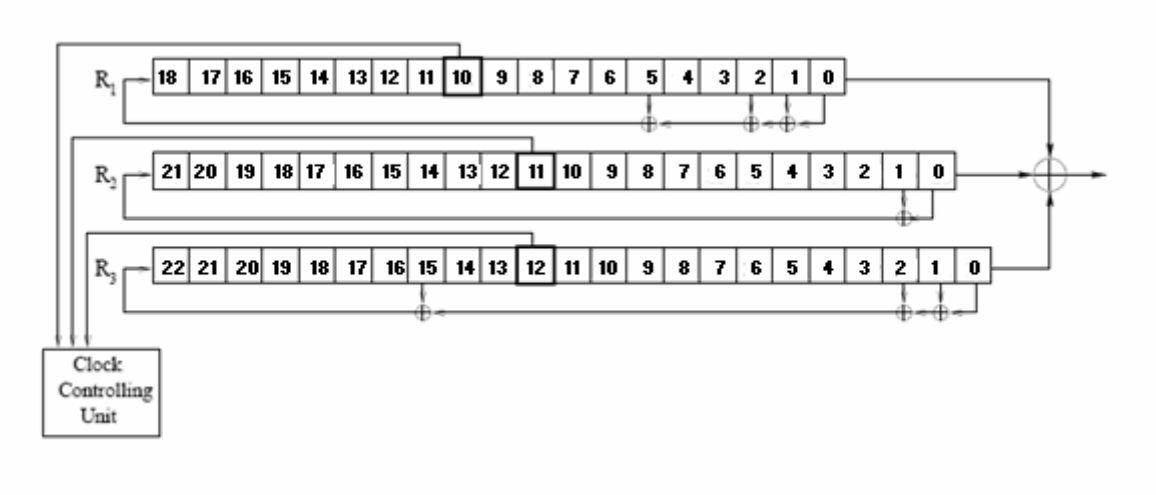
A Matrix Algebra approach to retrieve the Key in Stream Ciphers

A5/1 explained II

By Juan Chamero, juan.chamero@intag.org, as of April 2006

A serial encoding algorithm could be considered a “states’ machine”, with a complex scrambling process partially linear and partially non linear. It could be implemented via LFSR’s which stands for Linear Feedback Shift Registers as the ones depicted above to implement the A5/1 algorithm that generates ciphering “masks” of most mobile telephone (GSM wireless) in Europe and any other regions of the world.



Its feedback mechanism works controlled by the following Primitive Polynomials:

R1 (19 bits) controlled by the Primitive Polynomial: P19(x)= x^19 + x^5 + x^2 + x^1 + 1
R2 (22 bits) controlled by the Primitive Polynomial: P(22)= x^22 + x^1 + 1
R3 (23 bits) controlled by the Primitive polynomial: P(23)=x^23 + x^15 + x^2 + x^1 + 1

And XORed bits (Tap bits) from right to left are then:

R1: bits ⇔ powers at 0, 1, 2, 5 ⇔ 18, 17, 16, 13
R2: bits ⇔ powers at 0, 1 ⇔ 21, 20
R3: bits ⇔ powers at 0, 1, 2, 15 ⇔ 22, 21, 20, 7

And its registers shift activation mechanism is ruled by a “Majority Function” whose argument is one three bits Clock Vector [C1 C2 C3], where Ci’s are unique Clock Tap bits, located approximately in the middle of each register: positions 8, 10, and 10 respectively. As we will see, at each time step two or there registers are activated depending of the structure of this vector whose content changes at each time step.

Output – bit wise delivered- are the most significant bits coming out of each register (MSB corresponds to bit 0 as depicted in the figure, numbered from left to right). Two to three registers’ outcomes, depending of the majority function structure, are XORed to finally determine the algorithm “output” that bit a bit configures the cipher “mask”.

