# A Hybrid Hiding Encryption Algorithm (HHEA) For Data Communication Security

Mahmoud Shaar[1], Magdy Saeb[2], Usama Badawi[1]
1. Cairo University, Faculty of Science
Mathematics Dept., Computer Science Division,
Cairo, Egypt
2. Arab Academy for Science, Technology & Maritime Transport
School of Engineering, Computer Department
Alexandria, Egypt

**Abstract**
In this work, we present an encryption algorithm that can be used to for hardware-implemented applications to secure data communications. This encryption algorithm is based on hiding a number of bits from plain text message into a random vector of bits. The locations of the hidden bits are determined by a key known to the sender and receiver. We call this algorithm hybrid hiding encryption algorithm (HHEA). The name demonstrates the two basic operations of this algorithm. These are operations include inserting part of the plaintext bits into a cover to hide it from recognition. That is there are no conventional operations on the ciphered text, just plain hiding in a random bit string. The name "Hybrid" is used to show that the algorithm has built-in features that are inherited from data hiding techniques or "Steganography." The distinctive features of this algorithm are as follows:

- Key length is variable: the key length can be varied from 16 up to any larger value depending on the security level required.
- Word length is variable: the block size can be varied between 1 to 16 bit or 1 to 32 and so on. That is, encryption can be performed on 16 or 32 or 64 bit blocks. This, in turn, can be used on different processor architectures employing 16, 32, or 64 bit registers.
- The algorithm, therefore, provides variable degrees of security. However, this increased security level will be at the cost of increased size of the cipher-text.
- The number of rounds is variable: the whole process can be repeated r times using the same key.

The method is quite suitable for hardware implementation employing Field programmable gate arrays (FPGA).
**Key words:** encryption, data security, steganography, data hiding, data communication, FPGA.

## 1. Introduction

In this work, we present an encryption algorithm that can be designed for hardware-implemented applications to secure data communications. The name demonstrates the two basic operations of this algorithm. These operations are based on inserting part of the plaintext bits into a cover to hide it from recognition. That is there are no conventional operations on the ciphered text, just plain hiding in a random bit string. The name "Hybrid" is used to show that the algorithm has built-in features that are inherited from data hiding techniques or "Steganography." We call this algorithm hybrid hiding encryption algorithm (HHEA).
In the following few sections, we provide a run through existing algorithms with some comparative evaluation. We then turn to describe our algorithm.

The first algorithm that we discuss is RC4 [2, 4]. This algorithm is a variable-key-size stream cipher. The general performance problem with RC4 is that almost every statement depends immediately on the statement before it, including the table index computation and the associated table accesses, limiting the amount of parallelism. On the other hand, SEAL [2, 4] is a software-efficient stream cipher. The major problem with this algorithm is the strong dependency between consecutive operations, allowing only minimal parallelism.

In block-encryption algorithms, DES [1, 5] recognized world-wide, it set a precedent as the first commercial-grade modern algorithm with openly and fully specified implementation details. The problem with DES is that the size of the key space is too small to be really secure. Khufu algorithm [2, 4] can be simply implemented in hardware at one clock per round. IDEA [2, 4] is used for message encryption in Pretty Good Privacy (PGP). In fact, the speed of hardware depends dramatically on how much cost could be absorbed due to the need for dedicated multipliers which are not cheep. Skipjack [2] is NAS-developed algorithm for the Clipper and Capstone chips and it is considered secure. REDCO II [2, 3] is secure because using the brute force attack, $2^{160}$ operations are required to recover the key. A summary of block encryption algorithms is shown in Table 1.

*Table 1*
*Summary of Block-encryption algorithms*

| | Key Length | Block Length | Problem |
|---|---|---|---|
| DES | 56 bits | 64 bits | key too small |
| Khufu | 64 bits | 64 bits | key too small |
| REDCO II | 160 bits | 80 bits | Secure |
| IDEA | 128 bits | 64 bits | Patented |
| Skipjack | 80 bits | 64 bits | Secret |

## 2. Design methodology of the proposed algorithm

On designing this algorithm, we have considered that the crypto analyst knows all details of the algorithm. This conforms to "Kerckhoffs' Principle" in cryptography, which holds that "the security of a cryptographic system should rely only on the key material".

The basic idea of our proposed encryption algorithm is hiding a number of bits from plain text message into a random victor of bits. The location of the hiding bits are determined by a pre agreed-upon key by the sender and the receiver. The following subsection gives more details about our algorithm.

### 2.1 The Encryption process

The method is reasonably simple. We have a key matrix $K_{Lx2}$ where,

$$k_{ij} \in \{1,2,3,4,5,6,7,8\} \begin{cases} \forall i = 1,...,L \text{ ; L} \geq 16 \\ \forall j = 1,2 \end{cases}$$

This key is known only to the sender and receiver. When the first party wants to send a message M to the second party, he/she determines the key $K_{L\times 2}$ and every character from the message is replaced by a binary value. An eight-bit octet is generated randomly and set in a temporary vector V. the bits in the vector V from position K [1,1] to position K[1,2] are replaced by bits from the secret message. Then the resulting vector V is stored in a file. As long as the message file has not reached its end yet, we move to the next row of the key matrix and another octet is generated randomly and the replacement is performed repeatedly and the resulting vector is stored in the file. The previous procedure is repeated over and over again pending the end the message. The resulting file is sent to the receiver who beforehand has the key matrix. If the key length is not enough to cover the whole message during the encryption process, the key will be reapplied over and over again

until the encryption of the whole message is completed. This formal algorithm is shown next.

