The Bit-Flip Pattern-Free Cipher

Gideon Samid
Department of Electrical Engineering and Computer Science
Case Western Reserve University, Cleveland, OH
BitMint, LLC
Gideon@BitMint.com

Abstract: Proposing a cipher based on the Bit-Flip protocol, where t letters of an alphabet $\{A\}_t = A_1, A_2,... A_t$ are each associated with a distinct bit string comprised of n random bits where n is an even number: $\{S\}_t = S_1, S_2,...S_t$. Alice will send an arbitrary letter A_i to Bob, by applying the bit-flip protocol to S_i, generating S'_i. By construction S'_i and S_i will agree over exactly 0.5n bits -- randomly distributed. Bob will compare S'_i to $S_1, S_2,...S_t$ in some order, and so will identify the letter A_i which Alice communicated. It will be the only letter that shares *exactly* 0.5n bits with the communicated string. Security is credibly determined by combinatorics, and can be adjusted to any desired level by proper selection of n, and t. The key, namely $\{S\}_t$, is comprised of open-ended amount of randomness. Each key string (per letter) generates a new randomized cipher-letter each time. There is no pattern to discern, no algorithmic complexity to crack. Brute force cryptanalysis – defective because the amount of randomness is open-ended -- is the only strategy. Alice could drown her message by sending over strings that do not evaluate to any of the $\{A\}_t$ letters. This way Alice can set up a fixed rate communication channel to Bob without leaking any information about the content, not even the pattern, or the existence of any information flow. And same for the counter communication from Bob to Alice. This casts Alice and Bob into a "black hole" mode. Similarly a "black hole" state may exist over a multi party conversation. It also can be extended over several size alphabets in parallel, distinguished by the size of their respective bit string $\{S\}_n$. This cipher reflects a new crypto-paradigm: replacing complex number-theoretic computations with simple bit-wise operations wielding large undisclosed quantities of randomness, which are very readily handled via modern memory technology, and fast and efficient communication technology.

Introduction

Given a bit string X comprised of |X|=x bits, (x is an even number), and given the fact that this string was constructed by randomly flipping 0.5x bit from an input string Y, the observer who is not aware of Y will be looking at x bits, each of which has an equal chance for being what it is, and an equal chance of being the opposite. The knowledge of X though, restricts the scope of possible Y strings, since X and Y must agree on the identity of half of their bits.

By straight forward combinatorics the number of Y string candidates is:

$$1.....x!/(0.5x)!^2$$

which will be regarded as the flip-range formula. And the ratio of the number of Y candidates given X relative to not knowing X is:

$$2.....x!/((0.5x)!^2 * (2^x))$$

which will be regarded as the flip-ratio formula. The value of x then determines both (1) what is the chance to guess Y given X, and (2) what is the chance to generate X, without knowledge of Y, such that a Y holder will find that X and Y have agreement over exactly 0.5x bits. It can be easily seen that x can be selected such that both probabilities will be as low as desired.

Please study the following table 1 constructed from the equations above:

|X|	Flip-Candidates (X)	Flip-Ratio (X)
20	184756	0.18
50	1.26E+14	0.11
100	1.01E+29	0.08
250	9.12E+73	0.05
1000	2.70E+299	0.02

The table shows that for an X string comprised of |X|=x=50 bits there are $1.26*10^{14}$ candidates Y, and if Y is perfectly randomized there is no hope for a shortcut in determining it, only the brute force approach. For a string of x=250 bits the number of candidates is more than 10^{73}. Paradoxically, of sorts, as the flip-range grows exponentially with the size of the string, so the ratio of these candidates relative to all possible strings is getting lower.

The price paid for having lower probabilities as above (namely, better security) is the burden of handling larger quantities of randomness. But that is a very low price to pay for three reasons: (1) the mathematical manipulation involved in this process is simple bit-wise: counting bits and flipping them; (2) the cost of storing large number of bits is subject to Moore's law, and hence is very low, and getting ever lower. And (3) communication technology hammered down the price of sending a bit around the globe.

