

Abstract:

TEA is a small algorithm which is being used in Smart Cards, embedded systems etc. It is an efficient and fast algorithm which performs very efficiently on modern systems. It is a feistel cipher with 64 bit block size and 128 bit key. It takes 32 cycles (64 rounds) of processing to produce a Ciphertext. TEA is easy to implement and cryptographically strong. Whenever we implement and use correctly, TEA can be proved to be excellent choice, particularly when we encrypt and decrypt small data in resource constrained devices

Smaller devices like PDA, smart cards etc. require fast and efficient ciphers for encryption. To encrypt information securely, 128 bit key needs more computing and cost than 64 bits. It is always not true that "Larger Key size, provides more security". In this age of Information Technology, every information is communicated through internet and the security to these information systems are inevitable. Different systems needs security for different time period.

Funds are transferred electronically all over the world through net. Security is required for brief period for this transaction

Confidential information of an individual like aadhar card may need security for its lifetime.

For such scenarios, the security is important and this type of information has to be encrypted in a faster and efficient way. it is discussed that why information security is of so much importance. It may be expensive to protect information but we can do it by using some economic models to secure the information

The need for security in small devices and short lived transactions is increasing day by day. People want quick transactions on their small gadgets. They are using lighter devices and want more security in their transactions. So the need of lighter and faster security algorithms is also increasing

So, here we reduce the key size of 128 bits to 64 bits, thereby reducing the cost and time to encrypt the data and checking robustness using bitsum algorithm

It is noted that to complete 5 million encryption at a total of 237.29 seconds a single TEA encryption with this implementation takes about 47.46ms.

Literature Survey:

Research paper	Technique	Aim	Pros	Cons	Inference
Secure image transferring	Tiny Encryption Algorithm	Scrambling the given image using permutation and then encrypting using TEA algorithm	Highly secured using password It uses character map	It is difficult for a single algorithm to satisfy all performance parameters.	This system will be useful in transferring sensitive images like medical images
On the Security of reduced key Tiny Encryption Algorithm	Tiny Encryption Algorithm	Implementing TEA by reduced key size of 64 bits instead of 128 bit	Reducing the no. of keys without decreasing its performance	All keys in TEA are not secure with reduced key	For quick transactions on the small gadgets, with more security this research will be useful.
The Development of TEA Crypto-Core for Mobile Systems	Tiny Encryption Algorithm	Encrypting and decrypting the plain text using matlab	Minimizing the memory footprint and maximizing the speed.	TEA has a few weaknesses most notably from equivalent keys and related-key attacks	This system makes use of MATLAB which helps in understanding the structure of TEA.

FPGA Based Implementat ion Scenarios of TEA Block Cipher	Tiny Encryption Algorithm	Hardware implementat ion of TEA cipher algorithm using Verilog HDL to get insight of performance and resource utilization.	Utilizes the least resources with reasonable power utilization	Its throughput is minimum due to more number of clock cycles	The throughput of the design can further be improved by introducing the pipeline approach at specific design registers.
Securing Ubiquitous and Low-Cost RFID Using Tiny Encryption Algorithm	Tiny Encryption Algorithm	For pervasive, ubiquitous applications employing RFID devices, low-cost and secure RFID's tags	The area, time, and power specification meeting low-cost RFID requirement	Limited to medium secure systems	Since,the system is limited for small devices ,this should be further developed to meet up other devices.

Extended Inference:

