HARPOCRATES:An Approach Towards Efficient Encryption of Data-at-rest

Md Rasid Ali [⋆], Debranjan Pal [⋆⋆], Abhijit Das[⋆⋆⋆], and Dipanwita Roychowdhury[†]

Indian Institute of Technology Kharagpur

Abstract. This paper proposes a new block cipher called HARPOCRATES, which is different from traditional SPN, Feistel, or ARX designs. The new design structure that we use is called the substitution convolution network. The novelty of the approach lies in that the substitution function does not use fixed S-boxes. Instead, it uses a key-driven lookup table storing a permutation of all 8-bit values. If the lookup table is sufficiently randomly shuffled, the round sub-operations achieve good confusion and diffusion to the cipher. While designing the cipher, the security, cost, and performances are balanced, keeping the requirements of encryption of data-at-rest in mind. The round sub-operations are massively parallelizable and designed such that a single active bit may make the entire state (an 8×16 binary matrix) active in one round. We analyze the security of the cipher against linear, differential, and impossible differential cryptanalysis. The cipher's resistance against many other attacks like algebraic attacks, structural attacks, and weak keys are also shown. We implemented the cipher in software and hardware; found that the software implementation of the cipher results in better throughput than many well-known ciphers. Although HARPOCRATES is appropriate for the encryption of data-at-rest, it is also well-suited in data-in-transit environments.

Keywords: Block cipher · substitution convolution network · statistical attacks

1 Introduction

The design principles of modern block ciphers depend on how we achieve confusion and diffusion properties. Diffusion is the distribution of plaintext over ciphertext such that each plaintext bit effectively acts on many ciphertext bits. It is usually a linear operation and can be achieved through permutation or linear transformation. Confusion [34] obscures the relationship between the key and the ciphertext and makes each ciphertext bit dependent on many key bits.

[⋆] rasid@iitkgp.ac.in
[⋆⋆] debranjanpal@iitkgp.ac.in
[⋆⋆⋆] abhij@cse.iitkgp.ac.in
[†] drc@cse.iitkgp.ac.in

The security of a one-time pad relies only on confusion. Another confusion-only cipher is a simple substitution cipher . Due to the absence of a proper diffusion mechanism, all the plaintext's fundamental properties remain in the ciphertext.

In the literature, there exist several design principles to achieve confusion and diffusion. Rijndael [19], ITUbee [20], 3D [28] use an 8-bit invertible S-box, which is the composition of two operations. First, the inverse function in $GF(2^8)$ is used where the zero is mapped to itself. Then, one performs an affine transformation of over $GF(2)$. ARIA [22] uses two different 8-bit S-boxes. Both S-boxes consist of the inverse function $y = x^{-1}$ but different affine transformations. Similarly, in Camellia [1], four 8-bit S-boxes are used. Crypton [24] and Whirlpool [5] also use 8-bit S-boxes, made of three 4-bit S-boxes. Though the initial S-box design choice of CLEFIA [35] and ANUBIS [4] is a random S-box, finally, two types of 8-bit S-boxes are selected. The first one is made of two 4-bit S-boxes, and the second one is an inverse function in $GF(2^8)$. For DES and TWIS [32], the S-box input and output lengths are different. For optimizing hardware resources, PRESENT [8], LED [16], TWINE [36], Piccolo, MIBS [18], and RECTANGLE [40] use single 4-bit S-boxes, but their securities are compromised to some extent. For balancing hardware resources and security criteria, Midori [2], LBlock [39], mCrypton [25], and Serpent [6] use multiple 4-bit S-boxes. A theoretical work by Nyberg [29] shows that the AES S-box gives the highest possible non-linearity. This is the reason why the AES S-box has gained wide acceptance in the design of many ciphers. The authors of Twofish [33], GOST [15], REDOC II [10] and PRINTcipher [21] use key-dependent random S-boxes, mainly to withstand several statistical attacks.

Similarly, several approaches are used for achieving the diffusion in the ciphers mentioned above. RECTANGLE and TWINE do fixed amounts of cyclic rotations of each row. Twofish and Piccolo uses simple byte permutation, whereas MIBS, mCrypton and LBlock apply nibble permutations in each round to achieve diffusion. In PRESENT and PRINTcipher, fixed bit permutations in each round are used for improved hardware efficiency. The designers of CLEFIA choose the Diffusion Switching Mechanism (DMS) for acquiring diffusion. In DMS, the diffusion matrix switches among multiple matrices in the round function in a predefined order. In TWIS, 3D, LED, ITUbee, Camellia, and AES, single MDS matrices are used.

In this paper, we propose a new block cipher HARPOCRATES [1] which achieves confusion and diffusion using a new approach. The design strategy is called Substitution Convolution Network (SCN). Our security analysis shows that the proposed cipher is immune against known cryptanalytic techniques, including linear attacks, differential attacks, impossible differential attacks, slide attacks, and structural attacks.

The rest of the paper is organized as follows. Section 2 presents the specification of HARPOCRATES. More specifically, Subsections 2.3, 2.4, and 2.5 respectively elaborate the encryption algorithm, the decryption algorithm, and *lut* generation. Section 3 motivates the design choices of HARPOCRATES. In

[1] Harpocrates was the Greek god of silence, secrecy, and confidentiality.

Section 4, we discuss the security of the cipher against known attacks. In Section 5, we discuss about the software and hardware implementations. Finally, Section 6 concludes the paper.

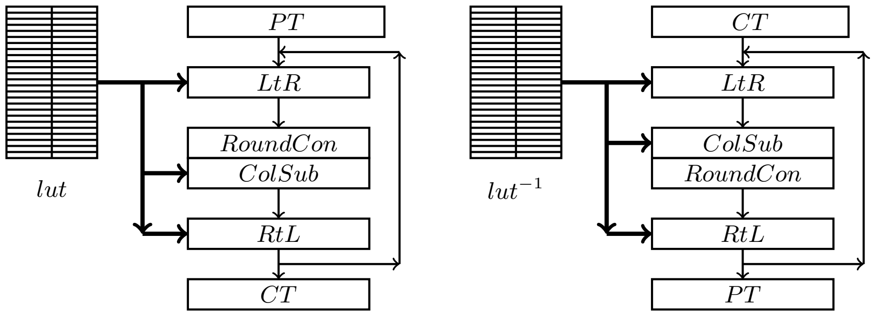
2 Specification of HARPOCRATES

The block length of the HARPOCRATES cipher is 128 bits arranged in an 8×16 binary matrix (Figure 3) in row-major order. Here, the key is only used as an entropy source [3] to a random bit generator for shuffling the secret lookup table (*lut*). In data-at-rest environments where the data do not move, we can use a non-deterministic random bit generator to shuffle the table. In data-in-transit environments, the parties can agree with the secret seed (key) so that the recipient can generate lut^{-1} and perform the decryption. The sender may also shuffle the table in a non-deterministic way and transfer the whole *lut* to the decryption operation recipient. The specification of HARPOCRATES consists of an encryption algorithm, a decryption algorithm, and an algorithm for the generation of the *lut*.