The Bit Flip protocol [Samid 2016] describes how to use this randomized procedure for Alice to authenticate herself to Bob by proving to him she is in possession of Y through sending Bob X. Here we extend this procedure to full fledged communication.

The Basic Bit Flip Pattern-Less Cipher

We consider an arbitrary alphabet $\{A\}_t$ comprised of t letters: A_1, A_2, ... A_t. We associate each letter with a unique and random bit string comprised of n bits each: $\{S\}_t = S_1, S_2,....S_t$ respectively.

We define the set FlipRange(S) as the collection of all the bit strings that can be generated by flipping exactly 0.5|S| bits in S.

We will now define the non-overlap condition over the set $\{S\}_t$:

$$(3).....FlipRange(S_i) \cap FlipRange(S_j) = \emptyset \text{ for all } i \neq j \text{ over } i,j=1,2,...t$$

Namely a string S'_i that has 0.5*|S| bits in agreement with string S_i, will not have exactly 0.5*|S| bits in agreement with any of the strings: S_1, S_2,...S_{i-1}, S_{i+1},....S_t.

Let Alice wish to communicate to Bob an arbitrary message M expressed through the $\{A\}_t$ alphabet: M =A_i - A_j - A_k,..... To do so Alice applies the Bit Flip protocol over S_i to randomly generate a same length string S'_i which has exactly 0.5n bits in common with S_i, then she applies the Bit Flip protocol over S_j to generate S'_j, followed by applying the Bit Flip protocol to S_k to generate S'_k, and so on, until she finishes with her message, say, comprised of m $\{A\}_t$ letters.

Alice then concatenates all the bit-flipped strings: M' = S'_i || S'_j || S'_k..... m strings all together, and sends M' over to Bob.

Bob parcels M' into blocks of n bits, and then checks every block against the $\{S\}_t$ set in his possession. Bob will readily determine that the first n bits represent a flipped S_i string, and conclude that the first letter Alice communicated to him was A_i. Bob will then check the 2nd block of n bits against the same set $\{S\}_t$, and identify A_j as the second letter Alice sent over to him. The third letter is A_k, and so on. Bob will decipher the full message Alice sent him.

This is the basic Bit Flip cipher.

Bit-Flip Calculus

We present a few definitions, lemmas, and the separation theorem. Lemma 1: The FlipRange function is symmetrical. Namely, if X' is a member of the set FlipRange(X), then X is a member of the set FlipRange(X'). This is because if it takes 0.5x bits to generate X' from X, then flipping back the same 0.5x bits in X' will generate X (x is the bit count of X and X'):

$$(4).....X' \in FlipRange(X) <=> X \in FlipRange(X')$$

Definitions: Every two random strings of same size X and Y: |X|=|Y|=n define a set of n-bits strings that are members of the two FlipRanges.

The set of strings Z such that $Z \in FlipRange(X) \cap Z \in FlipRange(Y)$ is regarded as the shared range: SharedRange(X,Y).

Let two bit strings X and Y such that |X|=|Y|=n have c bits in common (same identity), and hence (n-c) bits of opposite identity. Illustration: String A= 1101 and B=0011 have one bit in common c=1 (bit number 4).

The Range Separation Theorem: Every two bit strings of same even number of bits, n, which agree on the identity of an odd number of bits, c -- have c bits in common -- have an empty shared range.

In other words: let two bit strings X and Y such that |X|=|Y|=n, where n is even, have c (odd) bits in common. Then there is no string Z such that:

$$(5).....Z \in FlipRange(X) \cap Z \in FlipRange(Y)$$

The non-separation theorem: Every two bit strings of same even number of bits, which have an even number of bits in common, have a non empty shared range.