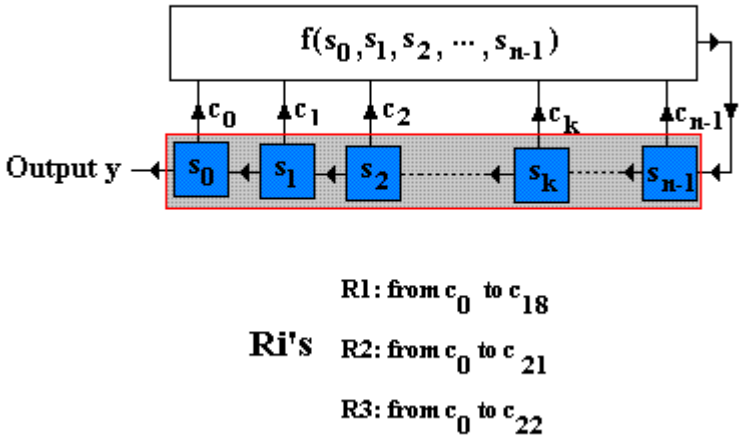
In the figure below we depict a detail of how these Ri's work. Feedback bits are calculated at time step j as a polynomial transformation of registers' states Si(j), let's say a linear combination of state's bits, one for each register, being the binary c coefficients the Tap bits read from leftmost bit as 0 bit. Computationally this linear combinations that generate feedback bits could be instrumented with "mask vectors" whose 1's correspond to the Primitive Polynomial non zero coefficients (Tap bits) being the rest bit positions set equal to zero:

R1 Mask ⇔ [1110010000000000000]
R2 Mask ⇔ [11000000000000000000000]
R3 Mask ⇔ [11100000000000010000000]

Where for instance R1 c's coefficients in the figure, which correspond to powers of x: x^0, x^1, x^2, and x^5, are set equal to 1 and the rest equal to 0. The output "y" bits coming out of registers, from y(1) to y(n) at each time step are initially those corresponding to the initial state of them:

yi(1) = Si(0)
yi(2) = Si(1)
yi(3) = Si(2)
……..
yi(n) = Si(n-1),

Being y's from these time onward feedback bits generated n time steps before as depicted in the figure.



Note: As we will see the linear combination core (sxc + f => f) is computed via AND for (x) and XOR(for (+).

Linear and non linear steps

As long as LFSR's operate ignoring the "Clock" activation system that controls the activation of registers, in a "Stop and Go" mode, working synchronized all at a time, the machine behaves linearly. As LFSR's follow a GF2 algebra we may go either forward/backwards between states indistinctly. A5/1 algorithm operation could be seen in four steps as it is depicted in the figure below. A mechanism the machine has to control the output is the inhibition of it along step 3 meanwhile a long enough mixing processes is performed with the Clock system activated. This mixing process assures the quality of the enciphering when time of the proper mask generation comes (step 4). Enciphering masks could be of any length. These machines are in theory enabled to generate (2^n – 1) different pseudo random states, for example (2^64 – 1). In the A5/1 enciphering masks have 228 bits divided in two blocks of 114 bits each.

Non linearity is instrumented via a “Stop and Go” mechanism, controlling the registers activation (along steps 3 and 4). This mechanism could be built as a linear combination of pre selected bits of registers (Tap Clock bits) giving for example a binary value that at its turn enable/disable the registers activation. The mechanism chosen for A5/1 is straightforward but effective: it activates registers that “match” the majority value; in fact those register whose clock Tap bit matches that value. As we will see this mechanism determines that statistically each register shift, in the average, ¾ of times and at least two of them at a time. This simple process introduces however a strong non linearity. Too close states with minor differences in their distances may generate very different outcomes!. The matrices approach for going from one state to the other do not work within non linear zones because the pseudo random activation of registers. Linear navigation through states presupposes registers working all at a time like brotherly concatenated hand to hand, all “aging” either forward or backward exactly the same. However with matrix algebra we may have a mechanism that builds eventual chains of registers content but of different ages instead!. Let see this with an example.