2.1 Notations

Throughout the paper, we use following notations.

state	128-bits arranged in an 8×16 binary matrix
lut	Secret table containing a permutation of $\{0, 1, 2, \ldots, 2^l - 1\}$
lut^{-1}	Involution function of *lut*, so that $lut^{-1}[lut[x]] = x$
RC_i	An array contains n_{rounds} round constants
stride	It is the sliding after substitution or the convolution operation
N	The block length of the *state* in bits
n	The number of bits in a row of the *state* matrix
n_{round}	The number of rounds in the encryption/decryption process
n_{row}	The number of rows
l	The length of the *lut* in bits
s	Stride or the number of bits we skip after each substitution
$S[i{:}j]$	Select bits from i to $(j-1)$ of the bitstring S
$\|$	Concatenation of two bit strings
$\oplus$	Bitwise exclusive-OR operation
LeftToRight	Left-to-right convoluted substitution operation (*LtR*)
RightToLeft	Right-to-left convoluted substitution operation (*RtL*)
ColSub	Column-substitution operation
DRBG	Deterministic Random Bit Generator
NRBG	Non-deterministic Random Bit Generator
PT	Plaintext
CT	Ciphertext



(a) Encryption algorithm overview (b) Decryption algorithm overview

Fig. 1: Overview of the cipher

2.2 Substitution Convolution Network

In SCN structure a randomly shuffled lookup table (*lut*) is generated and all substitutions are performed using the *lut*. Two primary operations in SCN are the convoluted substitution and column substitution operations. The convoluted substitution operation provides the diffusion in each row of the binary matrix. Column substitution operation, on the other hand, substitutes the column and mixes all bits of a column.

To understand the scenario better, let us assume that a N-bit block is arranged in a binary matrix of size $l \times n$. In Figure 2 the convoluted substitution for a row of the binary matrix is shown. First of all, a l-bit region at the left-hand side of the row is selected and replaced with the value from the *lut*. Then the convolution happens; the l-bit substitution window shifts s-bits towards a specific direction (as left to right convoluted operation is shown here, the window shifts to the right-hand side). After that, a combination of a previous substituted random value (l-s bit) and a new value (s-bit) that comes under the substitution window is selected for the next substitution.

The overlapping substitution and convolution operations continue until the substitution window reaches the end of the row. Figure 2 shows the convoluted substitution operation where the substitution window slides from the left side to the right-hand side of a row (*Left-to-Right*). In SCN, there is another operation called *Right-to-Left* in which the substitution operation starts in the right-hand side of the row, and the substitution window shifts until the window reaches the left-hand side of the row. Both *Left-to-Right* and *Right-to-Left* convoluted substitution operation diffuses each row of the binary state matrix row-wise. In SCN architecture, the bits in the column are the same as the number of bits in the permutation. When the *Left-to-Right* convoluted substitution operation finishes for all rows, the column substitution operation starts. In this operation the bits of a l-bit column gets substituted by the random *lut* values. This operation diffuses the bits in a column-wise fashion.

Diffusion in SCN Diffusion describes the influence of input bits on output bits. Ideally, every output bit should be dependent on all input bits. In the HARPOCRATES cipher, the lut is a pseudo-random permutation. An l-bit region is selected and substituted with the random mapped value from the lut. After every substitution, the active substitution region slides s-bit ($s \leq \frac{l}{3}$) bit positions, and a mixture of $(l-s)$-bit intermediate substituted value and s-bit new value is selected for the next substitution. All subsequent substitutions recursively depend on previously performed substitutions. This mechanism provides the desired diffusion.

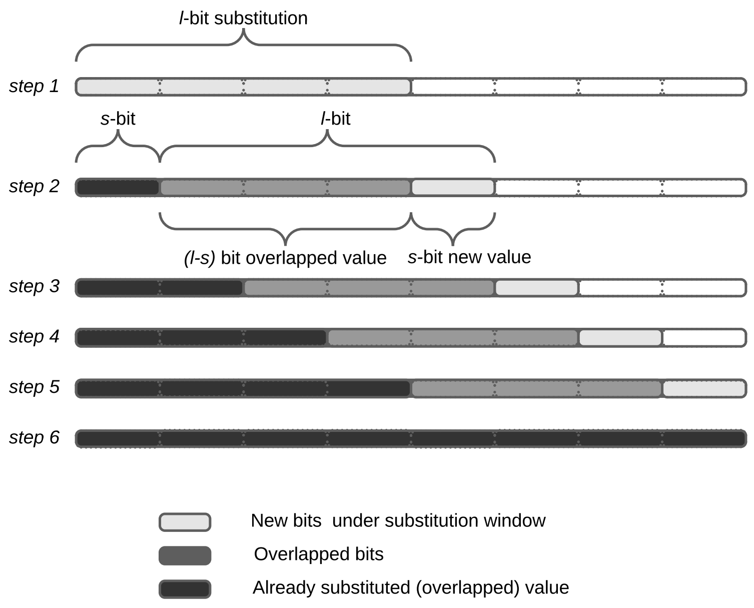


Fig. 2: Convoluted substitution operation (Left-to-Right)

2.3 Encryption algorithm

The encryption process of HARPOCRATES consists of eight iterative rounds. In every round, the Left-to-Right convoluted substitution (LtR) is applied first. Then, a round-dependent constant value is added. This is followed by column substitution ($ColSub$), which substitutes the eight bits from every column. Finally, the Right-to-Left convoluted substitution (RtL) is applied. In Algorithm 1, the encryption process is illustrated. In Section 2.5, we discuss how the lut is generated using the key.

