2007

A Differential Power Analysis Resistant Randomized Algorithm using Multiple AES Ciphers

Richard Tran
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the Computer Sciences Commons

Recommended Citation
DOI: https://doi.org/10.31979/etd.ehqt-4zmj
https://scholarworks.sjsu.edu/etd_projects/39

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.
A Differential Power Analysis Resistant Randomized Algorithm using Multiple AES Ciphers

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Richard Tran

Fall 2007
APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Robert Chun

Dr. Mark Stamp

Dr. David Taylor

APPROVED FOR THE UNIVERSITY
ABSTRACT

A Differential Power Analysis Resistant Randomized Algorithm using Multiple AES Ciphers

by Richard Tran

Differential power analysis (DPA) side channel attacks have been shown to have great effectiveness in breaking ciphers (such as the Advanced Encryption Standard or AES) that were previously though to be unbreakable. There are currently many methods published that prevent differential power analysis on AES. The method proposed for this project is based on the increased usage of multiprocessors and multicore processors. By using multiple copies of the same AES cipher, a randomly chosen cipher is used to encrypt each plaintext. The other ciphers are then used to obfuscate the data made available to the attacker for DPA in the hope of making DPA impossible or require a statistically large amount of data that it is practically impossible to run successfully.
Acknowledgements

I thank my advisor, Dr. Robert Chun, for his support in my project. Dr. Chun not only helped guide me to my project topic but he also provided me with various software licenses to aid in the completion of this project. I would also like to express my gratitude to Dr. Mark Stamp and Dr. David Taylor for participating as my committee members.

Dr. Stamp’s cryptography class was a huge inspiration in my chosen topic as I had no knowledge on security prior to it. It was in his class that I was first introduced to such concepts as AES and side channel attacks.

Dr. Taylor has played a huge role in launching me into the Computer Science Master’s program with his letter of recommendation. My first three classes at San Jose State University were also taught by him.

Dr. Chun, Dr. Stamp, and Dr. Taylor are huge assets to the Computer Science program at San Jose State University. The passion for the classes they teach is quite infectious and I would never have got this far without them.
## Table of Contents

1 Introduction 1  
2 Side Channel Attacks 2  
   2.1 Passive Side Channel Attacks 2  
   2.2 Active Attacks 4  
3 Differential Power Analysis 5  
4 DPA on AES 7  
   4.1 Advanced Encryption Standard 7  
      4.1.1 Add Round Key 7  
      4.1.2 Byte Substitution 8  
      4.1.3 Shift Rows 9  
      4.1.4 Mix Columns 10  
   4.2 A DPA Attack 10  
      4.2.1 The Simulated Attack 12  
      4.2.2 The Actual Attack 15  
5. Related Work 18  
   5.1 Hardware Method (Wave Dynamic Differential Logic) 18  
   5.2 Algorithmic Methods (Bit Masking) 19  
   5.3 Material Considerations 20  
6. Proposed Randomized Algorithm 21  
   6.1 Randomized Algorithm with Multiple Ciphers 21  
   6.2 Randomized Algorithm with Multiple Ciphers and Hash 22  
   6.3 Randomized Algorithm with Multiple Ciphers and Hash and Varying Key 24  
7. Implementation 26  
   7.1 Software Implementation 26  
   7.2 Security Analysis 28  
   7.3 Overhead Analysis 33  
8. Conclusion 36  
9. Future Work 37  
Appendices  
   Appendix A. Source Code 38  
   Appendix B. Sample Output 81  
References 121
List of Tables

Table 1. AES Class Methods. ............................................................................................26
List of Figures

Figure 1. Simple Power Analysis Trace of DES .................................................................3
Figure 2. Higher Resolution View of DES Trace ...............................................................4
Figure 3. DPA Trace of DES ...............................................................................................6
Figure 4. Add Round Key ....................................................................................................8
Figure 5. Byte Substitution ..................................................................................................9
Figure 6. Shift Rows ............................................................................................................9
Figure 7. Mix Columns .....................................................................................................10
Figure 8. Fastcore Crypto-chip Block Diagram .................................................................12
Figure 9. Correlation Between $M_1$ and $M_4$ ................................................................14
Figure 10. Correlation Between $M_1$ and $M_4$ For Different Measurements ...............14
Figure 11. Correlation Between $M_4$ and $M_7$ ...............................................................16
Figure 12. Correlation Between $M_4$ and $M_7$ For Different Measurements ...............16
Figure 13. Wave Dynamic Differential Logic ..................................................................18
Figure 14. AES Power Graph with and without WDDL ...................................................19
Figure 15. The $i^{th}$ Round of AES with and without Masking Countermeasure .........20
Figure 16. Initial Randomized Algorithm .........................................................................21
Figure 17. Flaw in Initial Algorithm ..................................................................................22
Figure 18. Revised Algorithm with Hash .........................................................................24
Figure 19. Pseudo Code for Randomized Algorithm with Multiple Ciphers and Hash and Varying Key .................................................................25
Figure 20. Sample Output for Standard AES ....................................................................28
Figure 21. Sample Output for Randomized Algorithm with Multiple Ciphers and Hash 29
Figure 22. Sample Output for Randomized Algorithm with Multiple Ciphers and Hash and Varying Key .................................................................30
Figure 23. Number of Times Cipher 1 is chosen from N Ciphers in the Randomized Algorithm with Multiple Ciphers and Hash and Varying Key .................................................................31
Figure 24. Consecutive Number of Times Cipher 1 is chosen from N Ciphers in the Randomized Algorithm with Multiple Ciphers and Hash and Varying Key .................................................................32
Figure 25. Performance Overhead Test ..........................................................................34
1. Introduction

Differential power analysis (DPA) side channel attacks have been shown to have great effectiveness in breaking ciphers (such as the Advanced Encryption Standard or AES) that were previously though to be unbreakable [20]. There are currently many methods published that prevent differential power analysis on AES [1][8][9][10][23][24]. The proposed method was designed based on the increased usage of multiprocessors and multi-core processors. By using multiple copies of the same AES cipher, I randomly chose a cipher to encrypt each plaintext. The other ciphers are then used to obfuscate the data made available to the attacker for DPA in the hope of making DPA impossible or require a statistically large amount of data that it is practically impossible to run successfully.
2. Side Channel Attacks

Side channel attacks allow attackers to find out information on a cipher without directly attacking the security algorithm. This method utilizes information that can be found based on the cipher’s physical characteristics. These unintended “side channels” of information allows attackers to figure out how a computation is performed. There are many types of side channel attacks available today in both passive and active forms.

2.1. Passive Side Channel Attacks

Passive attacks attempt to crack a cipher based upon observed data only. The attacker is not allowed to interact at all with the cipher itself or its users. Passive attacks are possibly more dangerous than active attacks to the cipher user as they would be unaware that their cipher is being compromised and would not be able to react accordingly. Passive attacks include timing attacks, TEMPEST attacks, and power analysis attacks.

Timing attacks study the amount of time it takes to process different inputs. This could apply to branching and condition statements, cache hits, and processor instructions. Paul Kocher introduces the possibility of a timing attack on many popular ciphers such as Diffie-Hellman, RSA, and DSS [11]. For Diffie-Hellman [7] and RSA [21], Kocher analyzes the time it takes the cipher to run a modular exponentiation and is able to back calculate the secret key (the exponent) based on whether it is running fast or slow as it calculates each bit of the exponent. The number of samples needed is then proportional to exponent size.
TEMPEST [14] refers to the study of unintentional emanations from which the information processed by a piece of equipment can be received. The term TEMPEST is often used analogously for the field of emission security (EMSEC). Wim van Eck developed a process called Van Eck phreaking where the contents of a CRT display could be detected by its electromagnetic emissions [25]. Van Eck was able to accomplish this using $15 dollars worth of equipment at a range of hundreds of meters.

Power analysis [19] studies the power consumption of a cryptographic device in order to find out what that device is doing. It allows the attacker to look inside tamperproof black boxes. Kocher et al. uses simple power analysis (SPA) in order to analyze DES [12]. Kocher et al. manages to find out the key schedule, permutations, comparisons, multipliers and exponentiators used in a DES implementation.

Figure 1. Simple Power Analysis Trace of DES [12]

Figure 1 clearly shows the 16 DES rounds in action.
At clock cycle 6 in figure 2, the upper trace shows a jump being performed as opposed to the bottom trace.

Differential power analysis is another type of power analysis that will be discussed in greater detail later.

2.2 Active Attacks

While active attacks are faster than passive attacks there is a higher chance of alerting the cipher users and allowing them to attempt to prevent the exploit. Differential fault analysis [4] is the process where faults are introduced into a cipher in order to recover information. Eli Biham and Adi Shamir use ionizing or microwave radiation in order to obtain the subkey for DES [15] and 3DES [18] by causing a fault in one of the bits in a register. Finding the rest of the key is then trivial as the subkey contains most of the bits.
3. Differential Power Analysis

Differential power analysis (DPA) uses statistical analysis of power consumption on multiple iterations of a cipher in order to calculate key blocks of data [12]. In comparison with the SPA example shown previously, DPA analysis allows the attacker to get the full DES subkey by guessing the key block for each S box.

Kocher defines the following DPA selection function [12]:

\[ D(C, b, K_s) = \text{The value of bit } 0 \leq b < 32 \text{ of the DES intermediate } L \text{ at the beginning of the } 16^{th} \text{ round for ciphertext } C, \text{ where the 6 key bits entering the S box corresponding to bit } b \text{ are represented by } 0 \leq K_s < 2^6 \]

Kocher then defines the differential trace to be the differences between the average of the traces where \( D = 1 \) and the average of the traces where \( D = 1 \). When \( K_s \) is incorrect this trace should be close to 0.
Figure 3. DPA Trace of DES [12]

Figure 3 shows a differential trace at the bottom where a direct correlation to the key bit can be seen at clock cycle 6.

The attackers are not limited to the DPA function defined by Kocher. Different functions can be used for different ciphers and for each cipher there may be many types of functions that will help reduce the number of samples needed.
4. DPA on AES

DES was selected as the Federal Information Processing Standard (FIPS) for the United States since 1976. Since then DES has been superseded by the new Advanced Encryption Standard (AES) in 2002 [16]. As a result most current DPA research is modeled on AES ciphers.

4.1. Advanced Encryption Standard

Realizing that DES was outliving its usefulness, the National Institute of Standards and Technology (NIST) had a contest to see what would be called the Advanced Encryption Standard (AES). The winner of this contest was the Rijndael algorithm [6].

AES is a block cipher with a three possible block sizes (128, 192, or 256 bits) and three possible key lengths (128, 192, or 256 bits). The number of rounds will vary from 10 to 14 based on the key length. Each round consists of the following four stages.

4.1.1. Add Round Key

A subkey is derived from the main key using Rijndael’s key schedule. Each byte of the state is combined with a byte of the round subkey using the XOR operation.
4.1.2. Byte Substitution

Each byte in the array is updated using an S-box similar to DES.
4.1.3. Shift Rows

This operation consists of simple cyclic shifts of the bytes in each row of the byte array by a certain offset. The offset will vary depending on the block size used.
4.1.4. Mix Columns

Each column is treated as a polynomial over a Galois Field and is then multiplied by modulo $x^4 + 1$.

![Figure 7. Mix Columns](http://en.wikipedia.org/)

The previous four phases are invertible making AES capable of both encryption and decryption.

4.2. A DPA Attack

Siddika Berna Ors, Frank Gurkaynak, Elisabeth Oswald, and Bart Preneel were the first to publish an actual DPA attack on AES [20]. Ors et al. used correlation analysis
to make their statistical comparisons. The correlation analysis is based off of the Pearson correlation coefficient,

\[ C(T, P) = \frac{\left( E(T * P) - E(T) * E(P) \right)}{\sqrt{\text{Var}(T) * \text{Var}(P)}} \text{,} \]

where \( T \) is the set of traces, \( P \) is the set of Predictions, \( E(T) \) is the expectation trace of the set of traces \( T \) and \( \text{Var}(T) \) is variance of the set of traces \( T \).

Ors et al. did an attack on both simulated data and actual data on the Fastcore crypto-chip (see figure 8).
4.2.1. The Simulated Attack
The goal of the simulated attack was to avoid as much outside noise as possible. DPA depends on very small variations in power usage and is sometimes overshadowed by noise and measurement errors.

Ors et al. attempts to predict the power consumption used when the 8 most significant bits of the round key was stored in the register. They produced a simulated power consumption file based on N random plaintexts and one randomly fixed key. The power consumption file is stored in an Nx10 matrix $M_1$. They then calculated an $Nx2^L$ matrix $M_4$ which contained the prediction for the bit changes in the Register for a particular guess of the $L$ attacked key bits of the initial key. The correlation coefficient is calculated based on the following equation,

$$c_{i,j} = C(M_1(1:i,1), M_4(1:i,j)),$$

where $i = 1, \ldots, 10000$ and $j = 0, \ldots, 2^L-1$

The results are shown in figure 9 and 10.
Figure 9. Correlation Between $M_1$ and $M_4$. [20]

Figure 10. Correlation Between $M_1$ and $M_4$ For Different Measurements. [20]
Ors et al. were able to get the 8 MSBs of the key within 400 measurements for the simulated attack.

