTOGRAPHY

MANNING

One-Time Pad

- A cipher which combines a **message key** with a **plaintext** message to produce a **ciphertext**.
- The key stream is the **same length as the message**, one character of key for each character of plaintext.
- The key stream is **unpredictable**, also called **cryptographically secure**. Given part of the key stream, an opponent cannot determine any other part of the key stream. (This is **not** the same as random!)

History

Invented by Frank Miller, a banker from Sacramento CA, in 1882.

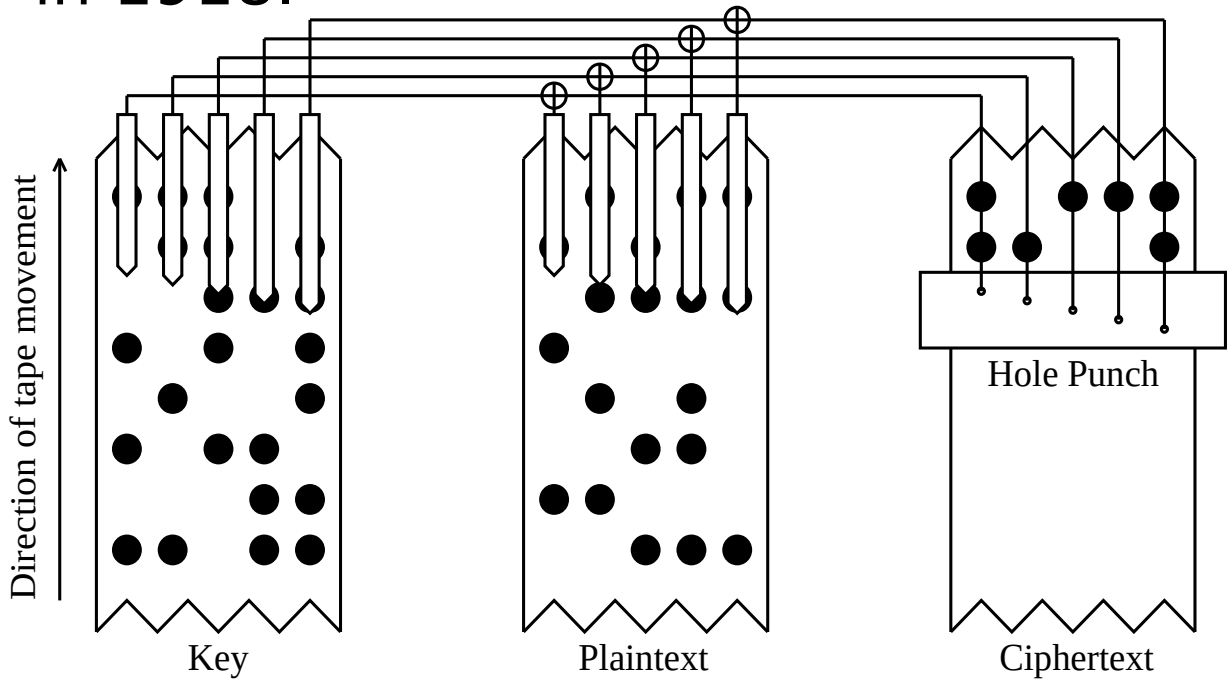
Contract 60401 Plaintext

\+ 372 Key

60773 Ciphertext

One-Time Pad device

Invented by Gilbert S. Vernam of AT&T in 1918.



Key Tape

- 1000 characters long
- Produced by a person using a (forerunner of) Friden Flexowriter
- Not very random, mostly letters
- Had to be delivered or transmitted in clear
- Tapes got reused, with different starting points
- Later version had 2 tapes, 999 and 1000

One-Time (paper) Pads

Invented by Werner Kunze of the German Pers Z S (Signal Intelligence Agency) around 1922. Decimal, using non-carrying addition.

Invented independently by the British about 1926.