Each encryption round consists of three main operations,

- Substitution: This means random substitution from the lut.
- Round-constant addition: A round-dependent constant value is added to break the round similarity.

$$\text{state}=\begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & \dots & b_{0,15} \\ b_{1,0} & b_{1,1} & b_{1,3} & \dots & b_{1,15} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ b_{7,0} & b_{7,1} & b_{7,2} & \dots & b_{7,15} \end{bmatrix}$$

Fig. 3: 128-bit state in 8×16 matrix

Algorithm 1 Encryption Algorithm

```
1: procedure EncryptHarpocrates($plaintext, lut, N, n, l, s, n_{round}, RC$)
2: // $N$ is block length, $n$ is row length, $n_{row}$ is number of rows
3: // $l$ is $lut$ length, and $s$ is $stride$; all in bits
4: // $state$ is a placeholder that gets 128-bit $plaintext$ in a $8 \times 16$ binary matrix
5:     $state \leftarrow plaintext$
6:     $n_{row} \leftarrow \frac{N}{n}$
7:     for $i \in (1, .., n_{round})$ do
8:         $state \leftarrow LeftToRight(state, lut, l, n_{row}, n, s)$
9:         $state \leftarrow RoundConstant(state, RC, i, n_{row}, n)$
10:        $state \leftarrow ColSub(state, lut, n)$
11:        $state \leftarrow RightToLeft(state, lut, l, n_{row}, n, s)$
12:    end for
13:    $ciphertext \leftarrow state$
14:    return $ciphertext$
```

– Diffusion: The two convoluted substitution operations and $ColSub$ together ensure the diffusion.

Figure 4 depicts the $LeftToRight$ convoluted substitution operation. Figure 5 shows the round-constant array for the first round. Figure 6 shows the $ColSub$ operation. The $RightToLeft$ convoluted substitution operation is shown in Figure 7. The algorithms for these operations are elaborated below,

***LeftToRight* convoluted substitution** This operation as detailed in Algorithm 2 consists of $\left(\frac{n-l}{s}+1\right)$ substitutions. In every step, l bits are substituted by taking the random value from the lut. After that, the substitution window moves s bits to the right. A mixture of $(l-s)$ bits from the previous substituted value and the s-bit new value (a total of l bits) are used for the next substitution. This operation continues until the end of the row is reached.

Round constant addition To break the rounds' self-similarity, a constant value is added to the $state$ matrix in each round. This operation makes the cipher resistant against invariant attacks. Fig 5 shows the round-constant for first round. Each row of the first round constant circularly left shifted $(2 \times i)$-th position to generate i-th round constant.

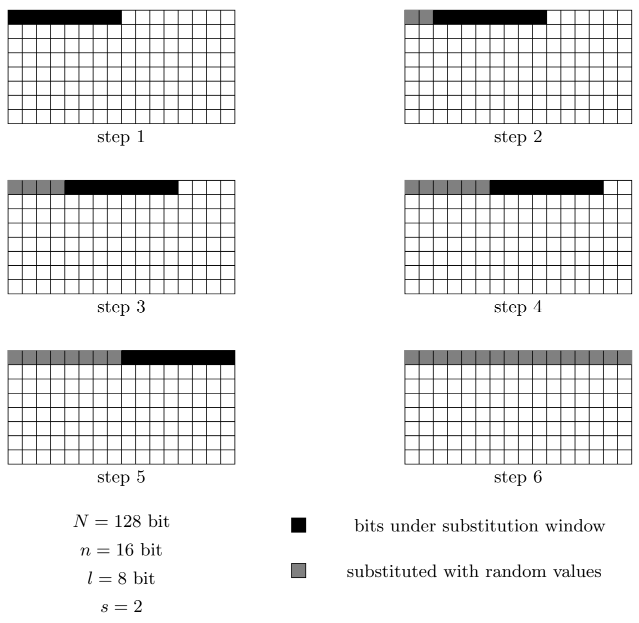


Fig. 4: LeftToRight convoluted substitution (shown for row 1)

Algorithm 2 *LeftToRight* Convoluted Substitution

```
1: procedure LeftToRight($state$,$lut$,$l$,$n_{row}$,$n$,$s$)
2:     $steps \leftarrow \left(\frac{n-l}{s}\right) + 1$
3:     for $i \in (1, \ldots, n_{row})$ do
4:         $temp \leftarrow state[i]$
5:         for $j \in (0, 1, \ldots, steps - 1)$ do
6:             $temp \leftarrow temp[0 : j \times s] \mid\mid lut[temp[j \times s : j \times s + l]] \mid\mid temp[j \times s + l : n]$
7:         $state[i] \leftarrow temp$
8:     end for
9:     return $state$
```

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0
\end{bmatrix}$$

Fig. 5: Round constant for the first round

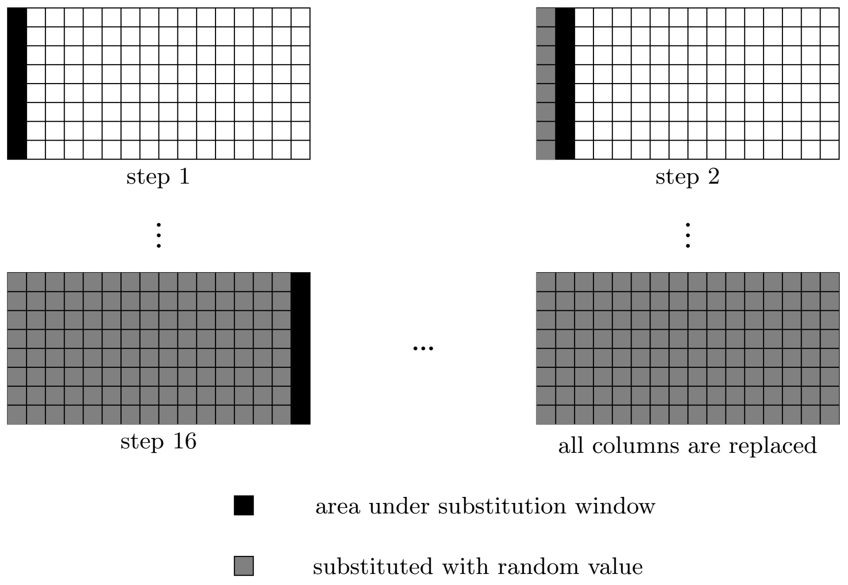


Fig. 6: *ColSub* operation

***ColSub* column substitution** *ColSub* (see Algorithm 3) substitutes every column of the state matrix using the random *lut*. This operation diffuses the value of every row after convoluted substitution. After 16 *ColSub* operations, all columns are replaced by random values, as shown in Figure 6.

