           FC1 Algorithm Ushers In The Era Of Post-Alien Cryptography

              draft-fabbrini-algorithm-post-alien-cryptography-02

Abstract

   This memo aims to introduce the concept of "post-alien cryptography",
   presenting a symmetric encryption algorithm which, in our opinion,
   can be considered the first ever designed to face the challenges
   posed by contact with an alien civilization. FC1 cipher offers an
   unprecedented grade of confidentiality. Based on the uniqueness of
   the modular multiplicative inverse of a positive integer a modulo n
   and on its computability in a polynomial time, this non-deterministic
   cipher can easily and quickly handle keys of millions or billions of
   bits that an attacker does not even know the length of.
   The algorithm's primary key is the modulo, while the ciphertext is
   given by the concatenation of the modular inverse of blocks of
   plaintext whose length is randomly chosen within a predetermined
   range. In addition to the full specification here defined, in a
   related work we present an implementation of it in Julia Programming
   Language, accompanied by real examples of encryption and decryption.

Status of This Memo

Table of Contents

1. Introduction

   In a symmetric key encryption scheme, a single key is used for both
   encryption and decryption. An algorithm can be considered safe if the
   only way to guess the key is to explore all the possibilities given
   by the different combinations of zeros and ones. This is called a
   "brute-force attack" and under certain circumstances it can be very
   difficult, if not impossible, to implement. A 256-bit key (the length
   used by the current AES encryption standard) is at present considered
   "unbreakable" even by the next generation of quantum computers. So it
   appears that approved standards can ensure a good level of
   confidentiality for many decades to come. Nevertheless, if we look at
   some structural aspects of them, we can find some relevant weaknesses
   that could jeopardize the security of encrypted data in light of some
   new challenges that we are likely to face in a near future. But

before going into the technical details of the weaknesses, it is useful to dwell on the implicit hypothesis that underlies the alleged security of encryption standards. In fact, they are designed to instantly transfer encrypted data between two different points in space. But if we consider sending data to a different point in time, then the algorithms used would be perhaps inadequate to protect the confidentiality of the original text. For example, suppose Alice wants to transmit an AES-256 encrypted message to Bob who will use the symmetric key to decrypt it in 50 years. How can Alice be sure that the technological development of the coming years will not lead to the ability of testing 2^{256} different sequences of 0's and 1's in a reasonable time, so making the key in Bob's possession useless? Now let's see the structural aspects that may compromise the safety of the accepted standards. Current standards have four main aspects in common. The first two are related to the key: its length is known and does not exceed 256 bits. The last two relate to the algorithms: they are deterministic and convert a fixed block of plaintext into a ciphertext block of the same length. An algorithm is deterministic if a given plaintext always produces a given ciphertext. These four points are generally considered irrelevant and do not raise security concerns. In our opinion, there are however two specific scenarios that could change the rules of the game. The first one, which we call the "internal" scenario, relates to the prospect of an upcoming world war, or at least of a long period of involution in the reciprocal relations among states. A tragic war is hitting Europe. Again. Diplomatic relations are cooling down and scientific discoveries become military secrets. In this context of separation and conflict, how can one be sure that a certain technological level has not already been reached by the adversary? So, for example, how can we be sure that no one in the world is able to test 2^{256} different possibilities in a reasonable time? But if the internal context is worrying, perhaps the "external" one is even more so. By "external" scenario we refer to the possibility of an encounter with an alien civilization that could render current encryption schemes ineffective, thus posing serious security risks for the entire human species. FC1 was designed by imagining attackers possessing a computational capacity far superior to that of our present and future computers.

[2](#). The Need Of A Post-Alien Cryptography

Imagine waking up tomorrow and discovering aliens exist, they're here and they're not good guys. Then you decide to join others to organize a resistance. The first thing you realize is that you need a secure communication channel to coordinate with others. The second bad news of the day is to understand that the AES-xxx is unable to protect your messages because the aliens use cryptanalytic techniques and

computing power capable of breaking any deterministic algorithm.
But without imagining alien invasion scenarios, let's think about
how much the chances of a first contact with aliens will increase
thanks to the synergy between the public and private sectors in
space trips. Humanity is about to become a multiplanetary species and
soon our spaceships will venture into deep space in search of planets
to explore and on which to perpetuate the human race. It is logical
to assume that aliens are technologically more advanced than us and
have perhaps another math, but that they are able to understand ours.
It also makes sense to assume that however advanced their
computational abilities are, they are somehow "bounded". Since a
brute-force attack implies in any case the use of computing power,
it may be a good idea "to raise the bar", thus passing from keys of
256 bits to keys of hundreds of thousands or millions of bits.
Likewise, it can be helpful not to publicly disclose the key length.
But in the face of considerable computing power, the use of a larger
key may not be sufficient. It is necessary to break the mold and move
without delay from deterministic to non-deterministic algorithms,
making the relationship between input and output more complex and
unpredictable. These are the main lines that have guided the
construction of the FC1 algorithm, the first one we made public in a
class of algorithms designed to face the exciting challenges of our
future.
We believe that FC1 ushers in the era of "post-alien cryptography"
and we hope that the debate we have stimulated will lead to realize
we need to have a vision oriented to the design of algorithms that
can defend human life from any possible threat.