4.2.2. The Actual Attack

Similar to $M_1$, Ors et al. define another $N \times 1000$ matrix $M_5$. This time they use real data instead of simulated. A pre-processing technique is then used to reduce the amount of noise, 

$$e_{i,j} = E(M_5(i, D * (j -1) + 1 : D * j)),$$

where $i = 1,\ldots,N$, $j = 1, 2$ and $D$ is the number of data points measured in a single clock cycle.

Ors et al. define $M_7$ as their pre-processed measurement data,

$$M_7(i, j) = E(M_5(i, D + 1 : D + j)) - E(M_5(i, D + 1 - j : D)),$$

where $i = 1,\ldots,N$ and $j = 1,\ldots,D$.

The results are shown in Figure 11 and 12.
Figure 11. Correlation between $M_4$ and $M_7$. [20]

Figure 12. Correlation between $M_4$ and $M_7$ for different measurements. [20]
Ors et al. were able to get the 8 MSBs of the key within 4000 measurements for actual attack. This is ten times more than the measurements needed for the simulated attack.
5. Related Work

Due to the increased success that DPA has seen on cracking ciphers that were previously though unbreakable with today’s technologies, cryptanalysis have come up with various methods to counteract this side channel attack. It is possible to prevent DPA with hardware and algorithmic methods. It should also be possible to reduce the likelihood of a DPA attack being successful on a material level.

5.1. Hardware Method (Wave Dynamic Differential Logic)

In 2004, Kris Tiri and Ingrid Verbauwhede introduced a new method of DPA prevention called Wave Dynamic Differential Logic (WDDL) [23]. This method uses a precharge wave in order to induce a constant power consumption level (see figure 13). The idea behind this is that with less variance in power consumption, there will be less information available for DPA.

![Wave Dynamic Differential Logic](image)

Figure 13. Wave Dynamic Differential Logic [23]
David D. Hwang, Kris Tiri, Alireza Hodjat, Bo-Cheng Lai, Shenglin Yang, Patrick Schaumont and Ingrid Verbauwhede have showed significant smoothing of the power consumption level on a 0.18-µm CMOS with the use of WDDL (see Figure 14) [10].

![AES Power Graph with and without WDDL](image)

**Figure 14. AES Power Graph with and without WDDL [10]**

5.2. Algorithmic Methods (Bit Masking)

Bit masking was first introduced by Mehdi-Laurent Akkar and Christophe Giraud (Akkar, 2001). This method uses both a binary additive mask and a nonzero multiplicative mask. The nonzero multiplicative mask is used on the data passing through the non linear part of the AES S-box. The user has to convert the binary additive mask into a multiplicative mask at the input of each non linear part and then reproduce the additive mask from the multiplicative mask at the output of each non linear part. This method has been expanded in recent publications [9][24] but the basic idea to randomize the intermediate results that are produced during a cipher computation remains the same.
5.3. Material Considerations

Both ASIC and FPGA depend on CMOS technology. Most CMOS devices are made using silicon dioxide as the gate oxide material. Intel has been researching high-k dielectric materials as an alternative to silicon dioxide [5]. High-k dielectrics will allow for 45 nanometer technologies. The other benefit of high-k dielectric is the reduction of gate leakage current. Both of these effects will help make DPA much harder. The DPA measuring device will need to have a high sampling frequency due to the faster technology. Also the power trace signals will be much smaller and harder to discern with the current reduction.
6. Proposed Randomized Algorithm

6.1. Randomized Algorithm with Multiple Ciphers

Similar to WDDL, the ideal algorithmic design that will not change the way a cipher was implemented. The initial plan was to use \( n \) identical copies of a cipher and randomly select one to receive the actual cipher text and have the other \( n-1 \) ciphers sent random data (see figure 16).

![Figure 16. Initial Randomized Algorithm](image)

The idea was that while the attacker knows what the plaintext is, he would have a \( 1/n \) chance of a specific cipher receiving the plaintext. If there were \( m \) plaintexts, then the analysis of any specific cipher would probabilistically only get \( m/n \) plaintexts and \( 1-m/n \)
garbage values. This would make DPA analysis more difficult unless the attacker maintained a trace of all of the ciphers.

However this system did not consider the possibility that the attacker can passively scan the inputs to each cipher in order to find out which was getting the plaintext and which was getting a random string (see Figure 17). This would allow the attacker to bypass the randomization completely.

The solution here was to encrypt the data streams from the randomizer to the ciphers. For this purpose I decided to use a hash function. I chose the SHA-1 [17] hash function although other hash functions (like Tiger [2]) will probably work just as well.

6.2. Randomized Algorithm with Multiple Ciphers and Hash
After further consideration, I decided to use the hash functions to generate a one time pad which would be used on the plaintext similar to the Galois/Counter Mode [13]. The randomizer will hash an initialization vector (IV) shared among all the ciphers and a counter that is randomly offset by a value unique to each cipher.

\[ H(IV, \text{Counter} + \text{Offset}_i) = \text{One Time Pad}, \]

where \(i\) is the cipher identifier from 1 to \(n\).

During this time, the ciphers will also be generating their own one time pad, except they will always use their own unique offset.

The randomizer will choose an offset randomly. It will then XOR the plaintext and the offset with the one time pad to generate the message to be sent to all ciphers (see figure 18).

\[ \text{Message} = (\text{plaintext}, \text{Offset}_i) \oplus \text{one time pad} \]

Once the ciphers receive the message, they will XOR it with the one time pad they have calculated. If the resulting string contains their unique offset, the cipher will know that it was chosen by the randomizer and that it has decrypted the plaintext successfully. The other ciphers will get a resulting garbage string.
The encrypting cipher will increment its counter after each message it sends out and the decrypting ciphers will increment their counters for each message they have received and processed.

Now if the attacker were to listen in on the communication between the randomizer and the ciphers they would not be able to tell which cipher got the plaintext unless they were able to break the hash.

The revised method makes the assumptions that the IV and the initial counter value are known to both the randomizer and the ciphers. It also assumes that the randomizer knows the unique offset of each cipher.

However, another problem is still present with this method. For DPA incorrect inputs to a cipher could be useful. Given enough samples, DPA could still figure out the contents of the key. In order to fix, the cipher key would have to be varied.

6.3. Randomized Algorithm with Multiple Ciphers and Hash and Varying Key
Each cipher will keep one master key. This is the key that is used for correct encryption and decryption. However when the cipher detects that it was not chosen by the randomizer for that round. It replaces the key with the hash value. This will cause the cipher to attempt to decrypt the garbage cipher text with a garbage key

The pseudo code for the cipher would look like the following:

*Calculate One Time Pad*

*Store One Time Pad in Bad Key*

*Upon Receiving a Message*

*XOR Message with One Time Pad*

*If Offset in Message equals My Offset*

*Run Plaintext on Good Key*

*Else*

*Run Garbage String on Bad Key*

Figure 19. Pseudo Code for Randomized Algorithm with Multiple Ciphers and Hash and Varying Key

Now the attacker will only gain useful data through DPA if they can correctly guess which cipher received the plaintext. For n ciphers ran for k iterations, the probability of an attacker randomly guessing the correct decrypting ciphers for all k iterations is (1/n)^k.
7. Implementation

7.1. Software Implementation

The randomized algorithm with multiple ciphers and hash and varying key is implemented in Java to view the effectiveness of the generated garbage output.

The algorithm is implemented using an open source Rijndael implementation [3] and an MD5 hash. The source code for the application can be found in Appendix A.

Implementation Details:

Plain Text: “OriginalPlainTxt”

Master Key: “DefaultCipherKey”

Initialization Vector: “0123456789012345”

Cipher Offset: 12

The same plain text is used in multiple iterations to illustrate the point that even with a fixed input the obfuscated output will still work effectively. The number of iterations and the number of ciphers is left open as a variable.

The AES class has 6 public functions:

<table>
<thead>
<tr>
<th>stdEncryption</th>
<th>Performs the basic AES encryption algorithm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdDecryption</td>
<td>Performs the basic AES decryption algorithm</td>
</tr>
<tr>
<td>hashEncryption</td>
<td>Randomly chooses 1 of n possible decrypting cipher offsets to generate a one time pad through an MD5 hash of the chosen offset, counter, and initialization vector. The hash is then used</td>
</tr>
</tbody>
</table>
on the cipher text generated from the basic AES encryption algorithm to create a new ciphertext that is sent to all n possible decrypting ciphers.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hashDecryption</td>
<td>This cipher will generate a one time pad through the MD5 hash of its offset, counter, and initialization vector. The hash is then used on the ciphertext to generate a new ciphertext that will be decrypted with the basic AES decryption algorithm.</td>
</tr>
<tr>
<td>randomKeyEncryption</td>
<td>Randomly chooses 1 of n possible decrypting cipher offsets to generate a one time pad through an MD5 hash of the chosen offset, counter, and initialization vector. The hash is then used on the cipher text generated from the basic AES encryption algorithm to create a new ciphertext that is sent to all n possible decrypting ciphers.</td>
</tr>
<tr>
<td>randomKeyDecryption</td>
<td>This cipher will generate a one time pad through the MD5 hash of its offset, counter, and initialization vector. The hash is then used on the ciphertext to generate a new ciphertext that will be decrypted with the basic AES decryption algorithm. The key used in algorithm will either be the correct key if the received offset, counter, and initialization vector matches the cipher’s offset, counter and initialization vector. Otherwise, the algorithm will use the one time pad for a key.</td>
</tr>
</tbody>
</table>

Table 1. AES Class Methods
7.2 Security Analysis

The standard encryption worked as expected. As long as the plaintext and key are the same, the encrypting cipher will generate the exact same ciphertext and if the ciphertext and key are the same the decrypting cipher will generate the exact same plaintext. This method is highly susceptible to DPA attacks as the information for N iterations will be more uniformed due to the fixed structure of the input and output.

ENCRYPTION 1
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Ciphertext: 0Ú‡HdØ’ÅÉxȧ Å_Eb_f

DECRYPTION 1
Ciphertext: 0Ú‡HdØ’ÅÉxȧ Å_Eb_f  Key: DefaultCipherKey
Plaintext: OriginalPlainTxt

...

ENCRYPTION N
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Ciphertext: 0Ú‡HdØ’ÅÉxȧ Å_Eb_f

DECRYPTION N
Ciphertext: 0Ú‡HdØ’ÅÉxȧ Å_Eb_f  Key: DefaultCipherKey
Plaintext: OriginalPlainTxt

Figure 20. Sample Output for Standard AES

The hash encryption worked better by sending ciphertext that can only be decrypted by the randomly chosen decrypting cipher. For the remaining ciphers this ciphertext would generate garbage plaintext. It should be noted that for the same plaintext, the ciphertext generated will be different during consecutive iteration even if the same decrypting cipher is chosen. A key flaw in this algorithm is that while the ciphertext is randomized, the key remains the same. This algorithm does not afford the user any additional protection against DPA than the standard encryption algorithm.
because even garbage ciphertext can reveal details on the key. The following figure shows the output as seen from a single encrypting cipher and a single decrypting cipher from among 10 possible decrypting ciphers

<table>
<thead>
<tr>
<th>ENCRYPTION 1</th>
<th>Plaintext: OriginalPlainTxt</th>
<th>Key: DefaultCipherKey</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Counter: 1  Offset: 423</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ciphertext: %j,źÆ¶*Ö7Ý·ÝΩ©?5³?`¶</td>
<td></td>
</tr>
<tr>
<td>DECRYPTION 1</td>
<td>Ciphertext: %j,źÆ¶*Ö7Ý·ÝΩ©?5³?`¶</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Key: DefaultCipherKey  Counter: 1  Offset: 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Plaintext: Đ«°&amp;_p_ä7™bZ„”</td>
<td></td>
</tr>
<tr>
<td>ENCRYPTION 2</td>
<td>Plaintext: OriginalPlainTxt</td>
<td>Key: DefaultCipherKey</td>
</tr>
<tr>
<td></td>
<td>Counter: 3  Offset: 72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ciphertext: HΩVE_rD_</td>
<td></td>
</tr><tr>
<td>ottle{A}{H}z)¹?İ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECRYPTION 2</td>
<td>Ciphertext: HΩVE_rD_</td>
<td></td>
</tr><tr>
<td>ottle{A}{H}z)¹?İ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Key: DefaultCipherKey  Counter: 3  Offset: 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Plaintext: __•1ëáåønÃ×P_Â];</td>
<td></td>
</tr>
<tr>
<td>ENCRYPTION 3</td>
<td>Plaintext: OriginalPlainTxt</td>
<td>Key: DefaultCipherKey</td>
</tr>
<tr>
<td></td>
<td>Counter: 4  Offset: 423</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ciphertext: Áû[İ?'Ý„â¥_x?jltöi;</td>
<td></td>
</tr>
<tr>
<td>DECRYPTION 3</td>
<td>Ciphertext: Áû[İ?'Ý„â¥_x?jltöi;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Key: DefaultCipherKey  Counter: 4  Offset: 12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Plaintext: °_áz9öiÎtÅxE(Åf®</td>
<td></td>
</tr>
</tbody>
</table>

...  