Algorithm 3 *ColSub* Column Substitution

```
procedure ColSub($state$,$lut$,$n$)
// $col_{state}[i]$ selects the $i$ th column of $state$ matrix
// $temp$ placeholder holds a column
    for $i \in (1, \ldots, n)$ do
        $temp \leftarrow col_{state}[i]$
        $temp \leftarrow lut[temp]$
        $col_{state}[i] \leftarrow temp$
    end for
    return $state$
```

(4) *RightToLeft* convoluted substitution This operation has a total of $\left(\frac{n-l}{s}+1\right)$ steps. Starting from the right end of each row, the substitution window occupies l-bits and keeps on shifting left by s bits until it reaches the beginning of the row (Figure 7). In each position of the window, we use a total of l bits

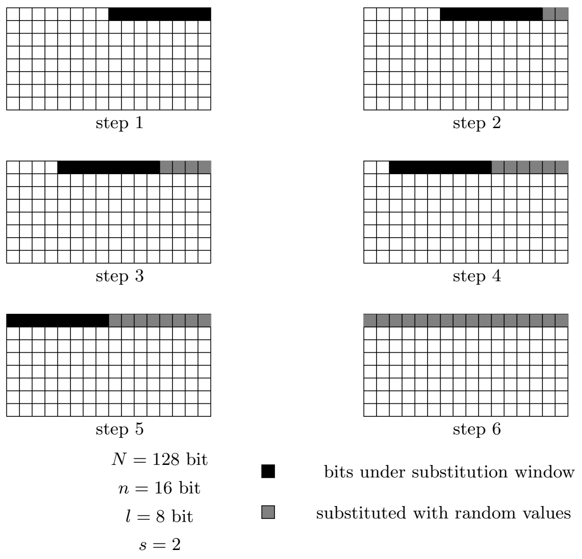


Fig. 7: RightToLeft convoluted substitution (shown for row 1)

consisting of the s-bit substituted value of the $LeftToRight$ operation and $(l-s)$ bits of the previous substituted value.

2.4 Decryption algorithm

Algorithm 4 explains the decryption algorithm of HARPOCRATES. The process (see Figure 1(b)) is almost the same as the encryption procedure with the following exceptions.

– During decryption, the $LeftToRight$ and $RightToLeft$ uses lut^{-1}.
– The round-constant addition is performed with the $(n_{round} - i)$-th element of RC array in the i-th decryption round.
– $ColSub$ operates before $RoundConstant$.

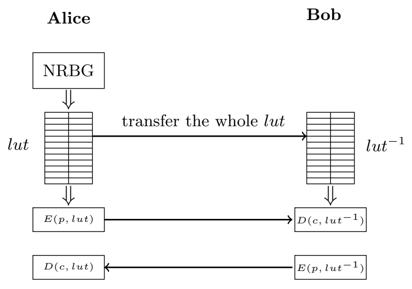
2.5 Generation of *lut*

The generation of lut is a one-time pre-processing task. The lut is kept secret and acts as the key for the cipher. Initially, we take an array containing the elements $0, 1, \ldots, 2^l - 1$, where l is the length of each lut element in bits. We perform a random permutation on the array elements. Since the lut contains 2^l elements, the number of possible permutations is $2^l!$. To perform the random permutation on the array's initial content, we need a uniform shuffling algorithm and a random bit generator.

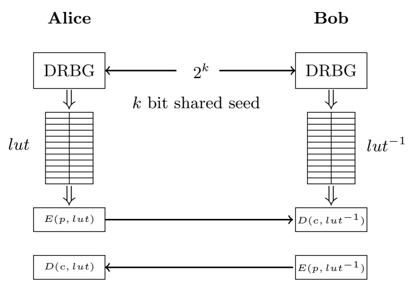
Algorithm 4 Decryption Algorithm

```
1: procedure DecryptHarpocrates($ciphertext$,$lut^{-1}$,$N$,$n$,$l$,$s$,$n_{round}$,$RC$)
2:     $state \leftarrow ciphertext$
3:     for $i \in (1, .., n_{round})$ do
4:         $state \leftarrow LeftToRight(state, lut^{-1}, l, n_{row}, n, s)$
5:         $state \leftarrow ColSub(state, lut^{-1})$
6:         $state \leftarrow RoundConstant(state, RC, n_{round} - i, n_{row}, n)$
7:         $state \leftarrow RightToLeft(state, lut^{-1}, l, n_{row}, n, s)$
8:     end for
9:     $plaintext \leftarrow state$
10:    return $plaintext$
```

Random bit generation In the data-at-rest mode, where the data stays at one place, we can use a non-deterministic($NRBG$) or true random bit generator [3,17] to shuffle the *lut*. In the data-in-transit environment, where the data moves between two parties, we need to agree with the same *lut* for performing the substitution and reverse substitution operations at the two ends. In this case, either one party non-deterministically shuffles (Figure 8(a)) the *lut* and sends the whole table to the other party or both the parties can agree with a shared secret seed value to generate the random bits deterministically [3,14] (see Figure 8(b)). Using NRBG, the shuffled *lut* may achieve $\log(2^l!)$ bits of entropy. In DRBG, using k-bit secret seed value, a maximum of 2^k different *lut*s can be obtained.



(a) Using NRBG



(b) Using DRBG

Fig. 8: *lut* Agreement

Shuffling the *lut* We use the Fisher-Yates [13] shuffling mechanism; presented as Algorithm 5. It works in linear time and provides a uniform shuffling that may result in $2^l!$ possible rearrangements for an l-bit *lut* when operated with a sufficient source of randomness.

Algorithm 5 Shuffling algorithm

```
1: procedure FisherYatesShufflingAlgorithm($lut$,$l$)
2: //$lut$ is an array of length $2^l$, containing $[0, 1, .., 2^l - 1]$
3: //$rand(p, q)$ returns a random number between $p$ to $q$
4:     $n \leftarrow 2^l$
5:     for $i \in (0, 1, \ldots, n-2)$ do
6:         $j = rand(i, n-1)$
7:         swap($lut[i], lut[j]$)
8:     end for
9:     return $lut$
```

3 Design Rationale

In this section, we briefly discuss the design principles and the selection of parameters for HARPOCRATES.

- **Substitution convolution structure** Most of the traditional block ciphers have either SPN, Feistel, or ARX structures. These ciphers generally use some non-linear known bijective function. For our cipher, the design structure is a substitution convolution network. We use a randomly shuffled *lut* with no statistical relationship between the input and output values. For achieving diffusion, we perform the substitution in a convoluted manner. When a *LeftToRight* or *RightToLeft* operation works in a row, every substitution is dependent on all previous substitutions. Moreover, the *ColSub* operation diffuses bits in the columns. *LeftToRight* and *RightToLeft* operations can be performed in every row parallelly. Similarly, the *ColSub* operation can be performed independently in every column in a parallel fashion. In Appendix A, Figure 10 shows that after one single round, even a single bit change may diffuse the entire 8×16 *state*.
- ***lut* size** For both deterministic and non-deterministic *lut* generation, we suggest the value of l as 8. Though larger *luts* may provide better non-linearity, it also increases the *lut* size and the one-time permutation time rapidly. Which increases the overall code size. If an *NRBG* is used, the *lut* stores one of the $2^8!$ permutations uniformly at random. If a *DRBG* is used, the entropy of the *lut* set is bounded by the entropy of the secret seed.
- **Stride size** During the convolution operation, we stride, that is, slide the convolution window by s bits. Larger strides make encryption faster but less convoluted, making bits loosely coupled. We should have $s \leq \frac{l}{3}$ and $(n - l) \bmod s = 0$ must satisfy so that the convoluted substitution ends evenly at the end of the row. For $l = 8$ possible stride values are $s = 1$ and $s = 2$. For HARPOCRATES the *stride* is fixed to 2. In the Appendix B, we provide test vector with $l = 8$ and $s = 2$.
- **Key length** This cipher is best suited for the encryption of data-at-rest, that is, data archived in storage systems. In that case, we can use an *NRBG* which is considered a truly random bit generator [17,3,14] having access to

an entropy source. For data-in-transit, we have two possibilities. First, a party may shuffle the table in a non-deterministic way and send the entire table securely to another party. Second, both parties agree upon a secret seed and generate the same *lut* separately. In this case, we recommend the length of the key or the seed to be 256 bits.

