A Block Cipher Masking Technique for Single and Multi-Paired-User Environments

Ray R. Hashemi
Amar Rasheed
Jeffrey Young
Department of Computer Science
Georgia Southern University, Armstrong Campus
Savannah, GA, USA
e-mails: {rayhashemi, amarrasheed, alanyoung10101}
@gmail.com

Azita A. Bahrami
IT Consultation
Savannah, GA, USA
e-mail: Azita.G.Bahrami@gmail.com

***Abstract*— A ciphertext inherits some properties of the plaintext, which is considered as a source of vulnerability and, therefore, it may be decrypted through a vigorous datamining process. The vulnerability increases when a community of users is communicating with each other. Masking the ciphertext is the solution to this vulnerability. We have developed a new block cipher masking technique named Vaccine for which the block size is random and each block is further divided into segments of random size. Each byte within a segment is instantiated using a dynamic multi-instantiation approach, which means (i) the use of Vaccine does not produce the same masked outcome for the same given ciphertext and key and (ii) the choices for masking different occurrences of a byte are extremely high. Vaccine is tested in both single-paired-user and multi-paired-user communities with the revoking option. A key agreement is used to manage key changes required by the revoking option. For testing in a single-paired-user environment, two sets (100 members in each) of 1K long plaintexts of *natural* (borrowed from natural texts) and *synthesized* (randomly generated from 10 characters to increase the frequency of characters in the plaintext) are built. For each plaintext, two ciphertexts are generated using Advanced Encryption System (AES-128) and Data Encryption Standard (DES) algorithms. Vaccine and two well-known masking approaches of Cipher Block Chaining (CBC), and Cipher Feedback (CFB) are applied separately on each ciphertext. On average: (a) the Hamming distance between masked and unmasked occurrences of a byte using Vaccine is 0.72 bits higher than using the CBC, and CFB, and (b) Vaccine throughput is also 3.4 times and 1.8 times higher than the throughput for CBC and CFB, correspondingly, and (c) Vaccine *masking strength* is 1.5% and 1.8% higher than the masking strength for CBC and CFB, respectively. For testing in a multi-paired-user community with the revoking option, the findings remain the same for every single-paired-user. However, there is an overhead cost related to re-keying and re-profiling, which is caused by the revoking of a user from the community or expanding the community of users. The overhead cost is linearly related to the size of community.**

Keywords- Cyber Security; Masking and Unmasking Ciphertext; Variable-Block Cipher Vaccination; Masking Strength; Key Aggregation; Re-keying; Re-Profiling

I. INTRODUCTION

Protecting sensitive electronic documents and electronic messages from unintended eyes is a critical task. Such protections are provided by applying encryption. However, the encrypted text (ciphertext) is often vulnerable to datamining. The root of such vulnerability is in the *inherited-features* of the plaintext by the ciphertext. We have addressed the problem and its solution in the past for a secure *single-paired-user environment* [1]. (The distinct property of this environment is that there is no community of users, but there are individual pairs of users.) However, we expand the previously reported paper to address not only the single-paired-user environment but also a secure *multi-paired-user* environment with the option of revoking user.

The distinct properties of such an environment are: (i) presenting the environment as a graph in which users make the vertices and the communications established among the users make the edges of the graph, (ii) having a community-based key, which is the aggregation of the single-paired-user keys, and (iii) having the option to revoke the membership of a user in the community. The side effect of the last property demands a re-keying of the remaining single-paired users in the community whenever membership of a user is revoked. Regardless of having a secure single or multi-paired-user environment, the problem of inherited-features will not disappear and the problem is not limited only to primitive encryption modes such as displacement but also it can be observed in the outcome of more sophisticated encryption modes such as CBC and CFB [2][3][4]. The following examples provide some evidence.

As an example related to primitive encryption modes, let us consider the plaintext message of: "*The center is under an imminent attack"*. The plaintext may be converted into the following ciphertext using, for instance, a simple *displacement* encryption algorithm: "xligirxvmwyrhivermqqmrirxexxego". The features of the plaintext are also inherited by the ciphertext—a point of vulnerability. To explain it further, the word "attack" is

among the key words related to security. The characteristics of the word are: (i) length is six, (ii) the first and the fourth characters are the same, and (iii) the second and the third characters are the same. Using these characteristics, one can mine the given ciphertext and isolate the subtext of "exxego" that stands for "attack" which, in turn may lead to decryption of the entire message.

As an example related to more sophisticated encryption modes, the block cipher techniques that employ CBC/CFB encryption mode to produce distinct ciphertexts are vulnerable to information leakage. In the case of CBC/CFB using the same Initial Text Vector (IV) with the same encryption key for multiple encryption operations could reveal information about the first block of plaintext, and about any common prefix shared by two different plaintext messages. In CBC mode, the IV must, in addition, be unpredictable at encryption time; in particular, the (previously) common practice of re-using the last ciphertext block of a message as the IV for the next message is insecure (for example, this method was used by Secure Sockets Layer (SSL) 2.0). If an attacker knows the IV (or the previous block of ciphertext) before he specifies the next plaintext, he can check his guess about plaintext of some block that was encrypted with the same key before (this is known as the Transport Layer Security (TLS) CBC IV attack) [5].

The problem of inherited-features becomes a bigger concern when communication takes place within a secure community of users with a user revoking option. Upon revoking a user from the community, all the keys used for communication between any two users that collectively make a *community-based aggregate key*, need to be changed (re-keying process) on the fly. As a result, the inherited-features problem needs to be addressed in two environments: *Single-paired-user* and *multi-paired-user* [6][7].

In either environment, the fact remains the same that the logical solution for the inherited-features problem is to mask the ciphertext using a masking scheme that is *dynamic* and supports a high degree of *multi-instantiations* for each byte. A dynamic masking scheme does not produce the same masked outcome for the same given ciphertext and the same key. The high degree of multi-instantiation masking scheme replaces the *n* occurrences of a given byte in the ciphertext with *m* new bytes such that *m* is either equal to *n* or extremely close to *n*.

The goal of this research effort has two prongs:

(1) Introducing and building a dynamic masking scheme, named Vaccine for a single-paired-user environment. The Vaccine also supports a high degree of multi-instantiations that can mask the inherited-features of a ciphertext in the eye of a data miner while providing for transformation of masked ciphertext into its original form, when needed and

(2) Adapting the Vaccine for use in a multi-paired-user community that has the revoking option.

The Vaccine has the following three unique traits, which makes it a powerful masking scheme: It (1) divides the ciphertext into random size blocks, (2) divides each block into random size segments, and (3) every byte within each segment is randomly instantiated into another byte. All three traits are major departures from the norm of masking schema.

The rest of the paper is organized as follows. The Previous Works is the subject of Section II. For single-paired-user environment, the Methodology is presented in Section III, the Empirical Results are discussed in Section IV, and the findings are covered in Section V. For a multiple-paired-user environment, the adoption of the Vaccine is the subject of Section VI and the findings are discussed in Section VII. The conclusion and future research are presented in Section VIII.

II. PREVIOUS WORKS

In a secure single-paired-user environment, masking the features of a ciphertext that are either inherited from the plaintext or generated by the encryption scheme itself is the essential step in protecting a ciphertext. The block cipher and stream cipher mode of operations provide for such a step. We are specifically interested in CBC [8][9][10] and CFB [11] as samples of the block cipher and stream cipher mode of operations. They are to some degree comparable to the proposed Vaccine.