More specifically: let two bit strings X and Y, such that |X|=|Y|=n, where n is even, have c (even) bits in common. Then the size of the shared range is:

$$(6).....|SharedRange(X,Y)| = (c!/(0.5c!)^2)* ((n-c)!/(0.5(n-c)!)^2$$

Proof: Let's divide the c shared bits into two categories α and β, each comprised of 0.5c bits. Similarly, let's divide the (n-c) opposite-identity bits to two equal size categories: γ and δ. We shall now construct a string Z (|Z|=n), such that Z = RFlip(X). We

shall do it in the following way: (1) we first flip all the bits in the α category, then (2) we flip all the bits in the γ category. Thereby we have flipped 0.5n bits, so that the resultant $Z \in \text{FlipRange}(X)$.

We shall now construct a string Z' (|Z'| = n), such that Z' = FlipRange(Y). We shall do it in the following way: (1) we first flip all the bits in the α category, then (2) we flip all the bits in the δ category. Thereby we have flipped 0.5n bits, so that the resultant $Z' \in \text{FlipRange}(Y)$:

It is easy to see that Z=Z'. In both strings the same α bits were flipped, and since they were the same before the flipping they do agree now, after the flipping. The γ category of bits were flipped in X. Each of these bits in X was opposite to its value in Y (bits are identified by their position in the string), so now that these bits were flipped in X, they are the same as in Y. And the way we constructed Z' was without flipping the γ category in Y, so the γ bits are the same in Z and Z'. Symmetrically the δ bits are the same in Z and Z'. They were not changed in Z, and they were all flipped in Z'. And hence we have proven that Z=Z', which means that $Z \in \text{SharedRange}(X,Y)$. To find the size of the shared range set we ask ourselves how many ways can the c bits be divided to α and β categories, and then in how many ways can the (n-c) bits be divided to the γ and δ categories, and thus we arrive at the result indicated in the theorem, Eq #6.

We can now prove the separation theorem: since c is odd and n is even, then (n-c) is odd too. And therefore the unshared (n-c) bits cannot be divided into two equal size categories γ and δ. And therefore we cannot exercise here the procedure taken for the even (n-c) case, and hence we cannot construct the same string, by flipping 0.5n bits in both X and Y. In the closest case the γ category will have (n-c+1)/2 bits and δ will have (n-c-1)/2. So at least two bits will be off when comparing Z and Z'.

Illustration: Let X = 11001101 and Y = 10111010. These strings have c=2 bits in common: bit 1 and bit 5. We set bit 1 to be the α category, and bit 5 to be the β category. The 6 remaining bits where X and Y disagree we divide to category γ: 2,3,4 and category δ: bits 6,7,8.

We shall now generate string Z by flipping the α category and the γ category in X: 00111101. In parallel we generate Z' by flipping the α category in Y and the δ category in Y: 00111101 -- resulting in the same string: Z=Z'.

However, if we use the same X but change Y by flipping its first bit: Y=00111010 then now X and Y have only one bit in common (bit 5). And since the number of disagreeing bits is odd (7), it is impossible to exercise the above protocol, and hence these X and Y above have no member in the set of their shared range.

Exercising The Non-Overlap Condition

One way is to randomize the identity of the first string S_1, then randomize the identity of S_2, and evaluate the number of shared bits (bits in common, of same identity), c. If c is even, then re-randomize S_2, test again for the number of shared bits with S_1, and keep randomizing until c is odd. This would be a quick exercise because in randomizing S_2 q times, the chance that all the results for c will be even is 2^{-q}, which disappears very rapidly.

Having established S_1 and S_2, one will randomize S_3, and evaluate the common bits between S_3 and the previously established strings S_1, and S_2. S_3 may have to be randomized several times until a selection is picked that has odd number of bits in common with each of the formerly established strings. This procedure will continue until all the t strings are established such that no string Z of same size n bits will fit into the Flip Range of more than one of the t strings.

Given the size of the t strings (each n bits long), one could apply the Flip-Ratio formula, and compute the ratio of flip related string to each string in the set S_1, S_2, ...S_t. Let that ratio be ρ. For the t strings to be mutually separable we will have to require that $t*\rho < 1$. And the closer $t*\rho$ is to 1, the more difficult will it be to find t mutually exclusive strings (strings that do not allow for interpretation ambiguity per the Bit-Flip protocol). So for t=4 and n=6 there would be no satisfactory solution of the separation condition because the bit-ratio formula (Eq #1),indicates a ratio of 31.25% so that 4*0.3125 > 1. It will be hard to find a solution even for t=3 because 3*0.3125 = 0.9375.