4 Security analysis

We now analyze the security of HARPOCRATES against known attacks.

4.1 Linear cryptanalysis

Linear cryptanalysis [27,23,11] is one of the most significant attacks on block ciphers. This is an instance of known plaintext attacks. That is, we check whether there exists any probabilistic linear relationship between a portion of plaintext bits and a subset of bits after substitutions are performed in the last round. To determine the complexity of linear cryptanalysis of a cipher, the largest value of the Linear Approximation Table (LAT) is combined with the number of active bundles after certain rounds. From the results of Appendix A, we get ≈ 43 expected active bundle for two round HARPOCRATES. For the expected maximum value of LAT, we use the results from [30]. With reference from [30] let us assume $\pi : Z_2^n \rightarrow Z_2^n$ is a random permutation and $\lambda(\pi)$ is the linearity of π. $E(\lambda(\pi, 2k))$ denote the expected number of entries in LAT of size $2k$. $E(\lambda(\pi, 2k))$ in the Equation 1 tends to zero as a function of k we are likely to obtain an upper bound on $\lambda(\pi)$.

$$E(\lambda(\pi, 2k)) = \frac{2 \times (2^l - 1)^2 \times (2^{l-1}!)^2}{2^l!} \times \binom{2^{l-1}}{2^{l-2} + k}^2 \tag{1}$$

Using Equation 1 the value of k is obtained as 19 for an 8-bit random permutation. So, using the results of Appendix A, two-round expected linear characteristics of HARPOCRATES is $\approx 2^{-118}$. Linear cryptanalysis ultimately depends on the non-linearity of the round function. For a word-oriented cipher structure like AES, the round non-linearity reduces to the non-linearity of the S-box. Though for a random permutation, the expected non-linearity is lesser than the maximum possible value, for HARPOCRATES, the overlapped sub-operations cause a bundle or parts of the bundle to get substituted multiple times. Thus, we believe the non-linearity of the HARPOCRATES round function shall be much higher than the expected non-linearity of a random *luts*.

4.2 Differential cryptanalysis

Differential cryptanalysis [23,11] is a chosen-plaintext attack. The attacker selects plaintext messages and examines the corresponding ciphertext messages in an attempt to derive the key. Differential cryptanalysis tries to correlate plaintext differences with ciphertext differences with high probability and works by exploiting the properties of the S-boxes used by the cipher.

HARPOCRATES, on the other hand, uses l-bit random permutation. Appendix A shows that sub-operations $LeftToRight$, $RightToLeft$ and $ColSub$ together generate ≈ 43 active bundles in one round transformation from one active bit. A theoretical study of O'connor [31] shows that for a uniformly random l-bit bijective mapping, the highest probability of differential characteristics in a non-trivial case is expected to be at most $\frac{2l}{2^l}$. So, the two-round HARPOCRATES is expected to have differential characteristics 2^{-172} ($\approx (2^{-4})^{43}$) when $l = 8$ and $s = 2$. So, we expect that the differential characteristics are not a potential threat to HARPOCRATES.

4.3 Boomerang attack

This is an extension of differential cryptanalysis [37]. In this attack, the cipher is divided into two sub-ciphers such that $E(m) = E_1(E_0(m))$. We need two differential characteristics for the two sub-ciphers. One is $\Delta \rightarrow \Delta^*$ for forward differential E_0, and the another one is $\nabla \rightarrow \nabla^*$ backward differential for E_1^{-1}. Plaintext-ciphertext pair (P, C) is chosen, and Δ, and ∇ are applied on P and C, respectively. Finally, it is checked whether the initial forward differential of E_0 holds. The boomerang quartet probability is p^2q^2, where $p = \sqrt{\sum_{\Delta^*} Pr^2[\Delta \rightarrow \Delta^*]}$, and $q = \sqrt{\sum_{\nabla^*} Pr^2[\nabla \rightarrow \nabla^*]}$ are the maximum differential probabilities of first and second sub-cipher, respectively. The values of p and q are bounded by the number of active bundles in each sub-ciphers. Drawing results from Appendix A, we found HARPOCRATES cipher achieves a high number of expected active bundles and resists differential cryptanalysis very well. As the boomerang quartet probability is p^2q^2, we expect boomerang attack is much more difficult than differential cryptanalysis.

4.4 Weak *keys* (*luts*)

The set of keys that make an encryption scheme weaker than other keys is called a weak key set. In HARPOCRATES, choosing a 256-bit key is analogous to choosing one permutation uniformly at random from 2^{256} potential permutations. A set of *luts* with more number of similar mappings may form a weak *lut* set. As, 8-bit lut can form $2^8!(\approx 2^{1684})$ permutations, when DRBG is used all potential *luts* are expected to skip $2^{1428}(\approx 2^8!/2^{256})$ permutations between two consecutive potential *luts*. Table 1 lists the expected number of the same mapping between two *luts* between generated *luts*.

We believe that the attacker can gain an advantage if, throughout the encryption process using multiple *luts*, the substitution occurs only within the same mappings of the *luts*. In HARPOCRATES, one round comprises 96 random substitutions. Let us assume that attacker gets two black box encryption engines that possess *luts*, which has 32 same entries (this is a bizarre assumption). The

Available *luts*	Expected #same mapping
2^8	1
2^{16}	2
2^{80}	10
2^{256}	32

Table 1: Number of expected same mappings

probability that both encryption engines will use identical mappings is $(2^{-3})^{192}$. This is sufficient to tell that a weak *lut* set where *luts* has 32 same entry will not pose a threat to HARPOCRATES.

4.5 Slide attack

Ciphers that use identical substitution-permutation functions in every round are vulnerable to the slide attack [7]. Unlike differential and linear cryptanalysis, slide attack does not depend on the number of rounds. Two main criteria for performing slide attack are:

- The encryption process is splittable into multiple rounds of identical functions.
- One-round encryption function is vulnerable to a known-plaintext attack.