	[0] [0] [0] =>	[1] [0] [1] =>	[2] [1] [1] =>	[3] [1] [2] =>	[3] [2] [3] =>	[4] [3] [3] =>	[5] [4] [4] ……
Time steps =>	1	2	3	4	5	6	7
	[0] [0] [0] =>	[1] [1] [1] =>	[2] [2] [2] =>	[3] [3] [3] =>	[4] [4] [4] =>	[5] [5] [5] =>	[6] [6] [6]

In the first sequence we have activated the Clock System and within [] we show the “age” of each register if measured in its virtual internal clock. The clocking system enables at least that two registers shift at a time. Only in the first and seventh clock steps all three registers were enabled to shift. In the second sequence we may appreciate that all registers “age” at the same time. Another important difference is that when clock system is activated the whole machine “ages slowly”, in the average ¾ of its hypothetical functioning with Majority Function deactivated.

How may we go forwards and backwards

We may use a modified reverse algorithm function for going “backwards” from any state S(j) along an ancestor’ states tree, that essentially determines for any actual state its possible ancestors as in MM, Markovian Models. With Matrix Algebra we may also go either forward and backward deterministically within linear “zones” and probabilistically within non linear zones as we will see.

Types of Mixing

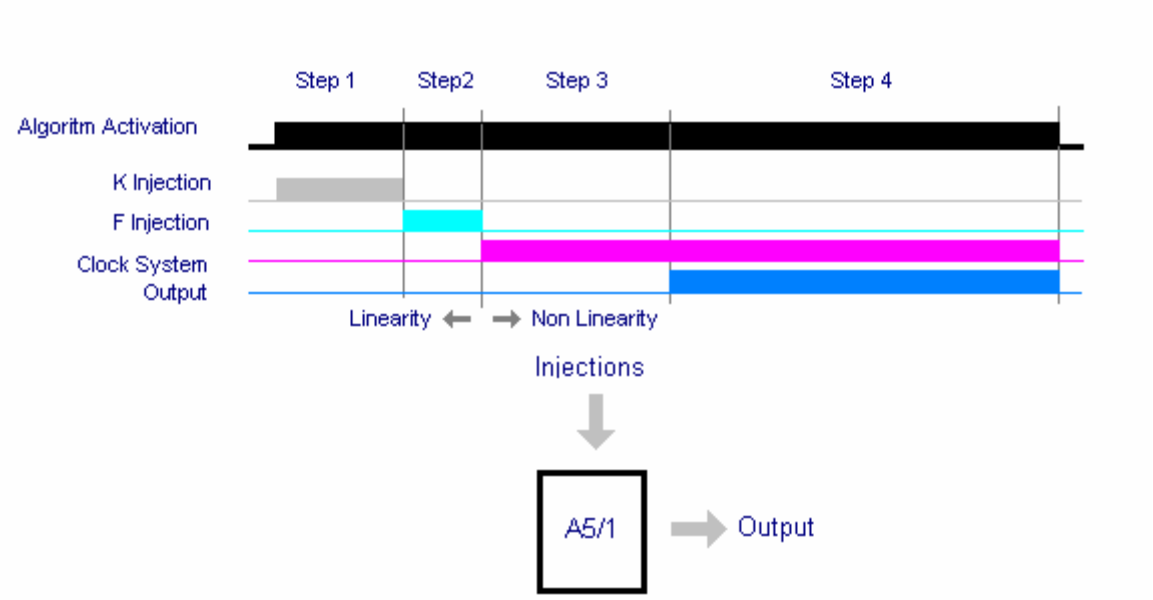
Several types of mixing to add randomness quality are instrumented. As these machines are in fact deterministic a given state S(j) at time j should be enforced to be “different”, specially at outcome stage when delivering ciphering masks. A trivial procedure is to mix the registers content with keys, in the A5/1 case with a Private key Kc and a Public key F, one for each “frame” of conversation.