Illustration, let t=3 and n=40. We ranomnly choose S_1 = 1111111111100100101000100010010100110100. We randomize S_2 = 0001100011000110010010010101010011101001 They share 16 bits. So we try again: S_2 = 0110110110101111011110110111111100100111. This time we have c=21 shared bits. We now randomize the third string: S_3 = 1100001110000001010001010000000010000000. Checking S_3 versus S_1, we find c=19 shared bits, another odd number. Now what remains to check is S_3 versus S_2: they also share c=19 bits. So all bi lateral relationships between S_1, S_2, and S_3 are associated with an odd number of shared bits. Which means that there would be no ambiguity at the interpretation side since there is no 40 bits bit-string that fits into the flip-range of more than one string.

Illustration

Let's apply the bit-flip pattern-free cipher to the simplest case: a two letters alphabet X and Y. (t=2). We randomly choose $S_1 = S_x$= 110011 (n=6) and $S_2 = S_y$ = 100110. We find that the two strings share 3 bits in common: 1,3,5. So they qualify as working strings because there is no risk for any ambiguity.

Note: it may look curious that because S_x and S_y have c=0.5n bits in common that they both fit into each other flip range:

(7).....S_x = FlipRange(X_y); S_y = FlipRange(S_x)

However, no n-bits string Z will fit into both S_x and S_y. So no ambiguity.

Now assume that Alice wishes to send Bob the message (plaintext) p = XYY. She will proceed as follows. Using a randomizer she will generate S'_x by randomly flipping 0.5n=0.5*6=3 bits in S_x: S'_x =000010. She could then send S'_x to Bob. Bob will check S'_x against S_x and find that the two strings share the values of 3 bits: 3,4, and 5. Checking with S_y Bob finds that they have 4 bits in common, not 3: bits 2,3,5,6. Bob then concludes that Alice sent him the letter X.

Next Alice wishes to send the letter Y, she will generate a flipped string of S_y: S'_y = 010111 and send it to Bob. Bob will first check the string against S_x and find that the two strings have four bits in common: bits 2,3,5,6. He concludes then that the new string is not Y. Checking versus S_y he finds that the two strings have 3 bits in common: bits 3,4,5, and concludes that Alice sent him here the letter Y.

Alice will then apply the bit flip protocol over S_y again (because her message to Bob is XYY). She will generate: S''_y = 101011. She will send it to Bob. Bob will again check it against S_x and find four bits of same identity: bits 1,4,5,6, and so conclude that the new letter from Alice is not X. He would check against S_y and find exactly three bits of same identity: bits 1,2 and 5. And so conclude that Alice, again sent the letter Y. All in all Bob will be able to recreate Alice's message XYY.

Alice could then add strings that don't evaluate neither to X nor to Y. For example the string Z=011101 has 2 bits in common with S_x (bits 2, and 6). It has one bit in common with S_y (bit 4). So Z does not evaluate to either X or Y. Bob will dismiss it -- and any other string that does not evaluate to either X or Y. Eve, the eavesdropper will not be able to discern that some of the bits sent over from Alice to Bob are perfectly meaningless.

Security of the Basic Bit-Flip Cipher

Assuming that the bit strings $\{S\}_t$ are randomly constructed, and assuming that the Bit Flip protocol is randomly executed, then given the flipped string X_f of X:

$$(8).....X_f = BitFlip(X)$$

there appears to be no chance for a 'shortcut' to identify X from X_f. The chance of every member X_r of the FlipRange(X_f) to be X is the same:

$$(9)..... Pr[X=X_r \mid X_r \in FlipRange(X_f)] = 1/FlipRange(X) = ((0.5n!)^2)/n!$$

And hence for a cryptanalyst in possession of M, and of knowledge of the values of n and t, the most efficient way, is the only way: brute force: to construct all plausible messages written in the $\{A\}_t$ alphabet, and check each of which against M. One may note that by choosing the values of t and n to be sufficiently large, and by keeping the bit-flip protocol strictly random, Alice and Bob can practically insure that no two bit-flipped strings will be identical, even for a very large plaintext (check out table 1). And this absence of repetition will insure that the brute force cryptanalysis set forth above will remain the most effective one.

