Security of TomsRotaryCipher during a brute force attack

A brute force attack is trying all possible key combinations to decipher a message. This boils down to knowing what exactly are the "keys", and are there so many options as to make deciphering any given message even worthwhile?

The best-case scenario, if I kept TomsRotaryCipher to myself, and no one had access to the code, no one would have a clue as how a particular cipher text was produced. However, a public encryption routine with a known key structure gives the hacker good information about the odds of a successful brute-force attack.

The analysis:

First, we start with the Seeds, or 4-byte numbers:

~~oSeeds.SeedXOR // 4 bytes~~

~~oSeeds.SeedNotchPlan // 4 bytes~~

oSeeds.SeedStartPositions // 4 bytes

oSeeds.SeedTurnOverPositions // 4 bytes

oSeeds.SeedIndividualRotors // 4 bytes \* number of total rotors

4 bytes can contain a 32-bit signed integer from -2,147,483,648 to 2,147,483,647. If we pass a 32-bit signed integer into System.Random as seed, negative signs are removed, which leaves us with 2,147,483,648 possible numbers to choose from.

Let’s omit the use of XOR seed scrambling and the NotchPlan (for HotchScotch skipping) to optimize security and maximize the user of rotors. We will assume 3 moving cipher rotors, which means 5 rotors in total which includes plug board and reflector. This means we will have a 4(2) + 4(5) = 4(7) = 28-byte key. Only 1 number out of 2,147,483,648 possible numbers for each key will be correct, and there are 7 such keys. We have 2,147,483,648 ^ 7 = 2.1062 E 65 possible combinations of numbers. It’s all or nothing.

Security Strength of Seeds = 2.1062 E 65 possible options.

Then we have the number of rotors:

Then we have chosen options, I won't count these options towards “security”.

public enum EnigmaMode { WithReflector, NoReflector }

public enum NoReflectorMode { None, Encipher, Decipher}

public enum NotchPlan { Sequential, Sigaba}

public enum CBCMode { None, Forward, Reverse }

Total Maximum-Security Strength = log10( 2.1062 E 65 ) / log10( 2 ) ~ 217 bits.

How secure is a 217-bit key?

DES used a 56-bit key, considered insecure. AES can use 128, 192, 256-bit keys, considered state of the art. Since TomsRotaryCipher uses a 217-bit key with 3 cipher rotors. Make of this what you will. You can use more rotors, which will increase message space.

Let’s say for the sake of argument you decide to go with 100 cipher rotors:

2,147,483,648 ^ (2 + 102)=

Total Maximum-Security Strength = log10( 3.3167 E 970 ) / 0.30103 ~ 3224 bits.

We can see adding rotors substantially increases security from a brute force attack.

If you add another encryption method, security increases exponentially. Example, you can encipher your plaintext with AES, then further scramble the ciphertext with TomsRotaryCipher. The odds of a successful brute-force attack become nearly zero, assuming the attacker knows what 2 methods of encryption was used, and in what order. All the attackers can see is cipher text. Of course, the downside is there are 2 sets of keys and 2 encryption methods to keep secure.

Another useful add-on is to scramble the cipher text with some basic algorithm. Switch characters within the cipher text using a template overlay. The more distractions like this, the better the hacker does not know what to look for if you have a lot of options in the toolbox. All they might see is a toolbox and cipher text. But the more complicated you make it, the more you must track, this becomes a double-edged sword.

Ideally, we keep things in our memory, NEVER write anything down. If we are supposed to commit passwords to memory, instead we could memorize the individual steps needed to recover a password using our encryption tools. OK, now things are becoming impossible for us. Imagine we cannot uncover plain text because of a faulty memory, no different from forgetting the password entirely.

It’s important to stay creative and use your imagination but also remain practical.

Rev: 3/31/25