CBC divides the ciphertext into fixed–length blocks and masks each block separately. The use of fixed-length block demands padding for the last partial block of the ciphertext, if the latter exists. The CBC avoids generating the same ciphertext when the input text and key remain the same by employing an Initial Text Vector (IV).

CFB eliminates the need for possible padding of the last block (that is considered a vulnerability for CBC [12]) by assuming the unit of transmission is 8-bits. However, CFB also uses IV for the same purpose that it was used for CBC.

In contrast, Vaccine splits the ciphertext into random size blocks and then sub-divides each block into segments of random size. Masking each pair of segments is done by using a pair of randomly generated patterns. As a result, Vaccine needs neither padding nor IV. The randomness of the block size, segment size, and patterns used for instantiation of a given character are the major departure points of Vaccine from the other block and stream cipher approaches.

In a secure multi-paired-user environment with a revoking option, the use of CBC and CFB demand key management, which could be executed either by a key distribution [13][14][15][16] or key agreement [17][18][19] process (both processes are considered as pure overhead.) Key management is also adopted by Vaccine, which in turn requires the re-keying operation of the entire community-based aggregate key. Such overhead may be reduced by using the Vaccine. The reduction in overhead cost comes from the fact that the Vaccine masking is an amalgamation of random patterns, a key, and the use of a randomly generated byte from the plaintext. Therefore, if the revoked

user knows only the keys then, it is not enough to correctly de-mask (or mask) a ciphertext as long as the patterns' profile changes. Changing the patterns' profile is less expensive than the re-keying process. Of course, re-keying always remains as another viable option.

III. SINGLE-PAIRED-USER ENVIRONMENT

First, we present our methodology for instantiation of a byte, which contributes into dynamicity of Vaccine. Second, we introduce our methodology for building Vaccine. The details of the two methodologies are the subjects of the following two subsections.

A. Instantiation

Instantiation is the replacement of a byte, c, by another one, c', such that c' is created by some modifications in c. To perform the instantiation, we present our two methods of *Self-substitution* and *Mixed-Substitution*. Through these methods, a number of parameters are introduced that are referred to as the *masking* parameters. At the end of this subsection, we present the masking parameters as a *profile*.

1) Self-Substitution: Consider byte 10011101 and let us (i) pick two bits in positions p_1 and p_2 such that $p_1 \neq p_2$, (ii) flip the bit in position p_1, and (iii) swap its place with the bit in position p_2—Two-Bit-One-Flip-Circular-Swap technique.

It is clear that the pairs (p_1=1, p_2=7) and (p_1=7, p_2=1) create different instances for the byte. Therefore, the order of p_1 and p_2 is important. The number of possible ways selecting a pair (p_1, p_2) from the byte is 7*8=56, which means a byte may be instantiated by 56 possible different ways using Two-Bit-One-Flip-Circular Swap technique. The technique name may be generalized as *W-R-Bit-M-Flip-Circular-Swap*. For the above example (W=2 and M=1) the technique is shown in Figure 1. As a more general example, (W=8 and M=4) is also shown in Figure 1. As a rule, the value of parameter M is always less than the value of parameter W. This is necessary for not diminishing the effect of the swapping step. (We introduce the parameter R shortly.)

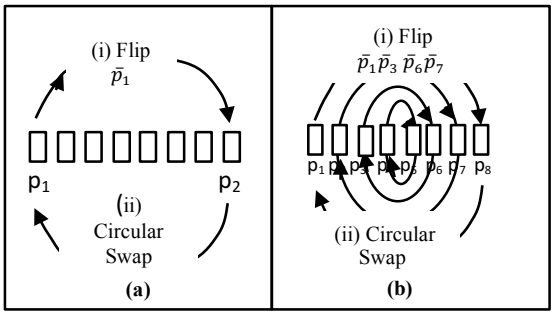

Figure 1. W-Bit-M-Flip-Circular-Swap Technique: (a) W=2 and M=1 and (b) W=8 and M=4

One may pick 3-bits (W=3) to instantiate the byte. Let us assume 3 bits randomly selected that are located in the positions p_1, p_2, and p_3. There are many ways that M-Flip-Circular-Swap technique can be applied:

a. (One-Flip-Circular-Swap) Flip one of the three bits and then make a circular swap among p_1, p_2, and p_3.
b. (Two-Flip-Circular-Swap) Flip two out of the three bits and then apply circular swapping.
c. (Three-Flip-Circular-Swap) is not used because it violates the rule of M being smaller than W.

The number of possible combinations grows to 5040.

Using the W-R-Bit-M-Flip-Circular-Swap for all possible values of W (W=2 to 8) and M (M=1 to W-1) generates the total of (X=1,643,448) possible substitutes for a given byte. If either W=1 or M= 0 then, the self-substitution has not been enforced and in this case X=1 (the byte itself). Now, we explain the role of parameter, R (where, R is a byte long) in the W-R-Bit-M-Flip-Circular-Swap.

Let us refer to the case of W=2 and M=1 one more time where it is able to facilitate the generation of 56 instantiations of a given byte using all the possible pairs of (p_1=•, p_2=•). That is, the two positions of p_1 and p_2 could have any value from 1 to 8 as long as $p_1 \neq p_2$. What if one is only interested in those instantiations resulting from the pairs of (p_1=3, p_2=•), which by definition also includes instantiations resulting from the pairs of (p_1=•, p_2=3)? The chosen value (bit) of interest for p_1 is a value from 1 to 8 that is expressed by setting the bit of interest in R. (Since p1 =3, the bit number 3, in R, is set to 1.) The number of bits that are set to "1" in R is always equal to M. For our example, R="00000100".

The pairs represented by (p_1=v, p_2=•) are the set of seven pairs of {(p_1=v, p_2=1), . . ., (p_1=v, p_2=8)}. The seven pairs are named the *primary set* for the *primary signature* of (p_1=v, p_2=•). The (p_1=*, p_2=v), which is a tweaked version of (p_1=•, p_2=v) is the *complementary signature* of (p_1=v, p_2=•) and stands for the other set of seven pairs {(p_1=8, p_2=v), . . ., (p_1=1, p_2=v)}. These seven pairs make the *complementary set* for (p_1=*, p_2=v). (Values of p_1, in the complementary set, are in reverse order of values of p_2 in the primary set.)

The primary and complementary sets also referred to as the *primary sub-pattern* and *complementary sub-pattern*, respectively. The two sub-patterns collectively make a *pattern* and the triplet of (W=2, M=1, R= "00000100") make the pattern's *stamp*, where W, M, and R are masking parameters. It is clear that M cannot be equal to W, because, when M= W, the primary and complementary sets are the same and they have only one member. This is another reason for supporting the rule of M must be smaller than W.)

The stamp of (W=4, M=3, R = "00001011") means four bits are chosen from the byte out of which three bits (M=3) in positions 1, 2, and 4 are the positions of interest (p_1=1, p_2=2, p_3=4.) Therefore, the primary signature and the Complementary signatures are, respectively, defined as (p_1=1, p_2=2, p_3=4, p_4=•) and (p_1= *, p_2=2, p_3=4, p_4=1).

When none of the bits in R is set to "1", it means R has not been enforced. In this case, we do not have the primary

and complementary sets. However, to apply Vaccine, we ought to have both sets. To do so, the default value of R, which is R with its M least significant bits set to "1" is used.)

