Abstract — This paper introduces a permutation generation mechanism based on a shared secret key. The generated permutation vectors are used as encryption keys in a stream ciphering cryptosystem. We investigated various types of attacks on the known stream cipher RC4 and patched most of its loopholes, especially biased-byte and state-related attacks. Unique to our approach, we prove mathematically that the complexity of brute-forcing such a system is $\Omega(2^n)$, where $n$ is the key size in bytes. This paper also presents a complete security model using permutation-based encryption, in order to handle privacy. In addition, our approach achieved higher performance than that of existing peer techniques, while maintaining solid security. Experimental results show that our system is much faster than the existing security mechanisms, such as AES and DES.

Index Terms — Biased byte attack, exponential brute force, network security permutation vector generation, stream cipher

I. INTRODUCTION

A permutation describes an arrangement, or ordering, of objects [1]. Many algorithmic problems seek the best way to order a set of objects, including traveling salesman (the least-cost order to visit $n$ cities), width (order the vertices of a graph on a line so as to minimize the length of the longest path), and graph isomorphism (order the vertices of one graph so that it is identical to another). Any algorithm for solving such problems must construct a series of permutations along the way.

There are $n!$ permutations of $n$ items, which grow exponentially to generate all permutations. Numbers like these should calm the urge of anyone interested in exhaustive search and help explain the importance of generating random permutations.

Fundamental to any permutation-generation algorithm is a notion of sequence order, the sequence in which the permutations are constructed, from first to last. The most natural generation order is lexicographic, the order in which permutations would appear if they were sorted numerically. Lexicographic order for $n = 3$ is $\{1, 2, 3\}$, $\{1, 3, 2\}$, $\{2, 1, 3\}$, $\{2, 3, 1\}$, $\{3, 1, 2\}$, and finally $\{3, 2, 1\}$. Although lexicographic order is aesthetically pleasing, there is often no particular reason to use it. Indeed, non-lexicographic orders lead to faster and simpler permutation generation algorithms [1][2].

The generation of random permutations is important for simplifying security algorithms. One way to do this [3] is the following two-line, linear-time algorithm. We assume that $Random(i,n)$ generates a random integer between $i$ and $n$.

$$\text{for } i \leftarrow 1 \text{ to } n \text{ do}$$

$$a_i \leftarrow i$$

/* $a = (1, 2, ..., n)$*/

$$\text{for } i \leftarrow 1 \text{ to } n \text{ do}$$

$$\text{swap}(a_i, a_{Random(i,n)})$$

It is not obvious that this algorithm generates all permutations uniformly. However, the validity of a security algorithm that is based on such linear generation of permutation vectors is yet to be proven in relation to peer algorithms. Permutations are also used to achieve “diffusion”, a critical characteristic of a secure cipher [4], in symmetric-key encryption algorithms such as DES [5], Twofish [6] and Serpent [7]. Some permutations in cryptographic algorithms are not one-way only. For instance, the Expansion Permutation in DES maps some bits in the source data vector to multiple destinations in the result data vector [8].

The rest of this paper is organized as follows. Section 2 presents a description of the RC4 stream cipher with some flaws that made it insecure. In Section 3, we propose a permutation technique to be used in building secure stream ciphers. The base theorem of our crypto system is presented in Section 4 as well as some useful lemmas. Section 5 presents in detail the theorem proof which covers all cases of forming a new permutation vector. Our SDES crypto system is briefly presented in Section 6. Section 7 shows some simulation experiments and comparison of SDES with the state-of-the-art security mechanisms in terms of throughput. The conclusion is given in Section 8.

II. RELATED WORK

Stream cipher algorithms are an important class of encryption techniques. They encrypt individual characters (usually binary digits) of a plaintext message one at a time, using an encryption transformation that varies with time [9]. In contrast, block ciphers tend to simultaneously encrypt groups of characters of a plaintext message using a fixed encryption transformation [10]. Stream ciphers
are generally faster than block ciphers [11] and have less complex hardware circuitry. They are also more appropriate, and in some cases mandatory (e.g., in some telecommunications applications), when buffering is limited or when characters must be individually processed as they are received.

RC4 is one of the dominant stream ciphers used in secure data communications [12]. Ron Rivest of RSA Data Security Inc developed the RC4 cipher in 1987, the details of which were published in 1996. RC4 is a stream cipher encryption system, which uses a shared key to shuffle a permutation vector, S, and randomly selects elements from it to encrypt and decrypt messages transferred during a particular communication session [13].

RC4 consists of two parts, as shown in Figure 1. The first is a key-scheduling algorithm, KSA, which turns a random key (whose typical size is 40-256 bits) into an initial permutation vector S of \{1, ..., n\}; the second is a pseudo-random generation algorithm, PRGA, which uses S to generate a pseudo-random output sequence.

<table>
<thead>
<tr>
<th>KSA(K):</th>
<th>PRGA(K):</th>
</tr>
</thead>
<tbody>
<tr>
<td>/<em>Initialization</em>/</td>
<td>/<em>Initialization</em>/</td>
</tr>
<tr>
<td>for i ← 1 to n do</td>
<td>i ← 1</td>
</tr>
<tr>
<td>S ← i</td>
<td>j ← 1</td>
</tr>
<tr>
<td>j ← 1</td>
<td>/<em>Generation loop</em>/</td>
</tr>
<tr>
<td>/<em>Scrambling</em>/</td>
<td>i ← i + 1</td>
</tr>
<tr>
<td>for i ← 1 to n</td>
<td>j ← j + S_i</td>
</tr>
<tr>
<td>j ← j + S_i</td>
<td>swap(S_i, S_j)</td>
</tr>
<tr>
<td>swap(S_i, S_j)</td>
<td>z ← S_{i+j}</td>
</tr>
</tbody>
</table>

Figure 1: Key scheduling and pseudo-random generation algorithms of RC4

There are several methods of attempting a brute force attack on RC4 that are classified into two categories: KSA-based attacks and PRGA-based attacks. Knowing that the initial state is enough to predict all of the keystream bits (regardless of the shared key K), PRGA-based attacks look for contradictions in the chosen keystream (in order to detect incorrect guesses) and discover some of the initial state entries. There has been considerable analysis of the probabilities of any given value being output by RC4. Most of these analyses have approached RC4 by looking at a given output.

Even though RC4 uses a permutation vector as its internal state box, the generated keystream is not necessarily redundancy-free. Fluhrer and McGrew [14] and Mantin and Shamir [15] defined a class of predictive states in which a non-negligible bias appears in the keystream. In their search for a polynomial-space distinguisher, they came up with a startling theorem, claiming that if \( S_2 = 0 \) and \( S_i \neq 2 \), then \( z_i = 0 \) with probability of 1.

In a good keystream generator, each bit of the output will depend on the entire key for its value; the relationship between the key and a given bit (or set of bits) should be extremely complicated [9]. However, RC4 uses the shared key only once (in the KSA); the shared key is not involved at all in the keystream generation. Recall that at each step of the PRGA, S changes in, at most, two locations; thus we can still expect the prefix of the output stream generated by RC4 from some permutation, S, to be highly correlated with the stream generated from the same S (or a slightly modified one) by RC4 [12].