**Algorithm HHEA**
[Given a plain text message M and key matrix $K_{L\times 2}$ where

$$k_{ij} \in \{1,2,3,4,5,6,7,8\} \begin{cases} \forall i = 1,...,L \text{ ; L} \geq 16 \\ \forall j = 1,2 \end{cases}$$

The aim of the algorithm is hiding a number of bits from plain text message (M) into a random vector (V) of bits. The locations of the hidden bits are determined by the key $K_{L\times 2}$ ]

**Input:**
M[plain text message], $K_{L\times 2}$ [Key array]

**Algorithm Body:**
First: in a plain text file, each character is sequentially replaced by its binary value.

```
i:=0
m := first digit in M file
while (m ≠ EOF) [EOF: End Of File]
i:=i mod L
Generate 8-bits randomly and set them in V
Vector
if (K[i,1]≤K[i,2]) then
                    for j=K[i,1] to K[i,2]
                      if (m ≠ EOF)
                        then do
                          V[j] = m
                          m := next m in M file
                        end do
                    next j
else
                    for j=K[i,1] downto K[i,2]
                      if (m ≠ EOF)
                        then do
                          V[j] = m
                          m := next m in M file
                        end do
                    next j
Save V in output file
i:= i+1
end while
Output: encrypted file
End algorithm.
```

## 2.2 The Decryption process

For decrypting the received encrypted file the following steps are taken. An octet is read from the encrypted binary plain text message EBPM file, then it is set in a temporary vector V, from this vector, bits are extracted from position K(1,1) to position K(1,2) and set in a BPM file. Since the EBPM file is nonetheless not empty, the next octet is read from the EBPM file and then it is set in a temporary vector V. From this vector, bits are extracted from position K (2, 1) to position K (2, 2) and added to the binary plain text message BPM file. The above steps are repeated over and over again until the EBPM file becomes empty. Every octet form the BPM file is transformed to the corresponding character, and then is put in the plaintext file. When the EPBM is empty the plaintext file becomes the message.
In case that the key length is not enough to cover the whole message during the decryption process, the key will be reapplied over and over again till the decryption of the whole message is completed.

## 2.3 Key Length

Now we will show the number of possible keys, i.e., the key space when the key length is 16. The probability of replacing a string of bits whose length ranges from 1 to 8 bit in an octet is 1/64. Consequently, if the key length is 16 there are $64^{16} = 7.9\text{x}10^{28}$ possible keys.
So we can say that if the attacker has a cipher text and he knows that the key length is 16, there are $7.9\text{x}10^{28}$ attempts to find the correct key, i.e. , there are $7.9\text{x}10^{28}$ attempts to find the correct plaintext or secret message.
Assuming that a supercomputer working in parallel is able to try $10^{12}$ attempts per second, it will take $2.5\text{x}10^{9}$ years to find the secret message. Note that the universe is only $10^{10}$ years old. This eliminates brute force attack; however other types of attacks will be discussed in future work.

## 3. Algorithms analysis

The worst case, regarding storage requirements, occurs when replacing one bit only from message to the V vector. Hence, cipher text equal eight times the size of the plain text. We have analyzed worst case running times for our encryption algorithm and found that it has linear complexity of O(n). Moreover, we have studied the following:

- Key length is variable: the key length can be varied from 16 up to any larger value depending on the security level required.
- Word length is variable: the block size can be varied between 1 to 16 bits or 1 to 32 bits and so on. That is, encryption can be performed on 16, 32 or 64 bit blocks. This, in turn, can be used on different processor architectures employing 16, 32, or 64 bit registers.
- The algorithm, therefore, provides variable degrees of security. However, this improved security levels will be at the cost of increased size of the cipher text.
- The number of rounds is variable: the whole process can be repeated r times using the same key.

.

## 4. Implementation

For the implementation of the above algorithm, windows C++ program is developed, as it is shown in the figure below:

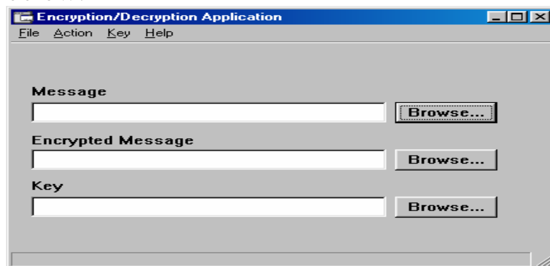


*Figure 1: A view of the C++ window program implementation of the HHEA algorithm.*

The user is asked to supply the name of the plaintext file, "message", and the name of the key file (key matrix). The user selects from the main menu "action\encrypt" in order to encrypt the plaintext.

## 5. Summary & Conclusions

Comparing our method with one time pad we observe the following

1. An obvious drawback of the one-time pad is that the key should be as long as the plaintext, which increases the difficulty of key distribution and key management. Regarding our method, the key length is shorter than the plain text in general.
2. In the one time pad the key can never be used again. While in our method the key can be used to encrypt more than one message.
3. The one-time pad is suitable for a few short messages. Our method is suitable for all messages, regardless its length

For higher degrees of data security, we perform the following:

1. The process can be repeated on the resulting ciphered file. That is, using more than one round of encryption cycles. The number of rounds should be agreed upon by sender and receiver before transmission begins.
2. Encryption can be performed on 16, 32 or 64 bit blocks.

## References


[1] Menezes, A.. *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1997.

[2] Schneier, B. *Applied Cryptography*. New York, Wiley, 1996.

[3] T.W. Cusick and M.C. Wood, “The REDOC-II Cryptosystem,” *Advances in Cryptology—CRYPTO ‘90 Proceedings,* Springer-Verlag, 1991, pp. 545-563.

[4] Bruce Schneier and Doug Whiting. "Fast software encryption: Designing encryption algorithms for optimal software speed on the Intel Pentium processor," In Biham [3], pages 242-259, 1997.

[5] Stinson, D. *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 1995