Karacell 3

Stream-friendly symmetric block cipher based on the Subset Sum problem and a chaotic iterator of a known limit cycle

Stuart Christmas / Russell Leidich

June 15, 2014

Abstract

We describe a stream-friendly block cipher, Karacell, that rather being based on layers of apparent complexity that may belie hidden weakness, is based simply on a known and well-studied NP-Complete problem combined with a chaotic iterator of known cycle length. We assert that the algorithm that achieves this is also naturally hardened against attack by quantum computer, allows for extremely parallel operation and is thus as fast as the host hardware allows, is immune to known-plaintext attack, produces completely deniable ciphertext, and is also hardened against corrupted ciphertext, insofar as the message recipient is tolerant of error. The scheme also incorporates native modular hash support.

Overview of Paper

This paper is designed as a compliment to the technical paper available at www.Karacell.info, which primarily focuses on the implementation of the algorithm and directly references source code; however some text is taken from that paper and reproduced here.

This paper is written in six parts. Part One discusses the motivations and claims of Karacell, with short a reasoning behind the claims. Part Two gives a detailed description of the algorithm. Part Three looks at the claims in more detail, and offers more robust explanations. Part Four is a complexity analysis, and also highlights best known cracking methods. Part Five highlights some of the criticisms the system has faced, with rebuttals from the authors, and finally Part Six highlights the Karacell cracking contest, in which cracking a small 15-byte test file, encrypted with the smallest supported key, is significantly financially rewarded.

Part One

Motivation

Karacell was initially designed by an expert in quantum physics and supercomputing architectures, as a way of providing decades more security against quantum computing attacks than current symmetric systems. This is achieved by forcing large contemporaneous memory requirements by virtue of the large data tables involved, which are still small enough to fit in the CPU cache of a classical machine, adding to the overall speed of the algorithm. Whereas all symmetric systems known to the authors require squaring the key length to offer the same robustness against quantum computing attack, Karacell has no such limitations thus key lengths can be kept within the bounds of human memory, as opposed to being digitally stored.

Karacell was also designed to produce completely deniable ciphertext, to allow encrypted communication in oppressive regimes that ban such communication, with no recourse to the communicating parties. As such, the Karacell file format has no discernible bitlane bias in its header, body, or footer, nor any detectable association between 2 or more bits, absent knowledge of the key.

The system was also designed to be highly parallel in its operation. Karacell works on 4KiB blocks (or partial blocks when the situation arises), and the cryption masks can be pre-fabricated, so as to allow parallel execution as data is ready for cryption.

Karacell is simple and transparent in its cracking complexity. Many encryption systems are designed by combining many layers that look complex at the outset, in the hope of achieving synergies between each component. However in many cases one layer of complexity can negate another, and subsequent analysis of the system becomes difficult if not impossible, which is highly dangerous from a cryptographic perspective. Karacell by contrast is designed to be very clearly an implementation of the subset sum problem, which is known to be NP-Complete, and driven by a long-period chaotic oscillator of a known and well studied limit cycle.

By designing Karacell to be amenable to microtransactions of any size, it was also possible to design the system to encrypt any size packet on-the-fly, so whilst not being a true stream cipher in the academic sense, it has been demonstrated to handle real time voice and video traffic with no perceivable latency.

Part Two

Cryption Overview

Cryption is achieved by generation of a specific "XOR Mask", which is a string of bits unique by way of the Karacell algorithm to the key, an initial value (IV) and the current block number (the "Block ID"), and is independent of the plaintext. The string of bits is then XOR'd with the plaintext, as is typical with many encryption schemes, to produce the ciphertext, and vice versa to revert to plaintext.

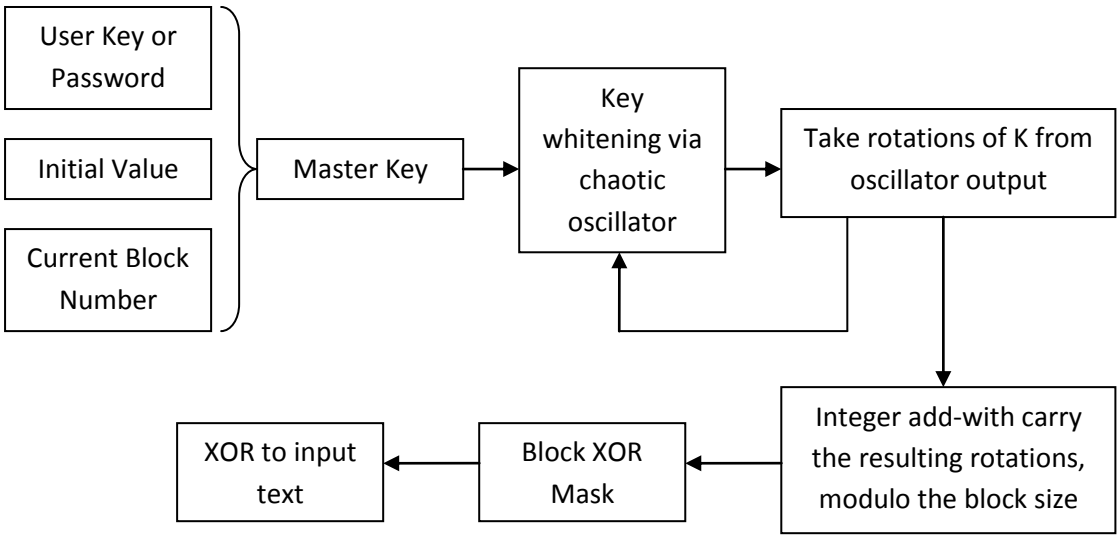
Karacell is very nearly a Feistal structure, aside from the addition and removal of a small header and footer, which are described later in this section. An implementation of Karacell that does not include the small header and footer would indeed be a true Feistal structure.