It is intuitively clear that for may reasonable combinations of (t, n, m) the cryptanalyst will end up with rich equivocation -- a very large number of plausible messages that Alice could have sent over to Bob. And there would be nothing in M that would help the cryptanalyst narrow down the list.

In principle, the values of n and t may remain part of the cryptographic secret.

Inherent Chaff

As discussed above, it is common to embed cryptograms in a larger flow of randomized data where only the intended reader knows to separate the wheat from the chaff. In most of these schemes the means of such separation are distinct from the decryption algorithm. What is unique with the Bit-Flip cipher is that the chaff is inherent, namely, only by knowing the key can one separate the wheat from the chaff. Say then that for any cryptanalytic effort (carried out, of course, without knowing the key), the chaff will look exactly like the wheat, and will have to be treated as such. The intended reader will regularly identify the chaff, as bit strings that don't evaluate to any letter in the alphabet. For the cryptanalyst any string is potentially a letter, and it participates in the cryptanalytic hunt. By adding sufficient chaff -- strings that don't evaluate to any alphabet letter -- the sender will build a chance for terminal equivocation where even the brute force cryptanalysis will be helpless.

Design Considerations of the Bit Flip Pattern-Free Cipher

The Bit-Flip Pattern-Free cipher will work on a binary alphabet, as well as on a large as desired alphabet $2 \leq t \leq \infty$. The value of t will determine a low level for n to insure separation and lack of any ambiguity. But there is no limit on the high level of n. Since brute force cryptanalysis is the only envisioned attack strategy, given the extensive randomization of the data and its processing, then the more bits there are to resolve, the greater the security of the cipher. Hence cipher security is proportional to $t*n$. Accordingly the Bit Flip cipher designer will opt to use high t and n values.

On the other hand the larger the values of n and t, the more randomness has to be shared between Alice and Bob, in the form of a the shared key ($t*n$- bits). But the larger the value of t (the size of the alphabet) the less information must be sent over by Alice to

Bob. For a fixed n value, if the alphabet is binary, and one uses, say the ASCII table then 8 bits are needed to communicate an ASCII symbol, and hence an ASCII symbol will require 8n bits to pass through. The ASCII table can also be covered by words comprised of 4 letters of an alphabet of 4 letters: $4^4 = 256$, and in that case a byte will be communicated using only 4n bits. If the entire table is comprised of letters, then n bits will be needed per symbol. Yet, the larger the number of letters (larger t) the more work needed for the decryption -- on average t/2 letters will have to be checked out to find which letter Alice communicated. This number may be improved owing to the naturally low entropy of a typical text. A common predictive algorithm will be able to rank the next letter according to probabilities, and use this ranking to decide on the order by which to evaluate the incoming ciphertexts.

All in all this Bit Flip Alphabet cipher takes advantage of two strong trends in modern technology: (i) memory is cheap and gets cheaper, and (2) bit communication is fast and getting faster -- more throughput, less cost. So Alice and Bob will likely be willing to store some more randomness, and communicate some more randomness in order to secure their data to their desired degree.

This cipher being part of the new wave expressed in “Randomness Rising” [Samid 2016R], also shifts the security responsibility from the cipher designer to the cipher user. By selecting the values of t and n, the user determines the security of his data. By operating two or more parallel sets of alphabets, the user will be able to designate some portion of his data for extra high security.

This cipher may be designed as a "shell" where the user selects, t, n, and then generates $t*n$ random bits -- the key. The processing being so minimal that there is no practical way to engineer a backdoor. What is more -- the chip for the bit wise operations of this cipher may be freely designed and manufactured using commercially available chip design programs.