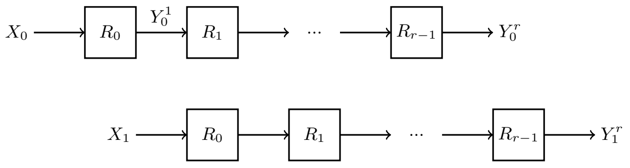


Fig. 9: Slide attack

Let X_0, X_1 be two plaintext messages, and Y_0^i, Y_1^i the ciphertext after i rounds of encryption. The attack succeeds if the one-round decryption of Y_1^r gives Y_0^r (see Figure 9).

HARPOCRATES does XORing of round-dependent constant values in every encryption round with the state, thereby breaking the similarity among the different rounds. Since identical rounds are needed for the slide attack, our cipher should be immune to this attack.

4.6 Impossible differential attack

Impossible differential cryptanalysis exploits the slow diffusion of block ciphers. This attack makes use of intermediate differential characteristics that never occur. A particular plaintext differential propagates through rounds produces some predefined patterns in intermediate states in the forward direction. Then, guessing one candidate key, the ciphertext is decrypted up to the desired rounds and checked whether the intermediate state pattern follows or not. Thus, eliminating the impossible keys. In the HARPOCRATES cipher, the diffusion mechanism is strong; predicting such state differential patterns in intermediate rounds is not straightforward, as that depends on secret lut. Thus, we conclude that this attack technique is not elementary to the HARPOCRATES cipher.

4.7 Algebraic attack

Algebraic attacks such as Higher-order differential attack [9], Cube attack [12] are mainly applicable for stream ciphers and block ciphers that have a low degree in round transformation. For performing algebraic attacks over-defined system of algebraic equations is derived. The degree of such over-defined equations for certain rounds is estimated as the degree of non-linear operation to the power of the number of times the non-linear operation is performed. Expected value of algebraic degree [38] of a random permutation of l-bit is $(l-1)$. In a single round transformation of HARPOCRATES cipher, a single bundle or fraction of a bundle is substituted multiple times due to the overlapped round operations. We believe that the expected degree of a single round transformation will be much higher than the degree of lut, and algebraic attacks shall be inapplicable for our cipher.

4.8 Structural attack

Structural attacks such as integral attacks [2] and saturation attacks [26] are generally applicable to block ciphers where the state has a word-like structure (like a collection of bytes as in AES [19]). Although HARPOCRATES uses substitution of l-bit words, the substitution window's shift is by an amount smaller $(s \leq \frac{l}{3})$ than the word length. As a result, the word-level alignment of substitution is destroyed during both $LeftToRight$ and $RightToLeft$ operations. Consequently, structural attacks are unlikely to apply to our cipher.

5 Performance evaluation

We implemented the cipher in software and hardware. The inherent structure of HARPOCRATES allows a high degree of parallelism. *LeftToRight*, *RightToLeft* operations can be performed independently in every row. Similarly, *ColSub* can be operated in all the columns parallelly. Although, for software implementation, we do not exploit the parallelization feature in performance comparisons.

5.1 Software Implementation

We perform software implementation of the cipher and compare the performances with the benchmarked implementations of many lightweight ciphers us-

CPU	`Intel(R) Core(TM) i3-10110U`
CPU Clock	`2.10GHz`
Architecture	`x86`
Instruction Mode	64-bit
Hardware acceleration used ?	No
Cache	4 MB
Operating System	Ubuntu `20.04` LTS
Compiler	`GCC (version 9.3.0)`
Compilation Optimization	`-O2`

Table 2: Platform Details for Software Implementation

ing FELICS[2] (Fair Evaluation of Lightweight Cryptographic Systems) framework. The environment details of our software implementation and comparison are shown in Table 2. The comparison results with other ciphers are shown in Table 3. FELICS framework only lists benchmark implementations of lightweight ciphers, no ciphers with key size greater than 128-bit is listed there. HARPOCRATES cipher only consists of substitution and XOR operations, which take much lesser CPU cycles than arithmetic and finite field operations. We found the proposed cipher has higher throughput than some 128-bit versions of cipher, like, AES, SKINNY, PRINCE. It also provides higher throughput than PRESENT, TWINE, and LED, which are some 80-bit ciphers. However, the code size of the cipher is larger than most of the ciphers, as shown in Table 3. But, the larger code size is justifiable; the *lut* is randomly generated, and its input-output relation shall not be represented by mathematical equations and need to be stored explicitly in the code.

5.2 Hardware Implementation

We formalized hardware implementation of three different approaches for the proposed cipher. The first approach focuses on throughput, whereas the resource consumption is optimized in approach two, and approach three tries to reduce the number of Input-Output Blocks (IOBs). In approach one round sub-operations are performed in partial-parallel fashion and significant higher throughput is achieved. In approach two, no parallelization is adopted and all the sub-operations are performed in a single row or column at a time, resulting lesser utilization of hardware resources. Approach three uses divider circuits in feeding the input and the output is made available in eight chunks of 16-bit each, reducing the number of used IOBs. Table 4 shows the environment for hardware implementation and Table 5 lists the resource consumption for all three approaches.

[2] FELICS framework is used by cipher designers to compare new primitives with state of the art

Algorithm	Function	Block Size (bit)	Key Size (bit)	Code Size (byte)	CPU Cycle
AES	Enc	128	128	833	158906
	Dec	128	128	953	241812
SPECK	Enc	64	128	119	2953
	Dec	64	128	122	3518
SKINNY	Enc	128	128	564	63924
	Dec	128	128	588	62476
PRESENT	Enc	64	80	487	498522
	Dec	64	80	547	497936
TWINE	Enc	64	80	629	48654
	Dec	64	80	607	52782
Piccolo	Enc	64	80	412	31992
	Dec	64	80	351	35256
LED	Enc	64	80	1478	3799376
	Dec	64	80	1472	3925274
PRINCE	Enc	64	128	370	40428
	Dec	64	128	386	41986
LEA	Enc	128	128	3399	2528
	Dec	128	128	3551	2930
HARPOCRATES	Enc	128	256	2754	40290
	Dec	128	256	2754	41050

Table 3: Performance Comparison of Software Implementation

Language	Verilog HDL
Synthesis Tool	Xilinx ISE
Version	ISE 14.7
Family	Artix 7
Device	XC7A100T
Package	CSG324

Table 4: Platform Details for Hardware Implementation

Approach Number	# of LUT Slices	# IOBs Used	Frequency (MHz)	Throughput (Megabits)
1	1447	258	136	79.01
2	1256	258	136	39.52
3	1267	36	136	31.40

Table 5: Post Synthesis Result Details

6 Conclusion

This paper proposes a new block, cipher HARPOCRATES. Its design principle is Substitution Convolution Network, which is entirely different from the conventional SPN, Feistel, and ARX models. In this cipher, the key is used to shuffle a *lut* randomly. The *lut* substitutions are performed in a convoluted manner, which in association with $ColSub$, introduces good confusion and diffusion. This cipher may flourish full diffusion in a single round. The round sub-operations are highly parallelizable, and due to the use of low CPU cycle consuming operations, the software implementation also has higher throughput than many popularly known ciphers. Our analysis shows that the cipher is immune to statistical attacks, boomerang attacks, algebraic attacks, structural attacks, and slide attacks.