These keys could be injected anywhere and at any time and at any mode. In A5/1, Kc and F, one after the other, are injected bit wise at time 0 over all its range (64 bits) previously zeroing registers, and in parallel over the three registers, at XOR mode and injecting at their LSB extreme. Clock system is not activated along these steps. A complementary mixing could be instrumented by enabling registers run along a “sufficient time” disregarding the outcome.

Equations that rule States Transformation

Given [S(0), K, F]
S(j) = S1(j)OS2(j)OS3(j)….OSr(j), being r number of registers
Si(j+1) = Ci(Bi(j))xSi(j) + H(j+1)

Note: O stands for “concatenation”, x by product (with AND and XOR operators), + (XOR)



Where

i=1, 2, 3, …r, amount of registers; r=3 for A5/1;
j=0, 1, 2, ….64, ……, 86, ……186, [187, 188, ……, 413, 414], machine internal time. Red numbers correspond to end of time steps 1, 2, 3, and 4 respectively. Along time steps [187, 188, ……, 413, 414] the 228 bits outcome is generated.
Si, are State Vectors of size n(i) such as n = n1 + n2 + n3 = 64 = 19 + 22 + 23
Si(j) state vector content at "time" j
S(j) equivalent state vector of size n formed by concatenating (O) its three contents S1(j), S2(j), and S3(j).
H(j+1) stands for an exogenous injection vector. It could also be considered a "pulse" of injected h bits that starts at a critical time j* and ends at time (j* + (h-1)).
Bi(j) are i Boolean Vectors, one for each register that rule their activation. In A5/1 case Bi(j)'s are predetermined and equal to 1 from j=1 to j=86 and ruling a non linear behavior from that time onwards.
Ci(Bi(j)) are three two valued function that depends on the value of Bi(j) as follows: If Bi(j)=1 Ci = Ai and on the contrary Ci = I when Bi(j)=0, being Ai the Characteristic Matrix of register i and I the Unit Matrix.

Each Ci rules then two types of transformation, a "Null" T0 transformation when the register is not activated remaining its content unchanged (except by exogenous injections whether they exist) and a T1 transformation that corresponds to a regular shifting with its corresponding feedback. It's the same as doing either

T1 ⇔ Si(j+1) = AixSi(j) + H(j+1) or
T0 ⇔ Si(j+1) = Si(j) + H(j+1), depending of the value of Bi(j).

Usually H(j+1) is inexistent when registers are activated in a "stop and go" mode by a particular function so transformations are in those cases simplified as:

T1⇔ Si(j+1) = AixSi(j) or
T0⇔ Si(j+1) = Si(j), depending of the value of Bi(j).

The A5/1 seen as state's machine has an output y(j) that could be formally defined by:

$$y(j) = (B1(j)xg1(j)xS1(j-j1^*) \text{ XOR } (B2(j)xg2(j)xS2(j-j2^*)) \text{ XOR } (B3(j)xg3(j)xS3(j-j3^*))$$

Where:

gi(j) are Primitive Polynomial vector masks that applied to the registers content generate their output bits. Outcome yi's XORed all together generate step by step (from step 187 to 414) the 228 A5/1 outcome bits.
The outcome is inhibited at steps 1, 2 and 3 along times j=1 to 186.

Note: This formal expression is not trivial to apply in A5/1 hard and soft versions. Because the strong pseudo random non linearity introduced via the Majority Function register shifts are almost always less than time steps and for this characteristic we put j1*, j2* and j3 * instead of 19, 22 and 23 respectively. Time j runs from 1 to 414 in the A5/1 case meanwhile "local time" for registers go slower with gaps that at random apart more and more from j. As explained above this is about ¼ j: for example when j=100, register shifts will be in the average around 75.
However to compute y's things are easy because they are the bits that "fall" at each shift: yi(j+1) = bit Bi0 of state Si(j) at each time j, being Bi0 the leftmost bit of register Ri and Si(j) Ri state vector at time j.