The processing of the data may be done in software, firmware or hardware -- for extra speed. It may be done with special purpose quite primitive integrated circuits because the operations are limited to basic bit-wise instructions.

Alphabet Variety

The Bit Flip alphabet cipher works on any alphabet from a simple binary one to any size t. The binary strings associated with the letters of a given alphabet will be of the same fixed size. However, Alice and Bob may use in parallel two or more alphabets.

Consider that Alice and Bob use two alphabets: $\{A\}_t = A_1, A_2,.....A_t$, and $\{A'\}_t' = A'_1, A'_2,....A'_t'$. The first alphabet is associated with strings of size n bits, and the second alphabet is associated with strings of size n' bits.

Alice will be able to communicate to Bob encrypted messages of either alphabet. She will then have to communicate to Bob the size of the string (n or n'). There are several established ways to do it. One simple way would double the size of the communicated message: The communication flow from Alice to Bob will be comprised of encrypted bits and meta bits (all the rest). The plaintext bits will be written as follows: 0 ➔ 01, 1 ➔ 10. For meta bits we have: 0 ➔ 00 and 1 ➔ 11. This way there will be no confusion as to whether the bits represent a cryptogram or some auxiliary data. The auxiliary, meta, data could be used to mark the boundaries of the Bit Flip Cipher blocks. This will allow the sender to shift at will from one alphabet to another, and give more security to more sensitive data within the same file.

One could of course extend this practice to any number of alphabets.

Use: one alphabet may be used for digits only; another for letters, and a third for a special code book that offers shortcuts to frequently used terms. Alternatively the same alphabet may be associated with two or more strings set. A simple alphabet for non-

critical encryption will have a small string size, n; while a more critical encryption over the same (or different) alphabet will be encrypted/decrypted with large string size, n'.

Brute-Force Implementation

The imposition of the non-overlap condition on the t strings $\{S\}_t$ contaminates to a negligible degree the pure randomness of their content. This contamination may be rectified if deemed important by ignoring this condition, and thereby admitting a certain -- known, and usually small -- chance for ambiguity. Namely a string sent by Alice is interpreted by Bob as fitting to letter A_i and letter A_j ($i \neq j$). This ambiguity can readily be resolved by having Alice exercise the protocol again. When applying the randomized protocol to sending the same letter to Bob, there is only an extremely low chance that exactly the same two letters will generate ambiguity for Bob. Chances are that only one of the two, either i or j will fit the new cryptogram. Or in a small likelihood a new pair will generate ambiguity: A_i and A_k. In that case it will be clear to Bob that A_i is the letter that corresponds to the S string he received.

This way, the stream will flow normally for most of the encrypted letters, and very rarely Bob will request Alice to resend a letter.

Advanced Bit Flip Alphabet Cipher

The Bit Flip cipher allows the sender to add randomized data to the plaintext, without limit, and without extra effort for decoding the stream, except that it will be proportional to the size of the incoming data flow. A sender will send over a bit-string of the right size of bits, but that would not evaluate to any letter according to the shared key (the t n-bits strings that represent the agreed upon alphabet). The decoder will spend the

same effort on this 'decoy' or 'chaff' string as on a regular string -- the letter evaluation effort.

This reality gives rise to advanced applications of the cipher:

- **Parallel Mutually Secret Messages**
- **cyber black holes.**

Parallel Mutually Secret Messages

Let us consider two alphabets, one comprised of t letters, and the other of t' letters: $\{A\}_t$, $\{A\}_{t'}$. Let each alphabet be associated with a key comprised of n-bits long strings. Let us construct the strings so that all strings are distinct. No string in one alphabet is the same as any string in the other alphabet.

Now consider the situation where Alice and Bob share the key for the first alphabet, and Alice and Carla share the key for the other alphabet. Let M be a message Alice wishes to communicate to Bob, and let M' be a message Alice wishes to communicate to Carla.