The Karacell Table

At the heart of the scheme is a large data table, known as the Karacell table (or 'K'), which is fixed in the specification and thus for all implementations of Karacell. The table consists of 2^{16} bits of equal numbers of 1's and 0's, re-arranged by a true random number generator. Furthermore, the table is considered as a ring, such that the bit in position 0 is effectively next to the bit in position 65,535. This allows for rotations of the table, of which there are 65,535 unique such rotations. These rotations are relative to a "zero-point" that is defined in the specifications.

The core of the algorithm selects various rotations of K, where the number of rotations selected is a function of key size, and is between 22 and 106 rotations. The reason for these numbers is described in Part Four at the end of the paper.

The general workflow is as follows, with each block in the diagram being explained further below. A more comprehensive diagram is given at the end of this section, following the explanation.



To begin, we assume the master key has already been generated from the three inputs. The Master key is always a 512 bit field, unique to its three inputs . The schema of this portion will be detailed later.

Key Whitening and Generating Rotations of K

The Master Key is then run ten times through a lag-0 Marsaglia oscillator, also known as a multiply-with-carry oscillator, by taking the lower 256 bits as the starting value (X_0) and the higher 256 bits as the initial carry, C_0. Finer details of the Marsaglia oscillator will not be discussed here, as it can be found on Wikipedia under http://en.wikipedia.org/wiki/Multiply-with-carry

Its general form is:

$$x_n = (ax_{n-r} + c_{n-1}) \bmod b,\ c_n = \left\lfloor \frac{ax_{n-r} + c_{n-1}}{b} \right\rfloor,\ n \geq r,$$

Where 'a' and 'b' are carefully chosen constants. Specifically, we choose b = 2^{256} to make the modulo operation natural to CPU registers, and we choose a = (b - 0xE66FAD12). This choice of the multiplier, 'a' is such that 'a' is a Sophie-Germain prime. The reason for this is expounded under the Wikipedia entry and also in George Marsaglia's original description of the oscillator, but in short having a multiplier that is a safe prime or a Sophie-Germain prime allows for easy calculation of the limit cycle. As far as this limit cycle is concerned, the largest calculated on Wikipedia is when b = 2^{256} and (ab-1) is a safe prime (which is a prime of form 2p+1, where conversely p is a Sophie-Germain prime), with 'a' also constituting 256 bits, is

$$2^{255}(2^{256}-9166)-1 \sim 2^{511} \sim 6.7 \times 10^{153}$$

Thus we assert the limit cycle of the Karacell oscillator is not a cause for any cryptographic concern, given the starting point on the oscillator is unique to the key, the Block ID and the IV, and we have the usual provision that the same IV should never be used with the same key more than once.

It should be pointed out at this stage that many references to this oscillator mention that it is not cryptographically secure. These statements refer to when it is used as a pseudorandom number generator, typically in the form of a lag-n oscillator. It is important to understand that in this case we are not using it for cryptographic random number generation, but to optimize (note: not maximize) the entropy of the Hamming distance between the master key and the first set of rotations of K, and thereafter between subsequent rotations of K. We also rely on the irreversibility of the oscillator absent knowledge of the specific rotations of K that were chosen, the exact order in which they were chosen, whether any repeated rotations were filtered, and how many iterations of ten cycles were used to reach the output state. This is explained in more detail in Part Four, and constitutes a one-way function within Karacell.