III. PROPOSED PERMUTATION VECTOR GENERATION

The generation of permutation vectors can be performed recursively. Given a permutation vector \( PV \) (a vector that contains all elements from 1 till \( n \), in a specific order), the generation of the next permutation vector \( PV' \) is based on \( PV \) and some other parameter that provides randomness. Our goal is to generate a large set of PVS whose sequence order is difficult to guess. In fact, the shared secret key (SK) utilization, in swapping the elements of PV, captures the notion of randomness in the abovementioned algorithm. Next is our linear algorithm to generate permutation vectors:

for \( i \leftarrow 1 \) to \( n \) do

\[
PV_i \leftarrow i
\]

for \( i \leftarrow 1 \) to \( n \) do

\[
\text{swap}(PV_i, PV_{SK}) \quad /^/ 1 \leq SK_i \leq n */
\]

The major advantage of using permutation vectors as encryption keys is the avoidance of biased byte analysis, in contrast with RC4 keystreams. In accordance with good keystream philosophy, an entry in the new generated permutation vector is a function of the entire key and the previous permutation vector, i.e., every bit in the new permutation vector is generated after performing exactly \( n \) swaps in the previous vector.

Another major contribution of our permutation generation algorithm is the continuous involvement of the shared key in the permutation vector generation. This will render the state-based attacks obsolete, since the attacker is forced to obtain the state and the key together in order to break the system. In order to increase the level of security, the system should update the shared key internally after each record. Therefore, the attacker is compelled to break a system with pseudo-multiple keys, instead of a single static key.

Most of cryptographic schemes are based on the “reducibility from hard problems” technique, which consists of proving that any successful protocol attack leads directly to the ability to solve a well-studied hard problem [11]. This “reference” problem is considered computationally infeasible, given current knowledge and an adversary with bounded resources, e.g., the “integer factorization” and the “discrete logarithmic” problems. Such analysis yields the so-called provably secure protocols, although the security is conditional on the reference problem’s being truly difficult. On the contrary, we will show that a cryptanalyst is cornered to the brute-force option only in order to guess the lexicographic order of the generated permutation vectors. Hence, we will prove a theorem that underlimits such brute-force algorithmic complexity to an exponential function \( \Omega(2^n) \).
with \( n \) the byte size of the shared secret key.

A. Permute function

A permutation function \( \text{Permute}(X, K) \) is a function that takes a permutation vector \( X \) (of size \( m \)) and a key \( K \) (of size \( m \)), and returns another permutation vector, as follows:

\[
\text{Permute}(X, K): \quad \text{Temp} \leftarrow X \\
\text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
\quad \text{swap}(\text{Temp}_i, \text{Temp}_k) \\
\text{return Temp}
\]

B. Reverse permutation function

Similarly, we can define a reverse function of \( \text{Permute}() \) as follows:

\[
\text{ReversePermute}(Y, K): \quad \text{Temp} \leftarrow Y \\
\text{for } i \leftarrow m \text{ downto } 1 \text{ do} \\
\quad \text{swap}(\text{Temp}_i, \text{Temp}_k) \\
\text{return Temp}
\]

C. Reduction function

A reduction function \( \text{Reduction}(V) \) of a permutation vector \( V \) of size \( m+1 \), is a function that reduces \( V \) to a permutation vector of size \( m \) as follows (illustrated in Figure 2):

\[
\text{Reduction}(V): \quad j \leftarrow 1 \\
\quad \text{for } i \leftarrow 1 \text{ to } m+1 \text{ do} \\
\quad \text{if } V_i \neq m+1 \text{ then} \\
\quad \quad /* \text{ skipping the } m+1 \text{ value}*/ \\
\quad \text{Temp}_j \leftarrow V_i \\
\quad j \leftarrow j + 1 \\
\text{return Temp} \\
\]

Thus, if \( R = \text{Reduction}(V) \), and \( V_p = m+1 \), then:

\[
R_i = \begin{cases} 
V_i, & \text{if } 1 \leq i \leq p \\
V_i, & \text{if } 1 \leq i \leq p
\end{cases}
\]

Figure 2: Multiple reduction of a permutation vector

D. Index map function

The index map \( \text{IMAP}_{X,Y} \) corresponding to two permutation vectors \( X \) (of size \( m \)) and \( V \) (of size \( m \) or higher) is defined as follows (illustrated in Figure 3):

\[
\text{IMAP}_{X,Y}(i) = j \text{ if and only if } X_i = V_j, \quad (i, j \in \{1, 2, \ldots, m \}).
\]

E. Reverse function

A reverse function \( \text{Reverse}(X) \) of a vector \( X \) of size \( m \), is a function that reverses backwards the coordinates of \( X \) (illustrated in Figure 4):

\[
\text{Reverse}(X) \\
\quad \text{Temp} \leftarrow X \\
\quad \text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
\quad \text{Temp}_i = X_{m-i+1} + 1 \\
\text{return Temp}
\]

Figure 4: The corresponding reverse vector of \( X = (2, 1, 5, 8, 2, 4) \)

IV. Theorem Statement

In order to strengthen our security mechanism, we present throughout this paper a detailed proof the following theorem: “Given two permutation vectors \( V \) and \( W \) of size \( m+1 \), there are \( 2^m \) different keys \( K \) (of size \( m+1 \)) that satisfy \( W = \text{Permute}(V, K) \).”

This theorem provides theoretical strength to cryptosystems in a way that a cryptanalyst who managed to obtain two consecutive encryption keys (permutation vectors), which is not an obvious task, will find it very hard to break the system and guess the secret key (\( 2^m \) possibilities) that is indispensible to calculate the next encryption keys.

A. Lemma 1

Given an index map function \( \text{IMAP}_{X,Y} \) of two permutation vectors \( X \) and \( V \), \( \text{IMAP}_{X,Y} \) remains unchanged when swapping any two elements in \( X \) and \( V \), \( \text{swap}(X_{i_1}, X_{i_2}) \) on \( X \) and \( \text{swap}(V_{j_1}, V_{j_2}) \) on \( V \), where \( i_1, i_2, j_1, j_2 \in \{1, \ldots, m \} \) only if \( X_{i_1} = V_{j_1} \) and \( X_{i_2} = V_{j_2} \) (illustration is shown in Figure 5).

\[
\text{Proof:} \quad \text{Suppose that } X_{i_1} = V_{j_1} = a, \text{ and } X_{i_2} = V_{j_2} = b, \text{ then } \text{IMAP}_{X,Y}(i_1) = j_1 \text{ and } \text{IMAP}_{X,Y}(i_2) = j_2. \text{ After performing swap}(X_{i_1}, X_{i_2}) \text{, } X_{i_1} = b \text{ and } X_{i_2} = a. \text{ Also, after performing swap}(V_{j_1}, V_{j_2}) \text{, } V_{j_1} = b \text{ and } V_{j_2} = a. \text{ Thus, } X_{i_1} = V_{j_1} \text{ and } X_{i_2} = V_{j_2}; \text{ then, } \text{IMAP}_{X,Y}(i_1) \text{ is still equal to } j_1 \text{ and } \text{IMAP}_{X,Y}(i_2) \text{ is still equal to } j_2. \text{ Since } i_1 \text{ and } i_2 \text{ are the only indices involved in swap}(X_{i_1}, X_{i_2}), \text{ then } \text{IMAP}_{X,Y}(i) \text{ still maintains the same value, for any other index } i \notin \{i_1, i_2\}.\]
V. Theorem Proof

The proof of the above mentioned theorem is obtained simply by induction, which is based on proving the base case and the induction step.

A. Base Case: \( m = 2 \)

Table I. All possible keys that permute certain permutation vector \( V \) to certain permutation vector \( W \) of size 2

<table>
<thead>
<tr>
<th>( V )</th>
<th>( W )</th>
<th>All Possible values for ( K )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( [1, 2] )</td>
<td>( [1, 2] )</td>
<td>( [1, 2], [2, 1] )</td>
</tr>
<tr>
<td>( [1, 2] )</td>
<td>( [2, 1] )</td>
<td>( [1, 1], [2, 2] )</td>
</tr>
<tr>
<td>( [2, 1] )</td>
<td>( [2, 1] )</td>
<td>( [1, 2], [2, 1] )</td>
</tr>
</tbody>
</table>

Table I shows that there are \( 2^1 = 2 \) different keys \( K \) that permute \( V \) into \( W \), for any permutation vectors \( V \) and \( W \) of size 2.