3. Specification

3.1. Modular Multiplicative Inverse

Definition - For a positive integer n, and a (an element of Z) we say
that a' is a multiplicative inverse modulo n if

                    a*a' is congruent to 1 mod n

It can be proven [1] that:

   1. a has a multiplicative inverse modulo n if and only if a and n
      are relatively prime
   2. if a' exists, then it is unique

Computation - There are various methods to compute the inverse modulo
n in a polynomial time [2] [3] which, if implemented in languages
like Julia, having built-in support for Arbitrary Precision
Arithmetic, make it possible to calculate a' in a few fractions of a

second even for numbers with hundreds of thousands of digits.

A note on Julia Programming Language - With origins in the Computer
Science and Artificial Intelligence Laboratory (CSAIL) and the Dep.
of Mathematics, Julia is a programming language created in 2009 by
Jeff Bezanson, former MIT Julia Lab researchers Stefan Karpinski, and
Viral B. Shah, and professor of mathematics Alan Edelman. The Julia
programming language is a flexible dynamic language, appropriate for
scientific and numerical computing. Julia provides software support
for Arbitrary Precision Arithmetic, which can handle operations on
numeric values that cannot be represented effectively in native
hardware representations, but at the cost of relatively slower
performance. To allow computations with arbitrary-precision integers
and floating point numbers, Julia wraps the GNU Multiple Precision
Arithmetic Library (GMP) and the GNU MPFR Library, respectively.
In an APA application the size of the integer is limited only by the
available memory.

3.2. Description

Basic concept - FC1 essentially relies on the uniqueness of the
modular multiplicative inverse of a positive integer a modulo n and
on the fact that it can be calculated in a polynomial time. Here the
modulo is the main key which, due to the algorithm's design, can be
any positive integer, while the ciphertext is the modular
multiplicative inverse. The plaintext, once tagged with a hash, is
divided into blocks, the length of which is chosen by a random number
generator, converted into ciphertext and sent over an insecure
channel.

Keys - Keys to be kept secret and transferred over a secure channel
are primary key (the modulo) and secondary key. The latter represents
the length of a random string that is placed at the beginning of the
ciphertext.

3.2.1 Encryption

Hash - The very first operation that is performed is the computation
of a hash of the plaintext using the SHA-256 function. This tag is
then appended at the end of the text. The purpose is to ensure the
integrity of the data transmitted. We denote the plaintext with the
final tag by 'tplain':

$$tplain = plaintext \ || \ hash$$

Ciphertext initialization - With the value of the secondary key, a
random string is created which we denote by 'startpad'. This is the
initial ciphertext:

                         c = startpad

   Fencrypt - A main function named 'fencrypt' has the task of
   controlling the flow, switching between different sections of the
   algorithm in relation to a certain threshold value of the length of
   the tagged plaintext that still remains to be encrypted. The
   threshold is fixed at 1.5 times the length of the modulo.

   Frand - In the first part of the algorithm fencrypt calls a random
   number generator in a given range, which we denote by 'frand'. The
   generated random integer represents the length of the i-block of
   tagged plaintext to be encrypted. We denote by |modulo| the length of
   the modulo. The frand function generates a random value between 1 and
   |modulo| − 3. From the length of the modulo, 3 bits are subtracted to
   define the upper limit of the random function because 2 bits space is
   used to append the leading and trailing 1 (see next point 'Fintgen').
   Moreover, since we want that the integer, whose modular inverse we
   are going to calculate, is less than the modulo, another bit is
   dropped:

                  $1 =< frandvalue =< |modulo| - 3$

   Fintgen - A leading and a trailing '1' are appended at each chunk of
   tagged plaintext whose length is randomly selected by frand function.
   The leading 1 is meant to make sure that the input of the function
   computing the modular inverse is a positive integer since the block
   could start with '0'. The trailing '1' serves to prevent the
   algorithm from blocking in the case of an even modulo and a tagged
   plaintext to be encrypted containing a long row of 0's. We denote by
   $tplain_i$ the i-block of tagged plaintext; then is:

                  $input_i = 1 || tplain_i || 1$

   Finv - Once the input has been prepared, it is possible to attempt to
   compute the modular inverse using the 'finv' function. If the input
   and the modulo are not coprime, finv cannot produce a result and it
   calls the main function fencrypt which calls frand again in order to
   try with a different random integer. Else, if they are coprime, the
   modular multiplicative inverse is computed in a polynomial time and
   passed to the next step.