2) Mixed-Substitution: We also extend the byte instantiation to the key. In a nutshell, the instantiation of the given byte, c, and each key byte are done separately. One of the instantiated key bytes is selected as the *key image* and the final instance of c is generated by XORing the key image and the instantiated c. The details are cited below.

Application of self-substitution with masking parameters of (W, M, and R) on a given byte generates the primary and the complementary sub-patterns of ($u_p^1 \ldots u_p^n$) and ($u_c^m \ldots u_c^1$). The subscripts p and c stand for these two sub-patterns and there are *n* and *m* members in the p and c sub-patterns, respectively. The key byte B_j is instantiated into another byte using the self-substitution with masking parameters of (W_j, M_j, and R_j, for j=1 to 4). Application of self-substitution on the individual four bytes of the key (B_1 . . . B_4) generates the primary and the complementary sub-pattern for each byte as follows:
($u_p^{1B1} \ldots u_p^{n1B1}$) and ($u_c^{m1B1} \ldots u_c^{1B1}$),
($u_p^{1B2} \ldots u_p^{n2B2}$) and ($u_c^{m2B2} \ldots u_c^{1B2}$),
($u_p^{1B3} \ldots u_p^{n3B3}$) and ($u_c^{m3B3} \ldots u_c^{1B3}$), and
($u_p^{1B4} \ldots u_p^{n4B4}$) and ($u_c^{m4B4} \ldots u_c^{1B4}$).

A byte, say c_1, using the first member of the primary sub-pattern, u_p^1, is instantiated to c_1'. The first byte of the key, B_1, using its first member of the primary sub-pattern, u_p^{1B1}, is instantiated to B_1'. The other three bytes are also instantiated into B_2', B_3', and B_4' using their first member of the primary sub-patterns, $u_p^{1B2}, u_p^{1B3}, and\ u_p^{1B4}$, respectively. The Hamming distance of *HD*(c', B_j'), for j=1 to 4, are measured and B'=Argmax[*HD*(c', B_j'), for j=1 to 4] is the key image. In the case that there are ties, the priority is given to the instantiated byte of B_1, B_2, B_3, and B_4 (and in that order.) The final substitution for c_1 is:

$$c_1'' = (c_1' \oplus B') \quad (1)$$

The next byte, c_2, within a given segment of ciphertext is instantiated to c_2' using u_p^2, and key bytes of B_1, B_2, B_3, and B_4 are instantiated to B_1', B_2', B_3', and B_4' using $u_p^{2B1}, u_p^{2B2}, u_p^{2B3}, and\ u_p^{2B4}$, respectively.

B'=Argmax[*HD*(c', B_j'), for j=1 to 4] and c_2''=(c_2'⊕ B'). The process continues until the segment of the ciphertext is exhausted. The bytes of the next sub-list and the key bytes are instantiated using the complementary sub-patterns. Therefore, the sub-patterns are alternately used for consecutive segments of the ciphertext.

Using mixed substitution, the number of possible combinations for each key byte is equal to X and for the key of four bytes is X^4 (>$1.19*10^{31}$ combinations.) The reader needs to be reminded that the four-byte key may be expanded to the length of N bytes for which the outcome of XOR is one of the X^{N+1} possible combination. For N=16 (128-bit key) The XOR is one of the X^{17} possible combinations (>$4.65*10^{105}$.)

3) Profile: Considering both self and mixed substitutions, the masking parameters grow to five triplets: The first triplet, (W, M, and R) for the instantiation of a byte of segment and the next four triplets of (W_j, M_j, and R_j, for j=1 to 4) for instantiation of the four bytes of the key. Therefore, patterns' profile, or simply profile, includes 15 masking parameters, which are accommodated by a 96-bit long binary string (Figure 2) as described below.

Since the possible values for each of the parameters W and W_j is eight (2 through 8 and value of 1 means the self-substitution has not been enforced), the value of each parameter can be accommodated by 3 bits (the total of 15 bits). Since the three bits make a decimal value between 0 and 7, we always add 1 to the decimal value to get the true value for W or W_j.) The parameters M and M_j have eight possible values (1 through 7 where the value of 0 means that self-substitution has not been enforced) and each parameter can also be accommodated by 3 bits (the total of 15 bits). The parameters R and R_j need eight bits each (the total of 40 bits). In addition, we use twenty-six bits as *prefix* of the profile (five bits as *reserved* bits for possible expansion, sixteen bits as the *Flag bits* and five bits as the *Preference bits*.)

The sixteen flag bits represent a decimal number (Δ) in the range of (0: 65,535). Let us assume that the length of the ciphertext that is ready to be masked is L_{ct}. Three bytes of f_1, f_2, and f_3 of the ciphertext are selected (flagged), which are in locations: $\delta_1 = \delta$, $\delta_2 = \lfloor L_{ct}/2+\delta/2 \rfloor$, and $\delta_3 = L_{ct} - \delta$, where, δ is calculated using the formula (2)

$$\delta = \begin{cases} \Delta\ Mod\ L_{ct}, & \Delta > L_{ct} \\ L_{ct}\ Mod\ \Delta, & \Delta \leq L_{ct} \end{cases} \quad (2)$$

The flagged bytes will not be masked during the vaccination process and they collectively make the *native byte* of $F=(f_1 \oplus f_2 \oplus f_3)$. Since the length of the ciphertext and the length of its masked version remain the same there is no need for including the length of the ciphertext in the profile. The question of why the flagged bytes are of interest will be answered shortly.

The purpose of preference bits is to build a *model*, which is influenced by both the key and flagged bytes. The model is used to create variable length blocks and segments. The minimum number of preference bits is five and can grow up to ten by consuming the reserved bits. The preference bits are partitioned such that the most significant bit is considered partition one and the rest of the bits make partition two.

To build the model, a desired byte number (z) of the key is identified by the bits of partition two. That is, one can select any byte from a maximum of a 512-byte long key. The key is treated as circular and two pairs of bytes $A_1=(z+1||z)$ and $A_2=(z+2||z-1)$ are obtained from the key. A new pair of bytes of $A_3=A_1 \oplus A_2 \oplus (F||F)$ is built. If the bit in the partition one is set to zero then, the model is A_3;

otherwise, the model is $a_1 \oplus a_2$, where, a_1 and a_2 are the pair of bytes in A_3.

Let us assume that there are two similar ciphertexts of CT_1 and CT_2 and we are using the same key and the same profile to mask the two ciphertexts, separately. As long as one of the three flagged bytes in CT_1 and CT_2 is different, the native bytes for the ciphertexts are different and so their models, which in turn make their masked versions different. This is one of the major advantages of Vaccine.

To summarize, using a 4-byte key, the number of bits needed for the profile is 96 bits. Dissection of a pattern profile is shown in Figure 2. The 24 hex digits representing the patterns' profile along with eight hex digits representing the 4-byte key that are collectively called *Masking Image*, may be sent to the receiver in advance or they may hide in the masked ciphertext itself:

a. In a predefined location/locations,
b. In location/locations determined by the internal representation of the key following some formula(s), or
c. A mixture of (a) and (b).