Practical problem

The difficulty with the One-Time Pad is that you need as much key material as the total length of all messages sent and received -- plus reserves for future use and sending messages twice.

What happens when you run out?

If you send additional keys, they need to be enciphered, which requires as much extra key material as the total of all keys sent, *ad infinitum*.

Random Number Generators

- True random numbers

  * Cosmic rays, nuclear decay, thermal noise

  * Nature photos: trees, pebbles, flocks of birds, crashing waves

  * Manmade: traffic, rushing people, paint spatters, keystrokes

- Too slow
- Overkill

Pseudorandom number generators

- Multiplicative congruential

  $X_n = aX_{n-1} \bmod p$

- Linear congruential

  $X_n = (aX_{n-1}+b) \bmod N$

- Chained addition

  $X_n = (X_{n-1} + X_{n-s}) \bmod N$

- Chained XOR

  $X_n = X_{n-1} \; X_{n-s}$

- Xorshift - Marsaglia

  Shift and XOR
- FRand

  Chained XOR with cyclic shifts

- Linear feedback shift register

  $X_n = X_{n-1} \quad X_{n-s}$
- Mersenne twister

Multiplicative congruential generator

$X_n = aX_{n-1} \bmod p$

p is a prime

a is a primitive root of p

a, a^2, a^3, ..., a^{p-1} (mod p) are distinct, that is, they are a permutation of 1, 2, 3, ..., p-1

- Use $\lfloor 256X_n/p \rfloor$ as the random byte (<u>not</u> $X_n \bmod 256$)

Wrong Way

Suppose you take 5 multiplicative congruential generators, S, G0, G1, G2 and G3. Use the *selector* S to generate a 2-bit pseudorandom number and use that to choose one of G0...G3 to produce the next random byte. Assuming $p=2^{31}-1$, this gives you an effective key size of $5\times31=155$ bits.

That does not work. At least one of G0...G3 will have 5 outputs among the next 17 output bytes. That's only 6188 possibilities. You can try them all. Only a small fraction of seeds will make all 5 of the corresponding plaintext characters be letters.

Once you have identified the positions for one of the G0...G3, there are only 1287 possible positions for the next generator.

Key Size

Suppose you choose $p=2^{31}-1$. This gives you a 31-bit key. If you use 4 such multiplicative generators, you have an effective key of $4\times31=124$ bits, secure by today's standards.

You can combine the 4 pseudorandom bytes adding them or by XOR-ing them.

To prevent your opponent from solving the 4 generators by linear algebra, make the 4 multipliers and 4 moduli distinct.

$$X_n = a_i X_{n-1} \bmod p_i \quad \text{for } i=1,2,3,4.$$

Linear combination

As before, use 5 multiplicative congruential generators, S, G0, G1, G2 and G3.

Use high-order 16 bits of S to form four 4-bit coefficients, c_0, c_1, c_2, and c_3. At least one of them must be odd.

Use the linear combination $(c_0X_0+c_1X_1+c_2X_2+c_3X_3)$ mod 256 as the random output byte.

Disadvantage: requires 5 pseudorandom numbers to produce each byte.

Combining Functions

Combine a key byte K with a message byte M. A, B and C are non-linear simple substitutions.

Up to now, everything was linear. A combining function can eliminate the linearity, and make it much harder for an opponent to determine any of the random outputs.

- **Addition**. Replace M by: K+M, A(K)+M, K+B(M), A(K)+B(M), C(K+M), C(A(K)+M), C(M+B(K)), C(A(K)+B(M)).
- **XOR**. Replace M by: K⊕M, A(K)⊕M, K⊕B(M), A(K)⊕B(M), C(K⊕M), C(A(K)⊕M), C(K⊕B(M)), C(A(K)⊕B(M)).
- Downside: setup time to mix the substitution alphabets.

Summary

By combining multiple pseudorandom number generators, then combining that key byte non-linearly with the message byte, you can achieve a strong stream cipher which is effectively equivalent to the one-time pad, without needing any true random numbers.