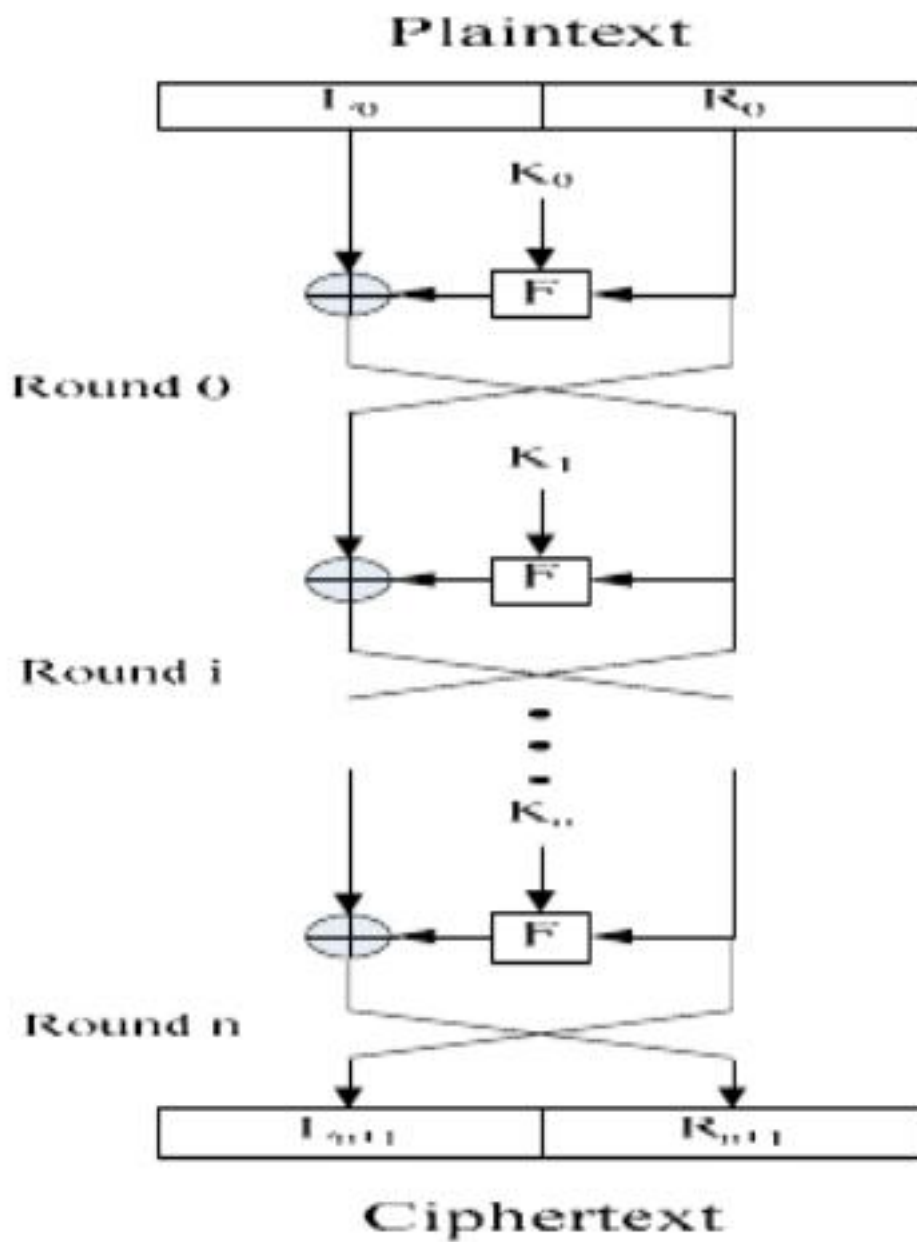
- 1) Secure image transferring uses to scramble the given image using permutation and then encrypting using TEA. This algorithm is useful in transferring sensitive information like medical images.
- 2) On the security of reduced key method used to reduce the key size of 64 instead of 128 bit without reducing its security.
- 3) The development of TEA crypto-core system uses the matlab for encryption and decryption of text in a simple and easy way.
- 4) FPGA based implementation is an hardware implementation of TEA cipher using verilog HDL to get insight of performance and resource utilization.
- 5)securing ubiquitous and low cost RFID using tiny encryption algorithm for pervasive ubiquitous applications employing RFID devices, low-cost and secure RFID tags.