Alice could use the Bit-Flip alphabet cipher to send these messages separately, but she could also mix them into one mixed string M" = per-letter-mix(M, M'). When Bob receives M" he will readily discard all the letters that belong to M' because all these letters will not evaluate to any of his alphabet. When Carla receives M" she will ignore all the letters written in Bob's key, and correctly interpret her message.

For example, Alice wishes to communicate to Bob the word: 'NORTH', and to Carla the word: 'SOUTH'. Marking letters sent over with Carla's key with /'/ we write: NS'OO'RU'TT'HH' or in some other mix: NOS'RO'TU'HT'H' where Bob will interpret as 'NORTH' and Carla as 'SOUTH'. Neither Carla, not Bob have to know that the letters sent

to them by Alice, which all look as meaningless garbage, are indeed a bona fide message for someone else.

This concept should not be limited to two alphabets and two parallel messaging. It can be applied to any number of parallel messages. There are several advantages to this configuration. We discuss: *Peer-to-Peer message distribution* and *Built-in Equivocation*.

Peer to Peer Message Distribution

Consider a peer-to-peer network where one peer is designated as a 'hub' and shares Bit Flip cipher keys with all other peers. The hub could mix some q messages, each designated to another peer, and send the package to an arbitrary peer in the network. That peer will check the package for a message to itself, and if it finds any, it will strip it from the package, and pass the stripped package ahead to any other peer. This passing on will continue until the package is emptied, and there is nothing to pass on. At that point it is also clear that all q peers received their message. The peer that would empty the package will signal to the hub that this package was fully distributed. The advantage of this procedure is that it handles well, off time of peers, and is very resilient against any interruptions to parts of the network. The variety of sequences that such a package can assume is astronomical: $p!$ for a p-peers network.

This P2P message distribution may also apply for the cases where peers are divided by blocks. Each block has the same key (the t Bit Flip strings). In that case, the number of the addressed peers in each block will be indicated in the contents of the message to these peers, and each peer reading this message will decrement the counter of how many more peers need to read it. The last reader will remove that message from the package.

Every arbitrary peer will be able to take advantage of this messaging regimen. That peer will send all its messages to the hub, using its shared key with the hub, requesting the hub to put a package forward.

Built-In Equivocation

Let M_1, M_2, ... M_k represent k messages that cover all the plausible messages relative to a given situation. Elaboration: A cryptanalyst is told that Alice sent Bob a message, and then the cryptanalyst is asked to list all the plausible messages that Alice could have sent. Messages that make sense given whatever the prevailing circumstances are. This list of plausible messages reflects the cryptanalyst's ignorance of the contents of the message Alice sent Bob. It only reflects his or her insight into the situation where the message took place. The aim of the cryptanalyst is to use the captured encrypted message to reduce the entropy of this set of messages, to build some probability distribution over them.

Now assume that Alice sent Bob M_1, but buried it in a mixed package where all the other (k-1) messages show up. For Bob there would be no confusion. He would only regard the bit strings that evaluate to his message, and ignore all the rest. Alas, a cryptanalyst, with full possession of the ciphertext but with no possession of Bob's Key, at best, with Omni powerful tools, will uncover all the keys for all the k messages and will end up with all the k messages as being plausible communication from Alice to Bob -- namely the cryptanalyst will face terminal equivocation that drains any value offered by possessing the ciphertext. This equivocation will be valid, although to a lesser degree, by padding the real messages with a smaller number of decoy or 'chaff' strings.

Cyber “Black Holes”

If Alice and Bob are not communicating -- it says something about them. If Alice and Bob are communicating with uncracked encrypted data -- they surrender a great deal of information just through the pattern of the data flow -- size of messages, frequency, back and forth relationship between Alice and Bob, etc. To stop this leakage of information flow Alice and Bob can build a "black hole" communication regimen.

In a "black hole" Alice and Bob send each other a constant stream of randomized bits. These bits may be raw randomness and carry no information -- which represents the case of no communication. Or, these random bits may hide bits that carry information according to some pattern.

