Group E

Cryptography Project 1

[redacted]

September 11, 2020

We at IMC were approached by your company concerning issues involving internal security breaches. Specifically, you expressed having issues where sensitive information has been sent via text to the wrong recipients. To keep your data and information secure, we have come up with a nontrivial method of encrypting these plaintext English messages. In this report we will go over the encryption process, as well as the mathematical underpinnings which keeps your data safe.

Before defining the theory for the current system, some definitions must be provided. First, a *dynamical system* is (loosely) a function $\phi_t(x) = \phi(t, x)$ mapping $T \times M \to M$ satisfying $\phi_0(x) = x$ and $\phi_{t_1+t_2}(x) = \phi_{t_2}(\phi_{t_1}(x))$. In other words, a dynamical system provides the flow of each point x under a time map, where the flow begins at x and can be naturally partitioned into subflows. A *chaotic* dynamical system is here defined to be a system which is *topologically mixing* and has *sensitive dependence on initial conditions*. The former means merely that, given two subsets A, B of M, if we flow the points of A long enough (say, for some time N) they will eventually intersect B, and at least one flowed point of A will continue to intersect B for all time after that (formally, $\exists N$ such that, $\forall n > N$, $\phi_n(A) \cap B \neq \varnothing$). We think of the flow as having 'mixed' the set A and B.

The latter term, namely *sensitive dependence on initial conditions*, merely means that nearby (but unequal) points get split apart (for at least some time) as the system evolves. Finally, a *discrete* dynamical system is merely a dynamical system such that the time component is given by iteration of some related function $\phi : M \to M$ (with $T = \mathbb{Z}$ or $T = \mathbb{N}$), i.e. $\phi_0(x) = x, \phi_1(x) = \phi(x), \phi_2(x) = \phi(\phi(x))$, and so forth.

With these definitions in mind, the basic encryption idea can be outlined as follows: take some dynamical system with an unknown parameter (representing the key), canonically embed the message into some points of the space domain for the system, flow the system forward for some time, determine the points where the embedded message flowed to, and turn these points back into some encrypted, alphanumeric message. To decrypt the message all one must do is re-embed the ciphertext and flow *backwards* in time until a readable message appears. To prevent precision or floating point errors with decryption one should force the system to be discrete, and to ensure decent encryption the system should be chaotic.

Fortunately, there is a very simple system which satisfies these constraints: *The Discrete Cat Map*, which we will hereafter refer to either as 'the cat map' or with the letter Γ. The simple version of the cat map we need here is defined to be the function $(\mathbb{Z}_N)^2 \to (\mathbb{Z}_N)^2$ given by $\Gamma(x, y) = (2x + y, x + y) \bmod N$ *(Note: throughout the rest of this paper, the letier N will be used only for this purpose)* Importantly, the cat map is known to chaotic and mixing, though proofs of these would take us far afield in this paper.

The cat map has another wonderful feature which makes it useful for a simple encryption system: it is periodic. What this means is that there exists some $k \in \mathbb{N}$ such that $\Gamma_n(x, y) = (x, y)$. As such, we can keep iterating the system and will get back what we started infinitely many times; this is known as Poincaré recurrence, and k is known as the return time. The importance of this feature is that instead of needing to flow backward in time to return to our original state, we can instead flow forward in time. Interestingly, the return time depends heavily on N, with no simple formula existing. In fact, it is not even true that as N increases the return times increases (for example, $N = 104$ yields $k = 25$, while $N = 124$ yields $k = 15$). What *is* known is that $k \leq 3N$, so the return time cannot be radically higher than N. Because of this, we can view the return time as being something easy to explicitly calculate from N, meaning the cat map has

only the single changing parameter N. With the motivation in the prior paragraph in mind, we will thus let N represent the key for our cryptosystem.

All that remains is to specify a way to embed and extract alphanumeric data into lattice points of $(Z_N)^2$ and extract/re-embed the encoded message. There are plenty of ways to do this, but the following is the way we have chosen to use: first all spaces, punctuation, and another unnecessary data is removed, leaving only letters, numbers, and symbols. We make everything alphabetic, representing numbers or symbols with a sequence of appropriate letters distinguishing them from the alphabetic text. For example, we might represent the number '2' with the letters 'qqtwoqq', with the pair of double 'q's signaling that the text contained within is a description of the number, though the choice of signifier and encoding can be chosen arbitrarily so long as it is readable and will not be confused with surrounding text. All letters are then made lowercase, and are made numeric in the natural way (a = 1, b = 2, ..., z = 26). What we have now is an ordered collection of integers k_n (where k_n is the integer in the nth position), and to embed these integers as lattice points we map each integer k_n to the point (k_n, n). At this point we need to view these points as living in $(\mathbb{Z}_N)^2$ for some $N \in \mathbb{N}$. Since our messages will be split into blocks of size at most 160 it suffices to choose $N > 160$.