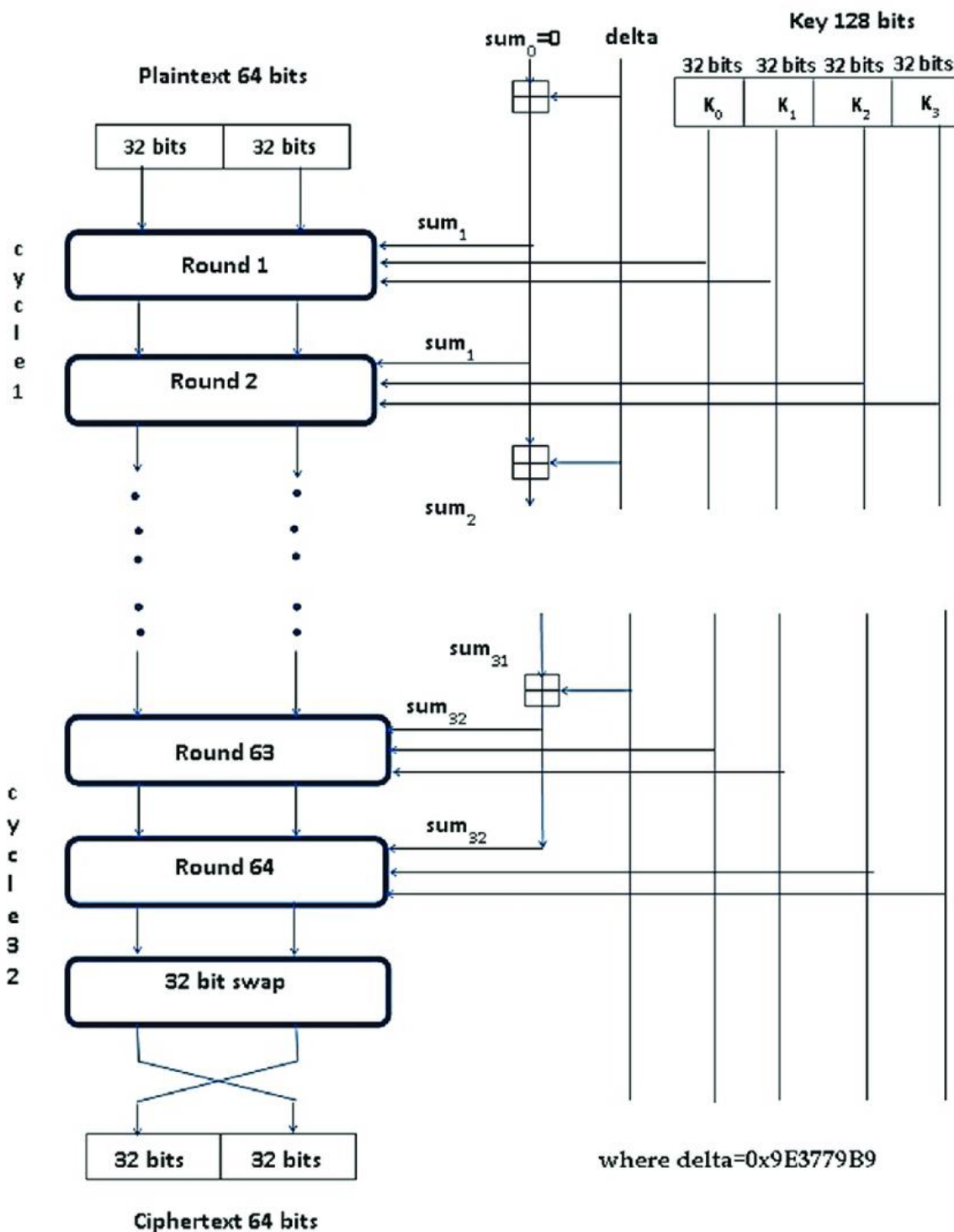
Proposed Work :

Architecture of TEA:

TEA is a Feistel Structured symmetric key algorithm. TEA is a block cipher that uses a 64 bit plain text with 64 rounds and a Key Length of 128-bit with variable rounds having 32 cycles. It does not contain S- boxes and same algorithm is used in reversed for decryption.