The reason for running the oscillator ten times comes from deep analysis by the authors. The results of this analysis show that the oscillator exhibits "scintillating entropy", with maximum entropy occurring almost always at 10 iterations, then 16, then 19, and beyond. In the interest of fast computation, the lower of these was chosen. The full analysis can be seen here:

http://leidich-message-digest.blogspot.sg/2013/02/harder-than-diehard-scintillating.html

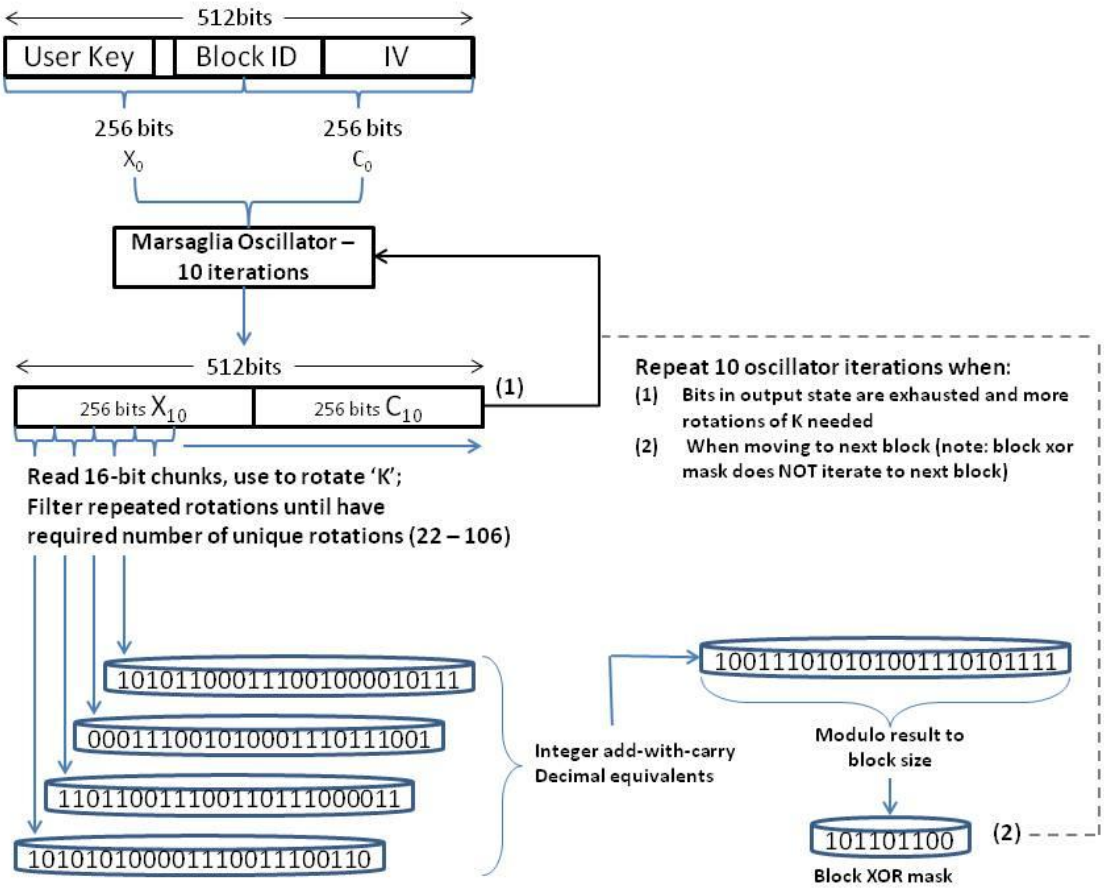
The output state of the oscillator after ten iterations is then an X_{10} of 256 bits, and a C_{10} also of 256 bits. Considering these as a continuous 512 bit field, i.e. the concatenation $[X_{10}||C_{10}]$, a 16-bit value is read off starting at the LSB, giving an integer between 0-65535. The Karacell table is rotated by this amount from its pre-defined "zero-point", and stored.

The next 16 bits are then read off, and the same process occurs. However, if the same rotation has occurred before, it is ignored, as repeated rotations weaken the resulting relationship to the subset-sum problem on which the scheme is based.

This process is repeated until the required number of rotations is found, as a function of the key size mentioned previously. Since the 512 bit field can only yield at most 32 such rotations, when the $[X_{10}||C_{10}]$ concatenation is exhausted, it is run through the Marsaglia oscillator another ten times, yielding $[X_{20}||C_{20}]$. This is then walked in the same manner from the LSB to extract a further 32 rotations (or less in the case of repeated results), and so on until the required number of rotations is found.

These N unique rotations of K are then treated as integers, and the add-with-carry operation is performed on all rotations, with the result taken modulo the current block size. Considering again the bitwise representation of the final result, we treat this as the XOR mask that is used for cryption purposes of the block.

Of course, computationally the rotations can be added as they are found rather than storing every one, but this paper is concerned with the basic theory behind the scheme rather than it's computational implementation. This is shown in the following diagram:



Hashing

Karacell does not include authentication but does include modular hash support. Karacell supports hashes in the LMD family, of which some are simply for accidental error-detection purposes, and some are full and complete cryptographic hashes. Current source code directly supports LMD7, LMD8, or no hash. Details of these hashing schemes can be found at the links below. Support for other hashing schemes can also be incorporated if required, and the option for no hash is also included in the specification and the source code.