Figure 21. Sample Output for Randomized Algorithm with Multiple Ciphers and Hash

The randomized algorithm with multiple ciphers and hash and varying key fixes this problem by running identically to the previous algorithm with the exception that if the decrypting cipher identifies itself to be the wrong cipher it will replace its key with
the hash text. This will in effect randomize the key in a similar manner that the ciphertext was randomized. The attacker will now need to be able to identify the correct cipher in order to avoid analyzing power information for the incorrect key.

Figure 22. Sample Output for Randomized Algorithm with Multiple Ciphers and Hash and Varying Key

In the following figure it can be seen that for N decrypting ciphers over M iterations, the number of times a specific cipher will be chosen will follow M/N.
Figure 23. Number of Times Cipher 1 is chosen from N Ciphers in the Randomized Algorithm with Multiple Ciphers and Hash and Varying Key.

The effectiveness of the randomization can be seen in the following figure which displays the average max number of consecutive times a cipher will be chosen in a row.
Figure 24. Consecutive Number of Times Cipher 1 is chosen from N Ciphers in the Randomized Algorithm with Multiple Ciphers and Hash and Varying Key.

It can be seen that the randomization is effective in mixing the ciphers up and that the attacker will have a hard time approximating the randomized distribution.

Now assuming that an attacker is willing to spend the time to manually determine if the decrypting cipher is the chosen by matching the decrypting cipher output with the original plaintext, it will be shown that the number of samples needed to be analyzed will make this attack extremely inefficient. From Section 4.2.2., it was shown that an actual attack required 4000 generated samples just to calculate 8 bits of a 128 bit key and therefore 64000 samples for the whole key. It can be seen that to gather the 64000 samples required, n*64000 samples will be needed from the improved algorithm where n is the number of decrypting ciphers in the system. For a 3-cipher system, the attacker will
need to manually analyze 192000 samples and then match the iterations where the cipher was chosen with the time interval on the power trace chart.

The figure also shows that the algorithms effectiveness increases logarithmically with the number of ciphers used. The analysis shows that a 3 cipher system generates a good performance within a reasonable cost increase.

7.3. Overhead Analysis

Even with an effective algorithm, overhead is a big issue. If the overhead outshadows the usefulness of such an algorithm then it will never be used in industry. In order to measure the processing overhead, a timing analysis was done on the three implemented algorithm for varying numbers of iterations.
The above figure shows that the three algorithms vary from each other by a constant factor. The randomized algorithm with multiple ciphers and hash and varying key and the randomized algorithm with multiple ciphers and hash have a similar overhead of about 250% that of the standard decryption algorithm. This can be further optimized through the use of a more efficient hash algorithm or a precalculation of the hash by the decrypting ciphers while the encrypting cipher is encrypting the plaintext.

As for hardware overhead, it can be seen that the the algorithm is more effective with the more ciphers used. However due to the cost of such an implementation it is recommended that the ciphers are limited in number. Three ciphers appear to be the most cost effective solution based on observation of the output given in Appendix B. It should
be noted with the decreasing cost of dual-core and quad-core chips, that a 4-cipher or an 8-cipher implementation might be feasible in the near future.
8. Conclusion

The proposed randomized algorithm with multiple ciphers and hash and varying keys is shown to be effective in randomizing the key enough to make a DPA attack on an AES cipher much more difficult. The cost and overhead of such an algorithm is roughly three times that of a normal AES implementation which makes this solution very attractive. It should be noted that all of the analysis in this project assumes that the attacker can only analyze the power traces from one cipher at a time instead of all ciphers and that it assumes the hash function used is not easily broken. Considerations for these issues are detailed in the next section.
9. Future Work

The analysis of this algorithm is done based on a pure software view. For a more accurate analysis the algorithms proposed in this project should be implemented in hardware such as an FPGA. With a hardware implementation, real DPA attacks can be performed to determine the algorithms’ effectiveness. Many of the previous methods described in this paper can be used concurrently with proposed randomized algorithm. A hybrid method using the proposed algorithm, WDDL and Bit Masking on a high-k dielectric CMOS would receive all of the benefits. Further work will also need to be done on a more intelligent attacker and creating a more secure hash function for the purpose of security analysis.
Appendix A: Java Source Code

import java.security.*;
import java.util.Random;

/**
 * This class runs the AES Cipher with the standard encryption/decryption algorithm
 * as well as the methods proposed in the paper A DPA Resistant Randomized Algorithm
 * using Multiple AES Ciphers by Richard Tran
 *
 * @author Richard Tran
 * @date 11/09/2007
 */

public class AES {

    private Rijndael rijndael;
    private final int keyBits = 128; // 128 bit encryption/decryption
    private final byte[] initVector;

    public final byte[] key; // Only public for testing purposes
    public int counter;

    // Encryption Specific Fields
    private final int NUM_CIPHERS;
    private final int[] offsetTable = {12, 124, 123, 16, 164, 72, 34, 765, 231, 423}; // Max 10 Ciphers
public int selectedOffset;

// Decryption Specific Fields
public final int offset = 12; // Offset for Cipher 1
private boolean selectedDecrypter;
public byte[] decrypterKey;

/**
* AES Constructor only valid for standard encryption and decryption purposes
* @param key AES key
* @param initVector Initialization Vector
*/
public AES(String key, String initVector) {
    NUM_CIPHERS = 0;

    this.initVector = initVector.getBytes();
    this.key = stringTo16ByteArray(key);
    rijndael = new Rijndael();
    counter = 0;
}

/**
* AES Constructor valid for all encryption and decryption methods
* @param key AES key
* @param initVector Initialization Vector
* @param numCiphers Number of Decryption Ciphers in the System
*/
public AES(String key, String initVector, int numCiphers) {
    NUM_CIPHERS = numCiphers;

    this.initVector = stringTo16ByteArray(key);
    this.key = stringTo16ByteArray(key);
    rijndael = new Rijndael();
    counter = 0;
}

/**
 * Standard AES Encryption
 * @param plainText Plaintext in a byte array
 * @return Ciphertext in a byte array
 */
public byte[] stdEncryption(byte[] plainText) {
    rijndael.makeKey(key, keyBits);
    byte[] cipherText = new byte[16];
    rijndael.encrypt(plainText, cipherText);

    return cipherText;
}

/**
 * Standard AES Decryption
 * @param cipherText Ciphertext in a byte array
 * @return Plaintext in a byte array
 */
public byte[] stdDecryption(byte[] cipherText) {
    rijndael.makeKey(key, keyBits);
    rijndael.initCounter(counter);
    byte[] plainText = new byte[16];
    rijndael.decrypt(cipherText, plainText);

    return plainText;
}
byte[] plainText = new byte[16];
rijndael.decrypt(cipherText, plainText);

return plainText;
}

/**
 * AES Encryption for Randomized Algorithm with Multiple Ciphers and Hash
 * @param plainText Plaintext in a byte array
 * @return Masked Counter and Ciphertext in a byte array
 */
public byte[] hashEncryption(byte[] plainText) {
    byte[] cipherText = stdEncryption(plainText);
    byte[] otpText = createEncryptionOTPKey(); // pad
    byte[] otpCipherText = oneTimePad(cipherText, otpText);
    byte[] byteCounter = integerToBytes(counter - 1);
    byte[] otpCounter = oneTimePad(byteCounter, otpText);

    return concatByteArrays(otpCounter, otpCipherText);
}

/**
 * AES Decryption for Randomized Algorithm with Multiple Ciphers and Hash
 * @param counterAndCipherText Counter and Ciphertext in a byte array
 * @return Plaintext in a byte array
 */
public byte[] hashDecryption(byte[] counterAndCipherText) {
    byte[] otpText = createDecryptionOTPKey();

    byte[] otpCounter = extractFirst4Bytes(counterAndCipherText);
    otpCounter = oneTimePad(otpCounter, otpText);
    int extractedCounter = bytesToInteger(otpCounter);

    if (counter21 == extractedCounter)
        selectedDecrypter = true;
    else
        selectedDecrypter = false;

    byte[] cipherText = removeFirst4Bytes(counterAndCipherText);
    byte[] newCipherText = oneTimePad(otpText, cipherText);
    return stdDecryption(newCipherText);
}

/**
 * AES Encryption for Randomized Algorithm with Multiple Ciphers and Hash
 * @param plainText Plaintext in a byte array
 * @return Masked Counter and Ciphertext in a byte array
 */
public byte[] randomKeyEncryption(byte[] plainText) {
    byte[] cipherText = stdEncryption(plainText);
    byte[] otpText = createEncryptionOTPKey(); // pad
    byte[] otpCipherText = oneTimePad(cipherText, otpText);
byte[] byteCounter = integerToBytes(counter - 1);
byte[] otpCounter = oneTimePad(byteCounter, otpText);
return concatByteArrays(otpCounter, otpCipherText);
}

/**
 * AES Decryption for Randomized Algorithm with Multiple Ciphers and Hash
 * and Varying Key
 * @param counterAndCipherText Counter and Ciphertext in a byte array
 * @return Plaintext in a byte array
 */
public byte[] randomKeyDecryption(byte[] counterAndCipherText) {
 byte[] otpText = createDecryptionOTPKey();
 byte[] otpCounter = extractFirst4Bytes(counterAndCipherText);
 otpCounter = oneTimePad(otpCounter, otpText);
 int extractedCounter = bytesToInteger(otpCounter);

 if (counter-1 == extractedCounter)
     selectedDecrypter = true;
 else
     selectedDecrypter = false;

 byte[] cipherText = removeFirst4Bytes(counterAndCipherText);
 byte[] newCipherText = oneTimePad(otpText, cipherText);
if (selectedDecrypter) {
    decrypterKey = key;
    return stdDecryption(newCipherText);
}
else {
    decrypterKey = otpText;
    rijndael.makeKey(decrypterKey, keyBits);
    byte[] plainText = new byte[16];
    rijndael.decrypt(newCipherText, plainText);

    return plainText;
}

/**
 * Generates a one time pad for the hashEncryption and randomKeyEncryption methods
 * One time pad is generated by a MD5 hash on the counter, randomly selected offset,
 * and the initialization vector
 * @return One Time Pad
 */
private byte[] createEncryptionOTPKey(){ // used to create the encryption onetimepad
    byte[] counterText = new byte[initVector.length+1];
    for (int i = 0; i < initVector.length; i++)
    {
        counterText[i] = initVector[i];
}
Randomly choose a decrypting cipher IV

Random r = new Random();
int cipherIndex = r.nextInt();
cipherIndex = Math.abs(cipherIndex);
cipherIndex = cipherIndex % NUM_CIPHERS;

selectedOffset = offsetTable[cipherIndex];
counterText[initVector.length] = new Integer(counter + selectedOffset).byteValue();
counter++;
return md5(counterText);
}

private byte[] integerToBytes(int value) {
byte[] output = new byte[4];
output[0]=(byte)((value & 0xff000000)>>24);
output[1]=(byte)((value & 0x00ff0000)>>16);
output[2]=(byte)((value & 0x0000ff00)>>8);
output[3]=(byte)((value & 0x000000ff));

return output;
}

/**
 * Generates a one time pad for the hashDecryption and randomKeyDecryption methods
* One time pad is generated by a MD5 hash on the counter, offset, and the
  * initialization vector
  * @return One Time Pad
  */

private byte[] createDecryptionOTPKey() { // used to create decryption onetimepad
    byte[] counterText = new byte[initVector.length+1];
    for (int i = 0; i < initVector.length; i++)
    {
        counterText[i] = initVector[i];
    }

    // Randomly choose a decrypting cipher IV
    counterText[initVector.length] = new Integer(counter + offset).byteValue();
    counter++;
    return md5(counterText);
}