Warning: The LFSR's transformations in order to adjust precisely to its formal algebra should be as follows: given a present state S(j) at time step (j), next state S(j+1) is generated by the linear transformation depicted in the figure above and "simultaneously" with the shift the leftmost bit comes out, virtually "falling", leaving the register, and the feedback bit entering by the register rightmost position as a slide rule. This characteristic should be carefully taken into account when designing hard and soft versions of LFSR's in order to keep working all their properties. If these versions proceed to first "clock" registers, and if a buffer to preserve the leftmost bits of the registers that are going to be activated was not considered, they will be lost and the outcome y(j+1) will correspond to next contiguous positions B1's instead of B0's. This anomaly could be easily overcome performing the y computation prior to shift registers at each time step.

Now to have defined the algorithm values at each time step we need to know the Bi(j) behavior from time 187 onward. This behavior is the one that provides the necessary non-linearity to the algorithm, basically for security reasons. This algorithm virtually calculates Bi's content via a Boolean "majority function" by XORing specific "Clock Tab bits", one per register. These tab bits are selected to be approximately in the middle of each register. As in this case we have three registers we always will have at least two registers whose clock bits matches the majority value at a given time (j).

$$Bi(j) = (Si(j)(ai) = z(j))$$

Where:

z(j) is the content of the majority function at time (j)
ai is the relative position of Clock bit within state vectors Si(j) at time step j
z(j) is the content value of the majority function that could be defined by the following table:

S1(a1)	S2(a2)	S3(a3)	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Now we are ready to compute all states given S(0), K, F by the iterative process:

$$S(j) = S1(j)OS2(j)OS3(j)$$
$$Bi(j) = (Si(j)(ai) = z(j))$$
$$Si(j+1) = Ci(Bi(j))xSi(j) + H(j+1)$$

$$y(j) = (B1(j)xg1(j)xS1(j)) \text{ XOR } (B2(j)xg2(j)xS2(j)) \text{ XOR } (B3(j)xg3(j)xS3(j))$$

A5/1 Case

Step 1: From time 1 to 64

Si(0) = 0
Si(j+1) = AixSi(j) + K(j+1), that gives place to the following “linear” recursion

Si(1) = AixSi(0) + K(1)
Si(2) = AixSi(1) + K(2)
……
Si(64) = AixSi(63) + K(64), that could be synthesized as

Si(64) = Ai^64xSi(0) + M(64)xK

Where K above stands for the Private Key Vector that is injected bit wise, in parallel to the three registers and along the first 64 time steps. M is a 64x64 Boolean sparse matrix that could be defined as the Characteristic Matrix of the A5/1 K Mixing process. As we will see this matrix is built with the first columns of the Ai powers, from 1 to 63.

Step 2: From time 65 to 86

Si(64) = Ai^64xSi(0) + M(64)xK
Si(j+1) = AixSi(j) + F(j+1), from j=64 to j=85 that gives place to the following “linear” recursion

Si(65) = AixSi(64) + F(1)
Si(66) = AixSi(65) + F(2)
……
Si(86) = AixSi(85) + F(22), that could be synthesized as

Si(86) = Ai^22xSi(64) + MF(22)xF

Where the F stands for the Public Key Vector. It is also injected bit wise, in parallel to the three registers along 22 consecutive time steps. MF is a 22x64 Boolean sparse matrix that could be defined as the A5/1 M Frame Mixing Matrix. As we will see this matrix is built with the first columns of the Ai powers, from 1 to 21.

Some preliminary reasoning before going to see steps 3 and 4

About the deterministic behavior of most encrypt algorithms

We may imagine steps 3 and 4 as a transformation with the ciphering algorithm working under non linear mode, with the clock system activated. Step 3 and 4 only differ in its outcome process: in step 3 it is disabled and enabled in step 4. However if we are only interested in the states we may arrive to some interesting conclusions. One supposition would be that *we know at least the state of the three registers at a given “time”*. This supposition is not trivial because all these algorithms work as hermetic black boxes, only delivering ciphering masks and in most cases we do not even have access to the ciphering masks but to the ciphered messages instead. However if we know at least one “Internal State” after being passed by steps 1 and 2 we may try to go “upwards” in order to retrieve the Private key K!.