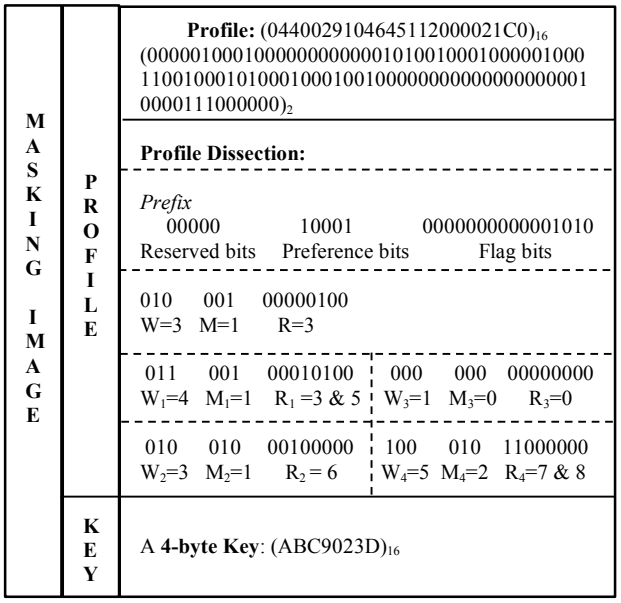

Figure 2. Dissection of the masking Image

B. Vaccine

Vaccine is a variable-block cipher methodology capable of masking and unmasking a ciphertext. The details of masking and unmasking of Vaccine are presented in the following two subsections.

1) Masking of the Ciphertext: Vaccine as a masking scheme is able to mask the features of a ciphertext in the eye of a text miner. Vaccine: (1) divides the ciphertext into random size blocks, (2) each block, in turn, is divided into a number of segments such that the length of each segment is random, and (3) every byte within each segment is randomly instantiated to another byte using self and mixed substitutions. The masking process is encapsulated in algorithm Mask shown in Figure 3.

The algorithm is made up of four sections. In section one, (Step 1 of the algorithm) the profile is dissected to extract masking parameters and they, in turn, generate primary and complementary sub-patterns for five patterns:($Pattern_p^0$, $Pattern_c^0$), ($Pattern_p^{B1}$, $Pattern_c^{B1}$), ($Pattern_p^{B2}$, $Pattern_c^{B2}$), ($Pattern_p^{B3}$, $Pattern_c^{B3}$), and ($Pattern_p^{B4}$, $Pattern_c^{B4}$) used for masking the chosen byte of the ciphertext and the four key bytes, respectively. The array of pt with five elements keeps track of those primary and complementary sub-patterns of the five patterns that are in use. The model is also extracted in this step.

Algorithm Mask
Input: A 32–bit key, a pattern's profile of 96-bit, and a ciphertext, CT.
Output: Delivering IC as the masking version of CT.
Method:
Step1- //Dissection of the profile and initializations
Dissection delivers primary and secondary sub-patterns of five patterns ($Pattern_p^0$, $Pattern_c^0$), ($Pattern_p^{B1}$, $Pattern_c^{B1}$), ($Pattern_p^{B2}$, $Pattern_c^{B2}$), ($Pattern_p^{B3}$, $Pattern_c^{B3}$), and ($Pattern_p^{B4}$, $Pattern_c^{B4}$).
κ ←Model obtained by using Preference bits, Flag bits, and key;
IC ← ""; C ← CT;
pt[5]← 0;//pt gives turn to the primary (pt[•]=0) and complementary (pt[•] =1) sub-patterns of the five patterns for initializing the CurrentP [5];
Step 2-Repeat until C is exhausted
a- Get the set of decimal numbers from κ in ascending order: D ={d_1, d_2, . . . d_{y-1}, d_y};
Get the next random size block,
β_n=Substr(C, 0, d_y);
b- CL = 0; //Current location in C
c- Repeat for i =1 to y-1
//Divide β_n into y-1 segments;
s_i = Substr(β_n, CL, d_i - CL);
CL = CL+ d_i;
CurrentP[m]=$Pattern_{pt}^m$ //for m =0 to 4;
d- Repeat for each byte, c_j, in s_i
d_1- If (c_j is a flagged byte) Then continue;
d_2- If (CurrentP[0] is exhausted)
Then CurrentP[0] = $Pattern_{pt}^0$;
d_3- c_j' = Flip c_j bits using CurrentP[0];
d_4- c_j' = Circularly swap proper c_j bits using CurrentP[0];
d_5- σ = Select(c_j', CurentP[1], CurrentP[2],CurrentP[3],CurrentP[4]);
d_6- a = c_j' ⊕ σ;
d_7- IC←IC || a;
End;
pt[•]++; pt[•] ← pt[•] mode 2;
End;
e- Remove block β_n from C;
f- Apply one-bit-left-rotation on κ;
End;
End;

Figure 3. Algorithm Mask

The second section (Step 2.a of the algorithm) identifies a random size block prescribed by κ—the model.

The identification process is done by creating y binary numbers using κ. The i-th binary number starts from the least significant bit of the κ and ends at the bit with the i-th value of "1" in κ. The binary numbers are converted into decimal numbers and sorted in ascending order, $\{d_1, d_2, \ldots d_{y-1}, d_y\}$. The block, β_n=Substr(C, 0, d_y), where C is initially a copy of the cipher text.

The third section (Step 2.c of the algorithm) divides block β_n into a number of random size segments. The size and the number of segments are dictated by the κ internal representation. Block β_n has y segments: $\{ s_0 \ldots s_{y-1}\}$.

The segment s_i starts from the first byte after the segment s_{i-1} (the location is preserved in variable CL) and contains $\lambda_i = d_{i+1} - d_i$ bytes. The number of segments and their lengths are not the same for different blocks.

To get the next block of the ciphertext, the block β_n is removed from C (Step 2.e) and κ is changed by having a one-bit-left-rotation (Step 2.f). Using the above process along with the new κ, the next block with a different size is identified. This process continues until C is exhausted. It is clear that the lengths of blocks are not necessarily the same. In fact, the lengths of the blocks are random. It needs to be mentioned that the length of blocks β_i and β_{i+8} are the same when κ is one byte long. When κ is two bytes long, the length of the blocks β_i and β_{i+16} are the same. And a block on average is 32,768 bytes long. As a result, the ciphertext, on average, must be longer than 491,520 bytes before the blocks' lengths are repeated.

Algorithm Select
Input: A byte (c), Key, and four patterns for the four key bytes.
Output: key image, k.
Method:
a. Repeat for (w = 1 to 4)
  If (CurrentP[w] is exhausted)
  Then CurrentP[w] = $Pattern_{pt}^{w}$;
End;
b. h ← -1;
c. Repeat for v= 1 to 4;
  i. c_v ← An instantiated version of $KeyByte_v$ using related sub-pattern.
  ii. If HD(c, c_v) >h //HD is Hamming distance function
  Then h = HD(c, c_v); k = c_v;
End;
End;

Figure 4. Algorithm Select

The fourth section (Step 2.d of the algorithm) delivers the masked version of the ciphertext, byte by byte, for a given segment. Flagged bytes are not masked (Step $2.d_1$). If the number of bytes in the segment s_i is greater than the cardinality of the pattern then, the pattern repeats itself (Step $2.d_2$). Each byte, c_j, of the segments s_i (for i=1 to y-1) are masked by applying (i) the relevant member of the current sub-pattern on byte c_j (Step $2.d_3$ and $2.d_4$), (ii) identifying the key image (Step $2.d_5$), by invoking the Algorithm Select (Figure 4), (iii) create c_j', the masked version of c_j, by XORing the outcome of process (i) and process (ii), (Step $2.d_6$), and (iv) concatenate the masked version of c_j, to the string of IC, which ultimately becomes the inoculated version of the inputted ciphertext (Step $2.d_7$).