/**
 * Converts a 4 Byte Array to an Unsigned Integer
 * @param bytes 4 byte array
 * @return integer
 */

private int bytesToInteger(byte[] bytes) {
    int output = ((bytes[0] & 0xFF) << 24)
        | ((bytes[1] & 0xFF) << 16)
        | ((bytes[2] & 0xFF) << 8)
        | ((bytes[3] & 0xFF) << 0);
| (bytes[3] & 0xFF);

return output;
}

/**
 * Concat two byte arrays
 * @param a Left Hand Side Byte Array
 * @param b Right Hand Side Byte Array
 * @return Concatenation of the two byte arrays
 */
private static byte[] concatByteArrays(byte[] a, byte[] b)
{
    byte[] output = new byte[a.length + b.length];

    for (int i = 0; i < a.length; i++)
        output[i] = a[i];

    for (int i = 0; i < b.length; i++)
        output[i+a.length] = b[i];

    return output;
}

/**
 * Returns the first four bytes from the byte array
 * @param array byte array
 * @return first four bytes from the byte array
 */
private static byte[] extractFirst4Bytes(byte[] array) {
    byte[] output = {array[0], array[1], array[2], array[3]};
    return output;
}

/**
 * Removes the first four bytes from the byte array
 * @param array byte array
 * @return byte array with the first four bytes removed
 */
private static byte[] removeFirst4Bytes(byte[] array) {
    byte[] output = new byte[array.length - 4];
    for (int i = 0; i < output.length; i++)
        output[i] = array[i+4];

    return output;
}

/**
 * Converts a string to a 16 byte array by padding with "0" if less
 * than 16 bytes and truncates it if over 16 bytes
 * @param str string to be converted
 * @return 16-byte array
 */
public static byte[] stringTo16ByteArray(String str) {
    if (str.length() >= 16)
        return str.substring(0, 16).getBytes();
else
{
    // Pad with 0
    for (int i = str.length(); i < 16; i++){
        str.concat("0");
    }

    return str.getBytes();
}
}

/**
 * Performs an XOR between two byte arrays for the length
 * of the first array
 * @param a first byte array
 * @param b second byte array
 * @return a byte array representing a^b
 */
private static byte[] oneTimePad(byte[] a, byte[] b) {
    byte[] otp = new byte[a.length];
    for (int i = 0; i < a.length; i++)
    {
        otp[i] = (byte) ((int) a[i] ^ (int) b[i]);
    }
    return otp;
}

/**
 * Performs an md5 hash on a given byte array

private static byte[] md5(byte[] content) {
    byte[] messageDigest = new byte[content.length];
    try {
        MessageDigest algorithm = MessageDigest.getInstance("MD5");
        algorithm.reset();
        algorithm.update(content);
        messageDigest = algorithm.digest();
    } catch (NoSuchAlgorithmException nsae) {
    }
    return messageDigest;
}
public class TestBench {
    private static final String INIT_VECTOR = "0123456789012345";
    private static final String KEY = "DefaultCipherKey";
    private static final int NUM_CIPHERS = 3; // Value must be between 1 to 10
    private static final int NUM_ITER = 100; // # of times to run test

    public static void main(String args[]) throws Exception
    {
        //testStandardAES(NUM_ITER);
        //testHashAES(NUM_ITER);
        testRandomKeyAES(NUM_ITER);

        //performanceTest(NUM_ITER);
    }

    /**
     * Test the Standard AES Encryption/Decryption
     * @param numIterations to be tested
     */
    private static void testStandardAES(int numIterations) {
        AES encrypter = new AES(KEY, INIT_VECTOR);
        AES decrypter = new AES(KEY, INIT_VECTOR);

        String plainText = "OriginalPlainTxt";
        for (int i = 0; i < numIterations; i++)
        {
        }
byte[] cipherText = encrypter.stdEncryption(AES.stringTo16ByteArray(plainText));
String output = new String(decrypter.stdDecryption(cipherText));

System.out.println("ENCRYPTION " + (i+1) + "\tPlaintext: " + plainText + "\tKey: " + new String(KEY) + "\tCiphertext: " + new String(cipherText));
System.out.println("DECRYPTION " + (i+1) + "\tCiphertext: " + new String(cipherText) + "\tKey: " + new String(KEY) + "\tPlaintext: " + output);
System.out.println();
}
}

/**
 * Test the AES Encryption/Decryption for the Randomized Algorithm
 * with Multiple Ciphers and Hash
 * @param numIterations to be tested
 */
private static void testHashAES(int numIterations) {
    AES encrypter = new AES(KEY, INIT_VECTOR, NUM_CIPHERS);
    AES decrypter = new AES(KEY, INIT_VECTOR, NUM_CIPHERS);
    String plainText = "OriginalPlainTxt";

    // testing counters
    int correctDecrypt = 0;
    int maxConsecutiveOccurences = 0;
    int consecutiveOccurences = 0;

    //
for (int i = 0; i < numIterations; i++)
{
    byte[] cipherText = encrypter.hashEncryption(AES.stringTo16ByteArray(plainText));
    String output = new String(decrypter.hashDecryption(cipherText));

    System.out.println("ENCRYPTION " + (i+1) + "\tPlaintext: " + plainText + "\tKey: " + new String(encrypter.key) + "\tCounter: " + encrypter.counter + "\tOffset: " + encrypter.selectedOffset + "\tCiphertext: " + new String(cipherText));
    System.out.println("DECRYPTION " + (i+1) + "\tCiphertext: " + new String(cipherText) + "\tKey: " + new String(decrypter.key) + "\tCounter: " + decrypter.counter + "\tOffset: " + decrypter.offset + "\tPlaintext: " + output);
    System.out.println();

    if (plainText.equals(output)) {
        correctDecrypt++;
        consecutiveOccurences++;
        if (consecutiveOccurences > maxConsecutiveOccurences)
            maxConsecutiveOccurences = consecutiveOccurences;
    } else {
        consecutiveOccurences = 0;
    }
}
System.out.println("Cipher 1 was chosen " + correctDecrypt + " out of " + NUM_ITER + " iterations.");
System.out.println("Cipher 1 was chosen at most " + maxConsecutiveOccurences + " times in a row.");
}

/**
 * Test the AES encryption/decryption for the Randomized Algorithm
 * with Multiple Ciphers and Hash and Varying Key
 * @param numIterations to be tested
 */
private static void testRandomKeyAES(int numIterations) {
    AES encrypter = new AES(KEY, INIT_VECTOR, NUM_CIPHERS);
    AES decrypter = new AES(KEY, INIT_VECTOR, NUM_CIPHERS);
    String plainText = "OriginalPlainTxt";

    // testing counters
    int correctDecrypt = 0;
    int maxConsecutiveOccurences = 0;
    int consecutiveOccurences = 0;

    for (int i = 0; i < numIterations; i++)
    {
        byte[] cipherText = encrypter.randomKeyEncryption(AES.stringTo16ByteArray(plainText));
        String output = new String(decrypter.randomKeyDecryption(cipherText));
System.out.println("ENCRYPTION " + (i+1) + " \tPlaintext: "+ plainText + " \tKey: " + new String(encrypter.key) +" \tCounter: " + encrypter.counter + " \tOffset: " + encrypter.selectedOffset + " \tCiphertext: " + new String(cipherText));

System.out.println("DECRYPTION " + (i+1) + " \tCiphertext: " + new String(cipherText) + " \tKey: " + new String(decrypter.decrypterKey) +" \tCounter: " + decrypter.counter + " \tOffset: " + decrypter.offset + " \tPlaintext: " + output);

System.out.println();

if (plainText.equals(output)) {
    correctDecrypt++;
    consecutiveOccurences++;
    if (consecutiveOccurences > maxConsecutiveOccurences)
        maxConsecutiveOccurences = consecutiveOccurences;
}
else {
    consecutiveOccurences = 0;
}
}

System.out.println("Cipher 1 was chosen " + correctDecrypt + " out of " + NUM_ITER + " iterations.");
System.out.println("Cipher 1 was chosen at most " + maxConsecutiveOccurences + " times in a row.");

55
}\

/**
* Runs the three encryption/decryption algorithms for the specified numbers of
* iterations to determine it's time/performance efficiency
* @param numIterations to be tested
*/
private static void performanceTest(int numIterations) {
    AES encrypter = new AES(KEY, INIT_VECTOR, NUM_CIPHERS);
    AES decrypter = new AES(KEY, INIT_VECTOR, NUM_CIPHERS);

    String plainText = "OriginalPlainTxt";
    byte[] cipherText = encrypter.stdEncryption(AES.stringTo16ByteArray(plainText));

    long currentTime = System.currentTimeMillis();
    for (int i = 0; i < numIterations; i++)
    {
        decrypter.stdDecryption(cipherText);
    }
    long elapsedTime1 = System.currentTimeMillis() - currentTime;

    byte[] cipherText3 = encrypter.randomKeyEncryption(AES.stringTo16ByteArray(plainText));
    currentTime = System.currentTimeMillis();
    for (int i = 0; i < numIterations; i++)
    {
        decrypter.randomKeyDecryption(cipherText3);
Long elapsedTime3 = System.currentTimeMillis() - currentTime;

byte[] cipherText2 = encrypter.hashEncryption(AES.stringTo16ByteArray(plainText));
currentTime = System.currentTimeMillis();
for (int i = 0; i < numIterations; i++)
{
    decrypter.hashDecryption(cipherText2);
}
Long elapsedTime2 = System.currentTimeMillis() - currentTime;
System.out.println("For " + numIterations + " iterations, ");
System.out.println("Standard Decryption took " + elapsedTime1 + " ms.");
System.out.println("Hash Decryption took " + elapsedTime2 + " ms.");
System.out.println("Random Key Decryption took " + elapsedTime3 + " ms.");
}
/**
 * Rijndael.java
 *
 * @version 1.0 (May 2001)
 *
 * Optimised Java implementation of the Rijndael (AES) block cipher.
 *
 * @author Paulo Barreto <paulo.barreto@terra.com.br>
 *
 * This software is hereby placed in the public domain.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
 * BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

public final class Rijndael {

    public Rijndael() {
    
    }
/**
 * Flag to setup the encryption key schedule.
 */
public static final int DIR_ENCRYPT = 1;

/**
 * Flag to setup the decryption key schedule.
 */
public static final int DIR_DECRYPT = 2;

/**
 * Flag to setup both key schedules (encryption/decryption).
 */
public static final int DIR_BOTH = (DIR_ENCRYPT | DIR_DECRYPT);

/**
 * AES block size in bits
 * (N.B. the Rijndael algorithm itself allows for other sizes).
 */
public static final int BLOCK_BITS = 128;

/**
 * AES block size in bytes
 * (N.B. the Rijndael algorithm itself allows for other sizes).
 */
public static final int BLOCK_SIZE = (BLOCK_BITS >>> 3);
* Substitution table (S-box).
 */

private static final String SS =
"\u637C\u777B\uF26B\u6FC5\u3001\u672B\uFED7\uAB76" +
"\uCA82\uC97D\uFA59\u47F0\uADD4\uA2AF\u9CA4\u72C0" +
"\uB7FD\u9326\u363F\uF7CC\u34A5\uE5F1\u71D8\u3115" +
"\u04C7\u23C3\u1896\u059A\u0712\u80E2\uEB27\uB275" +
"\u0983\u2C1A\u1B6E\u5AA0\u523B\uD6B3\u29E3\u2F84" +
"\u53D1\u00ED\u20FC\uB15B\u6ACB\uBE39\u4A4C\u58CF" +
"\uD0EF\uAAFB\u434D\u3385\u45F9\u027F\u503C\u9FA8" +
"\u51A3\u408F\u929D\u38F5\uBCB6\uDA21\u10FF\uF3D2" +
"\uCD0C\u13EC\u5F97\u4417\uC4A7\u7E3D\u645D\u1973" +
"\u6081\u4FDC\u222A\u9088\u46EE\uB814\uDE5E\u0BDB" +
"\uE032\u3A0A\u245C\uC2D3\uAC62\u9195\uE479" +
"\uE7C8\u376D\u8DD5\u4EA9\u6C56\uF4EA\u657A\uAE08" +
"\uBA78\u252E\u1CA6\uB4C6\uE8DD\u741F\u4BBD\u8B8A" +
"\u703E\uB566\u4803\uF60E\u6135\u57B9\u86C1\u1D9E" +
"\uE1F8\u9811\u69D9\u8E94\u9B1E\u78E9\uCE55\u28DF" +
"\u8CA1\u890D\uBFE6\u4268\u4199\u2D0F\uB054\uBB16";

private static final byte[]
  Se = new byte[256];

private static final int[]
  Te0 = new int[256],
  Te1 = new int[256],
  Te2 = new int[256],
  Te3 = new int[256];
private static final byte[]
        Sd = new byte[256];

private static final int[]
        Td0 = new int[256],
        Td1 = new int[256],
        Td2 = new int[256],
        Td3 = new int[256];

/**
 * Round constants
 */
private static final int[]
        rcon = new int[10]; /* for 128-bit blocks, Rijndael never uses more than 10 rcon values */

/**
 * Number of rounds (depends on key size).
 */
private int Nr = 0;

private int Nk = 0;

private int Nw = 0;