Hashes are always computed on blocks of size 4KiB, such that any smaller blocks including null blocks are padded with 0s prior to hashing. (Accelerated hashing could occur in the event of a short block, but this not currently optimized in the source). For a given block, the hash seeds are formed in exactly the same manner as the cryption mask, except that K is first rotated by 2^{15} bits (i.e., half way). Critically, there is no carry propagation from the 2^{15} cryption mask bits, into the next 2^{15} bits which are dedicated to hash seed formation; the additions could be performed in parallel. Only as many hash seed bits are generated as are required for a particular hash algorithm.

More information on the details and reasoning of the hash computations can be found in the full technical paper at www.Karacell.info. Details of the LMD family of hashes can be found at these links, the first reference being LMD4-6, the second being for LMD7 and the third LMD8.

http://leidich-message-digest.blogspot.sg/2012/04/the-lmd4-lmd5-and-lmd6-secure-hashes.html

http://leidich-message-digest.blogspot.sg/search?q=LMD7

http://leidich-message-digest.blogspot.sg/search?q=LMD8

Header and Footer

Karacell files formally begin with a header, although hashless operation is possible without one. The details of the header are in the technical paper and specifications, but in brief the header consists of the following fields:

Bytes	Description
32	Initial Value / Initialization Vector (IV)
8	Must Decrypt to Zero
8	Size Following
2	Build number of creator
2	Build number needed to decode
2	Reserved Zero
1	Hash Type
1	Hash Type Minus one

The 256-bit IV is the same concept as in any other cipher scheme, and the usual rules of never re-using an IV with the same key apply.

Must decrypt to zero is a 64 bit field that is zero before the encryption process. Thus, when decryption happens, Karacell checks this field has returned to zero. If not, then it reports an “incorrect key” error. This saves the recipient time should they have mistakenly entered an incorrect

key for a very large file. Although statistically insignificant one in every 2^{64} cases this will decrypt to zero by chance – however the worst consequence of this is the lost time to decrypt – the hash will still report an error.

This field has caused some controversy, with detractors saying that knowing that 8 byte field is zero in plaintext leaks information about the key and can be used to "unwind" the entire XOR mask back to the master key. We show in the cryptanalysis section in Part Four that not only is this not the case, but in fact we assume an attacker always knows reasonably large sections of the plaintext, and show that it is still not sufficient to attack the rest of the block, and much less backtrack to previous blocks and to the master key.

As a final note on this, if any cryption system fails if an attacker knows 8 bytes or less of the plaintext, that constitutes a serious problem, given that most file formats have constant or highly similar headers from file to file, and known signatures. Knowing these 8 bytes are all zeros in plaintext is of no consequence to attacking Karacell.

"Size following" states the bytes that follow this field, in order to check for pre-hash modifications, "build number" fields are used for version number control and enforcement, "reserved" is for future releases, and "hash type" can be 1,2 or 3 for no hash, LMD7 or LMD8 respectively.

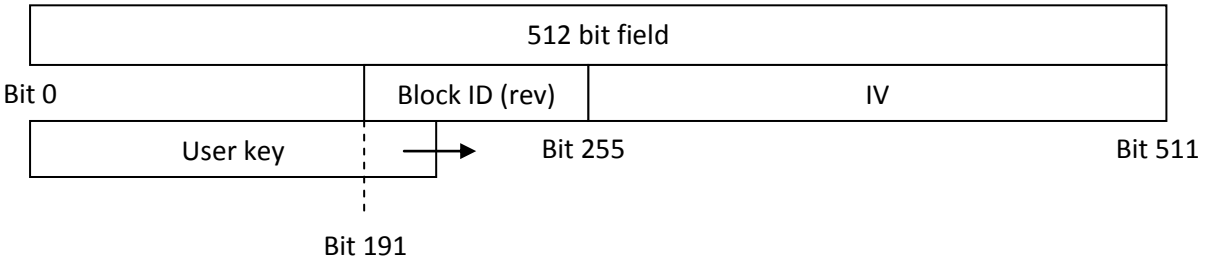
"Hash type minus one" is 1 less than the number of 32-bit integers in the hash. The only exception is that it must be 0 (meaning no hash) when hash type is 1. It must be checked for consistency against the hash type field. If the hash type is unrecognized, then the hash must be ignored (but its size still respected in payload size computation), which the caller can discover.

The footer of each block and subsequently each file is the hash value.

Master key generation

The master key that forms the initial input to the Marsaglia oscillator is a 512 bit field derived from the users key (or password), the IV, and the current block ID, which is a 64 bit number. There are many methods of producing this 512 bit field from the available inputs, and there are likely many optimal solutions. We selected a method that incurs minimal latency penalties, preserves as much entropy from the initial users key as possible, and allows the block ID to grow to large values without interfering with this entropy.

Consider the following 512 bit field. The regions of the three inputs are shown, and in the case of overlap, the XOR operation is performed on the overlapping values.