References

1. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-bit block cipher suitable for multiple platforms—design andanalysis. In: International Workshop on Selected Areas in Cryptography. pp. 39–56. Springer (2000)
2. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 411–436. Springer (2015)
3. Barker, E., Feldman, L., Witte, G.: Recommendation for random number generation using deterministic random bit generators. Tech. rep., National Institute of Standards and Technology (2015)
4. Barreto, P.S.: The anubis block cipher. NESSIE (2000)
5. Barreto, P., Rijmen, V., et al.: The whirlpool hashing function. In: First open NESSIE Workshop, Leuven, Belgium. vol. 13, p. 14 (2000)
6. Biham, E., Anderson, R., Knudsen, L.: Serpent: A new block cipher proposal. In: International workshop on fast software encryption. pp. 222–238. Springer (1998)
7. Biryukov, A., Wagner, D.: Slide attacks. In: International Workshop on Fast Software Encryption. pp. 245–259. Springer (1999)
8. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C.: Present: An ultra-lightweight block cipher. In: International workshop on cryptographic hardware and embedded systems. pp. 450–466. Springer (2007)
9. Courtois, N.T., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 267–287. Springer (2002)
10. Cusick, T.W., Wood, M.C.: The redoc ii cryptosystem. In: Conference on the Theory and Application of Cryptography. pp. 546–563. Springer (1990)
11. Daemen, J., Rijmen, V.: The wide trail design strategy. In: IMA International Conference on Cryptography and Coding. pp. 222–238. Springer (2001)
12. Dinur, I., Shamir, A.: Side channel cube attacks on block ciphers. IACR Cryptol. ePrint Arch. **2009**, 127 (2009)

13. Fisher, R.A., Yates, F.: Statistical tables for biological, agricultural and medical research. Hafner Publishing Company (1953)
14. Goldreich, O.: Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali. Morgan & Claypool (2019)
15. GOST, G.S.: 28147-89,“. Cryptographic protection for data processing systems,” Government Committee of the USSR for Standards (1989)
16. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The led block cipher. In: International workshop on cryptographic hardware and embedded systems. pp. 326–341. Springer (2011)
17. Gutmann, P.: Software generation of practically strong random numbers. In: Usenix Security Symposium (1998)
18. Izadi, M., Sadeghiyan, B., Sadeghian, S.S., Khanooki, H.A.: Mibs: a new lightweight block cipher. In: International Conference on Cryptology and Network Security. pp. 334–348. Springer (2009)
19. Joan, D., Vincent, R.: The design of rijndael: Aes-the advanced encryption standard. In: Information Security and Cryptography. springer (2002)
20. Karakoç, F., Demirci, H., Harmancı, A.E.: Itubee: a software oriented lightweight block cipher. In: International Workshop on Lightweight Cryptography for Security and Privacy. pp. 16–27. Springer (2013)
21. Knudsen, L., Leander, G., Poschmann, A., Robshaw, M.J.: Printcipher: a block cipher for ic-printing. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 16–32. Springer (2010)
22. Kwon, D., Kim, J., Park, S., Sung, S.H., Sohn, Y., Song, J.H., Yeom, Y., Yoon, E.J., Lee, S., Lee, J., et al.: New block cipher: Aria. In: International Conference on Information Security and Cryptology. pp. 432–445. Springer (2003)
23. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Annual International Cryptology Conference. pp. 17–25. Springer (1994)
24. Lim, C.H.: Crypton: A new 128-bit block cipher. NIsT AEs Proposal (1998)
25. Lim, C.H., Korkishko, T.: mcrypton–a lightweight block cipher for security of low-cost rfid tags and sensors. In: International Workshop on Information Security Applications. pp. 243–258. Springer (2005)
26. Lucks, S.: The saturation attack—a bait for twofish. In: International Workshop on Fast Software Encryption. pp. 1–15. Springer (2001)
27. Matsui, M.: Linear cryptanalysis method for des cipher. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 386–397. Springer (1993)
28. Nakahara, J.: 3d: A three-dimensional block cipher. In: International Conference on Cryptology and Network Security. pp. 252–267. Springer (2008)
29. Nyberg, K.: Differentially uniform mappings for cryptography. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 55–64. Springer (1993)
30. O’Connor, L.: Properties of linear approximation tables. In: International Workshop on Fast Software Encryption. pp. 131–136. Springer (1994)
31. O’connor, L.: On the distribution of characteristics in bijective mappings. Journal of Cryptology **8**(2), 67–86 (1995)
32. Ojha, S.K., Kumar, N., Jain, K., et al.: Twis–a lightweight block cipher. In: International Conference on Information Systems Security. pp. 280–291. Springer (2009)
33. Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., Ferguson, N.: Two sh: A 128-bit block cipher. AES submission (1998)
34. Shannon, C.E.: Communication theory of secrecy systems. The Bell system technical journal **28**(4), 656–715 (1949)

35. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit blockcipher clefia. In: International workshop on fast software encryption. pp. 181–195. Springer (2007)
36. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: Twine: A lightweight, versatile block cipher. In: ECRYPT Workshop on Lightweight Cryptography. vol. 2011 (2011)
37. Wagner, D.: The boomerang attack. In: International Workshop on Fast Software Encryption. pp. 156–170. Springer (1999)
38. Watanabe, D., Hatano, Y., Yamada, T., Kaneko, T.: Higher order differential attack on step-reduced variants of luffa v1. In: International Workshop on Fast Software Encryption. pp. 270–285. Springer (2010)
39. Wu, W., Zhang, L.: Lblock: a lightweight block cipher. In: International Conference on Applied Cryptography and Network Security. pp. 327–344. Springer (2011)
40. Zhang, W., Bao, Z., Lin, D., Rijmen, V., Yang, B., Verbauwhede, I.: Rectangle: a bit-slice lightweight block cipher suitable for multiple platforms. Science China Information Sciences **58**(12), 1–15 (2015)

Appendices

A Analysis of Differential and Linear Trails

The propagation of difference along the rounds of a cipher is called the differential characteristics. Figure 10 shows the differential characteristics of HARPOCRATES. Even with a single active bit, the whole block may become active after one round transformation. We call a l-bit differential that has a non-zero value an active bundle for the sake of discussion. After every substitution in $LeftToRight$ and $RightToLeft$ convoluted operation the substitution window moves s-bit towards a particular direction. $(l-s)$-bits of random value is overlapped between two consecutive substitutions. The skipped s-bits do not have any influence on the succeeding substitutions. So, the successive bundle is active with probability $p = (1 - \frac{(2^s-1)}{2^l})$. Similarly, for calculating successive active bundles, previous bundles being active is also considered. Using this phenomenon, the expected number of active bundles and the probability of each bit of an active bundle being set is calculated (Figure 11). If a bundle is active with probability p, then a particular bit of that bundle is active with probability $p \times h$, where $h = \frac{1}{2}$.