/**
 * Encryption key schedule
 */
private int rek[] = null;
/**
 * Decryption key schedule
 */

private int rdk[] = null;

static {
    /*
     * Te0[x] = Se[x].[02, 01, 01, 03];
     * Te1[x] = Se[x].[03, 02, 01, 01];
     * Te2[x] = Se[x].[01, 03, 02, 01];
     * Te3[x] = Se[x].[01, 01, 03, 02];
     * Td0[x] = Sd[x].[0e, 09, 0d, 0b];
     * Td1[x] = Sd[x].[0b, 0e, 09, 0d];
     * Td2[x] = Sd[x].[0d, 0b, 0e, 09];
     * Td3[x] = Sd[x].[09, 0d, 0b, 0e];
     */
    int ROOT = 0x11B;

    int s1, s2, s3, i1, i2, i4, i8, i9, ib, id, ie, t;
    for (i1 = 0; i1 < 256; i1++) {
        char c = SS.charAt(i1 >>> 1);
        s1 = (byte)((i1 & 1) == 0 ? c >>> 8 : c) & 0xff;
        s2 = s1 << 1;
        if (s2 >= 0x100) {
            s2 ^= ROOT;
        }
        s3 = s2 ^ sl;
        i2 = i1 << 1;
    }
}
if (i2 >= 0x100) {
    i2 ^= ROOT;
}

i4 = i2 << 1;
if (i4 >= 0x100) {
    i4 ^= ROOT;
}

i8 = i4 << 1;
if (i8 >= 0x100) {
    i8 ^= ROOT;
}

i9 = i8 ^ i1;
ib = i9 ^ i2;
id = i9 ^ i4;
ie = i8 ^ i4 ^ i2;

Se[i1] = (byte)s1;
Te0[i1] = t = (s2 << 24) | (s1 << 16) | (s1 << 8) | s3;
Te1[i1] = (t >>> 8) | (t << 24);
Te2[i1] = (t >>> 16) | (t << 16);
Te3[i1] = (t >>> 24) | (t << 8);

Sd[s1] = (byte)i1;
Td0[s1] = t = (ie << 24) | (i9 << 16) | (id << 8) | ib;
Td1[s1] = (t >>> 8) | (t << 24);
Td2[s1] = (t >>> 16) | (t << 16);
Td3[s1] = (t >>> 24) | (t << 8);
} /*
* round constants

```java
int r = 1;
rcon[0] = r << 24;
for (int i = 1; i < 10; i++) {
    r <<= 1;
    if (r >= 0x100) {
        r ^= ROOT;
    }
    rcon[i] = r << 24;
}
```