That possibility is attainable. There are many ways to accomplish this. One “trick” is to detect some pattern within the outputs generated by special internal states!. Let’s suppose that we know that some “Special States” at time step j=86 or even deeper, at time step j=186, generate a well defined pattern such as [1000000000000000] in the beginning of the output.

As A5/1 is of pseudo random type, at large deterministic, it could be seen as a machine that generates patterns within its outcome stream. If a given state generates a given pattern at distance h measured in time steps but starting at position p of the outcome we may argue that going “upward” j time steps it will generate the same pattern in the outcome stream starting at position (p-j), and conversely if that state is imagined going forward j time steps it will generate the same pattern but this time starting at position (p+j) of the outcome instead. Taking into account that A5/1 outcomes begin at time step 187 and end at time step 414, the span where these conjectures apply is within those values.

So in despite of being encapsulated, the presence of a given pattern within one ciphered mask is a signal that it was generated by a “hidden” state. Identified the suspected state the challenge would be to go “upwards“ to reach state S(86) because we know that once there we could unveil K!..

Here rest the central idea of the Birthday Paradox Attack (see Real Time Cryptanalysis of A5/1 on a PC, by Alex Biryukov, Adi Shamir, and David Wagner (2001)). A significant sample of pre computed special states characterized by generating a given pattern in the output from a given internal state (for instance from j=187 onwards) are stored in a Special States Database. As each communication “session” of length (t) measured in normal time has in the average N pieces of messages to encode (pieces of 228 bits in the A5/1 example) our deciphering problem could be stated in the following terms:

a) Detection within samples of N pieces of messages the existence of at least one well defined pattern!;
b) Once detected our problem would be to see if there exists a “collision” between this sample and the pre computed states. That’s would be nice but we forgot something: a “name” relating this pattern with the Special States stored in our database. The name of each special state could be the string of bits that follow the pattern at time of pre computing. In our example one of the “enforced” special states could be

[1010001101011000111] [00111010100011010100001] [11011110010100001010001]

That in the pre computation stage if run with A5/1 generated the following output stream:

[0000000000000001] followed by **[11000010101100010111000101010111001]**……..

We may choose then this “arbitrary” suffix of 35 bits (in red) as a “Short Name” to identify the state. So these special states should be kept in the database by pairs [State, Short Name], classified by Short Name. This reasoning presupposes that special states are generated by a A5/1 software clone specially adapted for this purpose.

What happens if the probability of collision is too low?. We may increase this probability by pre computing all 2^64 -1 pairs [states, names] but they are too much states for the actual state of the art of computing. If this procedure were feasible it entails the pre computation of approximately 2^(64 – p) being p the size of the pattern of bits, usually known as the “prefix”. Fixing p=16 we need to pre compute 2^48 pairs, also too much!. It’s important to realize that the name or “suffix” must have – in theory- the minimum length (s=64 – p) in order to address the whole variety of states. In the example above we used a prefix of 16 bits and a suffix of 35 bits that pre determine a database with 2^35 Special States. Let’s see later how the authors of successful attacks justify these figures. Another way of increasing the possibility of collision would be to consider internal special states that generate the sought pattern in any position within the output ciphering mask.

What to do once we have got a successful collision

For time steps j>86, from 87 to 414 the process in non linear. To go forward to such states we may use two procedures, namely:

a) The exact one by running a soft version of the algorithm or
b) A statistical approximation applying matrix algebra

In both cases the pair [K, F] should be known, being K the Private Key and F the actual frame number that works as the Public Key that "tag" the pieces of messages to be ciphered under the control of the local Mobile Control Station.

How could we get those pieces of information?