B. Induction Step

Given two permutation vectors \( V \) and \( W \) of size \( m+1 \), we will prove that there are \( 2^m \) keys that permute \( V \) into \( W \), assuming that for any two permutation vectors \( X \) and \( Y \) of size \( m \), there are \( 2^{m-1} \) keys that permute \( X \) into \( Y \) (inductive hypothesis).

Given two permutation vectors \( V \) and \( W \) of size \( m+1 \) (where \( V_p = W_{p+1} = m+1 \)), we extract three permutation vectors \( X \), \( Y \) (Figure 6) and \( Z \) (Figure 7), of size \( m \), as follows. \( X \) and \( Y \) are the corresponding reductions of \( V \) and \( W \), respectively, i.e., \( X = \text{Reduction}(V) \) and \( Y = \text{Reduction}(W) \).

\( Z \) is obtained directly from \( Y \) as follows:

\[
Z_i = \begin{cases} 
Y_i, & \text{if } i < p_1 \\
Y_{p_1 - 1}, & \text{if } i = p_1 \\
Y_{i - 1}, & \text{if } p_2 < i < p_1 \\
Y_i, & \text{if } i \geq p_1
\end{cases}
\]

(1)

Due to the insertion of the element \( m+1 \) into \( X \), \( V_{i+1} = X_i \) when \( i \geq p_1 \); hence the next modification is necessary, in order to cope with the shifting of \( X \) elements, indexed from \( p_1 \) through \( m \), filling \( V \) elements, indexed from \( p_1 + 1 \) through \( m+1 \):
for \( i \leftarrow 1 \) to \( m+1 \) do
if \( i \neq p_1 \) then
if \( K^{V-W}_{i} \geq p_1 \) then increment \( K^{V-W}_{i} \)

Now, we will work on task (ii): i.e., moving \( m+1 \) in \( V \) from \( p_1 \) to \( p_2 \) (Figure 9). This can be done easily by setting \( K^{V-W}_{p_1} \) to \( p_2 \):

\[
K^{V-W}_{p_1} = p_2
\]

Figure 9: Modification of the \( p_1 \) entry to let \( m+1 \) migrate from \( p_1 \) to \( p_2 \)

However, this single change in \( K^{V-W} \) leads to two side effects. First, when \( \text{swap}(V_{p_1}, V_{K^{V-W}_{p_1} = p_2}) \) is executed, \( X_{p_2} \) and \( V_{p_1} \) are no longer identical, violating the strict equality of \( V_i = X_i \), when \( i < p_1 \) (Figure 10). In fact, \( X_{p_2} \) will equal \( V_{p_1} \).

![Figure 10: Violation of the reduction property when a unilateral swap occurs on \( V \) only](image)

Any further swapping involving \( X_{p_2} \) (using \( K^{V-Z} \)) should be translated to an access to \( V_{p_1} \) by \( K^{V-W} \):

for \( i \leftarrow p_1+1 \) to \( m+1 \) do
if \( K^{V-W}_{i} = p_2 \), then \( K^{V-W}_{i} \leftarrow p_1 \)

The second side effect is that, at the end of the permutation algorithm, the elements of the resulting vector between position \( p_2 \) and \( p_1 \) will be rotated to the left, compared to the expected \( W \).

Hence, we failed to obtain \( W \) by permuting \( V \) using \( K^{V-W} \), which was constructed based on \( K^{V-Z} \). Had we selected a mysterious permutation vector identical to \( Y \), except with the element range between \( p_2 \) and \( p_1 \) rotated right by one step, we would have obtained our target \( W \) (Figure 11). This mysterious vector is exactly \( Z \). Therefore, we can generate \( K^{V-W} \) using \( K^{V-Z} \) instead of \( K^{V-Z} \).

![Figure 11: The violation of the reduction property misleads the swapping into a vector other than \( W \)](image)

Here is a three-step algorithmic depiction of the above discussion:

**Step I**

1. We skipped the case of \( X \rightarrow Y \), we need to show, based on discussion:

2. **Figure 11:** The violation of the reduction property misleads the swapping into a vector other than \( W \).

3. **Step 2**
   for \( i \leftarrow 1 \) to \( m+1 \) do
   if \( i \neq p_1 \) then
   if \( K^{V-W}_{i} \geq p_1 \) then increment \( K^{V-W}_{i} \)

4. **Step 3**
   for \( i \leftarrow p_1+1 \) to \( m+1 \) do
   if \( K^{V-W}_{i} = p_2 \), then \( K^{V-W}_{i} \leftarrow p_1 \)

In order to prove that the newly constructed key \( K^{V-W} \) permutes \( V \) into \( W \), we will perform \( \text{Permute}(V, K^{V-W}) \) in parallel with \( \text{Permute}(X, K^{V-Z}) \), as shown in Table II. Then, we will show that the result of \( \text{Permute}(V, K^{V-W}) \) is identical to \( W \).

**Phase I:** At the end of this phase, \( \text{IMAP}_{xTemp,vTemp} \) will be set as follows (based on \( \text{IMAP} \) Definition):

\[
\text{IMAP}_{xTemp,vTemp}(i) = \begin{cases} 
  i & \text{if } 1 \leq i < p_1 \\
  i+1 & \text{if } p_1 \leq i \leq m 
\end{cases}
\]

**Phase II:** In order to maintain the same \( \text{IMAP}_{xTemp,vTemp} \) after executing Phase II, we need to show, based on Lemma 1, that \( xTemp_1 = vTemp_1 \), and \( xTemp_{k_1} = vTemp_{k_1} \) for \( 1 \leq k < p_1 \).

1. \( xTemp_1 = vTemp_1 \) (based on \( \text{IMAP}_{xTemp,vTemp} \)).
2. If \( K^{V-W}_{i} < p_1 \), then \( K^{V-W}_{i} \) \( K^{V-Z}_{i} \), based on (2). Since \( xTemp_{j} = vTemp_{j} \), for \( j < p_1 \) (based on \( \text{IMAP}_{xTemp,vTemp} \)), then \( xTemp_{K^{V-Z}_j} = vTemp_{K^{V-Z}_j} \) (substituting \( j \) with \( K^{V-Z}_j \) and \( K^{V-W}_j \)).
3. If \( K^{V-W}_{i} > p_1 \), then \( K^{V-W}_{i} = K^{V-Z}_{i} + 1 \), based on (2) and (3). Since \( xTemp_{j} = vTemp_{j+1} \), for \( j > p_1 \) (based on \( \text{IMAP}_{xTemp,vTemp} \)), then \( xTemp_{K^{V-Z}_j} = vTemp_{K^{V-W}_j} \) (substituting \( j \) and \( j+1 \) with \( K^{V-Z}_j \) and \( K^{V-W}_j \), respectively).

Thus, the same \( \text{IMAP}_{xTemp,vTemp} \) is still maintained at the end of Phase II. We skipped the case of \( K^{V-W}_{i} = p_1 \), because it is inconsistent with (3).