In the case of a user key between 191 and 511 bits, the XOR operation is performed as mentioned. In the event the user key is between the minimum 121 bits and 191 bits, there will be zero bits between the end of the user key and the start of the block ID. This may initially appear insecure, however we make two assertions. (1) this provides no more insecurity than is already risked by using a 121 bit key, and (2) that the entropy afforded by the ten iterations of the Marsaglia oscillator completely obliterates any traces of this small section of zeros, and the final output from the oscillator is still entirely key-dependant.

Additionally, the Block ID is stored as its bitwise reverse (not inverse!). This allows for extension should the Block ID need to exceed 2^{64}, without interfering with the low bits of the users key, to afford more uniform entropy coming from the key.

Part Three

Quantum Computing

The first claim to expound upon is that of resistance to quantum computing attacks. The theory is based on the complexity of maintaining ever larger states of qubits in quantum entanglement. Maintaining several qubits already requires incredible amounts of cooling, magnetic shielding, and vacuum conditions. Adding another qubit disproportionately increases these requirements verses the classical analogue.

How many qubits would be required to attack Karacell? The quantum computer must, at minimum, have to store the current addition of all rotations of the Karacell table, and the next rotation, at all times. It must also have to, in the meantime, store the current output state of the Marsaglia oscillator. Even at this point, we are at (65536 + 65536 + 512) = 131,584 bits. This does not include maintaining other things such as the key, block number tracking, and the actual algorithm itself. And we of course take it as known that quantum storage cannot be linked to classical storage for this purpose, as such a linkage would immediately cause collapse of the wavefunctions and trivialize the use of a quantum computer in the first place.

The current state of quantum computing is that of highly refuted claims of 128 qubits being held in superposition, to help solve a very narrow range of combinatorial problems. Even if this is true, moving to being able to hold each additional qubit beyond this represents an enormous engineering challenge far greater than for the previous qubit. So even the far too generous estimation of needing 131,584 qubits to test Karacell puts a realistic quantum attack decades or more away. If that day approaches, the defence is simply to increase the size of the Karacell table by an order of magnitude or so. Moores Law will certainly be able to accommodate that on classical machines by that time. But for the foreseeable future, a 130KiB quantum computer remains a decades-distant pipedream for even the most optimistic quantum expert.

Insofar as key length is concerned, quantum computing attacks on symmetric algorithms, to the best of our knowledge, effectively reduce the key length by a square root factor. Thus, a 256 bit key becomes worth only 16 bits (thus totally insufficient). To maintain the security of a 256 bit key, a 65,536 bit key must be used. This is beyond the scope of human memory, necessitating the key to be

"written down", either digitally or in analog form, which immediately poses security issues, and in almost all cases causes a massive slowdown of the algorithm. Asymmetric key exchanges could be used, but it is assumed a general purpose quantum computer will have no issue dealing with this, unless the keys are so large that the slowdown in the exchange and the algorithm becomes prohibitive.

Karacell, by contrast, puts the burden on the quantum computer to manage the large data tables, and until a machine of sufficient size is constructed (which we assert to be decades away in the most optimistic case), a key length of memorable size can be maintained, with no loss of speed. And as mentioned above, if that great day for quantum computing does arise, these data tables can be increased in size by an order of magnitude or so, which will pose no issue as classical computers are already capable of handling this situation.

Deniability

Karacell has been designed to operate in the venue of repressive regimes which seek to prevent encrypted communication, chiefly by means of packet filtering. As such, the Karacell file format has no discernible bitlane bias in its header, body, or footer, nor any detectable association between 2 or more bits, absent knowledge of the key. (It does, however, have a minimum size, namely, that of its header, which is not a significant impediment to deniability. In the worst case, smaller junk files could be sent at random times, in order to achieve a Poisson distribution of file sizes, and thereby obscure the misuse of Karacell to send multiple null packets). Under circumstances where all highly entropic packets are dropped for fear that they might be encrypted, one might implement a verbose aliasing system which maps dense binary to plausible uncompressed text, audio, or graphics streams, then converts them back to binary upon receipt; a form of Karacell-based stenography if you will.

The claims of deniability have been testing by asserting that a truly deniable packet is indistinct from random noise. Various Karacell-encrypted files, data streams and encapsulated IP packets were tested with the standard "DieHard" tests for randomness, and passed dramatically. The results do not need to be reproduced here, as for one it cannot be proven that Karacell data was indeed used versus pseudorandom data from a physical source, hence the reader is encouraged to download and compile the source code, encrypt data, and run the battery of tests themselves. The details of each test can be found here: http://en.wikipedia.org/wiki/Diehard_tests and a suite of DieHard tests can be downloaded here: http://manpages.ubuntu.com/manpages/precise/man1/dieharder.1.html

Parallel operation

One of the key advantages to Karacell is its amenability to parallelization. Given the almost ubiquitous nature of parallel computing and multi-threading today, it seems counter-intuitive to use heavily serialized cryptographic systems.