A "conversation" could be assimilated then to a train of pieces of information assembled as a stream of 228 bits of length, bursts (frames in the wireless Jargon) synchronized with the Frame Number broadcasted by the control station, a numbering system that goes cyclically from 0 to (2^22 -1). A train of messages could then be seen as a train of messages of the form [m1, m2, m3, ….., m(N)] where each m could be imagined like a string of 228 bits representing equal number of bits of a digitalized piece of conversation, masked via a XOR operation with an enciphering mask of 228 bits generated by the A5/1 algorithm. These trains of bursts begin at any moment within the continuous beating of the frame counter as follows:

…….Fs F(s+1) F(s+2) F(s+3) …….F(s+N)……

[m1 m2 m3 …….. mN]

That is N frames were emitted under the "session key" K and each one paired with a different Public Key F, from F(s+1) to F(s+N). The stream may look then as follows:

Origin side [m1m2 m3 …… mN]
Origin side [c1 c2 c3…… cN]

m's XOR c's: c's generated with pair [Korigin F]
In the "air" we have then [mi* m2*m3*…… mN*]

m's XOR c's: c's generated with pair [Kdestiny F]

Destiny side [c1 c2 c3…… cN]
Destiny side [m1 m2 m3 …… mN]

At origin messages m are encoded with the c's generated masks by the A5/1 algorithm activated by the pair [Korigin, F] and send over the air as m*. At destiny coded messages are decoded with the same c's but generated with its particular pair [Kdestiny F] in order to maintain both origin and destiny privacy. As we see masks c's are different at both sides of the communication being the "secrecy" of it under the custody either of the Control Station or a higher level of control. Private Key deciphering could be attained by several methods, but mainly two: a) by knowing pairs [m m']; b) by only knowing c's.

By only knowing c's

The ciphering masks stream could be obtained by XORing the received message pieces at destiny with the encoded messages m*'s sent by the Control Station to destiny provided that access to them is attainable!. Effectively meanwhile messages going from origin to the Control Station are encoded with the pair [Korigin F] the same messages are retransmitted by the Control Station to destiny encoded with the pair [Kdestiny F]. Then the "trick" is to cross combine messages as follows:

m's (from origin to Control Station) XORed with m's (received from the Control Station at destiny and decoded by using the pair [Kdestiny F]. As the message pieces are the same at both sides, that is messages received at destiny are the same as messages emitted by the origin, the XOR gives us the c's stream generated by the origin.*

Now we have c's but what next?. We need at least one A5/1 internal state related to any pair [Korigin F]!. Let's suppose that we got state S(J) at "time" j. The sequence of a stream encoder algorithm is in fact "deterministic", behaving like a deterministic states' machine meaning the following:

[K F] => a practically "unique" states sequence of the form =>
S(0) S(1),…..,S(64), ..S(86),…S(186),…..S(414)

in such a way that from S(187) to S(414) the encoding algorithm generates the enciphering 228 bit masks c's. As this sequence is unique, or it could be considered as unique, any internal state S(j) could be considered the ancestor of the corresponding ciphering mask!. Only one peculiar S(j) working at the right "time" j generates it. As we have seen in our analysis transformations between states are only linear in steps 1 and 2 that is from time 1 to 86. However we could go from any S(j) either in the non linear or linear region upward to state S(0) via matrix algebra or throughout a Markovian type tree browsing algorithm. In order to speed up this browsing we are going to use a linear approximation taking into account the predictive nature of the register behavior within their non linear zones.