Alice and Bob may use the Bit Flip Alphabet cipher to mix bits that represent letters in their agreed upon alphabet with bits that don't evaluate to any of the alphabet letters. Only the holder of the key ($\{S\}_t$) will be able to separate the raw randomness from the meaningful message.

This black hole status may be extended to a multi party communication. It would also be extended to n communicating parties where each pair of parties shares a key. This configuration is very wasteful and very burdensome for its key management. A simpler solution is a hub configuration where one of the parties has a bilateral shared key with all the other participants, and any party i wishing to communicate with a party j will send its encrypted message to the hub, using the shared key i-hub, and the hub will send it to j, or to any other parties which part i identifies as intended recipients. For each recipient the hub will use the shared key with that recipient.

Use Cases

The Bit Flip cipher seems ideal for Internet of Things applications where some simple devices will be fitted with limited bit-wise computation power to exercise this cipher. IOT devices may read some environmental parameter which fluctuates randomly, and use this reading to build the ad-hoc flipping randomness. Smart but cheap devices may be fitted with the hardware necessary for operating this simple cipher, and no more. This will prevent attempts to hijack such a device. The simple BitFlip cipher is too meager a machine to turn around for ill purpose.

One may note that while the data flow is much greater than with a nominal cipher where the ciphertext is as large as the plaintext, once the message is decoded, it is kept in its original size. So the larger ciphertext is only a communication imposition. But since most secrets are in textual form, this will be not much of a burden, compared to communicating a regular photo today.

.

Because of the ultra simplicity of the cipher and its great speed, it may find a good use in many situations. Some are discussed:

The Bit Flip cipher may be used for audio and video transfer, say, a store will sell a pair of headphones, or headphone attachment where each element of the pair is equipped with the same key (randomized t strings), and will be used to encrypt and decrypt the spoken word.

The cipher could be used to communicate across a network through a hierarchy of clusters where the members of each cluster share a key. Messages between random peers in the network will have to be encrypted and decrypted several times, but the speed of the operation will minimize the overhead.

The speed of the cipher could be used for secure storage. All stored data will be BitFlip encrypted before storing, and then decrypted before using. The keys will be kept only in that one computer, in a fast processing chip, likely. This option will also relax worries about the security of data, which a third party backs up in the cloud.

There are several applications where the cyber black hole mode will come in handy hiding communication pattern between two financial centers for example.

Personal privacy: most personal computing devices today allow for an external keyboard, and an external display to be attached to the machine. By fitting a BitFlip chip between these peripherals and the computer, two parties (sharing the same BitFlip chip box) will be able to communicate truly end-to-end with the BitFlip chip box (the box that houses

the shared chip which has ports for the keyboard and the screen) serving as a security wall against any malware that may infect the computer itself: like keyboard loggers.

Reference:

Mate 2015: “Survey on Cryptographic Obfuscation” Ma´t´e Horva´th 9 Oct 2015 International Association of Cryptology Research, ePrint Archive https://eprint.iacr.org/2015/412

Nies 2008: “Computability and randomness” Niels A. The University of Auckland, Clarendon, Oxford, UK

Samid 2010 “Shannon Revisited: Considering a More Tractable Expression to Measure and Manage Intractability, Uncertainty, Risk, Ignorance, and Entropy” https://arxiv.org/abs/1006.1055

Samid 2003 “Essential Shannon Security with Keys Smaller than the Encrypted Message” http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.1585&rep=rep1&type=pdf

Samid, 2016 “The Bit Flip Protocol” http://www.agsencryptions.com/BitFlip_H6n30.pdf

Samid, 2016R “Randomness Rising” http://www.agsencryptions.com/RandomnessRising_H6n08.pdf

Samid, 2001 “Re-Dividing Complexity between Algorithms and Keys: http://link.springer.com/chapter/10.1007/3-540-45311-3_31#page-1

Smart 2016: “Cryptography Made Simple” Nigel Smart, Springer.

Table of contents