2) Unmasking of the Ciphertext: For unmasking a masked ciphertext, those steps that were taken during the masking process are applied in reverse order. Therefore, the Algorithm Mask with a minor change in step 2.d can be used for unmasking. We show only the changes to Step d of Figure 3 in Figure 5.

d- Repeat for each byte, c_j', in s_i
  d_1- If (c_j is a flagged byte) Then continue;
  d_2- If (CurrentP[0] is exhausted) Then CurrentP[0] = $Pattern_{pt}^{0}$;
  d_3- σ = Select(c_j', CurentP[1], CurrentP[2], CurrentP[3], CurrentP[4];
  d_4- α = c_j' ⊕ σ;
  d_5- α = Circularly swap bits of α using CurrentP[0];
  d_6- α = Flip a bits using CurrentP[0];
  d_7- UM←UM||α; //UM is the unmasked ciphertext;
End;

Figure 5. The modified part of the Algorithm Mask

IV. EMPIRICAL RESULTS

To measure the effectiveness of the proposed Vaccine, we compared its performance with the performance of the well-established masking algorithms of CBC and CFB. The behavior of Vaccine was observed using three separate profiles of simple, moderate, and complex. These observations are named VAC_s, VAC_m, and VAC_c.

Two plaintext templates of *natural* and *synthetic* were chosen and 100 plaintexts were generated for each template. Each plaintext following the first template was selected from a natural document made up of the lower and upper-case alphabets and the 10 digits—total of 62 unique symbols. Each plaintext following the second template was randomly synthesized using the 10 symbols set of {A, b, C, L, x, y, 0, 4, 6, 9}. The goal was to synthesize plaintexts with high occurrences of a small set of symbols. Each plaintext created under both templates was 1K bytes long.

For each plaintext, two ciphertexts of C_a and C_d were generated using Advanced Encryption System (AES-128) and Data Encryption Standard (DES) algorithms [20][21][22]. The masking approaches of CBC, CFB, VAC_s, VAC_m, and VAC_c were applied separately on C_a and C_d generating the masked ciphertexts of:

$\{C_a^{cbc}, C_a^{cfb}, C_a^{vac_s}, C_a^{vac_m}, C_a^{vac_c}\}$ and
$\{C_d^{cbc}, C_d^{cfb}, C_d^{vac_s}, C_d^{vac_m}, C_d^{vac_c}\}$.

When CFB was applied on C_a and C_d the key lengths were 64-bit and 128-bit, respectively, and the IV was chosen from a natural document. (The least significant 64 bits of the 128-bit key was used as the key when CFB was applied on C_a. The key used by VAC_s, VAC_m, and VAC_c was also borrowed from the least significant 32 bits of the 128-bit key used for CFB.)

Let us consider the first set of masked ciphertexts $\{C_a^{cbc}, C_a^{cfb}, C_a^{vac_s}, C_a^{vac_m}, C_a^{vac_c}\}$ generated from C_a. The

following steps are used to compare the effectiveness of the proposed Vaccine with CBC and CFB. (The same steps are also followed to compare the effectiveness of the proposed Vaccine with CBC and CFB using the masked ciphertexts of $\{C_d^{cbc}, C_d^{cfb}, C_d^{vac_s}, C_d^{vac_m}, C_d^{vac_c}\}$.)

a. Get the list of unique symbols, which makes up the plaintext, List=$\{\sigma_1 \ldots \sigma_m\}$.
b. Get the frequency of symbol σ_i, for i = 1 to m, and calculate the average frequency of the symbols.
c. Repeating the next two steps for every symbol, σ_i, in the list.
d. Identify the locations for all the occurrences of the symbol, σ_i, in the plaintext, ($\ell^i_1 \ldots \ell^i_n$).
e. Identify the bytes in the locations of ($\ell^i_1 \ldots \ell^i_n$) within the $C_a^\bullet$ and calculate the Hamming distance, h_j, between the two bytes in location ℓ_j, for j=1 to n, in the plaintext and $C_a^\bullet$. The overall average of Hamming distance for the symbol σ_i is $h_{\sigma i}$=Average($h_1 \ldots h_n$),
f. Concluding that the underline masking methodology with the highest average values of the Hamming distances have a superior performance.

TABLE I. AVERAGE OF HAMMING DISTANCES BETWEEN THE TWO 100 PLAINTEXTS OF 1K BYTE LONG (GENERATED BY TWO TEMPLATES) AND THEIR RELATED MASKED CIPHERTEXTS: (A) ENCRYPTED BY AES AND (B) ENCRYPTED BY DES

		AES-128				
	Avg. Symb.	*CBC*	*CFB128*	*VAC$_s$*	*VAC$_m$*	*VAC$_c$*
Tem.	Freq.	Dist.	Dist.	Dist.	Dist.	Dist.
Syn.	103	3.568	3.570	4.415	4.373	4.411
Natu.	16.5	3.569	3.561	4.423	4.361	4.411
(a)						
	Avg. Symb.	DES				
		CBC	*CFB64*	*VAC$_s$*	*VAC$_m$*	*VAC$_c$*
Tem.	Freq.	Dist.	Dist.	Dist.	Dist.	Dist.
Syn.	103	3.527	3.526	4.182	4.153	4.223
Natu.	16.5	3.513	3.515	4.176	4.141	4.221
(b)						

TABLE II. THROUGHPUT AVERAGE IN MILISECOND FOR THE TWO 100 PLAINTEXTS OF 1K BYTE LONG (GENERATED BY TWO TEMPLATES): (A) ENCRYPTED BY AES AND (B) ENCRYPTED BY DES

		AES-128				
	Avg Symb.	*CBC*	*CFB128*	*VAC$_s$*	*VAC$_m$*	*VAC$_c$*
Tem.	Freq.	*TPut.*	*TPut.*	*TPut.*	*TPut.*	*TPut.*
Syn.	103	4545	11111	25000	33334	20000
Natu.	16.5	12500	10000	16667	20000	12500
(a)						
	Avg Symb.	DES				
		CBC	*CFB64*	*VAC$_s$*	*VAC$_m$*	*VAC$_c$*
Tem.	Freq.	*TPut.*	*TPut.*	*TPut.*	*TPut.*	*TPut.*
Syn.	103	3846	11111	20000	25000	14286
Natu.	16.5	10000	10000	14286	20000	11111
(b)						

The outcome of applying the above steps on the ciphertexts of $\{C_a^{cbc}, C_a^{cfb}, C_a^{vac_s}, C_a^{vac_m}, C_a^{vac_c}\}$ and $\{C_d^{cbc}, C_d^{cfb}, C_d^{vac_s}, C_d^{vac_m}, C_d^{vac_c}\}$ are shown in Table I.a and Table I.b. We have also used the system clock to calculate the average throughput (in millisecond) for the masking approaches of CBC, CFB, VAC_s, VAC_m, and VAC_c and reported in Tables II.a and II.b.

In addition, a *masking strength* of μ ($0 < \mu < 1$), is introduced that is defined as $\mu = N_{inst} / N_{occ}$, where N_{inst} is the number of unique bytes in the masked ciphertext representing the instantiations of the N_{occ} occurrences of symbol σ_i in the underlying plaintext of the masked ciphertext. The masking strength for CBC, CFB, VAC_s, VAC_m, and VAC_c are presented, respectively, in Tables III.a and III.b.