Client - server Architecture:

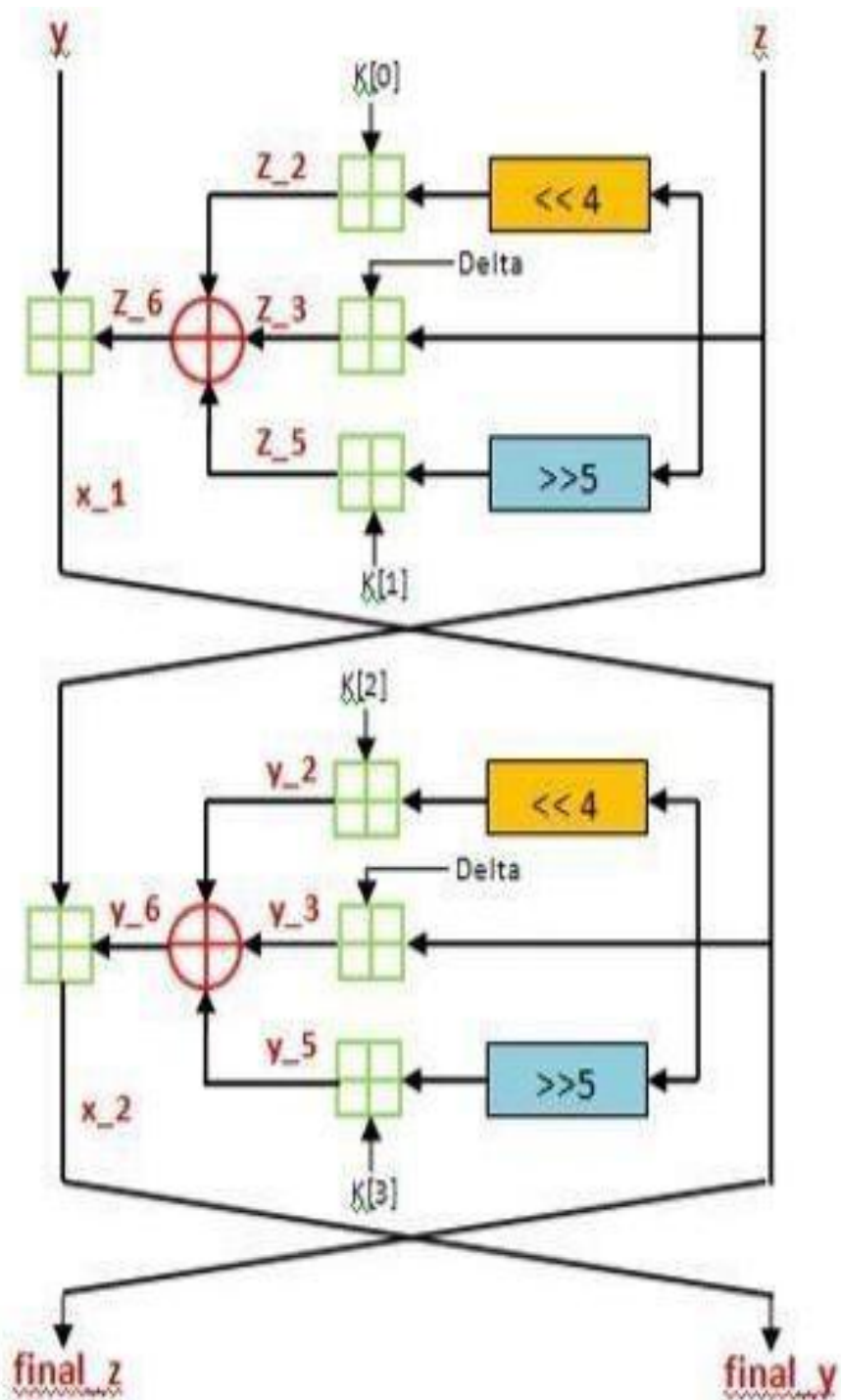




The Encryption of TEA:

Tiny Encryption Algorithm is a Feistel type routine design which uses addition and subtraction as the reversible operators . XOR and ADD alternately used in the routine provide non linearity . The Dual bit shifting in the routine cause all the bits and data mixed repeatedly . The three XOR, ADD and SHIFT operation will provide properties of diffusion and confusion necessary for a secure block cipher without the need for P-boxes and S-boxes . TEA is a feistel cipher that split the plain text into halves . A sub key will be applied to the one half of plain text in the round function,. Then the output of the F will be XOR with other half before the two halves are swapped . All same patterns applied to the entire round except the last round where there is often no swap . A Feistel cipher

where 64 bits of plain text is divided into halves which are equally 32 bits each part. 128 bits of key is used for the encryption and decryption process and it is spitted into 32 bits sub key .