Karacell owes its parallel operation to its independence from the plaintext. Most symmetric systems today use the plaintext as part of the cryption process, and also feed-forward that results to the next block, heavily serializing operation. Many of these can operate in so-called "counter mode" to offer a closer analogue to truly parallel operation, however there are strong cases made in the literature that such a mode of operation weakens the system, or at best is not well studied.

Because Karacell operates independently from the plaintext, XOR masks can be produced in advance of the arrival of data, at which point the mask can be XOR'd with as much data as is in the buffer, all in parallel; thus as much data can be crypted in parallel as can be stored in the memory buffer and handled by the number of cores and multithreading architecture. Thus the effective speed of Karacell scales with the advancements in parallel computing, where as serialized systems can only scale with increases in CPU clock speed (which of course benefit Karacell equally).

In the event the masks are not generated in advance, cryption can only happen as fast as the masks are created. However, several parts of the core engine can also operate in parallel, making this process as fast as the host hardware allows. The parts of the engine that can occur in parallel are the reading off up to 32 rotations of K, and the addition of the non-repeated rotations. Whilst these steps are happening, another CPU core can cycle the output state of the oscillator through another ten iterations , and then perform the parallel operating of reading off 16-bit sections, filtering and adding another 32 rotations, whilst a third core has already taken this output state and is running another ten iterations and so on. Eventually the first core will have generated it's XOR mask, and can take the output state of the current CPU core and repeat the cycle.

Known-Plaintext attacks

Known plaintext can be XOR'd with the ciphertext to reveal a portion of the cryption mask. Part Four on complexity analysis and cracking methods shows how this falls short at the point of landing the attacker at the foot of the subset sum problem, which we assert is the core strength of the algorithm in the first place. This attack is detailed in that section.

Corrupted Ciphertext

Files can be corrupted for many reasons, such as network errors, hard disk miswrites, interference on storage devices from optical or magnetic sources, as well as bit-flips occurring from cosmic neutrinos. In the case of serialized cryption schemes that depend on feeding-forward plaintext and/or ciphertext from one block to the next, even one bit-flip can mean the data is unrecoverable from that point on.

The case of Karacells' independence from the plaintext means that in such a situation, the bit in question (and hence the whole byte, word, double-word or whatever unit size is used) will be corrupt, but the rest of the file can still be recovered.

Of course, this would cause a hash error for the block (and hence the whole file), and in the majority of cases this would warrant rejection of the file (or request of a re-send) as the error may have been from malicious interception. However there are a great many cases in which the recipient may not be concerned with such an issue, or may be able to tell from the nature of the decrypted file whether the hash error was caused by malicious alteration or simply a random corruption. In these cases, Karacell has the strong advantage that a single bit-error or more does not cause the entire file to be unrecoverable.

Part Four

Complexity analysis

For a proper analysis we assume the most generous situations for the attacker and the most conservative for Karacell. Thus, we assume the attacker is armed with a section of known plaintext.

A hacker can easily discover a portion of a mask by simply XOR'ing this plaintext to the corresponding piece of ciphertext. Given the resulting mask fragment, the first part of the task is then to discover the rotations of K which gave rise to that mask fragment. This is tantamount to the subset sum problem, expounded on Wikipedia under "Subet Sum". Finding an incorrect set of rotations that coincidentally add up to the same fragment is useless.

The attackers task is made orders of magnitude harder, as they must also determine the number of rotations that are used, which is a function of key size. In the spirit of proper analysis stated above, we first assume the hacker knows the key size, and that the key size is the smallest Karacell supports. Thus, the attacker knows they must select a correct set of 22 rotations from a possible 2^{16}.

The attacker must select 22 rotations at random, perform the addition, and test the resulting block-sized mask to see if it produces a sensible output, at which point the attacker can be sure they have cracked the block. One could argue that in the vast majority of cases, given the small block size relative to most file sizes, determining a "sensible" output is near impossible, but again we offer the attacker the most generous situation and ourselves the most conservative, and assume that they can always be sure they have found the correct mask fragment.

But how realistic is finding such a fragment? The attacker must select the correct 22 rotations from a possible 2^{16}, which is $1/[22\ {}^{n}\mathbf{C}_r\ 2^{16}]$, in which ${}^{n}\mathbf{C}_r$ denotes the combination function, also expounded on Wikipedia under "Combination".

$[2^{16}\ {}^{n}\mathbf{C}_r\ 22]$ is given by Wolfram Alpha as:

8,133,190,060,616,455,136,766,836,726,183,441,930,279,044,436,931,442,220,772,110,590,856,824,532,484,309,811,200

Or 8.1×10^{84}, which is about 10,000 times the number of atoms in the visible universe.

$1/[22\ {}^{n}\mathbf{C}_r\ 2^{16}]$ is thus 1.229×10^{-85}. Considering the case of brute forcing the key, the probability of choosing the correct corresponding 121-bit key is $1/(2^{121}) = 3.76 \times 10^{-37}$.