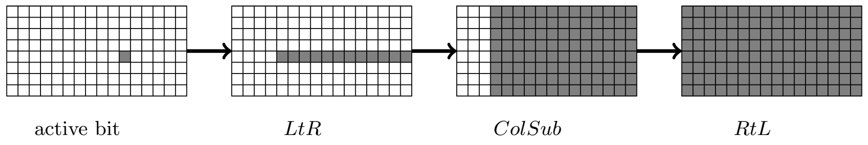


Fig. 10: Differential characteristics of HARPOCRATES

If a bit is active in a row after $LeftToRight$ convoluted substitution, that bit participates in the corresponding column's diffusion. And, that column shall be active after the $ColSub$ operation. So, all those columns which are having a set bit after the $LeftToRight$ operation become active, as shown in Figure 12. Similarly, while calculating the expected number of active bundles after the $RightToLeft$ convoluted substitution operation, it is observed whether the current bundle is active or not. After each lut substitution, again check whether the new region under the substitution

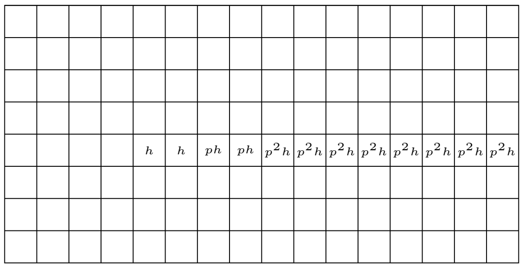


Fig. 11: Probability of being differentially active after first round $LeftToRight$

				h^2	h^2	ph^2	ph^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2
				h^2	h^2	ph^2	ph^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2
				h^2	h^2	ph^2	ph^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2
				h^2	h^2	ph^2	ph^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2
				h	h	ph	ph	p^2h	p^2h	p^2h	p^2h	p^2h	p^2h	p^2h	p^2h
				h^2	h^2	ph^2	ph^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2
				h^2	h^2	ph^2	ph^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2
				h^2	h^2	ph^2	ph^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2	p^2h^2

Fig. 12: Probability of being differentially active after first round *ColSub*

window is active or not. In HARPOCRATES a single round transformation consist of $(n + 2 \times \frac{N}{n} \times (\frac{n-l}{s} + 1))$ number of substitution from the random *lut*. As the round sub-operations perform substitutions in an overlapped manner, one bundle or part of a bundle becomes substituted multiple times. Thus the expected number of active bundles in HARPOCRATES is much higher than other ciphers after a particular round, which helps the cipher resist the differential and linear cryptanalysis even for a lesser number of rounds.

Example: expected number of active bundles with initial active bit is at position $b_{0,15}$ (Left most active bits shall generate lesser number of expected active bundles) $l = 8, s = 2$, $p = 253/256$ and $h = \frac{1}{2}$

Expected bundle weight after $LeftToRight$: 1
Expected bundle weight after $ColSub$: $h + h + h + h + h + h + h + h + 1 = 5$
Expected bundle weight after $RightToLeft$: $(\underbrace{7 \times (4.819)}_{\text{row}=1,\ldots,7} + \underbrace{4.92828}_{\text{row}=0} + 5) \approx 43$

B Test vectors

Values of $l = 8$ and $s = 2$. The *lut* is randomly generated. Plaintext, ciphertext, and *lut* are presented in the hexadecimal format.

lut = [da, 7a, e7, 93, d7, ae, d3, fa, 20, 60, 70, 62, c9, 9d, 5e, 6a, 4a, e1, 8d, b4, 74, ce, 55, ac, ea, c3, 3a, d0, 8b, 3d, 49, 7f, 82, f3, f6, 90, 6b, 3c, f9, ba, df, b1, 11, fe, 14, 73, 06, 2a, e3, 96, 6c, 0e, 13, 65, 2e, d1, 01, c2, cd, 47, 5a, d5, 4b, 10, d2, d8, 69, 2b, 1d, f5, 99, e5, b5, 03, 9a, f0, 37, cb, 7d, 23, 53, 81, 59, 16, 2d, 94, b2, a7, 40, 86, 24, eb, 95, 1f, 83, 3b, f8, f7, 0d, 28, fd, ca, 51, db, 97, 56, 43, 5d, 5f, 9b, 71, 63, 78, f1, 52, ff, 18, c5, 64, 32, 6f, 8c, 9e, dc, 30, ec, d9, a9, 42, a6, 0a, cc, 9f, 4c, a3, 08, ee, 57, 36, b0, ef, 0f, 25, 8e, e2, 76, 6e, fc, 35, 79, 5b, e0, 26, 98, f2, 88, f4, 67, aa, 33, 6d, d4, 41, a0, c0, 3f, 72, e4, 15, 07, c6, 7b, 44, ed, de, 91, 2f, 48, 04, 61, a8, 39, ad, b6, c8, c1, af, 7c, b9, a5, 27, 29, 0c, 58, 34, ab, 21, e8, 9c, 19, a2, 8a, 0b, 80, 68, b7, e9, a4, 89, 4f, 12, dd, c7, 1e, e6, 75, 66, 3e, 8f, cf, d6, 54, 5c, b8, 92, bf, 22, 85, 17, 05, bb, 4d, c4, 50, 02, b3, 77, 87, 1c, bc, 1a, 45, 2c, 09, 84, 00, 1b, be, 4e, 31, 7e, a1, 38, fb, 46, bd]

plaintext	ciphertext
00000000000000000000000000000000	5f5bc52acc8665cd08700f35b8ab66dc
10000000000000000000000000000000	99ebb44375eab43dc65c7ae5526f0a54
00000000000000000000000000000001	0d1998890f92f0b97e5199d6c5f10764
10000000000000000000000000000001	e5ac90467b7d76dd9a1d625225e15ff1