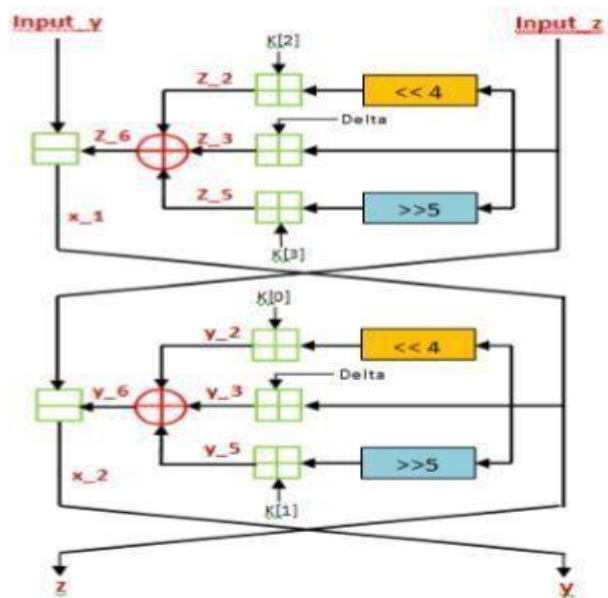
Code for encryption:

```
import sys
from ctypes import *
def encipher(v, k):
    y = c_uint32(v[0])
    z = c_uint32(v[1])
    sum = c_uint32(0)
    delta = 0x9e3779b9
    n = 32
    w = [0,0]
    while(n>0):
        sum.value += delta
        y.value += ( z.value << 4 ) + k[0] ^ z.value + sum.value ^ ( z.value >> 5 ) +
k[1]
        z.value += ( y.value << 4 ) + k[2] ^ y.value + sum.value ^ ( y.value >> 5 ) +
k[3]
        n -= 1
    w[0] = y.value
    w[1] = z.value
    return w
if __name__ == "__main__":
    key = [1,2,3,4]
    v = [1385482522,639876499]
    enc = encipher(v,key)
    print (enc)
```

Decryption:

The decryption of TEA is basically almost the same as the encryption of TEA with the function is being reversed. The decryption process started with the encrypted text is now treated as the input of the algorithm in which the final_y is indicated as input_y and final_z is indicated as the input_z. The input_z is initially being left-shifted with 4-bits and the result is then being added up with the third passkey which is the K[2] and the result is then being kept aside in the memory with the name of z_2. The input_z is then being utilized to add up with the Golden Ratio constant and the result is being saved in the memory z_3. The following step is to reuse again the input_z value to undergo a right-shift of 5-bits and the result is being added up with the fourth passkey which is the K[3] with the result being recorded in the memory named z_5. The decryption process continued with the z_2 being XORed with the z_3 and the result is again being XORed with z_5. This final result is then stored in the memory carrying the name of z_6. Next, the value of z_6 must be reduced

by the input_y value and the result is being saved in the memory x_1 in order for the one round TEA decryption to be utilized up to this point.