/**
 * Expand a cipher key into a full encryption key schedule.
 * 
 * @param cipherKey the cipher key (128, 192, or 256 bits).
 */
private void expandKey(byte[] cipherKey) {
    int temp, r = 0;
    for (int i = 0, k = 0; i < Nk; i++, k += 4) {
        rek[i] =
            ((cipherKey[k    ]    ) << 24) |
            ((cipherKey[k + 1] & 0xff) << 16) |
            ((cipherKey[k + 2] & 0xff) <<  8) |
            ((cipherKey[k + 3] & 0xff));
    }
    for (int i = Nk, n = 0; i < Nw; i++, n--) {
        temp = rek[i - 1];
```
if (n == 0) {
    n = Nk;
    temp =
    ((Se[(temp >>> 16) & 0xff] ) << 24) |
    ((Se[(temp >>> 8) & 0xff] & 0xff) << 16) |
    ((Se[(temp ) & 0xff] & 0xff) << 8) |
    ((Se[(temp >>> 24) ] & 0xff));
    temp ^= rcon[r++];
} else if (Nk == 8 && n == 4) {
    temp =
    ((Se[(temp >>> 24) ] ) << 24) |
    ((Se[(temp >>> 16) & 0xff] & 0xff) << 16) |
    ((Se[(temp >>> 8) & 0xff] & 0xff) << 8) |
    ((Se[(temp ) & 0xff] & 0xff));
    rek[i] = rek[i - Nk] ^ temp;
}
 temp = 0;
}

/*
 * Faster implementation of the key expansion
 * (only worthwhile in Rijndael is used in a hashing function mode).
 */

private void expandKey(byte[] cipherKey) {
    int keyOffset = 0;
    int i = 0;
int temp;

rek[0] =

(cipherKey[ 0] ) << 24 |
(cipherKey[ 1] & 0xff) << 16 |
(cipherKey[ 2] & 0xff) << 8 |
(cipherKey[ 3] & 0xff);

rek[1] =

(cipherKey[ 4] ) << 24 |
(cipherKey[ 5] & 0xff) << 16 |
(cipherKey[ 6] & 0xff) << 8 |
(cipherKey[ 7] & 0xff);

rek[2] =

(cipherKey[ 8] ) << 24 |
(cipherKey[ 9] & 0xff) << 16 |
(cipherKey[10] & 0xff) << 8 |
(cipherKey[11] & 0xff);

rek[3] =

(cipherKey[12] ) << 24 |
(cipherKey[13] & 0xff) << 16 |
(cipherKey[14] & 0xff) << 8 |
(cipherKey[15] & 0xff);

if (Nk == 4) {
    for (;;) {
        temp = rek[keyOffset + 3];
        rek[keyOffset + 4] = rek[keyOffset] ^
            ((Se[ (temp >>> 16) & 0xff ] ) << 24) ^
            ((Se[ (temp >>> 8) & 0xff & 0xff ] << 16) ^
            ((Se[ (temp ) & 0xff & 0xff ] << 8) ^
            (Se[ (temp ) & 0xff & 0xff ] << 8));
    }
}
((Se[(temp >>> 24) ] & 0xff) ) ^

rcon[i];
rek[keyOffset + 5] = rek[keyOffset + 1] ^
rek[keyOffset + 4];
rek[keyOffset + 6] = rek[keyOffset + 2] ^
rek[keyOffset + 5];
rek[keyOffset + 7] = rek[keyOffset + 3] ^
rek[keyOffset + 6];
if (++i == 10) {
    return;
}
keyOffset += 4;
}
}
rek[keyOffset + 4] =
(cipherKey[16] ) << 24 |
(cipherKey[17] & 0xff) << 16 |
(cipherKey[18] & 0xff) << 8 |
(cipherKey[19] & 0xff);
rek[keyOffset + 5] =
(cipherKey[20] ) << 24 |
(cipherKey[21] & 0xff) << 16 |
(cipherKey[22] & 0xff) << 8 |
(cipherKey[23] & 0xff);
if (Nk == 6) {
    for (;;) {
        temp = rek[keyOffset + 5];
        rek[keyOffset + 6] = rek[keyOffset] ^
((Se[(temp >>> 16) & 0xff] ) << 24) ^
((Se[(temp >>> 8) & 0xff] & 0xff) << 16) ^
((Se[(temp ) & 0xff] & 0xff) << 8) ^
((Se[(temp >>> 24)] & 0xff) ) ^

rcon[i];
rek[keyOffset + 7] = rek[keyOffset + 1] ^ 
rek[keyOffset + 6];
rek[keyOffset + 8] = rek[keyOffset + 2] ^ 
rek[keyOffset + 7];
rek[keyOffset + 9] = rek[keyOffset + 3] ^ 
rek[keyOffset + 8];
if (++i == 8) {
    return;
}
rek[keyOffset + 10] = rek[keyOffset + 4] ^ 
rek[keyOffset + 9];
rek[keyOffset + 11] = rek[keyOffset + 5] ^
rek[keyOffset + 10];
    keyOffset += 6;
}
)
}
rek[keyOffset + 6] =
(cipherKey[24] ) << 24 |
(cipherKey[25] & 0xff) << 16 |
(cipherKey[26] & 0xff) << 8 |
(cipherKey[27] & 0xff);
rek[keyOffset + 7] =
(cipherKey[28] ) << 24 |
(cipherKey[29] & 0xff) << 16 |
(cipherKey[30] & 0xff) << 8 |
(cipherKey[31] & 0xff);

if (Nk == 8) {
    for (;;) {
        temp = rek[keyOffset + 7];
        rek[keyOffset + 8] = rek[keyOffset] ^
            ((Se[(temp >>> 16) & 0xff] ) << 24) ^
            ((Se[(temp >>> 8) & 0xff] & 0xff) << 16) ^
            ((Se[(temp ) & 0xff] & 0xff) << 8) ^
            ((Se[(temp >>> 24) ] & 0xff) ) ^
            rcon[i];
        rek[keyOffset + 9] = rek[keyOffset + 1] ^
            rek[keyOffset + 8];
        rek[keyOffset + 10] = rek[keyOffset + 2] ^
            rek[keyOffset + 9];
        rek[keyOffset + 11] = rek[keyOffset + 3] ^
            rek[keyOffset + 10];
        if (++i == 7) {
            return;
        }
        temp = rek[keyOffset + 11];
        rek[keyOffset + 12] = rek[keyOffset + 4] ^
            ((Se[(temp >>> 24) ] ) ) << 24) ^
            ((Se[(temp >>> 16) & 0xff] & 0xff) << 16) ^
            ((Se[(temp >>> 8) & 0xff] & 0xff) << 8) ^
            ((Se[(temp ) & 0xff] & 0xff));
        rek[keyOffset + 13] = rek[keyOffset + 5] ^
            rek[keyOffset + 12];
        rek[keyOffset + 14] = rek[keyOffset + 6] ^
            rek[keyOffset + 13];
    }
}
rek[keyOffset + 15] = rek[keyOffset + 7] ^
rek[keyOffset + 14];

    keyOffset += 8;

}
}
}
*/

/**
 * Compute the decryption schedule from the encryption schedule .
 */
private void invertKey() {
    int d = 0, e = 4*Nr, w;
    /*
    * apply the inverse MixColumn transform to all round keys
    * but the first and the last:
    */
    rdk[d  ] = rek[e  ];
    rdk[d + 1] = rek[e + 1];
    rdk[d + 2] = rek[e + 2];
    rdk[d + 3] = rek[e + 3];
    d += 4;
    e -= 4;
    for (int r = 1; r < Nr; r++) {
        w = rek[e  ];
        rdk[d  ] =
            Td0[Se[(w >>> 24)       ] & 0xff] ^
            Td1[Se[(w >>> 16) & 0xff] & 0xff] ^
            Td2[Se[(w >>>  8) & 0xff] & 0xff] ^
            Td3[Se[(w <<  8) & 0xff] & 0xff] ^
Td3[Se[w   ] & 0xff] & 0xff];
w = rek[e + 1];
rdk[d + 1] =
    Td0[Se[w >>> 24] ] & 0xff] ^
    Td1[Se[w >>> 16] & 0xff] & 0xff] ^
    Td2[Se[w >>>  8] & 0xff] & 0xff] ^
    Td3[Se[w   ] & 0xff] & 0xff];
w = rek[e + 2];
rdk[d + 2] =
    Td0[Se[w >>> 24] ] & 0xff] ^
    Td1[Se[w >>> 16] & 0xff] & 0xff] ^
    Td2[Se[w >>>  8] & 0xff] & 0xff] ^
    Td3[Se[w   ] & 0xff] & 0xff];
w = rek[e + 3];
rdk[d + 3] =
    Td0[Se[w >>> 24] ] & 0xff] ^
    Td1[Se[w >>> 16] & 0xff] & 0xff] ^
    Td2[Se[w >>>  8] & 0xff] & 0xff] ^
    Td3[Se[w   ] & 0xff] & 0xff];
d += 4;
e -= 4;
}
rdk[d   ] = rek[e   ];
rdk[d + 1] = rek[e + 1];
rdk[d + 2] = rek[e + 2];
rdk[d + 3] = rek[e + 3];
}/***/
* Setup the AES key schedule for encryption, decryption, or both.
*
* @param cipherKey the cipher key (128, 192, or 256 bits).
* @param keyBits size of the cipher key in bits.
* @param direction cipher direction (DIR_ENCRYPT, DIR_DECRYPT, or DIR_BOTH).
*
public void makeKey(byte[] cipherKey, int keyBits, int direction)
        throws RuntimeException {
    // check key size:
    if (keyBits != 128 && keyBits != 192 && keyBits != 256) {
        throw new RuntimeException("Invalid AES key size (" +
                keyBits + " bits")");
    }
    Nk = keyBits >>> 5;
    Nr = Nk + 6;
    Nw = 4 * (Nr + 1);
    rek = new int[Nw];
    rdk = new int[Nw];
    if ((direction & DIR_BOTH) != 0) {
        expandKey(cipherKey);
        /*
         * for (int r = 0; r <= Nr; r++) {
         *     System.out.println("RK" + r + "=");
         *     for (int i = 0; i < 4; i++) {
         *         int w = rek[4*r + i];
         *         System.out.println(" " + Integer.toHexString(w));
         *     }
         */
        System.out.println();
    }
} */

if ((direction & DIR_DECRYPT) != 0) {
    invertKey();
}
}
}

/**
 * Setup the AES key schedule (any cipher direction).
 *
 * @param cipherKey the cipher key (128, 192, or 256 bits).
 * @param keyBits size of the cipher key in bits.
 */
public void makeKey(byte[] cipherKey, int keyBits) throws RuntimeException {
    makeKey(cipherKey, keyBits, DIR_BOTH);
}

/**
 * Encrypt exactly one block (BLOCK_SIZE bytes) of plaintext.
 *
 * @param pt plaintext block.
 * @param ct ciphertext block.
 */
public void encrypt(byte[] pt, byte[] ct) {
    /*
     * map byte array block to cipher state
     * and add initial round key:
int k = 0, v;

int t0 = ((pt[0]    ) << 24 |
    (pt[1] & 0xff) << 16 |
    (pt[2] & 0xff) << 8 |
    (pt[3] & 0xff)      ) ^ rek[0];

int t1 = ((pt[4]    ) << 24 |
    (pt[5] & 0xff) << 16 |
    (pt[6] & 0xff) << 8 |
    (pt[7] & 0xff)      ) ^ rek[1];

int t2 = ((pt[8]    ) << 24 |
    (pt[9] & 0xff) << 16 |
    (pt[10] & 0xff) << 8 |

int t3 = ((pt[12]   ) << 24 |
    (pt[13] & 0xff) << 16 |
    (pt[14] & 0xff) << 8 |
    (pt[15] & 0xff)     ) ^ rek[3];

/*
 * Nr - 1 full rounds:
 */

for (int r = 1; r < Nr; r++) {
    k += 4;

    int a0 =
        Te0[(t0 >>> 24)    ] ^
        Te1[(t1 >>> 16) & 0xff] ^
        Te2[(t2 >>> 8) & 0xff] ^
        Te3[(t3    ) & 0xff] ^
    rek[k    ];
}
int a1 =
    Te0[(t1 >>> 24) ^ Te1[(t2 >>> 16) & 0xff] ^ Te2[(t3 >>> 8) & 0xff] ^ Te3[(t0) & 0xff]]
    rek[k + 1];

int a2 =
    Te0[(t2 >>> 24) ^ Te1[(t3 >>> 16) & 0xff] ^ Te2[(t0 >>> 8) & 0xff] ^ Te3[(t1) & 0xff]]
    rek[k + 2];

int a3 =
    Te0[(t3 >>> 24) ^ Te1[(t0 >>> 16) & 0xff] ^ Te2[(t1 >>> 8) & 0xff] ^ Te3[(t2) & 0xff]]
    rek[k + 3];

t0 = a0; t1 = a1; t2 = a2; t3 = a3;
}

/*
 * last round lacks MixColumn:
 */

k += 4;

v = rek[k ];

c[t[ 0] = (byte)(Se[(t0 >>> 24) ] ^ (v >>> 24));
c[t[ 1] = (byte)(Se[(t1 >>> 16) & 0xff] ^ (v >>> 16));
c[t[ 2] = (byte)(Se[(t2 >>> 8) & 0xff] ^ (v >>> 8));
ct[ 3] = (byte)(Se[(t3 ) & 0xff] ^ (v ));

v = rek[k + 1];
ct[ 4] = (byte)(Se[(t1 >>> 24) ] ^ (v >>> 24));
ct[ 5] = (byte)(Se[(t2 >>> 16) & 0xff] ^ (v >>> 16));
ct[ 6] = (byte)(Se[(t3 >>> 8) & 0xff] ^ (v >>> 8));
ct[ 7] = (byte)(Se[(t0 ) & 0xff] ^ (v ));

v = rek[k + 2];
ct[ 8] = (byte)(Se[(t2 >>> 24) ] ^ (v >>> 24));
ct[ 9] = (byte)(Se[(t3 >>> 16) & 0xff] ^ (v >>> 16));
ct[10] = (byte)(Se[(t0 >>> 8) & 0xff] ^ (v >>> 8));
ct[11] = (byte)(Se[(t1 ) & 0xff] ^ (v ));

v = rek[k + 3];
ct[12] = (byte)(Se[(t3 >>> 24) ] ^ (v >>> 24));
ct[13] = (byte)(Se[(t0 >>> 16) & 0xff] ^ (v >>> 16));
ct[14] = (byte)(Se[(t1 >>> 8) & 0xff] ^ (v >>> 8));
ct[15] = (byte)(Se[(t2 ) & 0xff] ^ (v ));
}

/**  
 * Decrypt exactly one block (BLOCK_SIZE bytes) of ciphertext.  
 *  
 * @param ct ciphertext block.  
 * @param pt plaintext block.  
 */
public void decrypt(byte[] ct, byte[] pt) {
/* map byte array block to cipher state
 * and add initial round key:
 */

int k = 0, v;

int t0 = ((ct[ 0] ) <<< 24 |
          (ct[ 1] & 0xff) <<< 16 |
          (ct[ 2] & 0xff) <<< 8 |
          (ct[ 3] & 0xff) ) ^ rdk[0];

int t1 = ((ct[ 4] ) <<< 24 |
          (ct[ 5] & 0xff) <<< 16 |
          (ct[ 6] & 0xff) <<< 8 |
          (ct[ 7] & 0xff) ) ^ rdk[1];

int t2 = ((ct[ 8] ) <<< 24 |
          (ct[ 9] & 0xff) <<< 16 |
          (ct[10] & 0xff) <<< 8 |

int t3 = ((ct[12] ) <<< 24 |
          (ct[13] & 0xff) <<< 16 |
          (ct[14] & 0xff) <<< 8 |
          (ct[15] & 0xff) ) ^ rdk[3];

/*
 * Nr - 1 full rounds:
 */

for (int r = 1; r < Nr; r++) {
    k += 4;

    int a0 =
        Td0[(t0 >>> 24) ] ^
        Td1[(t3 >>> 16) & 0xff] ^
        Td2[(t2 >>> 8) & 0xff] ^
Td3[(t1     ) & 0xff] ^
rdk[k    ];

int a1 =
    Td0[(t1 >>> 24)   ] ^
    Td1[(t0 >>> 16) & 0xff] ^
    Td2[(t3 >>>  8) & 0xff] ^
    Td3[(t2     ) & 0xff] ^
    rdk[k + 1];

int a2 =
    Td0[(t2 >>> 24)   ] ^
    Td1[(t1 >>> 16) & 0xff] ^
    Td2[(t0 >>>  8) & 0xff] ^
    Td3[(t3     ) & 0xff] ^
    rdk[k + 2];

int a3 =