**Table II.** Running Permute(\( X, K^{V-Z} \)) and Permute(\( V, K^{V-W} \)) in parallel

<table>
<thead>
<tr>
<th>Phases</th>
<th>Permute(( X, K^{V-Z} ))</th>
<th>Permute(( V, K^{V-W} ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>( xTemp \leftarrow X )</td>
<td>( vTemp \leftarrow V )</td>
</tr>
<tr>
<td>II</td>
<td>for ( i \leftarrow 1 ) to ( p_1 ) do</td>
<td>for ( i \leftarrow 1 ) to ( p_1 ) do</td>
</tr>
<tr>
<td></td>
<td>( \text{swap}(xTemp_1, xTemp_{p_1+1}) )</td>
<td>( \text{swap}(vTemp_1, vTemp_{p_1+1}) )</td>
</tr>
<tr>
<td>III</td>
<td>( \text{swap}(vTemp_{p_1}, vTemp_{p_2}) )</td>
<td></td>
</tr>
<tr>
<td>IV</td>
<td>for ( i \leftarrow p_1+1 ) to ( m+1 ) do</td>
<td>for ( i \leftarrow p_1+1 ) to ( m+1 ) do</td>
</tr>
<tr>
<td></td>
<td>( \text{swap}(xTemp_{p_1+1}, xTemp_{p_2+1}) )</td>
<td>( \text{swap}(vTemp_{p_1+1}, vTemp_{p_2+1}) )</td>
</tr>
<tr>
<td>V</td>
<td>return ( xTemp )</td>
<td>return ( vTemp )</td>
</tr>
</tbody>
</table>

**Phase III:** Since swapping is performed on \( vTemp \) only, a new \( \text{IMAP}_{xTemp,vTemp} \) is obtained:
Exponential Brute-Force Complexity of A Permutation Based Stream Cipher

Phase IV: In order to maintain the same IMAP<sub>xTemp,vTemp</sub> after executing this set of swaps, we need to show, based on Lemma 1, that V<sub>i</sub> = X<sub>i-1</sub>, and xTemp<sub>\text{K}_{i-1}^{X\to Z}</sub> = \text{vTemp}<sub>K_{i-1}^{Y\to W}</sub>, for i > p.<br />
i. X<sub>i-1</sub> = \text{V}_{i-1} (based on IMAP<sub>xTemp,vTemp</sub>).<br />
ii. If K<sub>i-1</sub><sup>Y\to W</sup> < p<sub>i</sub>, then K<sub>i-1</sub><sup>Y\to W</sup> = K<sub>i-1</sub><sup>X\to Z</sup>, based on (2). Also, K<sub>i-1</sub><sup>Y\to W</sup> = p<sub>2</sub>, based on (4). Since xTemp<sub>j</sub> = \text{vTemp}<sub>p</sub>, for j < p<sub>i</sub> and j ≠ p<sub>2</sub> (based on IMAP<sub>xTemp,vTemp</sub>), then xTemp<sub>\text{K}_{i-1}^{X\to Z}</sub> = \text{vTemp}<sub>K_{i-1}^{Y\to W}</sub> (substituting j with K<sub>i-1</sub><sup>X\to Z</sup> and K<sub>i-1</sub><sup>Y\to W</sup>, respectively).<br />
iii. If K<sub>i-1</sub><sup>Y\to W</sup> = p<sub>i</sub>, then K<sub>i-1</sub><sup>X\to Z</sup> = p<sub>2</sub> based on (4). But xTemp<sub>p<sub>2</sub></sub> = vTemp<sub>p<sub>1</sub></sub>, based on IMAP<sub>xTemp,vTemp</sub>. Therefore, = \text{vTemp}<sub>K_{i-1}^{Y\to W}</sub>.

Phase IV: In this phase, the result of permuting \text{X} and \text{V} is returned. Based on the inductive hypothesis, Permute(\text{X}, K<sub>i-1</sub><sup>X\to Z</sup>) results in \text{Z}, i.e., xTemp = Z. However, we still need to show that vTemp is identical to W.

Based on the last version of IMAP<sub>xTemp,vTemp</sub>, we have:

\[
x\text{Temp}_1 = \begin{cases} 
\text{vTemp}_1, & \text{if } 1 \leq i \leq p_2 \\
\text{vTemp}_1, & \text{if } i = p_2 \\
\text{vTemp}_1, & \text{if } p_2 < i < p_1 \\
\text{vTemp}_{i+1}, & \text{if } p_1 \leq i \leq m 
\end{cases}
\]

Therefore (substituting xTemp with Z):

\[
Z_i = \begin{cases} 
\text{vTemp}_1, & \text{if } 1 \leq i \leq p_2 \\
\text{vTemp}_1, & \text{if } i = p_2 \\
\text{vTemp}_1, & \text{if } p_2 < i < p_1 \\
\text{vTemp}_{i+1}, & \text{if } p_1 \leq i \leq m 
\end{cases}
\]

Then, based on (1), we have:

\[
Y_i = \text{vTemp}_1, \quad \text{if } 1 \leq i \leq p_2 \\
Y_{p_1-1} = \text{vTemp}_{p_1}, \quad \text{if } i = p_2 \\
Y_{i-1} = \text{vTemp}_1, \quad \text{if } p_2 < i < p_1 \\
Y_i = \text{vTemp}_{i+1}, \quad \text{if } p_1 \leq i \leq m
\]

which is equivalent to:

\[
Y_i = \text{vTemp}_1, \quad \text{if } 1 \leq i \leq p_2 \\
Y_i = \text{vTemp}_{i+1}, \quad \text{if } p_2 < i < p_1 - 1 \\
Y_i = \text{vTemp}_{i+1}, \quad \text{if } p_1 \leq i \leq m
\]

After simplification:

\[
Y_i = \begin{cases} 
\text{vTemp}_1, & \text{if } 1 \leq i \leq p_2 \\
\text{vTemp}_{i+1}, & \text{if } p_2 < i < p_1 - 1 \\
\text{vTemp}_{i+1}, & \text{if } p_1 \leq i \leq m 
\end{cases}
\]

But \text{Y} = Reduction(W). Then, based on the reduction definition we have:

\[
\begin{cases} 
W_i = \text{vTemp}_1, & \text{if } 1 \leq i < p_2 \\
W_{i+1} = \text{vTemp}_{i+1}, & \text{if } p_2 \leq i \leq m 
\end{cases}
\]

Also, \text{vTemp}_{p_2} = m+1, based on (5), and W<sub>p_2</sub> = m+1 based on the induction step. Therefore:

\[
\begin{cases} 
W_i = \text{vTemp}_1, & \text{if } 1 \leq i < p_2 \\
W_{p_2} = \text{vTemp}_{p_2}, & \text{if } p_2 \leq i \leq m 
\end{cases}
\]

Hence: \text{vTemp} = W.

At this stage, we have proved that the constructed key K<sub>i-1</sub><sup>Y\to W</sup> does permute \text{V} into \text{W}. Therefore, given that there are 2<sup>m-1</sup> keys that permute \text{X} into \text{Z} (based on the induction hypothesis), we can expand them to generate 2<sup>m-1</sup> keys that permute \text{V} into \text{W}, following (2), (3) and (4) (see example in Figure 12).

Figure 12: First set of two keys that permute (1, 2, 3) to (3, 2, 1)

Second set of 2<sup>m-1</sup> keys

In order to complete the proof, we need to construct
another set of $2^{m-1}$ keys that permute $V$ into $W$. For this purpose, we will investigate the relationship between keys that permute $V$ into $W$, and keys that shuffle the reverse vector of $W$ ($W_r = \text{Reverse}(W)$) into the reverse vector of $V$ ($V_r = \text{Reverse}(V)$).

Suppose that there is a key $\hat{V}^{-W}$ (different from any of the $2^{m-1}$ keys generated in the first set) that permutes $V$ into $W$, i.e., $W_r = \text{Permute}(V, \hat{V}^{-W})$. Then, $V_r = \text{ReversePermute}(W, \hat{V}^{-W})$, based on the reverse permutation definition.