TABLE III. AVERAGE MASKING STRENGTH FOR THE TWO 100 PLAINTEXTS OF 1K BYTE LONG (GENERATED BY TWO TEMPLATES): (A) ENCRYPTED BY AES AND (B) ENCRYPTED BY DES

		AES-128				
	Avg. Symb.	*CBC*	*CFB128*	*VAC$_s$*	*VAC$_m$*	*VAC$_c$*
Tem.	Freq.	μ	μ	μ	μ	μ
Syn.	103	0.506	0.486	0.451	0.540	0.571
Natu.	16.5	0.882	0.878	0.845	0.890	0.889
(a)						
	Avg. Symb.	DES				
		CBC	*CFB64*	*VAC$_s$*	*VAC$_m$*	*VAC$_c$*
Tem.	Freq.	μ	μ	μ	μ	μ
Syn.	103	0.501	0.494	0.490	0.564	0.570
Natu.	16.5	0.878	0.894	0.880	0.909	0.893
(b)						

V. FINDINGS FOR SINGLE-PAIRED-USER ENVIRONMENT

The performance of the presented new cipher block approach, Vaccine, for masking and unmasking of ciphertexts seems superior to the performance of the well-known masking approaches of CBC and CFB.

The advantages of Vaccine over CBC and CFB are numerated as follows:

a. The key and patterns' profile may hide in the masked ciphertext.
b. The block size for Vaccine is not fixed and it is selected randomly.
c. Each block is divided into segments of random size.
d. The masking pattern changes from one byte to the next in a given segment.
e. Masking a ciphertext using Vaccine demands mandatory changes in the ciphertext. Therefore, the identity transformation could not be provided through the outcome of Vaccine. The simple proof is that the Hamming weight is modified.
f. The results revealed that on average:
   i. The Hamming distance between masked and unmasked occurrences of a byte using Vaccine is 0.72 bits higher than using CBC and CFB.
   ii. Vaccine throughput is 3.4 times and 1.8 times higher than throughput for CBC and CFB.

iii. Vaccine masking strength is 1.5% and 1.8% higher than masking strength for CBC and CFB.
iv. VAC_m masking strength is 3.6% and 3.7% higher than masking strength for CBC and CFB. And VAC_c masking strength is 3.9% and 4.2% higher than masking strength for CBC and CFB.

VI. MULTI-PAIRED-USER ENVIRONMENT

A secure multi-user environment [13][23][24] is represented by a graph, G(V, E), where V is a set of vertices representing the user members of the environment and E is a set of bi-directional edges indicating the communication between paired users. As an example, let us consider a secure multi-user environment, Figure 6, for which V is composed of the set $\{v_1, v_2, v_3\}$ and E is composed of the set $\{e_{1,2}, e_{1,3}, e_{2,3}.)$

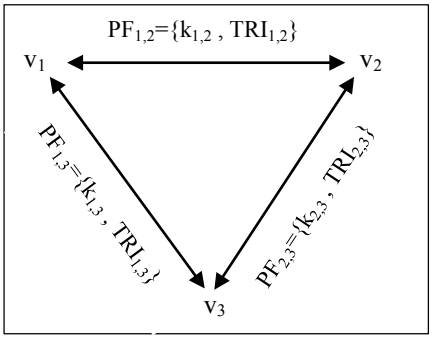

Figure 6. A safe multi-user environment with three cooperative users

There is a pairwise masking image, $MI_{i,j}$ associated with each edge, $e_{i,j}$, that enables the two users of v_i and v_j to communicate securely with each other (i.e., vaccinated messages can flow between v_i and v_j through the edge of $e_{i,j}$.) The pairwise $MI_{i,j}$ has two major components of a pairwise key, $k_{i,j}$, and a pairwise profile, $PF_{i,j}$, Figure 2. For a pairwise $k_{i,j}$ of four-byte long the $PF_{i,j}$ includes a prefix, $PRE_{i,j}$, and five sets of triplets (W, M, R) , $TRI_{i,j} = \{Tri^0_{i,j}, \ldots, Tri^4_{i,j}\}$, that each one prescribes a set of patterns. The number of bits to accommodate one triplet in PF_{ij} is 14. The pairwise key $k_{i,j}$ can grow as many bytes as desired (the maximum of 512 bytes.) The number of triplets in $TRI_{i,j}$ also grows with the growth of key length. For each added byte to the key, a new triplet is added to the PF_{ij}. Beyond the key length of sixteen bytes, the length of prefix also grows one bit at a time. Each added bit to the prefix doubles the length of key in bytes. Since there are five reserved bits in PF_{ij} and they can be used to increase the length of the prefix, the key length can grow up to 512 bytes. Each one of these bytes can be addressed by the second partition of the preference bits (i.e., all the preference bits excluding the most significant bit.) For the key length of L_k bytes, the PF length, L_p, is calculated in bits using formula (3):

$$L_p = 14(L_k+1) +26 \quad (3)$$

Since the pairwise masking images are associated with edges, we also refer to MI, k, and PF as the edge masking image, edge key, and edge profile, respectively.

The secure multi-user environment of our interest has the option of revoking existing users. That is, upon revoking a user the keys become vulnerable and a key management approach needs to be employed. Key management is achieved in two ways: key distribution [13][14][15][16] and key agreement[17][18][19]. In the key distribution approach, each user has its own private key and it is only shared with a key-distribution center (KDC). In the case that v_k needs to communicate with v_m, KDC is asked for a session key that will be generated and delivered to both v_k and v_m.

In the key agreement approach, a number of users agree on having one community-based key, K, and all users participate in building such a key by donating their individual keys. The community-based key remains private to the members.

The revoking option using key distribution has the same overhead cost for using CBC, CFB, and Vaccine. However, adaptation of the Vaccine into the key agreement approach suggests some interesting developments that need to be discussed.

The key agreement approach requires that as soon as a user, v_k, is revoked all the edges be re-keyed, re-profiled, or both (regardless of substituting or not substituting the revoked user.) Let us take a closer look at these three options. During the masking process conducted by Vaccine, edge key and edge profile, play a role. In fact, the masking of a ciphertext is completed by the use of the *model* and *native byte* (both terms explained in Section III), which in turn was generated by employing the prefix (PRE), triplets (TRI), and key (k), along with the plaintext. Therefore, the revoked user cannot correctly mask (or de-mask) a ciphertext as long as one of the three parameters of PRE, TRI and k changes. However, both re-keying and re-profiling may provide a higher masking strength.

To implement the Vaccine adaptation into the key agreement approach we present three algorithms of Keys, Carve, and Profile. The Keys algorithm dynamically creates N pairwise keys of a given length for N edges that collectively make the community-based key, K. The algorithm Carve randomly generates a set of triplets based on the key length and also enforces the internal constraints on the randomly generated triplets. The Algorithm Profile generates either one profile used by all pairwise keys or N profiles for the N edges. The details of the three algorithms are covered in the following three subsections.