Thus we have already shown that even giving the attacker all possible advantages, it is still better (by almost 50 orders of magnitude) to brute force the key than to try and "break" Karacell, even given a known portion of plaintext.

But we're not done yet. If we assume the hacker is the luckiest person in the universe (or finds a more efficient solution to the subset sum problem, which is tantamount to proving P=NP and thus wins them the Millennium prize), and does manage to find the 22 correct rotations from the 2^{16} possible, they can decrypt the block, of which they already knew a large chunk to begin with (as it was our assumption). The value of cracking a 4KiB block of which a portion is already known is

debatable, but the task now is to backtrack this knowledge to the previous block. If this can be continued though all blocks, the hacker can backtrack to the master key.

But even if they do find those correct set of rotations by a divine miracle, the task now it to re-create the last output state of the Marsaglia oscillator. Whilst no longer part of the subset sum connection, this task is perhaps even more difficult, if not impossible. Whilst the addition of the rotations is commutative, to recreate the oscillator output state the attacker must know in what order those rotations occurred, in order to correctly re-populate the 512-bit output state of the oscillator.

Assuming no repeated rotations were encountered and filtered, there are already [22 ${}^{n}\mathbf{P}_{r}$ 32] possible ways to arrange those rotations back onto the 512 bit output state. [${}^{n}\mathbf{P}_{r}$] in this case represents the permutation function, expounded on Wikipedia under “Permutation”.

[22 ${}^{n}\mathbf{P}_{r}$ 32] is 72,511,804,710,563,693,278,003,200,000, or 7.25×10^{28}. The probability of the attacker choosing this correctly is 1.38×10^{-29}, which we assert is still unlikely enough as to be considered impossible, especially when combined with the $\sim 10^{-85}$ chance of picking the correct 22 rotations in the first place. However, if no repeated rotations were filtered, and the attacker has indeed managed to guess the right order, they have populated 22 * 16 = 352 bits of the required 512. Thus there are still 10 x 16 = 160 bits of oscillator state that are fundamentally unknown. These will have to be brute-forced, so it is therefore obvious that brute forcing the 121 bit key is more efficient than brute-forcing the remaining 160 bits of oscillator state.

Of course, this gives the hacker the benefit that they know that no rotations were filtered, which is fundamentally unknowable. Given that provision, it is also fundamentally unknowable how many iterations of 10 cycles the oscillator went through before reaching that final output state. Given these two fundamentally unknowable quantities leads to a complexity explosion that simply cannot be calculated.

So we have shown that given the most generous situation to the attacker and the most conservative to Karacell, and given the attacker knows a portion of the plaintext, their first task is tantamount to the subset sum problem, which is only solvable in non-deterministic polynomial time, and 50 orders of magnitude “harder” than brute-forcing the key. Whilst this is enough to show robust encryption, another layer of astronomical complexity is encountered when needing to re-create the latest oscillator state, given unknowable quantities such as if and when repeated rotations were filtered, and the number of 10-cycle iterations that were subsequently required to produce the required number of rotations.

Best known cracking method

So we have linked the complexity of the first stage of an attack to the subset sum problem, which is one of the NP-complete class of mathematical problems and hence extremely well studied. Hence we assert that mathematicians have been implicitly attempting to crack Karacell for a great many years.

Here we analyse the best known solution to date, which still falls many orders of magnitude short of simply brute-forcing the key.

In 1972, Horowitz and Sahni produced an efficient algorithm for the solution of Subset Sum. It runs in $O(2^{0.5N})$ for N possible addends. Since then, several accelerated approaches have been discovered for special cases. In 2013, Bernstein, Jeffery, Lange, and Meurer produced a quantum computer algorithm which runs in roughly $O(2^{0.24N})$, better than a square root factor improvement.

However, there is a faster approach specifically tailored to Karacell, which we call "Sparse Horowitz and Sahni". This is more applicable to an attack on Karacell as it focuses on cases in which the possible addends form a very sparse subset of all possible addends. Even in the case of the largest possible Karacell key, there are still only 106 possible rotations from a possible 2^{16}, so sparse Horowitz and Sahni is the optimal case.

The algorithm runs in $\mathbf{O}((2^{16}\ {}^{n}\mathbf{C}_r\ R/2) / (R\ {}^{n}\mathbf{C}_r\ R/2))$ time for R possible rotations. We have been unable to find or produce a plausible quantum acceleration.

The choice of rotations to use based on key length is based on the above best known cracking method, to ensure that cracking complexity exceeds the Sparse Horowitz and Sahni complexity, whist maintaining computational efficiency.