Table III. Modifying the ReversePermute() code

<table>
<thead>
<tr>
<th>Original execution of ReversePermute(W, $\hat{V}^{-W}$)</th>
<th>Modified version of ReversePermute(W, $\hat{V}^{-W}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$wTemp \leftarrow W$ for $i \leftarrow m+1$ downto 1 do</td>
<td>$wTemp \leftarrow W$ for $i \leftarrow 1$ to $m+1$ do</td>
</tr>
<tr>
<td>swap($wTemp_i$, $wTemp_{m+2-i}$) return $wTemp$</td>
<td>swap($wTemp_{m+2-i}$, $wTemp_{m+2-i}$) return $wTemp$</td>
</tr>
</tbody>
</table>

As shown in Table III, we modified the original code of ReversePermute(W, $\hat{V}^{-W}$) in order to resemble a regular Permute() function, i.e., the inner loop becomes to.downto.do instead of downto.do. Consequently, wTemp is scanned backward starting by $wTemp_{m+1}$, $wTemp_{m}$, $wTemp_{m-1}$, ..., $wTemp_{2}$, and $wTemp_{1}$. In order to restore the original form of scanning elements in an increasing order, we may assign $Wr$ to $wTemp$ instead of $W$, as shown next:

\[
\begin{align*}
    & wTemp \leftarrow Wr \\
    & \text{for } i \leftarrow 1 \text{ to } m+1 \text{ do} \\
    & \quad \text{swap}($wTemp_i$, $wTemp_{m+2-i}$) \\
    & \text{return } wTemp
\end{align*}
\]

Since $wTemp$ was reversed at the beginning of the above function, the returned vector will be also reversed at the end of execution, i.e., $wTemp$ will be identical to $V_r$ instead of $V$.

The above function looks like a typical Permute() function, except for the second parameter of swap(). For this matter, we will construct a new key $K_r^{W_r^{-V_r}}$ as follows:

\[
K_r^{W_r^{-V_r}} = m + 2 - K_r^{V_r^{-W_r}} \text{ for } 1 \leq i \leq m + 1
\]

(6)

And similarly,

\[
\hat{V}_i^{-W} = m + 2 - K_r^{W_r^{-V_r}} \text{ for } 1 \leq i \leq m + 1
\]

(7)

Therefore, the above algorithm will look like:

\[
\begin{align*}
    & wTemp \leftarrow Wr \\
    & \text{for } i \leftarrow 1 \text{ to } m+1 \text{ do} \\
    & \quad \text{swap}($wTemp_i$, $wTemp_{m+2-i}$) \\
    & \text{return } wTemp
\end{align*}
\]

which is identical to $\text{Permute}(Wr, K_r^{W_r^{-V_r}})$, which results in $V_r$.

Thus, we found a relationship between keys that permute $V$ into $W$, and keys that permute $Wr$ into $V_r$, referring to (6) and (7). Therefore, if we could generate keys that permute $V$ into $W$, we could easily infer the same number of keys that permute $Wr$ into $V_r$, and vice versa. Since $V_{p_1} = W_{p_2} = m+1$, then $V_{r_{m+2-p_1}} = W_{r_{m+2-p_2}} = m+1$. Then, we can set two new positions $p_1$ and $p_2$ to be $m + 2 - p_2$ and $m + 2 - p_1$, respectively. Since $p_1$ is greater than $p_2$, then $p_1$ is also greater than $p_2$. Based on the previous section, there are $2^{m-1}$ keys that permute $Wr$ into $V_r$, since $p_1 > p_2$. We will consider a key $K_r^{W_r^{-V_r}}$ from this set. Then, based on (7), we can infer a new key $\hat{V}_r^{-W}$ that permutes $V$ into $W$. Consequently, we can infer another set of $2^{m-1}$ keys in this manner.

Now, we will show that $\hat{V}_r^{-W}$ does not belong to the first set of $2^{m-1}$ keys that we generated previously. For this purpose, we will choose a key, $K_r^{W_r^{-V_r}}$, from the previously generated keys (first set).

We know that $K_r^{W_r^{-V_r}} = \hat{p}_1$, when $i < \hat{p}_1$, based on (2) and (3).

Therefore, $m + 2 - \hat{V}_r^{-W} \neq \hat{p}_1$, for $i < \hat{p}_1$.

Then, $m + 2 - \hat{V}_r^{-W} \neq m + 2 - p_2$, for $i < m+2-p_2$.

Therefore, $\hat{V}_r^{-W} \neq p_2$, for $j > p_2$.

Hence, when $j = p_1$, $\hat{V}_r^{-W} \neq p_2$.

But, for any key $K_r^{W_r^{-V_r}}$ from the first set, we have $K_r^{W_r^{-V_r}} = p_2$, based on (2).

Therefore, $\hat{V}_r^{-W}$ does not belong to the $2^{m-1}$ set of generated keys.

To conclude this subsection: given two permutation vectors $V$ and $W$ of size $m+1$, where $V_{p_3} = W_{p_2} = m+1$, and $p_1 > p_2$, we constructed $2^{m-1}$ keys that permutes $V$ into $W$, and inferred a second, different set of $2^{m-1}$ keys that also permutes $V$ into $W$, for a total of $2^m$ keys (see example in Figure 13).

Hence, we finished the proof by induction for the case of ($p_1 > p_2$).

Figure 13: Second set of two keys that permute (1, 2, 3) to (3, 2, 1)

V = (1, 2, 3)  \hspace{1cm} W = (3, 2, 1)

\[
\begin{align*}
    & \text{Inversion} \\
    & \text{Reduction} \\
    & \text{Modification based on (1)} \\
    & \text{Keys that permute Wr to Zr} \hspace{1cm} \text{Keys that permute Vr to Wr} \\
    & \{(1, 1, 2), (2, 1, 2)\} \hspace{1cm} \{(1, 1, 2), (2, 1, 2)\}
\end{align*}
\]

\[
\begin{align*}
    & \text{Step 1} \\
    & \text{Step 2} \\
    & \text{Step 3} \\
    & \{(1, 1, 2), (2, 1, 2)\} \hspace{1cm} \{(3, 3, 3), (3, 2, 2)\}
\end{align*}
\]
(b) Case of $p_1 < p_2$

In this case, the element $m+1$ moves forward from position $p_1$ in $V$ to position $p_2$ in $W$. We will try to find a specific pair of permutation vectors, $\tilde{V}$ and $\tilde{W}$, where the element $m+1$ moves backward from $\tilde{V}$ into position $\tilde{W}$ also permutes $V$ into $W$. Since there are 2\textsuperscript{m} keys that permute $\tilde{V}$ into $\tilde{W}$ (based on case (a)), we can infer that there are also 2\textsuperscript{m} keys that permute $V$ into $W$.

**Index Map Set**

An index map set, $IMAPS_{X,W}(A)$, corresponding to a set of indices, $A$, is the set of indices that are produced by the function $IMAP_{X,Y}$ over the set $A$, i.e., $IMAPS_{X,W}(A) = \{IMAP_{X,W}(i) \mid i \in A\}$.

**Lemma 2:** For any given set of indices, $A$, $|A| = |IMAPS_{X,W}(A)|$, where $X$ and $V$ are two permutation vectors.

**Proof:**

1. If $|A| < |IMAPS_{X,W}(A)|$, then $IMAP_{X,Y}$ is not a function.

2. If $|A| > |IMAPS_{X,W}(A)|$, then there exist two different indices $i, j \in A$ where $IMAP_{X,W}(i) = IMAP_{X,W}(j)$.

Therefore, if $r = IMAP_{X,W}(i) = IMAP_{X,W}(j)$, then $X_i = V_r$ and $X_j = V_r$. Thus, $X_i = X_j$, which is a contradiction, since $X$ is a permutation vector.