A. Algorithm keys

Dynamically creating a Community-based key, K, which is composed of a large number of edge keys (one per edge and for the purpose of re-keying) is encapsulated by the algorithm Keys, Figure 7. In the N iterations of Step 3, which is the number of edge keys, the algorithm delivers N pairwise keys with the length of L_k bytes that are randomly

generated. This is accomplished by randomly generating L_k binary strings with the length of L_k = 256-bit (Step 2.) Eight bits from each one of the L_k strings are selected randomly (Step 3.a.) The obtained bytes are concatenated in a random order to make one edge key of L_k bytes long (Step 3b.) The resulting edge key, k, is added to the Community-based aggregated key, K, (Step 3.c.)

Algorithm Keys (L_k, N)
Input: L_k, which is the length of key in bytes and N is the number of edge keys needed.
Output: A Community-based key, K, made up of N edge keys randomly generated on fly.
Method:
Step 1: k = ""; λ =0; J =0;
Step 2: Create L_k binary numbers of 256-bit long: S_1, . . ., S_{Lk};
Step 3: Repeat (while λ < N)
  a: Select eight bits randomly from each string: B_1, . . ., B_{Lk};
  b: Repeat (while j < L_k)
    a1: Randomly pick a byte, B', from the set{ B1, . . ., B_{Lk}};
    a2: k = k|| B'; j = j++;
  End;
  c: K = K||k;
  d: k = "";
  e: j =0;
  f: λ = λ++;
End;
End;

Figure 7. Algorithm Keys

B. Algorithm Carve

The algorithm Carve, shown in Figure 8, accepts an edge key with length of L_k bytes and randomly creates a binary number, TRI, of the length L_p delivered by formula (3) (Step 2.)

Considering Figure 2, the length of $PRI_{i,j}$ in the edge profile of $PF_{i,j}$ is 26 bits. As we mentioned previously the number of bits needed to express each triplet of (W, M, R) in $PF_{i,j}$ is 14 bits. Therefore, each 14 bits after the 26-bit prefix is made up of three parts (Part1, Part2, and Part3). Part1 (W) is 3 bits long and starts from the location r1 = 26 in the profile. Part2 (M) is also 3 bits long and starts immediately after Part1 (i.e., location r2 = r1 +3.) Part3 (R) is 8 bits long and starts 6 bits after Part1 (i.e., location r3 = r1 +6.) (Step 3 gets the starting points of Part1, Part2, and Part3.)

The locations of Part1of all the triplets are separated by 14 bits and the same is true for the locations of Part2 and Part3 of all triplets. The three parts of each triplet are obtained in Step 4.a and converted to decimal numbers in Step 4.b. In reference to the three parts of the triplets we have three concerns that need to be addressed. These concerns are in reference to the constraints on W, R, and M values in W-R-Bit-M-Flip-Circular-Swap technique.

The first concern is about the validity of the equivalent decimal value carried by (Part1+1) of the triplet. If the value is less than 2, the value changes to 2 (Step 4.c.)

The second concern is about the equivalent decimal value in Part2 that must be at least one less than the value in Part1 (a constraint rule between W and M.) If the value of Part 2 is greater than or equal to the value in Part1, it is reduced to make it smaller such that the value of Part 1 is higher by 1 (Step 4.d.)

Algorithm Carve (k, L_k)
Input: An edge key, k, with the length of L_k bytes.
Output: Dynamically generating a set of triplets, TRI, that is in agreement with k.
Method:
Step 1: i = 0;
Step 2: Create a random binary number of $L_p = 14(L_k+1) +26$ bits long, TRI;
Step 3: $r_1 = 26$; r2 = r_1 +3; r3 = r_1 +6;
Step 4: Repeat while (i <14 L_k)
  a: $Part_i^1$ = SUBSTR(TRI, r_1+i, 3);
    $Part_i^2$ = SUBSTR(TRI, r_2+i, 3);
    $Part_i^3$= SUBSTR(TRI, r_2+i, 7);
  b: f_1 = DECIMAL($Part_i^1$)+1; f_2 = DECIMAL($Part_i^2$) ;
    /*DECIMAL function converts a given binary number into decimal number*/
  c: If (f_1 < 2) Then f_1 = 2;
  d: If ($f_1 \le f_2$) Then $f_2 = f_1$-1;
  e: If (COUNT($Part_i^3$) – f_2 > 0)
    Then randomly flip (COUNT($Part_i^3$) – f_2) bits in $Part_i^3$ to 0;
    If (COUNT($Part_i^3$) – f_2 < 0)
    Then randomly flip |COUNT($Part_i^3$) – f_2| bits in $Part_i^3$ to 1;
  f: SUBSTR(TRI, r_1+i, 3) = BINARY(f_1);
    SUBSTR(TRI, r_2+i, 3) = BINARY(f_2); /*BINARY function converts a given decimal number into binary number*/
    SUBSTR(TRI, r3+i, 8)= $Part_i^3$;
  g: i = i+14;
End;
End;

Figure 8. Algorithm Carve

The third concern is about the validity of the content of Part3 of the triplet. The count of 1s in Part3 must be equal to the equivalent decimal value carried by Part2 of the triplet. If the count of 1s is higher, randomly enough 1s are flipped to zero and if the number of 1s is lower, randomly enough 0s are flipped to one to solve the problem (Step 4.e.)

Replace the three parts of the triplet with their new changes (Step 4.f.) By adjusting the index of i (Step 4.g) all the parts in TRI are inspected and corrected.

C. Algorithm Profile

Now we are ready to look into two cases of using:

(a) a new aggregated community-based key, K, and (i) one new profile for all edges or (ii) one new profile per edge and
(b) the existing aggregate key for the community and (i) one new profile for all edges or (ii) one new profile per edge.

One may raise the question of why case (b) is a valid case to begin with. The question is an important one because

keeping the existing aggregate key may jeopardize the overall security of the system. To answer the question, as it was mentioned before, the PF (composed of PRE and TRI) and k are needed to complete the vaccination. As long as either PF or k changes, the vaccination results are different. Therefore, case (b) is a legitimate one. The set of steps for implementing case (a) and case (b) are given in the algorithm Profile shown in Figures 9.

```
Algorithm Profile (Option, Lk, N, EK)
Input: Option (Option = "10" means a new aggregate key
       and one new profile is needed for the entire
       aggregate key. Option = "11" means a new
       aggregate key and one new profile is needed for
       each edge key within the aggregate key. Option =
       "00" means one new profile is needed for the entire
       aggregate old key. Option = "01" means one new
       profile is needed for each edge key within the old
       aggregate key.), a desired key length, Lk, the number
       of needed edge keys, N, and an existing community-
       based key, EK.
Output: (a) A new community-based key K and one or N
            new profiles or
        (b) One or N new profiles
Method:
 Step 1: Switch (Option)
          a: Case "10":
              K = Keys(Lk, N);
              k = one individual key in the community-
                  based key K;
              Carve(k, Lk);
              Break;
          b: Case "11":
              K = Keys(Lk, N);
              Repeat for every key, ki, in the community-
                | based key K
                Carve(ki, Lk);
              End;
              Break;
          c: Case "00":
              ek = one individual key in EK.
              Carve(ek, Lk);
              Break;
          d: Default:
              Repeat for every key, eki, in EK.
                | Carve(ki, Lk);
              End
 End;
```

Figure 9. Algorithm Profile

The parameter Option is 2 bits long and the values "1" and "0" for the most significant bit represent case (a) and case (b), respectively. The option (i) in both cases is represented by the least significant bit set to "0". The option (ii) in both cases is represented by the least significant bit set to "1".