For the sake of proper cryptosecurity in your forthcoming appliction, the choice of N should be made very large (and should probably be ensured to have a period at least as large as N), though for the sake of testing purposes and demonstration it suffices to use three digit keys (all of which should be greater than 160). (*Note: This also ensures the 800 character requirement is met*). We then flow the system, resulting in a sequence of lattice points. Since $N > 26$, many of these points will have entries far larger than 26, so we need to find a way to (invertibly) encode these points alphanumerically. The way we will do this is by taking considering each lattice point, splitting the first lattice point into single digits, replacing each with the appropriate letter, then concatenating together with the second entry in the lattice point. As an example, the lattice point $(169, 967)$ would be replaced by the alphanumeric string 'afi967' (the digit 0 will be represented by 'z' if it appears). We then concatenate all lattice points, preserving order (see the example below for more details). Since this is clearly invertible, we can later re-embed our encoded message as lattice points.

All the hard work is now done; all that remains is to actually iterate the system and encrypt the data. The amount of time the system is flowed does not matter, so long as the time we flow is not the return time. (*Technical note: for some N the message will be encoded backward at half the return time, so one should not flow to this point either. Nevertheless, for very large N most return times will be quite long, making the choice of these points quite unlikely. If it still occurs, just choose a different amount of time to flow and repeat.*) We now provide a simple example using the text 'Bob ate 2 rolls', with $N = 124$. (*Note: the chosen example message is far shorter than 124 characters, so this choice of N will work here. It was chosen for this example only due to its relatively short period so that a reader could verify the calculations if desired*).

First, the text 'Bob ate 2 rolls' is made alphabetic and lowercase, becoming 'bobateqqtwoqqrolls'.

This string is then made numeric, becoming the ordered list

$$\{2, 15, 2, 1, 20, 5, 17, 17, 20, 23, 15, 17, 17, 18, 15, 12, 12, 19\}$$

which become ordered lattice points as follows:

$$\{[2,0], [15,1], [2,2], [1,3], [20,4], [5,5], [17,6], [17,7], [20,8],\\ [23,9], [15,10], [17,11], [17,12], [18,13], [15,14], [12,15], [12,16], [19,17]\}$$

The cat map is then run on each lattice point in order, resulting in the sequence:

$$\{[4,2], [31,16], [6,4], [5,4], [44,24], [15,10], [40,23], [41,24], [48,28],\\ [55,32], [40,25], [45,28], [46,29], [49,31], [44,29], [39,27], [40,28], [55,36]\}$$

Running the cat map four more times results in the sequence:

$$\{[54,110], [26,115], [40,54], [6,33], [16,120], [100,73], [107,23], [38,57], [112,8],\\ [62,83], [25,49], [10,69], [65,103], [85,68], [121,61], [33,54], [88,88], [22,11]\}$$

We will use this for our encoded message. We first split and alphabetize the digits of the first entry of each coordinate (recall 0 is mapped to 'z', since it should represent letter before 'a'):

$\{[ed, 110], [bf, 115], [dz, 54], [f, 33], [af, 120], [azz, 73], [azg, 23], [ch, 57], [aab, 8],$
$[fb, 83], [be, 49], [az, 69], [fe, 103], [he, 68], [aba, 61], [cc, 54], [hh, 88], [bb, 11]\}$

We then concatenate everything together into a string, yielding:

ed110bf115dz54f33af120azz73azg23ch57aab8fb83be49az69fe103he68aba61cc54hh88bb11

When we wish to decode this message, we can of course invert the process back until we get the sequence

$\{[54, 110], [26, 115], [40, 54], [6, 33], [16, 120], [100, 73], [107, 23], [38, 57], [112, 8],$
$[62, 83], [25, 49], [10, 69], [65, 103], [85, 68], [121, 61], [33, 54], [88, 88], [22, 11]\}$

at which point running the cat map ten more times will return the original sequence

$\{[2, 0], [15, 1], [2, 2], [1, 3], [20, 4], [5, 5], [17, 6], [17, 7], [20, 8],$
$[23, 9], [15, 10], [17, 11], [17, 12], [18, 13], [15, 14], [12, 15], [12, 16], [19, 17]\}$

which can be quickly decoded into the original message.

All that remains it be specific on how messages longer than 160 characters are broken up into blocks. The order taken will be to first make the message entirely alphabetic in the manner above (e.g. removing spaces and replacing numbers and symbols with text equivalents) and to then extract text blocks of length 133 until less than 133 characters remain, making this the last block. The same N is then used on each block for encryption and decryption purposes. The only exception to this rule is when extracting a block of size 133 would only partially grab the text expansion of a symbol (e.g. if the symbol '2' were expanded as 'qqtwoqq' and the 133rd character was the any character of that expanded symbol except for the last 'q'); in this case, the message is to be cut-off just prior to the symbol expansion, so the symbol will be sent it its complete form in the next message. Upon encryption these messages will not exceed 800 characters given the recommendation for a 3-digit choice of N greater than 160, since in this case each coordinate becomes, under iterations the cat map, at most a pair of 3-digit numbers. Since the ciphertext contains at most 133 pairs of 3-character strings, the length of the ciphertext for each block is at most 798 characters.

In summary, we have created a simple encryption system using Arnold's Cat Map. By using this chaotic dynamical system, we create a simple yet decently robust encryption of your sensitive data. Our solution offers a great deal of speed in encryption and decryption, since the encrypted message can be returned to its original state, given the key, merely by repeated iteration of the cat map. Being a chaotic and discrete system, our solution additionally offers increased security and precision over a generic choice of encryption via dynamical systems. With all this in mind, we feel our proposed solution will be more than sufficient to meet your security needs.