**Lemma 3: (Equilibrium Principle)** Given two permutation vectors, $V$ and $W$, where $V_{p_1} = W_{p_2} = m+1$, and $p_1 < p_2$, there exists an index $j \neq p_1$ such that $j > IMAP_{V,W}(j)$. In other words, since the element $m+1$ moves forward from position $p_1$ in $V$ into position $p_2$ in $W$, there should be another element, $v_1$, that moves backward from position $j$ in $V$ into position $IMAP_{V,W}(j)$ in $W$ (Figure 14).

**Proof:** By contradiction. Suppose that $j \leq IMAP_{V,W}(j)$, for any $j \neq p_1$. Based on that, if $j > p_1$, then $IMAP_{V,W}(j) > p_1$. Given that $IMAP_{V,W}(p_1) = p_2$, $p_2 \in \{p_1 + 1, \ldots, m+1\}$ since $p_1 < p_2$, therefore $IMAPS_{V,W}(p_1, \ldots, m+1) \subseteq \{p_1 + 1, \ldots, m+1\}$, which is a contradiction based on Lemma 2, since $\{p_1 + 1, \ldots, m+1\} \not\subseteq \{p_1 + 1, \ldots, m+1\}$.

So, based on Lemma 3, there exists $j_1 \neq p_1$ where $j_1 > IMAP_{V,W}(j_1) = j_2$, and $V_{j_1} = W_{j_2} = v_1$.

Then, we will construct a permutation vector, $\tilde{V}$, of size $m+1$ as follows:

\[
\tilde{V}_i = \begin{cases} 
V_i & \text{if } i \neq p_1 + 1 \\
V_{p_1 + 1} & \text{if } i = p_1 \\
V_{p_1 + 1} = m + 1 & \text{if } i = j_1
\end{cases}
\]

And another permutation vector $\tilde{W}$ of size $m+1$ as follows:

\[
\tilde{W}_i = \begin{cases} 
W_i & \text{if } i \neq p_2 + 1 \\
W_{p_2 + 1} = v_1 & \text{if } i = p_2 \\
W_{p_2 + 1} = m + 1 & \text{if } i = j_2
\end{cases}
\]

In other words, we perform swap($V_{p_2}$, $W_{j_2}$) in $V$ to get $\tilde{V}$, and swap($W_{p_2}$, $V_{j_2}$) in $W$ to get $\tilde{W}$ (Figure 15).

![Figure 15: Utilization of the equilibrium principle to generate different vectors V and W](image)

Therefore, $\tilde{V}$ and $\tilde{W}$ are also two permutation vectors of size $m+1$. Since $j_1 > j_2$, we have 2\textsuperscript{m} keys that permute $\tilde{V}$ into $\tilde{W}$ based case (a). Then, all we need is to show that these keys also permute $V$ into $W$.

**Lemma 4: (Value-independent property of the ‘Permute’ function)** Given two permutation vectors $A$ and $B$ of size $m+1$, a key $K$ of size $m+1$, a pair $(a, b) \in \{(1, \ldots, m+1)\}^2$, a position $p$ where $A_p = a$ and $B_p = b$, two permutation vectors $C = Permute(A, K)$ and $D = Permute(B, K)$; therefore, if $C_q = a$ for certain position $q$, then $D_q = b$.

**Proof:** by loop invariant. Suppose that we run $Permute(A, K)$ and $Permute(B, K)$ in parallel as shown in Table IV. We will set the loop invariant as “if $aTemp_p = a$ for some position $p$, then $bTemp_p = b$.”

**Table IV. Running Permute(A, K) and Permute(B, K) in parallel**

<table>
<thead>
<tr>
<th>Permute(A, K)</th>
<th>Permute(B, K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$aTemp \leftarrow A$</td>
<td>$bTemp \leftarrow B$</td>
</tr>
<tr>
<td>for $i \leftarrow 1$ to $m+1$ do</td>
<td>for $i \leftarrow 1$ to $m+1$ do</td>
</tr>
<tr>
<td>swap($aTemp_p$, $aTemp_{p+i}$)</td>
<td>swap($bTemp_p$, $bTemp_{p+i}$)</td>
</tr>
<tr>
<td>return $aTemp$</td>
<td>return $bTemp$</td>
</tr>
</tbody>
</table>

**Initialization:** Before executing the two loops, $aTemp = A$ and $bTemp = B$. Therefore, $aTemp_p = a$ and $bTemp_p = b$ for position $p$ defined in Lemma 4.

**Maintenance:** Suppose that before executing the $i$\textsuperscript{th} loop, there exists a position $t$ such that $aTemp_t = a$ and $bTemp_t = b$. Also, we will set $j = K$.

i. If $i = t$ and $j = t$, then $a$ and $b$ will remain in their position $t$ inside $aTemp$ and $bTemp$, respectively, after executing the $i$\textsuperscript{th} loop.

ii. If $i = t$ and $j \neq t$, then $a$ and $b$ will swap to position $j$ inside $aTemp$ and $bTemp$, respectively, after executing the $i$\textsuperscript{th} loop.

iii. If $i \neq t$ and $j = t$, then $a$ and $b$ will swap to position $i$ inside $aTemp$ and $bTemp$, respectively, after executing the $i$\textsuperscript{th} loop.

iv. If $i \neq t$ and $j \neq t$, then $a$ and $b$ will be not involved at all in the $i$\textsuperscript{th} swap operation. Therefore, $a$ and $b$ will remain in their position $i$ inside $aTemp$ and $bTemp$.  

Copyright © 2013 MECS

respectively, after executing the $i^{th}$ loop.

Therefore, $a$ and $b$ will share the same position ($t$, $i$, or $j$) in $aTemp$ and $bTemp$, respectively, after executing the $i^{th}$ loop, based on the above cases. Hence, the loop invariant is still valid at the end of the $i^{th}$ loop.

Termination: When $i = m+2$, Permute($A$, $K$) and Permute($B$, $K$) will return $aTemp$ and $bTemp$, and store them in $C$ and $D$, respectively. Therefore, if $C_{q} = a$ for some position $q$, then $D_{q} = b$.

Next, we will choose a key $K^{V \rightarrow W}$ from the $2^{n}$ keys that permute $V$ into $W$ and observe the result of Permute($V$, $K^{V \rightarrow W}$). For this purpose, we will run Permute($V$, $K^{V \rightarrow W}$) and Permute($V', K^{V' \rightarrow W'}$) in parallel, as shown in Table V.

<table>
<thead>
<tr>
<th>Phases</th>
<th>Permute($V, K^{V \rightarrow W}$)</th>
<th>Permute($V', K^{V' \rightarrow W'}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>$vTemp \leftarrow V$</td>
<td>$vTemp' \leftarrow V'$</td>
</tr>
<tr>
<td>II</td>
<td>for $i \leftarrow 1$ to $m+1$ do</td>
<td>for $i \leftarrow 1$ to $m+1$ do</td>
</tr>
<tr>
<td></td>
<td>$vTemp_{p_{j}}$, $vTemp_{p_{j}'}$)</td>
<td>$vTemp_{p_{j}}$, $vTemp_{p_{j}'}$)</td>
</tr>
<tr>
<td></td>
<td>return $vTemp$</td>
<td>return $vTemp'$</td>
</tr>
</tbody>
</table>

We know that Permute($V$, $K^{V \rightarrow W}$) results in $W$. We will use Lemma 4 to show that Permute($V$, $K^{V \rightarrow W}$) also results in $W$, i.e., at the end of Permute($V$, $K^{V \rightarrow W}$), $vTemp$ will be identical to $W$. We will consider all elements of $vTemp$ and $vTemp'$ at Phase I ($vTemp_{j}$ and $vTemp'_{j}$, for $1 \leq j \leq m+1$).