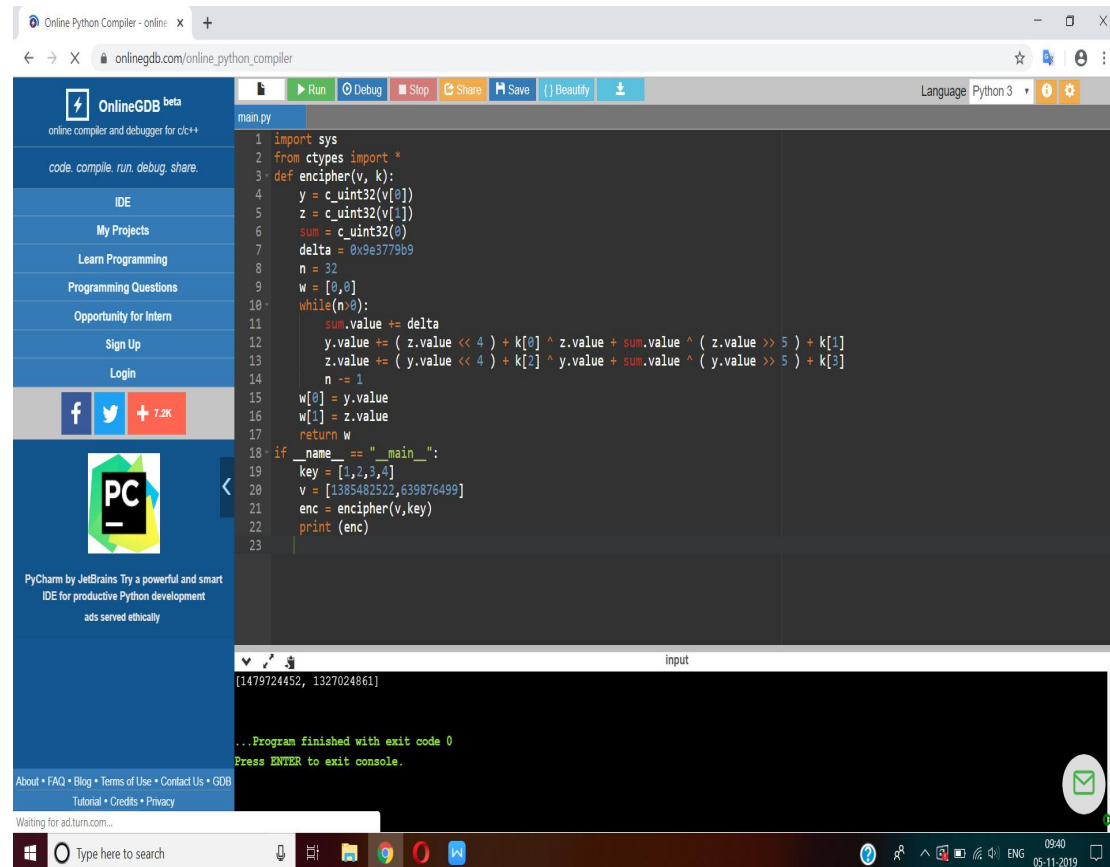
Code for Decryption:

```
def decipher(v, k):
    y = c_uint32(v[0])
    z = c_uint32(v[1])
    sum = c_uint32(0xc6ef3720)
    delta = 0x9e3779b9
    n = 32
    w = [0,0]
    while(n>0):
        z.value -= ( y.value << 4 ) + k[2] ^ y.value + sum.value ^ ( y.value >> 5 ) +
k[3]
        y.value -= ( z.value << 4 ) + k[0] ^ z.value + sum.value ^ ( z.value >> 5 ) +
k[1]

        sum.value -= delta
        n -= 1
        w[0] = y.value
        w[1] = z.value
    return w
if __name__ == "__main__":
    key = [1,2,3,4]
    v = [1385482522,639876499]
    enc = encipher(v,key)
    print (enc)
    print (decipher(enc,key))Results :
```


Result:

Enciper



The screenshot displays the OnlineGDB web interface for Python. The left sidebar contains navigation links such as 'IDE', 'My Projects', 'Learn Programming', 'Programming Questions', 'Opportunity for Intern', 'Sign Up', and 'Login'. The main editor area shows a Python script named 'main.py' with the following code:

```
1 import sys
2 from ctypes import *
3 def encipher(v, k):
4     y = c_uint32(v[0])
5     z = c_uint32(v[1])
6     sum = c_uint32(0)
7     delta = 0x9e3779b9
8     n = 32
9     w = [0,0]
10    while(n>0):
11        sum.value += delta
12        y.value += ( z.value << 4 ) + k[0] ^ z.value + sum.value ^ ( z.value >> 5 ) + k[1]
13        z.value += ( y.value << 4 ) + k[2] ^ y.value + sum.value ^ ( y.value >> 5 ) + k[3]
14        n -= 1
15    w[0] = y.value
16    w[1] = z.value
17    return w
18 if __name__ == "__main__":
19     key = [1,2,3,4]
20     v = [1385482522,639876499]
21     enc = encipher(v,key)
22     print (enc)
23
```

The output console at the bottom shows the result of the encryption:

```
input
[1479724452, 1327024861]
...Program finished with exit code 0
Press ENTER to exit console.
```

The Windows taskbar at the bottom indicates the system time as 09:40 on 05-11-2019.