For case (a) option (i), the algorithm does not re-key the existing community-based key, EK, however, it generates one profile used by all users (Step 1.a) and for the same case option (ii), no re-keying takes place and the algorithm generates a new edge profile for every edge (Step 1.b.) For the case (b) the algorithm reacts the same way that it has reacted for case (a) except that for both options of (i) and (ii) first, a new community-based, K, is generated and then one new edge profile (Step 1.c) or N edge profiles (one per edge) are generated (Step 1.d.)

It is worth mentioning that in the case of the pair-wise community growth, the above algorithms easily can provide for re-keying and re-profiling of only the new edges without disturbing the existing edge masking images.

VII. FINDINGS FOR MULTI-PAIRED-USER COMMUNITY

The findings about the behavior of the Vaccine in a secure multi-paired-user community are the same as those of a single-paired-user environment. The reason stems from the fact that the vaccine performs the same function in a single-paired-user system as it does on each individual paired-user in the community. However, the re-keying and re-profiling are pure overhead and this is true anytime that a key agreement is used. To have a better understanding of this overhead cost we have completed the time complexity calculations for the three algorithms of Keys, Carve, and Profile, Table IV. The notations of L_k and N are used for the key length and the community size, respectively.

TABLE IV. TIME COMPLEXITY FOR ALGORITHMS KEYS, CARVE, AND PROFILE

Algorithm	Time Complexity
Keys	O(N): if $L_k < N$ $O(N^{\lceil L_k/N \rceil})$: otherwise
Carve	O(1)
Profile	*For Option = "10" and "11"* O(N): if $L_k < N$ $O(N^{\lceil L_k/N \rceil})$: otherwise
	For Option = "00" O(1)
	For Option = "01" O(N)

The findings reveal that the performance of the three algorithms in the worst case is linear to the size of the community assuming the size is larger than the length of the key given to the community members. The assumption is not far from reality and, in general, any linear growth in delivery of an algorithm with the large size of community is well accepted behavior.

VIII. CONCLUSION AND FUTURE RESEARCH

A new cipher block approach, Vaccine, for masking and unmasking of ciphertexts was introduced and implemented with one major goal in mind: Removal of inherited-features from a ciphertext. The methodology was first applied to a single-paired-user environment and the performance of the Vaccine was scrutinized by comparing it with the two well-known approaches of CBC and CFB modes. The advantages of the Vaccine application in a single-paired environment were numerated in Section V. The adoption of Vaccine for a multi-paired-user environment with the option of user revocation was also explored, which resulted in a methodology for re-keying

and re-profiling. One major property of the new keys and new profiles was that they were generated by starting with the creation of completely random and long binary strings. In general, using such strings provide a better protection for the keys and profiles against the adversarial attempts.

As future research, building a new version of the Vaccine is currently in progress to make the throughput and the masking strength of the methodology even higher. Study of Vaccine as an authentication method in a secure multi-paired-user environment has been scheduled.

REFERENCES

[1] R. Hashemi, A. A. Rasheed, A. Bahrami, and J. Young, "Vaccine: A Block Cipher Method for Masking and Unmasking of Ciphertexts' Features", The Second International Conference on Cyber-Technologies and Cyber-Systems (CYBER 2017), Barcelona, Spain, Nov. 2017, pp. 41-47.

[2] A. A. Rasheed, M. Cotter, B. Smith, D. Levan, and S. Phoha, "Dynamically Reconfigurable AES Cryptographic Core for Small, Power Limited Mobile Sensors", The 35th IEEE International Performance Computing and Communication Conference and Workshop, pp. 1-7, 2016.

[3] G. P. Saggese, A. Mazzeo, N. Mazzocca, and A. G. M. Strollo, "An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm", LNCS 2778, pp. 292-302, 2003.

[4] N. Pramstaller and J. Wolkerstorfer, "A Universal and Efficient AES Co-processor for Field Programmable Logic Arrays", Lecture Notes in Computer Science, Springer, Vol.3203, pp. 565-574, 2004.

[5] B. Moeller. *Security of CBC Cipher suites in SSL/TLS: Problems and Countermeasures. [Online]*. Available from: https://www.openssl.org/~bodo/tls-cbc.txt

[6] W. Stallings, "Cryptography and Network Security: Principles and Practice", Pearson, 2014.

[7] C. A. Henk and V. Tilborg, "Fundamentals of Cryptology: A Professional Reference and Interactive Tutorial", Springer Science & Business Media, 2006.

[8] N. Feruson, B. Schneier, and T. Kohno, "Cryptography Engineering: Design Principles and Practical Applications", Indianapolis: Wiley Publishing, Inc., pp. 63-64, 2010.

[9] W. F. Ehrsam, C. H. W. Meyer, J. L. Smith, and L. W. Tuchman, "Message Verification and Transmission Error Detection by Block Chaining", US Patent 4074066, 1976.

[10] C. Kaufman, R. Perlman, and M. Speciner, "Network Security", 2nd ed., Upper Saddle River, NJ: Prentice Hall, p. 319, 2002.

[11] National Institute of Standards and Technology (NIST), Advanced Encryption Standard (AES), Federal Information Processing Standards Publications 197 (FIPS197), Nov. 2001.

[12] S. Vaudenay, "Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS...", Lecture Notes in Computer Science, Springer, vol. 2332, pp. 534-546, 2002.

[13] S. Tanaka and F. Sato, "A Key Distribution and Rekeying Framework with Totally Ordered Multicast Protocols", Proceedings of the 15th International Conference on Information Networking, 2001, pp. 831-838.

[14] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-Hellman Key Distribution Ex-tended to Group Communication", Proceedings of the 3rd ACM Conference on Computer and Communication Security, 1996, pp. 31-37.

[15] C. Becker and U. Wille, "Communication Complexity of Group Key Distribution", In 5th ACM Conference on Computer and Communication Security, 1998, pp. 1-6.

[16] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system", Advances in Cryptology - EUROCRYPT'94, 1994.

[17] M. Steiner, G. Tsudik, and M. Waidner, "CLIQUES: A New Approach to Group Key Agreement", Proceedings of the 18th Internal Conference on Distributed Computing Systems, 1998, pp. 380-387.

[18] G. Ateniese, M. Steiner, and G. Tsudik, "New Multiparty Authentication Services and Key Agreement Protocols", IEEE Journal on Selected Areas in Communications, Vol. 18, No. 4, 2000, pp. 628-639.

[19] Y. Kim, A. Perrig, and G. Tsudik, "Communication-efficient group key agreement", 17th International Information Security Conference, 2001.

[20] H. Kuo-Tsang, C. Jung-Hui, and S. Sung-Shiou, "A Novel Structure with Dynamic Operation Mode for Symmetric-Key Block Ciphers", International Journal of Network Security & Its Applications, Vol. 5, No. 1, p. 19, 2013.

[21] H. Feistel, "Cryptography and Computer Privacy", Scientific American, Vol. 228, No. 5, pp 15–23, 1973.

[22] F. Charot, E. Yahya, and C. Wagner, "Efficient Modular-Pipelined AES Implementation in Counter Mode on ALTERA FPGA", (FPL 2003), Lisbon, Portugal, pp. 282-291, 2003.

[23] S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting", Proceedings of the ACMSIGCOMM , 1997, pp. 277-288.

[24] D. Wallner, E. Harder, and R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, 1999.