- If $vTemp_{j} = v_{1}$ at Phase I, then $vTemp_{j} = m+1$, based on (8). At Phase III, $vTemp'_{p_{j}} = W_{j_{2}} = m+1$, based on (9). Therefore, $vTemp'_{p_{j}} = m+1$, based on Lemma 4.

Hence, $vTemp_{p_{2}} = W_{p_{2}}$ (10)

- If $vTemp'_{j} = m+1$ at Phase I, then $vTemp'_{j} = v_{1}$, based on (8). At Phase III, $vTemp'_{p_{j}} = W_{j_{2}} = v_{1}$, based on (9). Therefore, $vTemp'_{p_{j}} = v_{1}$, based on Lemma 4.

Hence, $vTemp'_{p_{1}} = W_{p_{1}}$ (11)

- If $vTemp'_{j} \neq v_{1}$ or $m+1$ at Phase I, then $vTemp'_{j} = vTemp'_{j} = vTemp'_{j}$, based on (8). Thus, at Phase III, $vTemp'_{p_{j}} = vTemp'_{p_{j}}$, based on Lemma 4, when $vTemp'_{p_{j}} \neq v_{1}$ or $m+1$ (1 $\leq k \leq m+1$). Therefore, $vTemp'_{p_{j}} = vTemp'_{p_{j}}$, when $k \neq v_{1}$ or $m+1$, based on (9).

Hence, $vTemp'_{k} = W_{k}$, when $k \neq v_{1}$ or $m+1$ (12)

Based on (10), (11) and (12), $vTemp$ is identical to $W$ at Phase III.

Hence, $K^{V \rightarrow W}$ permutes $V$ into $W$.

To conclude this sub-section: given two permutation vectors $V$ and $W$ of size $m+1$, where $V_{p_{1}} = W_{p_{1}} = m+1$ and $p_{1} < p_{2}$, we constructed two new vectors $V'$ and $W'$ of size $m+1$, where $V_{j_{2}} = W_{j_{2}} = m+1$ and $j_{1} > j_{2}$. Based on (case (a)), there are $2^{n}$ keys that permute $V$ into $W$. Ultimately, we proved that these keys also permute $V$ into $W$ (see example in Figure 16).

Then, we finished the proof by induction for the case of ($p_{1} < p_{2}$).

$$V = (3, 2, 1) \quad \text{Swapping} \quad W = (1, 2, 3) \quad \text{Swapping} \quad \text{W} = (3, 2, 1)$$

Figure 16: Inferring a set of four keys that permute $(3, 2, 1)$ to $(1, 2, 3)$

(c) Case of $p_{1} = p_{2}$

Lemma 5: (Equilibrium Principle #2) Given two permutation vectors $V$ and $W$, where $V_{p_{1}} = W_{p_{1}} = m+1$, there exists an index $j \neq p_{1}$ such that $j \geq IMAP_{V,W}(j)$. In other words, since the element $m+1$ stays in position $p_{1}$ when permuting $V$ into $W$, there should be another element that either stays still or moves backward from position $j$ in $V$ into position $IMAP_{V,W}(j)$ in $W$ (Figure 17).

Proof: By contradiction. Suppose that $\forall j \neq p_{1}$, $j < IMAP_{V,W}(j)$. So, if $j > p_{1}$, then $IMAP_{V,W}(j) > p_{1}+1$. Therefore $IMAP_{V,W}(p_{1}+1, \ldots, m+1) \subseteq [p_{1}+2, \ldots, m+1]$, which is a contradiction based on Lemma 2, since $|[p_{1}+1, \ldots, m+1]| > |[p_{1}+2, \ldots, m+1]|$.

Notice that Lemma 5 is another version of Lemma 3, but it is not necessary to find an element moving backwards. Therefore, two cases are to be discussed:

1. There exists $j_{1} \neq p_{1}$ where $j_{1} > IMAP_{V,W}(j_{1}) = j_{2}$. Then, following the same steps in case (b) ($p_{1} < p_{2}$), we can infer $2^{n}$ keys that permute $V$ into $W$ (see example in Figure 18).
Figure 18: Inferring a set of four keys that permute (1, 3, 2) to (2, 3, 1)

2. For any index \( j, j = \text{IMAP}_v(j) \). This means that \( V \) and \( W \) are identical. Following the same steps of constructing keys in case (a), we will end up with \( 2^{m-1} \) keys that permute \( V \) into \( W \) (or \( V \)). We will choose a key \( K^{V \rightarrow W} \) for comparison purposes.

We need to find another set of \( 2^{m-1} \) keys that permute \( V \) into \( W \). Notice that we cannot use the same methodology of case (a), because the argument presented to prove that the new set of keys is different than the one constructed at case (a) is not valid in case \( p_1 < p_2 \).

Instead, we will try to come up with a second pair of identical permutation vectors \( \tilde{V} \) and \( \tilde{W} \), and we will show that the keys that permute \( \tilde{V} \) into \( \tilde{W} \) also permute \( V \) into \( W \).

2.1 if \( p_1 \neq 1 \):

We will construct a permutation vector \( \tilde{V} \) (also identical to \( \tilde{W} \)) of size \( m+1 \) as follows:

\[
\tilde{V}_i = \begin{cases} 
V_i, & \text{if } i \neq p_1, p_1 - 1 \\
V_{p_1}, & \text{if } i = p_1 - 1 \\
V_{p_1-1}, & \text{if } i = p_1 
\end{cases}
\]

In other words, we perform \( \text{swap}(V_{p_1}, V_{p_1-1}) \) in \( V \) to get \( \tilde{V} \) (also \( \text{swap}(W_{p_1}, W_{p_1-1}) \) in \( W \) to get \( \tilde{W} \)). Therefore, \( \tilde{V} \) and \( \tilde{W} \) are also two identical permutation vectors of size \( m+1 \). Following the same steps for constructing the keys in case (a), we obtain \( 2^{m-1} \) keys that permute \( \tilde{V} \) into \( \tilde{W} \). Since \( \text{Reduction}(V) = \text{Reduction}(W) = \text{Reduction}(\tilde{V}) = \text{Reduction}(\tilde{W}) = X = Y \), then the \( 2^{m-1} \) set of keys that permute \( V \) into \( W \), and the \( 2^{m-1} \) set of keys that permute \( \tilde{V} \) into \( \tilde{W} \) are constructed based on the same \( 2^{m-1} \) set of keys that permute \( X \) into \( Z \) (which is the modification of \( Y \) based on (1)). For comparison purposes, we will choose a key, \( K^{\tilde{V} \rightarrow \tilde{W}} \), from the \( 2^{m-1} \) set of keys that permute \( \tilde{V} \) into \( \tilde{W} \), such that \( K^{\tilde{V} \rightarrow \tilde{W}} \) and \( K^{V \rightarrow W} \) are constructed using the same key, \( K^{\tilde{V} \rightarrow \tilde{W}} \), that permutes \( X \) into \( Z \) (based on (2), (3), and (4)). Following the same methodology in case (b), we can prove that \( K^{\tilde{V} \rightarrow \tilde{W}} \) also permutes \( V \) into \( W \).