Decipher:

The screenshot shows a web browser with the OnlineGDB Python compiler. The code is as follows:

```

11 sum.value += delta
12 y.value += ( z.value << 4 ) + k[0] ^ z.value + sum.value ^ ( z.value >> 5 ) + k[1]
13 z.value += ( y.value << 4 ) + k[2] ^ y.value + sum.value ^ ( y.value >> 5 ) + k[3]
14 n -= 1
15 w[0] = y.value
16 w[1] = z.value
17 return w
18 def decipher(v, k):
19 y = c_uint32(v[0])
20 z = c_uint32(v[1])
21 sum = c_uint32(0xc6ef3728)
22 delta = 0x9e3779b9
23 n = 32
24 w = [0,0]
25 while(n>0):
26 z.value -= ( y.value << 4 ) + k[2] ^ y.value + sum.value ^ ( y.value >> 5 ) + k[3]
27 y.value -= ( z.value << 4 ) + k[0] ^ z.value + sum.value ^ ( z.value >> 5 ) + k[1]
28 sum.value -= delta
29 n -= 1
30 w[0] = y.value
31 w[1] = z.value
32 return w
33 if __name__ == "__main__":
34 key = [1,2,3,4]
35 v = [1365482522, 639876499]
36 enc = encipher(v, key)
37 print (enc)
38 print (decipher(enc, key))

```

The input is: [4192648365, 668894959]

The output is: [2213443429, 3463281144]

...Program finished with exit code 0
Press ENTER to exit console.

Discussion:

Algorithm	Key type	Plain Text/Cipher Text Length	Key Size	No. of S boxes	No. of Rounds	Run time (in sec)
Blowfish	symmetric	64 bits	128-448	4	16	11
Tea	symmetric	64 bits	128	-	64 (32 cycles)	0.15
RSA	Asymmetric	K-bits	1024 - 4096 bit typical	-	1	

References:

<https://ieeexplore.ieee.org/abstract/document/7130282/>

AÇ Bağbaba, B Örs, OS Kayhan... - 2015 23rd Signal ..., 2015 - ieeexplore.ieee.org
Recently, image compression and image encryption methods are very crucial in terms of transmission and security of images. Therefore, image compression and encryption were implemented by using JPEG standard and Tiny Encryption Algorithm

<https://ieeexplore.ieee.org/abstract/document/7867512/>

J Tian, X Gao - 2016 IEEE Advanced Information Management ..., 2016 - ieeexplore.ieee.org

With the increasing number of applications for security in small devices and short lived

transactions services, and the encryption algorithm which we need is faster, more convenient, and more secure. TEA is one of them

<https://ieeexplore.ieee.org/abstract/document/7421014/>

MA Hussain, R Badar - 2015 13th International Conference on ..., 2015 - ieeexplore.ieee.org
Transmission of sensitive data over some channel is a highly security constrained scenario and thus demands the application of some encryption algorithm. It is better to implement the algorithm in hardware as compared to software due to better computational speed

<https://ieeexplore.ieee.org/abstract/document/6507813/>

SAY Hunn, N binti Idris - 2012 IEEE International Conference ..., 2012 - ieeexplore.ieee.org
In this paper, a cryptographic algorithm design called Tiny Encryption Algorithm (TEA) is proposed in order to minimize the memory footprint and maximize the speed. The design was targeted for embedded and mobile systems which concern more on speed

<https://ieeexplore.ieee.org/abstract/document/1613621/>

P Israsena - 2006 1st International Symposium on Wireless ..., 2006 - ieeexplore.ieee.org
For pervasive, ubiquitous applications employing RFID devices, low-cost and secure RFIDs tags are becoming a necessity. The ICs for such systems have stringent requirements in terms of cost related to area and power consumption, making conventional encryption