    Td0[(t3 >>> 24)   ] ^
    Td1[(t2 >>> 16) & 0xff] ^
    Td2[(t1 >>>  8) & 0xff] ^
    Td3[(t0     ) & 0xff] ^
    rdk[k + 3];

t0 = a0; t1 = a1; t2 = a2; t3 = a3;
}
/*
 * last round lacks MixColumn:
 */
k += 4;

v = rdk[k    ];
pt[ 0] = (byte)(Sd[(t0 >>> 24) ] ^ (v >>> 24));
pt[ 1] = (byte)(Sd[(t3 >>> 16) & 0xff] ^ (v >>> 16));
pt[ 2] = (byte)(Sd[(t2 >>> 8) & 0xff] ^ (v >>> 8));
pt[ 3] = (byte)(Sd[(t1  ) & 0xff] ^ (v    ));

v = rdk[k + 1];
pt[ 4] = (byte)(Sd[(t1 >>> 24) ] ^ (v >>> 24));
pt[ 5] = (byte)(Sd[(t0 >>> 16) & 0xff] ^ (v >>> 16));
pt[ 6] = (byte)(Sd[(t3 >>> 8) & 0xff] ^ (v >>> 8));
pt[ 7] = (byte)(Sd[(t2   ) & 0xff] ^ (v    ));

v = rdk[k + 2];
pt[ 8] = (byte)(Sd[(t2 >>> 24) ] ^ (v >>> 24));
pt[ 9] = (byte)(Sd[(t1 >>> 16) & 0xff] ^ (v >>> 16));
pt[10] = (byte)(Sd[(t0 >>> 8) & 0xff] ^ (v >>> 8));
pt[11] = (byte)(Sd[(t3   ) & 0xff] ^ (v    ));

v = rdk[k + 3];
pt[12] = (byte)(Sd[(t3 >>> 24) ] ^ (v >>> 24));
pt[13] = (byte)(Sd[(t2 >>> 16) & 0xff] ^ (v >>> 16));
pt[14] = (byte)(Sd[(t1 >>> 8) & 0xff] ^ (v >>> 8));
pt[15] = (byte)(Sd[(t0   ) & 0xff] ^ (v    ));
}

/**
 * Destroy all sensitive information in this object.
 */

protected final void finalize() {
    if (rek != null) {
        for (int i = 0; i < rek.length; i++) {
            pt[ 1] = (byte)(Sd[(t3 >>> 16) & 0xff] ^ (v >>> 16));
            pt[ 2] = (byte)(Sd[(t2 >>> 8) & 0xff] ^ (v >>> 8));
            pt[ 3] = (byte)(Sd[(t1  ) & 0xff] ^ (v    ));

            v = rdk[k + 1];
            pt[ 4] = (byte)(Sd[(t1 >>> 24) ] ^ (v >>> 24));
            pt[ 5] = (byte)(Sd[(t0 >>> 16) & 0xff] ^ (v >>> 16));
            pt[ 6] = (byte)(Sd[(t3 >>> 8) & 0xff] ^ (v >>> 8));
            pt[ 7] = (byte)(Sd[(t2   ) & 0xff] ^ (v    ));

            v = rdk[k + 2];
            pt[ 8] = (byte)(Sd[(t2 >>> 24) ] ^ (v >>> 24));
            pt[ 9] = (byte)(Sd[(t1 >>> 16) & 0xff] ^ (v >>> 16));
            pt[10] = (byte)(Sd[(t0 >>> 8) & 0xff] ^ (v >>> 8));
            pt[11] = (byte)(Sd[(t3   ) & 0xff] ^ (v    ));

            v = rdk[k + 3];
            pt[12] = (byte)(Sd[(t3 >>> 24) ] ^ (v >>> 24));
            pt[13] = (byte)(Sd[(t2 >>> 16) & 0xff] ^ (v >>> 16));
            pt[14] = (byte)(Sd[(t1 >>> 8) & 0xff] ^ (v >>> 8));
            pt[15] = (byte)(Sd[(t0   ) & 0xff] ^ (v    ));
    }
}
rek[i] = 0;
}
rek = null;
}
if (rdk != null) {
    for (int i = 0; i < rdk.length; i++) {
        rdk[i] = 0;
    }
    rdk = null;
}
}
Appendix B: Select Output

Standard AES Algorithm for 10 iterations

ENCRYPTION 1  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
               Ciphertext: OÜĤĐÔ縠Ĥ%/âEbf

DECRIPTION 1 Ciphertext: OÜĤĐÔ縠Ĥ%/âEbf  Key: DefaultCipherKey
               Plaintext: OriginalPlainTxt

ENCRYPTION 2  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
               Ciphertext: OÜĤĐÔ縠Ĥ%/âEbf

DECRIPTION 2 Ciphertext: OÜĤĐÔ縠Ĥ%/âEbf  Key: DefaultCipherKey
               Plaintext: OriginalPlainTxt

ENCRYPTION 3  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
               Ciphertext: OÜĤĐÔ縠Ĥ%/âEbf

DECRIPTION 3 Ciphertext: OÜĤĐÔ縠Ĥ%/âEbf  Key: DefaultCipherKey
               Plaintext: OriginalPlainTxt

ENCRYPTION 4  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
               Ciphertext: OÜĤĐÔ縠Ĥ%/âEbf

DECRIPTION 4 Ciphertext: OÜĤĐÔ縠Ĥ%/âEbf  Key: DefaultCipherKey
               Plaintext: OriginalPlainTxt

ENCRYPTION 5  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
               Ciphertext: OÜĤĐÔ縠Ĥ%/âEbf

DECRIPTION 5 Ciphertext: OÜĤĐÔ縠Ĥ%/âEbf  Key: DefaultCipherKey
               Plaintext: OriginalPlainTxt
ENCRYPTION 6  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Ciphertext: O çıkmaٍٜٝٞٙ١٤٢٣٤٥
DECRYPTION 6  Ciphertext: Oствиеٍٜٝٞٙ١٤٢٣٤٥ Key: DefaultCipherKey
Plaintext: OriginalPlainTxt

ENCRYPTION 7  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Ciphertext: Oствиеٍٜٝٞٙ١٤٢٣٤٥
DECRYPTION 7  Ciphertext: Oствиеٍٜٝٞٙ١٤٢٣٤٥ Key: DefaultCipherKey
Plaintext: OriginalPlainTxt

ENCRYPTION 8  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Ciphertext: Oствиеٍٜٝٞٙ١٤٢٣٤٥
DECRYPTION 8  Ciphertext: Oствиеٍٜٝٞٙ١٤٢٣٤٥ Key: DefaultCipherKey
Plaintext: OriginalPlainTxt

ENCRYPTION 9  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Ciphertext: Oствиеٍٜٝٞٙ١٤٢٣٤٥
DECRYPTION 9  Ciphertext: Oствиеٍٜٝٞٙ١٤٢٣٤٥ Key: DefaultCipherKey
Plaintext: OriginalPlainTxt

ENCRYPTION 10  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Ciphertext: Oствиеٍٜٝٞٙ١٤٢٣٤٥
DECRYPTION 10  Ciphertext: Oствиеٍٜٝٞٙ١٤٢٣٤٥ Key: DefaultCipherKey
Plaintext: OriginalPlainTxt
Randomized Algorithm with Multiple Ciphers and Hash

Ran for 100 iterations with 3 decrypting ciphers

ENCRYPTION 1  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 1  Offset: 123  Ciphertext: "ëJŠAYíIlfÚEN-É×I-k

DECRYPTION 1  Ciphertext: "ëJŠAYíIlfÚEN-É×I-k  Key: DefaultCipherKey
Counter: 1  Offset: 12  Plaintext: 3?©ÖU7,ò'"b,ywV

ENCRYPTION 2  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 2  Offset: 12  Ciphertext: ŃûzI×'ü qÊÔÃ€9ïIà)

DECRYPTION 2  Ciphertext: ŃûzI×'ü qÊÔÃ€9ïIà)  Key: DefaultCipherKey
Counter: 2  Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 3  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 3  Offset: 123  Ciphertext: Ñ¥fpZyà:éügýý-KôG[0,M

DECRYPTION 3  Ciphertext: Ñ¥fpZyà:éügýý-KôG[0,M  Key: DefaultCipherKey
Counter: 3  Offset: 12  Plaintext: £ºèìÄ"ìÅ'ïåèâO

ENCRYPTION 4  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 4  Offset: 123  Ciphertext: C,FdÀA.ÁøswÉÔ'Ì•ì[}

DECRYPTION 4  Ciphertext: C,FdÀA.ÁøswÉÔ'Ì•ì[  Key: DefaultCipherKey
Counter: 4  Offset: 12  Plaintext: J¿ß=¿~u3ÂI<-4êÓ

ENCRYPTION 5  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 5  Offset: 123  Ciphertext: Ô6ÝM °'¿i°yêëH#ÔE

DECRYPTION 5  Ciphertext: Ô6ÝM °'¿i°yêëH#ÔE  Key: DefaultCipherKey
Counter: 5  Offset: 12  Plaintext: Ôë±-ëîÝ=²LØ
ENCRYPTION 6  Plaintext: OriginalPlainText  Key: DefaultCipherKey
Counter: 6  Offset: 12  Ciphertext: 〈ïvåÄ¬ø¬RïëMT+.〉<ù
DECRYPTION 6  Ciphertext: 〈ïvåÄ¬ø¬RïëMT+.〉<ù  Key: DefaultCipherKey
Counter: 6  Offset: 12  Plaintext: OriginalPlainText

ENCRYPTION 7  Plaintext: OriginalPlainText  Key: DefaultCipherKey
Counter: 7  Offset: 124  Ciphertext: ø¬éP$)§»Ä'³?©U|cÀ
DECRYPTION 7  Ciphertext: ø¬éP$)§»Ä'³?©U|cÀ  Key: DefaultCipherKey
Counter: 7  Offset: 12  Plaintext: XCM[u<^^"i^}(İB

ENCRYPTION 8  Plaintext: OriginalPlainText  Key: DefaultCipherKey
Counter: 8  Offset: 124  Ciphertext: ;G†i> êa³¬ÖåAbÅÔ
DECRYPTION 8  Ciphertext: ;G†i> êa³¬ÖåAbÅÔ  Key: DefaultCipherKey
Counter: 8  Offset: 12  Plaintext: -!|$v¬yU02@Äoça

ENCRYPTION 9  Plaintext: OriginalPlainText  Key: DefaultCipherKey
Counter: 9  Offset: 124  Ciphertext: ‹Î†yû ½x'ı'4Yû:š¬
DECRYPTION 9  Ciphertext: ‹Î†yû ½x'ı'4Yû:š¬  Key: DefaultCipherKey
Counter: 9  Offset: 12  Plaintext: x  "a6ÄÍJšÄ&T|©

ENCRYPTION 10  Plaintext: OriginalPlainText  Key: DefaultCipherKey
Counter: 10  Offset: 12  Ciphertext: <È®sõίèëk5"" конструкци
DECRYPTION 10  Ciphertext: <È®sõίèëk5"" конструкция  Key: DefaultCipherKey
Counter: 10  Offset: 12  Plaintext: OriginalPlainText

ENCRYPTION 11  Plaintext: OriginalPlainText  Key: DefaultCipherKey
Counter: 11  Offset: 123  Ciphertext: -7“BKêé’|üacZ’³g+çı
DECRYPTION 11  Ciphertext: -7¨BK±ê'´|üâçž³°+¿ Key: DefaultCipherKey Counter: 11 Offset: 12 Plaintext: Zoö.vpÜObebÁ+Ý

ENCRYPTION 12  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey Counter: 12 Offset: 12 Ciphertext: 8k;Íw·'âY*:z|§Z¡Oo

DECRYPTION 12  Ciphertext: 8k;Íw·'âY*:z|§Z¡Oo Key: DefaultCipherKey Counter: 12 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 13  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey Counter: 13 Offset: 12 Ciphertext: ÊŠêè"©•·I ø2øUêDoÑ

DECRYPTION 13  Ciphertext: ÊŠêè"©•·I ø2øUêDoÑ Key: DefaultCipherKey Counter: 13 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 14  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey Counter: 14 Offset: 123 Ciphertext: h%0B'aÅ}="mM`~VÜ¬Mµ7

DECRYPTION 14  Ciphertext: h%0B'aÅ}="mM`~VÜ¬Mµ7 Key: DefaultCipherKey Counter: 14 Offset: 12 Plaintext: ⁠ë¿ÆÇëvÔ°pÄ«kß

ENCRYPTION 15  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey Counter: 15 Offset: 124 Ciphertext: E9è``%åñ1,0e0I+ ðí;:Â

DECRYPTION 15  Ciphertext: E9è``%åñ1,0e0I+ ðí;:Â Key: DefaultCipherKey Counter: 15 Offset: 12 Plaintext: b*ID@X²×S§ùU¬,-

ENCRYPTION 16  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey Counter: 16 Offset: 124 Ciphertext: ø8çór%Uj¥rXE8± wand°fc

DECRYPTION 16  Ciphertext: ø8çór%Uj¥rXE8± wand°fc Key: DefaultCipherKey Counter: 16 Offset: 12 Plaintext: ðA7BØ;rfè}×Lz•
ENCRYPTION 17  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 17  Offset: 123  Ciphertext: ॰©8ôrWyjôXEô8±âV°bC

DECRYPTION 17  Ciphertext: ॰©8ôrWyjôXEô8±âV°bC  Key: DefaultCipherKey
Counter: 17  Offset: 12  Plaintext: ᴢ×× fetisch /R6¬±6AÅdı

ENCRYPTION 18  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 18  Offset: 12  Ciphertext: µtàEóu`สิ่งแวดล–ŌôiëùE15»ø`Ô

DECRYPTION 18  Ciphertext: µtàEóu`–ŌôiëùE15»ø`Ô  Key: DefaultCipherKey
Counter: 18  Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 19  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 19  Offset: 124  Ciphertext: \tœ¨›Æ{(eRª= Z?—Àp

DECRYPTION 19  Ciphertext: \tœ¨›Æ{(eRª=Z?—Àp  Key: DefaultCipherKey
Counter: 19  Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 20  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 20  Offset: 12  Ciphertext: ^ÂØ_2^äå«eý ?t"

DECRYPTION 20  Ciphertext: ^ÂØ_2^äå«eý ?t"  Key: DefaultCipherKey
Counter: 20  Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 21  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 21  Offset: 12  Ciphertext: D’×M®-6¬Iâ+’Éå¬0p

DECRYPTION 21  Ciphertext: D’×M®-6¬Iâ+’Éå¬0p  Key: DefaultCipherKey
Counter: 21  Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 22  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 22  Offset: 123  Ciphertext: ’`ÖYT=†%Å;?–Ô¿â®«*

DECRYPTION 22  Ciphertext: ’`ÖYT=†%Å;?–Ô¿â®«*  Key: DefaultCipherKey
Counter: 22  Offset: 12  Plaintext: ãÈ^%bagai’”)å|B
ENCRYPTION 23
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 23 Offset: 123 Ciphertext: þa†;½ ýžö*'Åöša„«1'
DECRYPTION 23
Ciphertext: þa†;½ ýžö*'Åöša„«1'
Key:
DefaultCipherKey Counter: 23 Offset: 12 Plaintext: Ōi¬yÅL=å?5`U,1"O

ENCRYPTION 24
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 24 Offset: 124 Ciphertext: !š nØ=.;3*BÖR^1K.ca
DECRYPTION 24
Ciphertext: !š nØ=.;3*BÖR^1K.ca
Key:
DefaultCipherKey Counter: 24 Offset: 12 Plaintext: U¬·ôS6-¢öwe

ENCRYPTION 25
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 25 Offset: 124 Ciphertext: ¬ô×1'g$8Œ%r®";d
DECRYPTION 25
Ciphertext: ¬ô×1'g$8Œ%r®";d
Key:
DefaultCipherKey Counter: 25 Offset: 12 Plaintext: ©Ωáµ§,ái,w,GAã

ENCRYPTION 26
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 26 Offset: 12 Ciphertext: ?FD¶pšÂçº"À‾:++−^UgÉ
DECRYPTION 26
Ciphertext: ?FD¶pšÂçº"À‾:++−^UgÉ
Key:
DefaultCipherKey Counter: 26 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 27