Unfortunately, \( K^{\tilde{V} \rightarrow \tilde{W}} \) and \( K^{V \rightarrow W} \) are not always different, which prevents the union of \( 2^{m-1} \) sets of keys that permute \( V \) into \( W \), and \( 2^{m-1} \) sets of keys that permute \( \tilde{V} \) into \( \tilde{W} \), to achieve \( 2^m \) different keys.

i. If \( K^{\tilde{V} \rightarrow \tilde{W}} \neq p_1 - 1 \), then \( K^{\tilde{V} \rightarrow \tilde{W}} \) and \( K^{V \rightarrow W} \) are different, since \( \tilde{V}_{p_1-1}^{p_1-1} = p_1 - 1 \).

ii. If \( K^{\tilde{V} \rightarrow \tilde{W}} = p_1 - 1 \), then \( K^{\tilde{V} \rightarrow \tilde{W}} \) and \( K^{V \rightarrow W} \) may be identical, since \( K^{\tilde{V} \rightarrow \tilde{W}} = p_1 - 1 \). In this case, \( \text{Permute}(V, K^{\tilde{V} \rightarrow \tilde{W}}) \) will be idle at step \( p_1 - 1 \);

\[
K^{\tilde{V} \rightarrow \tilde{W}} = \begin{cases} 
K^{V \rightarrow W}, & \text{if } i \neq p_1, p_1 - 1 \\
p_1, & \text{if } i = p_1 - 1 \\
p_1 - 1, & \text{if } i = p_1 
\end{cases}
\]

Now, \( K^{\tilde{V} \rightarrow \tilde{W}} \) has a different corresponding key \( K^{V \rightarrow W} \) that also permutes \( V \) into \( W \). Hence, we infer a second, different set of \( 2^{m-1} \) keys that permute \( V \) into \( W \), when \( V \) and \( W \) are identical (see example in Figure 19).

2.2 if \( p_1 = 1 \):

Following the same methodology of Section 2.1, we can proceed in the proof by setting \( \tilde{V} \) as follows (using \( p_1 + 1 = 2 \) instead of \( p_1 - 1 \)):

\[
\tilde{V}_i = \begin{cases} 
V_i, & \text{if } i \neq 1, \text{or } 2 \\
V_1 = m + 1, & \text{if } i = 2 \\
V_2, & \text{if } i = 1 
\end{cases}
\]

Hence, we finish the proof by induction for the case of \( (p_1 = p_2) \).

Figure 19: Constructing a set of four keys that permute (1, 2, 3) to (1, 2, 3)
that also permutes $V$ into $W$, based on case (b).

VI. SECURITY MODEL: SYNCHRONOUS DYNAMIC ENCRYPTION SYSTEM (SDES)

SDES is a stream cipher cryptosystem based on permutation vector generation [16][17][18]. The encryption function is simplified in order to minimize the overhead cost. Thus, a simple XOR is performed between the data record $d^j$ and one of the generated permutation vectors $PV^j$, resulting in a cipher $c^j$ to be transmitted (Figure 20 (a)). The decryption function is performed in the same manner as the encryption function. The cipher record $c^j$ is XORed with the same permutation vector $PV^j$ (generated at the recipient side), producing the original data record $d^j$ (Figure 20 (b)). Then, a new permutation vector is generated ($PV^{j+1} = \text{Permute}(PV^j, SK)$) to be used in the next encryption/decryption operations. Notice that the first permutation vector to be used in encryption $PV^0$ is the result of $\text{Permute}(PV^1, SK)$, where $PV^0$ is the initial permutation vector $(1, 2, \ldots, n)$, and $n$ is the key size. Figure 20: (a) Encryption at the sender side (b) Decryption at the recipient side.

A. Static shared key

This option provides a low security profile; compromising two consecutive $PV$s will break the entire system, since the corresponding $\text{IMAP}$ between $PV^j$ and $PV^{j+1}$ is identical to the $\text{IMAP}$ of $PV^{j+1}$ and $PV^{j+2}$. Therefore, this option is used only with the assumption that a cryptanalysis is unfeasible (e.g., transmitted data are proportionally limited).

B. Stream of shared keys

In order to alleviate the previous security loophole, a second option is also provided to modify the shared key after each data record encryption, for more $\text{IMAP}$ dynamics. Practically, we perform $SK^{j+1} \leftarrow SK + PV^j$ before generating the next permutation vector $PV^{j+1}$. Then, the shared key generation is not vulnerable to “biased byte” analysis since the involved permutation vector is a good source of byte diversity. However, in the event that more than one encryption session is opened (in parallel or at different times), the same stream of shared keys is generated, lacking security independence between sessions; i.e., breaking one session breaks all.

C. Session-based stream of shared keys

For ultimate security, the communicated data is involved in the shared key generation, as a third option. Basically, we perform $SK^{j+1} \leftarrow SK + PV^j + d^j$ before generating the next permutation vector, $PV^{j+1}$. Hence, a different sequence of shared keys is generated in every session (assuming that sessions are of different communication data).

VII. EXPERIMENTAL RESULTS

A prototype simulation of our technique proved to run two to three times faster than the state-of-the-art AES (Advanced Encryption Standard), DES (Data Encryption Standard), and Triple DES (Figure 21). Using the NS2 simulator (version 2.26), we designed a topology of five nodes connected to a router that routes packets to a sink node through a duplex connection of 1 Mbits/s maximum capacity. Each of the four nodes tries to send an exponential generated traffic data to the sink passing through the bottleneck 1 Mbits/s connection. The first node sends non-encrypted packets. The second, the third, the fourth and the fifth nodes send packets securely, encrypted with AES, DES, 3DES, and SDES, respectively.

Figure 21 shows that SDES (using the dynamic stream of shared keys option) achieves a maximum throughput of 896 Kbits/s. This result proves the higher efficiency of our SDES algorithm compared to other peer techniques (AES: 409 Kbps, DES: 528 Kbps, and Triple DES: 112 Kbps). The reason for achieving such better performance is the simplicity of our encryption/decryption function, since the function complexity is shifted to the dynamics of the key management (i.e., the permutation vectors generation).

VIII. CONCLUSION

In this paper, a simple mechanism to generate permutation vectors (based on a random secret key) is introduced for data encryption. Unique to our technique, we proved that $2^{m-1}$ different secret keys have the same effect on the generation of the next encryption key. Hence, even if an intruder (hypothetically) compromises two consecutive encryption keys, he is cornered to brute-force a massively huge key space (for $m = 256$). Moreover, the involvement of the entire permutation vector in the encryption process results in a much more diverse stream of keys than those of RC4, avoiding state-related attacks.

We also presented a cryptosystem implementation that utilizes permutation vectors in the process of encryption as well as in key management. Experimental results (using the NS2 simulator) showed that our security system outperformed peer security mechanisms, e.g., AES and Triple DES, due to the simplicity of both encryption and key management functions. Future work is related to the diversity of $\text{IMAPS}$ between consecutive encryption keys. On average, there is a chance of $1/n!$ for two subsequent secret keys to yield the same $\text{IMAP}$. Hence, it would be useful to investigate mechanisms to enhance secret key management in order to assert such diversity.
Figure 21: NS2 experimental results showing network throughput using secure transmission modes: (a) SDES (b) AES (c) DES (d) 3DES

REFERENCES


Dr. Mohammed Omari received his B.E degree from the University of Es-Senia Oran (Algeria) in 1995 and the M.S. degree from New Mexico Tech (New Mexico, USA) in 2002, as well as the Ph.D. degree in 2005. His major is in computer science. He is currently an associate professor at the computer science department, and a unit chair at the LDDI laboratory, University of Adrar, Algeria. His research interests include network security, cryptography, sensors and ad hoc networks protocols, image processing, neural networks, and genetic algorithms.

Prof. Hamdy S. Soliman Received his B.S. in 1978, from Alexandria University (Egypt), and his M.S. in 1983 from Florida Institute of Technology, and his Ph.D. 1989 from New Mexico State University (USA), all degrees are in Computer Science. He is currently a full Professor in the Computer Science & Engineering Department at New Mexico Tech (NMT). His research interests include Wireless Mobile sensor networks routing and security protocols, neural networks application in data/image processing, all-fiber optics networks admission control, computer and networks security, wireless security and protocols.