Given B, the bit position of the most significant bit of the key, then we can use the Sparse Horowitz and Sahni complexity to estimate the number of rotations, R, required such that the latter is a larger space than the key space. In other words, we can find the minimum number of rotations which will make it harder to reverse engineer the mask than to try every key. Subject to this constraint, we want to minimize R in order to minimize cryption latency. (For the sake of simplicity and better performance, odd numbers of rotations are not supported.) This is best shown by example.

Assume that we have a key, X, whose most significant bit position is 315. Thus:

$2^{315} \leq X < 2^{316}$

We solve the Sparse Horowitz and Sahni complexity for a result of 315, and find that:

$\text{floor}(\log_2((2^{16}\ {}^{n}\mathbf{C}_r\ 60)/(60\ {}^{n}\mathbf{C}_r\ 30))) = 315$

Thus 60 rotations of K. Trying the next even number of rotations we find that:

$\text{floor}(\log_2((2^{16}\ {}^{n}\mathbf{C}_r\ 62)/(62\ {}^{n}\mathbf{C}_r\ 31))) = 324$

So R = 60 might be sufficient, but we can't know that without much more number crunching. So we round up to R = 62, meaning that we need to generate 62 unique rotations for the sake of producing each XOR mask from block 0 through (N+1).

These calculations are fixed for fixed key lengths, so we pre-calculate them for supported key lengths and include them in a lookup table that is part of the Karacell specification. For key lengths of 121 to 512 bits, we find numbers of rotations from 22 to 106, increasing in a roughly linear fashion.

Although this approach to the subset sum problem has gone unoptimized since 1972, including a viable quantum approach, the numbers of rotations used is upward-biased, such that highly significant optimization to Sparse Horowitz and Sahni would have to occur to offer a faster cracking

time. In this eventuality, the lookup table of rotation counts can simply be updated and pushed as a critical update to Karacell.

Part Five

Common concerns and criticisms

"This looks like a pseudorandom number generator. There was once a cipher based on the Mersenne Twister which was weak. Therefore this is weak."

We address this by saying it's no more a pseudorandom number generator than any other encryption scheme. As we state above in the section on deniability, ideally the output of a cipher should appear as random data. Karacell appears to more closely resemble a PRNG since it generates its XOR masks without aid of the plaintext, by design. Karacell differs from a PRNG in the traditional sense as the latter can be reversed given the internal state of the machine and the current output. As we show at length in Part Four, Karacell cannot be reversed in this way, at least insofar as a great many orders of magnitude harder than brute-forcing the key.

"There was once a system based on the knapsack problem, which is an incarnation of subset sum, and that was shown to be weak. Therefore this is also weak."

This refers to the Merkle–Hellman knapsack cryptosystem, which can be found on Wikipedia. The article states it is based specifically on subset sum, where as in fact is it based on the broader knapsack problem. A brief review of the scheme on Wikipedia readily shows it does not relate to Karacell in the slightest, is an asymmetric system rather than symmetric, and the only commonality is the use of the keyword "subset sum". The algorithms are entirely distinct and dissimilar.

"People have been showing me [false] proofs of P=NP for years, this is no different."

Karacell makes no claims that P=NP or that P!=NP. All we state is that if a polynomial time algorithm is found to decipher the ciphertext quicker than brute-forcing the key, that is tantamount to proving P=NP since subset sum is NP-Complete. Or conversely, if someone ever does prove P=NP, the strength of Karacell is greatly diminished.

"The Marsaglia Oscillator is not suitable for cryptographic purposes"

We explain this in Part Two. To reiterate verbatim, these statements refer to when it is used as a pseudorandom number generator, typically in the form of a lag-n oscillator. It is important to understand that in this case we are not using it for cryptographic random number generation, but to optimize (note: not maximize) the entropy of the Hamming distance between the master key and the first set of rotations of K, and thereafter between subsequent rotations of K. We also rely on the irreversibility of the oscillator absent knowledge of the specific rotations of K that were chosen, the exact order in which they were chosen, whether any repeated rotations were filtered, and how many iterations of ten cycles were used to reach the output state.

"If you know that bytes 33-40 in the header are zero, you can backtrack the key and the whole thing unzips."

Again this was explained in Part Two, but to reiterate, not only is this not the case, but in fact we assume an attacker always knows reasonably large sections of the plaintext, in the cases of files with known headers or at least known statistical correlation. If an encryption scheme falls down if 8 bytes of the plaintext are known, you don't have an encryption scheme.

Part Six

Cracking contest

The Karacell website, Karacell.info, includes compilable source code, and a test file. Anyone able to demonstrate cracking the file and explaining the method used to do so, wins a prize that increases the longer it remains unclaimed, beginning at US$10,000. This is always divided in half, with half the prize money being awarded for cracking the file, and the other half for recovering the key used for encryption. Full contest rules are available at the same website.

References

Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer paper:
http://eprint.iacr.org/2013/199.pdf

Horowitz and Sahni paper:
http://dl.acm.org/citation.cfm?doid=321812.321823

Source Code, Contest Rules, and technical paper:
www.karacell.info