Statistical approach

As we are only interested about states and their content not about the output of the "states' machine" – via the trick of "collisions" explained above- what remains crucial is the estimation of the steps performed by registers along their non linear behavior. Being the probability of shifting equal for all of them (P= ¾) and of its stopping (Q= ¼) and the process statistically defined as responding to the Binomial Distribution, the expected number of shifts for n beats of the algorithm clock will be nP and its expected variance nPQ. For example if n=100 it is expected that each register (R1, R2 and R3) shifted in the average 75 times and the typical deviation estimated by 4.33. Then a first estimation would be to suppose that along 100 steps any register would shift a number ni within a predefined IC, Interval of Confidence, for example

$$IC = [(75 - 4.33z) <= n' <= (75 + 4.33z)]$$

Let's suppose z=1.15, then

IC = [70 <= n' <=80] = [70 71 72 73 74 75 76 77 78 79 80]

We have then defined a "neighborhood" around which it would be highly probable the real shifts appear!. Effectively calling n1, n2 and n3 the shifts of registers 1, 2 and 3 respectively we may state that the real state would be (with a certain pre designed probability) within the Cartesian product n1xn2xn3 = 1331 [S(1)OS(2)OS(3)] possible sequences.

Now which of these sequences is the right one provided that the right one is within this neighborhood?. It will be very easy. We apply to each member of the sequence their corresponding linear transformations n(i)P times as if the Clocking System does not work at all. If existent, one of them will generate the output c' "as it is" running forward a soft version of the algorithm. Once we have a right state located in the linear region, going upwards till S(0) and retrieve K becomes a linear matrix computation.

How to get a feasible internal state S(J) within the non linear region

We have discussed this above. We need to have a pairs [S Short Name] database sorted by name. These "special" states have the property of generating outputs with pre determined "prefixes". If prefixes are of length h it is possible to generate 2^(64-h) different 64 bits states that share the same property. Some authors define these states as "red states". One trivial way of generation is to compute all 2^64 states in their pseudo random sequence and take only those that generate as outcome the sought prefix (same authors name it as the "alpha" prefix). This is impractical as we have analyzed above: 2^64 is too large a number to deal with even using the most powerful farm of computers!. A by far more efficient and subtle method is to enforce their generation by "guessing" some bits and enforcing the rest to generate the "alpha" prefix as outcome.

Pairs ⇔ State [10100011010110000111][00111010100011010100001][11011110010100001010001]
⇔ Short Name [11000010101100010111000101010111001]

Remembering that "alpha" for a red state are in the beginning of the ciphering mask:

[[1000000000000000] [11000010101100010111000101010111001]……]

Where in red we have a sixteen bits "alpha" within the 228 bits mask, followed by its "name" of 35 bits. That's good but not enough in terms of probability of collisions because we are ignoring all possible appearances of alpha within the mask. As alpha plus its name takes 51 bits we would have available (228 – 51)=177 positions, the first bit of the mask included. Of course an appearance of alpha h positions shifted to the right means that its corresponding "red state" is also shifted accordingly as we explained above Those states that generate alpha prefix out of the first position of the mask but within it are named "green states" by some authors. Well you may argue, why don't we generate and store them to increase significantly the collision probability?. Because with 2^35 red states we are close to the limit of our actual computing capability and multiplying by 177 the size of our pre computed states reservoir is impractical. So we need to find another ingenious shortcut.

Warning: See in A5/1 Explained III why 35.

Finding alpha at position 57 for example means that a red state is virtually present but "hidden" 57 steps forward measured in time steps. Well at least we have detected a red state, but unfortunately this red state is not situated at its "right" place ⇔ time within the sequence. Remember that red states are always generated as being born at a well defined time, for instance at time step 186. In order to be in the linear region we have to go backwards 57 steps and virtually defining, at large, a green state!. So if going backwards we arrive at level 186 it means that we are arriving at a green state, If on the contrary, we fail because we get dead ends before getting that level, it means that our suspected green state do not exist. So the trick is to generate red states that have many green states as possible successors, what could be tagged as states with a high level of *"inverse prolific factor"*, too many ancestors within certain critic regions.

It involves to select among all possible 2^(48 - 35) sets one which members are highly inverse prolific in the average. Green states we are looking for are simply successors of red states that could go deep (exist) between states 101 and 277 in order to be potential candidates to be successors of "hidden" red states. Does it seem a words game?. Could be, in some cryptanalysis work we have to think symmetrically going to the other side of the Alice mirror!.