   Fblockgen - If the modular inverse is computable, a function called
   'fblockgen' comes into play comparing the modulo length with that of
   the modular inverse generated by finv. If the lengths are the same,
   fblockgen does not modify the string:

   If $|modulo| = |finvvalue|_i$

                         output_i = finvvalue_i

   Otherwise, if the modular inverse length is less than modulo length,
   fblockgen adds one or more leading zeros so that the lengths match:

   If |modulo| > |finvvalue|_i

                       output_i = 0..0 || finvvalue_i

   Final step of the first part - The block created by fblockgen is
   concatenated to the existing ciphertext and the main function
   fencrypt is called.

   Second part: ciphertext finalization - When threshold is crossed, the
   finalization functions are called. They have the task of
   simultaneously calculating the last and the second-last block of
   ciphertext. This design solution is necessary to prevent the case the
   modular inverse does not exist for the last portion of tagged
   plaintext, with the consequence of blocking the whole encryption
   process. The last step involves adding a random final padding whose
   length must be less than modulo length. This final padding, that we
   call 'endpad', is actually a third key that we can consider inferred
   from the other two. It is automatically added by the encryption
   algorithm. In the subsequent decryption phase, the algorithm will
   recognize it as its length is less than that of the modulo and
   finally it will discard it without attempting to decrypt it.

   Encryption flowchart - A detailed flowchart of the encryption process
   is provided in our related work [4].

3.2.2 Decryption

   We omit a complete description of the decryption algorithm since it
   is trivial. Note that, once the whole tagged plaintext has been
   decrypted, it is checked, through the hash function, that the final
   tag is correct and that therefore the integrity of the data is not
   compromised.

3.3 Recommended Parameters Set

   Primary key - We recommend a minimum length of 501 bits. At the same
   time, we encourage the use of 50.000-100.000 bits keys to fully
   exploit the potential offered by the algorithm. To maximize the speed
   we suggest the use of a modulo having as factors non-trivial prime
   numbers. If, on the other hand, the aim is to create further problems
   for a potential attacker, we recommend the inclusion of some trivial
   factors such as 3, 5, 11 and so on. Remember that you can safely use
   an even modulo without absolutely slowing down the algorithm.

Secondary key - It has no upper limit and can even be 0.

4. Implementation And Tests

We have coded the algorithm in Julia Programming Language and tested
it using keys of different length, from 10.000 bits to over
1.000.000.000 bits. Both the code and some of the tests were recently
published in a paper, available on IACR [4].

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

The minimum recommended primary key length we have seen is 501 bits.
The maximum length is instead not defined because it depends on the
limits of the system on which the algorithm is run. In our tests we
went as far as keys of over one Gigabit, which means a length of over
a billion of bits. Now, if by hypothesis the attacker knew the length
of the key, the startpad was zero and he could have any information
about the content of the first block of plaintext, for a brute-force
attack he would have to try about $2^{1.000.000.000}$ different
combinations. Since the attacker does not normally know the length of
the key, assuming the startpad equals zero, the number of attempts
would be:

$$\text{SUM } [2^i] \text{ from } i = 501 - 1, \text{ to } i = 1.000.000.000 - 1$$

We denote by |maxmodulo| the longest key that a system can handle in
a 'reasonably short time' and by |minmodulo| the minimum recommended
length of the primary key.
Generalizing and assuming that |tplain| > |maxmodulo| we have:

$$\text{SUM } [2^i] \text{ from } i = |minmodulo| - 1, \text{ to } i = |maxmodulo| - 1$$

FC1 therefore provides an incredible grade of confidentiality,
compared to the standards currently in use, which makes it suitable
for facing the difficult challenges of the next future. As far as
integrity is concerned, it is ensured by adding a tag generated by a
SHA-256 function. In a future work we will discuss in detail other
possible attacks (such as the 'replay attack') and we will show how
FC1 is immune to them.

[7](#). **Informative References**

[1] Victor Shoup (2009) A Computational Introduction to Number Theory
    and Algebra, Cambridge University Press; 2nd ed.
[2] Michele Bufalo, Daniele Bufalo, Giuseppe Orlando (2021) A Note on
    the Computation of the Modular Inverse for Cryptography, Axioms
[3] Niels Moeller (2007) On Schoenhage's algorithm and subquadratic
    integer GCD computation, MATHEMATICS OF COMPUTATION
[4] Michele Fabbrini (2022) FC1: A Powerful, Non-Deterministic,
    Symmetric Key Cipher, Cryptology ePrint Archive, Report 2022/567
    https://eprint.iacr.org/2022/567

Author's Address

   Michele Fabbrini
   Email: fc1_id@fabbrini.org