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 27 Offset: 124 Ciphertext: "ØBÜVW?£p.Ž(YŽ·q
DECRYPTION 27
Ciphertext: "ØBÜVW?£p.Ž(YŽ·q
Key:
DefaultCipherKey Counter: 27 Offset: 12 Plaintext: ©E,â§†δÈrXW¬ý·å

ENCRYPTION 28
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 28 Offset: 124 Ciphertext: ?ADuQX-X©”å DTÔ
ENCRYPTION 34  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
  Counter: 34  Offset: 124  Ciphertext: ITÖY;˘TÍY!i‰lkyë;iO  
DECRIPTION 34  Ciphertext: ITÖY;˘TÍY!i‰lkyë;iO  Key:  
DefaultCipherKey  Counter: 34  Offset: 12  Plaintext: dJ/þó^Ä&«lëÜÔ;  

ENCRYPTION 35  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
  Counter: 35  Offset: 124  Ciphertext: ?*îpöii"^t05âk·  
DECRIPTION 35  Ciphertext: ?*îpöii"^t05âk·  Key:  
DefaultCipherKey  Counter: 35  Offset: 12  Plaintext: ^pHÊmâGgæ+l×YÈçê  

ENCRYPTION 36  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
  Counter: 36  Offset: 12  Ciphertext: ©IpœöwyiFà800û-`s  
DECRIPTION 36  Ciphertext: ©IpœöwyiFà800û-`s  Key:  
DefaultCipherKey  Counter: 36  Offset: 12  Plaintext: OriginalPlainTxt  

ENCRYPTION 37  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
  Counter: 37  Offset: 12  Ciphertext: Ž^-âNÅO?ÝBêy Ñ#H©ô  
DECRIPTION 37  Ciphertext: Ž^-âNÅO?ÝBêy Ñ#H©ô  Key:  
DefaultCipherKey  Counter: 37  Offset: 12  Plaintext: OriginalPlainTxt  

ENCRYPTION 38  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
  Counter: 38  Offset: 12  Ciphertext: r"^ö=ûvz³PS³~'¼(Ê`Ø  
DECRIPTION 38  Ciphertext: r"^ö=ûvz³PS³~'¼(Ê`Ø  Key:  
DefaultCipherKey  Counter: 38  Offset: 12  Plaintext: OriginalPlainTxt  

ENCRYPTION 39  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
  Counter: 39  Offset: 124  Ciphertext: Ä^c*e„áêppãëæçg  
DECRIPTION 39  Ciphertext: Ä^c*e„áêppãëæçg  Key:  
DefaultCipherKey  Counter: 39  Offset: 12  Plaintext: ÜÔÈXQÏ±N[ÍNÝëê3
ENCRYPTION 40
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 40 Offset: 123
Ciphertext: Ä¹œ·à„á®pp®*́ëaqg

DECRYPTION 40
Ciphertext: Ä¹œ·à„á®pp®*́ëaqg
Key: DefaultCipherKey
Counter: 40 Offset: 12
Plaintext: φ„[[£®ŁÇ£] â²³

ENCRYPTION 41
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 41 Offset: 12
Ciphertext: ýœ s°â?°: R–ÄëfyØX

DECRYPTION 41
Ciphertext: ýœ s°â?°: R–ÄëfyØX
Key: DefaultCipherKey
Counter: 41 Offset: 12
Plaintext: OriginalPlainTxt

ENCRYPTION 42
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 42 Offset: 12
Ciphertext: 2{Ö<}$jÁët<âÖ–ých

DECRYPTION 42
Ciphertext: 2{Ö<}$jÁët<âÖ–ých
Key: DefaultCipherKey
Counter: 42 Offset: 12
Plaintext: OriginalPlainTxt

ENCRYPTION 43
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 43 Offset: 124
Ciphertext: gQ¿(?•ğwż–κ^NÇİ?H

DECRYPTION 43
Ciphertext: gQ¿(?•ğwż–κ^NÇİ?H
Key: DefaultCipherKey
Counter: 43 Offset: 12
Plaintext: Í,c9-,₆aK-NYÜêAK

ENCRYPTION 44
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 44 Offset: 123
Ciphertext: gQ¿(?•ğwż–κ^NÇİ?H

DECRYPTION 44
Ciphertext: gQ¿(?•ğwż–κ^NÇİ?H
Key: DefaultCipherKey
Counter: 44 Offset: 12
Plaintext: ü₆R®ş}óÜž?b|*

ENCRYPTION 45
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 45 Offset: 123
Ciphertext: %j,²È¶®7Ý ylabel?5?'`q
ENCRIPTION 51  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 51  Offset: 12  Ciphertext: T1?çòûÈòê²5?ò•ìâ
DECRYPTION 51  Ciphertext: T1?çòûÈòê²5?ò•ìâ  Key: DefaultCipherKey
Counter: 51  Offset: 12  Plaintext: OriginalPlainTxt

ENCRIPTION 52  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 52  Offset: 123  Ciphertext: 8Zè÷äÜ|@»¬Ý®²2»v
DECRYPTION 52  Ciphertext: 8Zè÷äÜ|@»¬Ý®²2»v  Key: DefaultCipherKey
Counter: 52  Offset: 12  Plaintext: ÜfêK~Ã!!î`üzh

ENCRIPTION 53  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 53  Offset: 124  Ciphertext: $µ¥è|›"I>=Åûi+üô
DECRYPTION 53  Ciphertext: $µ¥è|›"I>=Åûi+üô  Key: DefaultCipherKey
Counter: 53  Offset: 12  Plaintext: ÀxET¬їÈ«<Ü.ÉY×¼

ENCRIPTION 54  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 54  Offset: 123  Ciphertext: $µ¥è|›"I>=Åûi+üô
DECRYPTION 54  Ciphertext: $µ¥è|›"I>=Åûi+üô  Key: DefaultCipherKey
Counter: 54  Offset: 12  Plaintext: /A2QæZ?ããI‘7!

ENCRIPTION 55  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 55  Offset: 123  Ciphertext: †.DCZ>:?¬ýâé’.wôÍc
DECRYPTION 55  Ciphertext: †.DCZ>:?¬ýâé’.wôÍc  Key: DefaultCipherKey
Counter: 55  Offset: 12  Plaintext: 6j)é9Ux^XR‘ü

ENCRIPTION 56  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 56  Offset: 12  Ciphertext: ??`28QæM...wôO.alô°ui
DECRYPTION 56  Ciphertext: ??`28QæM...wôO.alô°ui  Key: DefaultCipherKey
Counter: 56  Offset: 12  Plaintext: OriginalPlainTxt
ENCRYPTION 57  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 57 Offset: 124 Ciphertext: I ¼Ã¬|:²±Äm¢ãDnûÔ Õ

DECRYPTION 57  Ciphertext: I ¼Ã¬|:²±Äm¢ãDnûÔ Õ  Key: DefaultCipherKey
Counter: 57 Offset: 12 Plaintext: '«®ˆs~¸Ä28›˜°4

ENCRYPTION 58  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 58 Offset: 123 Ciphertext: I ¼Ã¬|:²±Äm¢ãDnûÔ Õ

DECRYPTION 58  Ciphertext: I ¼Ã¬|:²±Äm¢ãDnûÔ Õ  Key: DefaultCipherKey
Counter: 58 Offset: 12 Plaintext: mÀí‾*÷¬³i—üe

ENCRYPTION 59  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 59 Offset: 123 Ciphertext: Ç%ÞˆùXzÔ^Skñ)Œh¬›

DECRYPTION 59  Ciphertext: Ç%ÞˆùXzÔ^Skñ)Œh¬›  Key: DefaultCipherKey
Counter: 59 Offset: 12 Plaintext: 6?/O¤ªêsá0ý2Tö

ENCRYPTION 60  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 60 Offset: 12 Ciphertext: ?Ç^›Oâ®ÆO2¨draqøºUI

DECRYPTION 60  Ciphertext: ?Ç^›Oâ®ÆO2¨draqøºUI  Key: DefaultCipherKey
Counter: 60 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 61  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 61 Offset: 124 Ciphertext: ¬}²â¦4réâ′˙°5NÆ«

DECRYPTION 61  Ciphertext: ¬}²â¦4réâ′˙°5NÆ«  Key: DefaultCipherKey
Counter: 61 Offset: 12 Plaintext: a#ëâ®®l@-ÉOSŠç^
DECryption 62  Ciphertext: \[\text{[...]}\]  Key: DefaultCipherKey
Counter: 62 Offset: 12 Plaintext: Šú·.Îlz_tÇÖü[é

ENCRYPTION 63  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 63 Offset: 12 Ciphertext: H®VyrÐÙ2ÁHz)¹?Í}
DECryption 63  Ciphertext: H®VyrÐÙ2ÁHz)¹?Í]  Key: DefaultCipherKey
Counter: 63 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 64  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 64 Offset: 123 Ciphertext: 2K¶]N--:*µ-Ø%×J®,Ψs~
DECryption 64  Ciphertext: 2K¶]N--:*µ-Ø%×J®,Ψs~ Key: DefaultCipherKey
Counter: 64 Offset: 12 Plaintext: DÅè–Ý~£˜€qk˜MÁ¬

ENCRYPTION 65  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 65 Offset: 12 Ciphertext: 8+ØQw+sYp0?¹x4GZ[]?¶
DECryption 65  Ciphertext: 8+ØQw+sYp0?¹x4GZ[]?¶ Key: DefaultCipherKey
Counter: 65 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 66  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 66 Offset: 124 Ciphertext: 0©l{Xujr+Kñ;Œ, êe®Iß¬
DECryption 66  Ciphertext: 0©l{Xujr+Kñ;Œ, êe®Iß¬ Key: DefaultCipherKey
Counter: 66 Offset: 12 Plaintext: /?w¥%N²Ý;Y0D×

ENCRYPTION 67  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 67 Offset: 124 Ciphertext: 1 0C;j|§IXÉ #h”tÜ|¬
DECryption 67  Ciphertext: 1 0C;j|§IXÉ #h”tÜ|¬ Key: DefaultCipherKey
Counter: 67 Offset: 12 Plaintext: 0$Ø¥$n3ø/ø]¼Xôn
ENCRIPTION 74  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 74 Offset: 123 Ciphertext: ÇòÒ`~Ýtáj1ÁıSò@üiÁí
DECRIPTION 74  Ciphertext: ÇòÒ`~Ýtáj1ÁıSò@üiÁí  Key:
DefaultCipherKey  Counter: 74 Offset: 12  Plaintext: øëôY^-z)!(?ıdpq

ENCRIPTION 75  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 75 Offset: 124 Ciphertext: 7È3xÍN1@uA+-(-úS?á
DECRIPTION 75  Ciphertext: 7È3xÍN1@uA+-(-úS?á  Key:
DefaultCipherKey  Counter: 75 Offset: 12  Plaintext: ø)qj²UGQe Mnôá

ENCRIPTION 76  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 76 Offset: 12 Ciphertext: ³”ÇàüjVÁKítõ"ê©J°-n+Á
DECRIPTION 76  Ciphertext: ³”ÇàüjVÁKítõ"ê©J°-n+Á  Key:
DefaultCipherKey  Counter: 76 Offset: 12  Plaintext: OriginalPlainTxt

ENCRIPTION 77  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 77 Offset: 123 Ciphertext: 'EDÜHÃ¥y@Á_jfI57S
DECRIPTION 77  Ciphertext: 'EDÜHÃ¥y@Á_jfI57S  Key:
DefaultCipherKey  Counter: 77 Offset: 12  Plaintext: w9lÉp?'¢'©DB {n

ENCRIPTION 78  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 78 Offset: 124 Ciphertext: #8ï²lâ·.șN(^{?t
DECRIPTION 78  Ciphertext: #8ï²lâ·.șN(^{?t  Key:
DefaultCipherKey  Counter: 78 Offset: 12  Plaintext: áëQÀOA(-i?·Wf2

ENCRIPTION 79  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 79 Offset: 123 Ciphertext: #8ï²lâ·.șN(^{?t

96
DECRYPTION 79  Ciphertext: ÊÁÇ±1 çaž°¿8(^[?■t? Key:
DefaultCipherKey  Counter: 79 Offset: 12  Plaintext: ÀZ'iI?Ø48Èay intelligent

ENCRYPTION 80  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 80 Offset: 123  Ciphertext: £V?−ІȘ*ü·m mē×GȘȘUV

DECRYPTION 80  Ciphertext: £V?−ІȘ*ü·m mē×GȘȘUV Key:
DefaultCipherKey  Counter: 80 Offset: 12  Plaintext: 7kšÄS{FiNg"ȘUș00

ENCRYPTION 81  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 81 Offset: 12  Ciphertext: 'Áluö·âWQò Äb±JZAð

DECRYPTION 81  Ciphertext: 'Áluö·âWQò Äb±JZAð Key:
DefaultCipherKey  Counter: 81 Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 82  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 82 Offset: 124  Ciphertext: %³zjo,cï(uœ,Á5^iÅw€

DECRYPTION 82  Ciphertext: %³zjo,cï(uœ,Á5^iÅw€ Key:
DefaultCipherKey  Counter: 82 Offset: 12  Plaintext: få)»a?−hcEe°)xXA

ENCRYPTION 83  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 83 Offset: 123  Ciphertext: %³yjo,cï(uœ,Á5^iÅw€

DECRYPTION 83  Ciphertext: %³yjo,cï(uœ,Á5^iÅw€ Key:
DefaultCipherKey  Counter: 83 Offset: 12  Plaintext: 'squirrel−YâA^PmO;

ENCRYPTION 84  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey

DECRYPTION 84  Ciphertext: Â=«?Šá−«R9Y? i□?~JÜ Key:
DefaultCipherKey  Counter: 84 Offset: 12  Plaintext: 7ÈEö−æŒ M@¬ ñLí
ENCRYPTION 85  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey

DECRYPTION 85  Ciphertext: Å=«−Šá−«R9Y? I,—?¬JÜ  Key:
  DefaultCipherKey  Counter: 85 Offset: 12 Plaintext: gnj□"FJGvDär.Ç~

ENCRYPTION 86  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
  Counter: 86 Offset: 12 Ciphertext: ÖVE,ŠÇÁÁśýµ£ß3Αµ

DECRYPTION 86  Ciphertext: ÖVE,ŠÇÁÁśýµ£ß3Αµ  Key:
  DefaultCipherKey  Counter: 86 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 87  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
  Counter: 87 Offset: 124 Ciphertext: VJ<¶−º¨¨"Ű,œ¼íѽ2

DECRYPTION 87  Ciphertext: VJ<¶−º¨¨"Ű,œ¼íѽ2  Key:
  DefaultCipherKey  Counter: 87 Offset: 12 Plaintext: ¿fÖ¦««|ž0ˆç!Ñ

ENCRYPTION 88  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
  Counter: 88 Offset: 123 Ciphertext: VJ<¶−º¨¨"Ű,œ¼íѽ2

DECRYPTION 88  Ciphertext: VJ<¶−º¨¨"Ű,œ¼íѽ2  Key:
  DefaultCipherKey  Counter: 88 Offset: 12 Plaintext: ¿œû2™ÉíÞ˜öz

ENCRYPTION 89  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
  Counter: 89 Offset: 124 Ciphertext: −­cDbtâÍ?Ôp:00û;4

DECRYPTION 89  Ciphertext: −­cDbtâÍ?Ôp:00û;4  Key:
  DefaultCipherKey  Counter: 89 Offset: 12 Plaintext: R5yeÔ¢ì,i*Z7Ió

ENCRYPTION 90  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
  Counter: 90 Offset: 12 Ciphertext: ôcx ¼î½−³Οuûfβ?á

DECRYPTION 90  Ciphertext: ôcx ¼î½−³Οuûfβ?á  Key:
  DefaultCipherKey  Counter: 90 Offset: 12 Plaintext: OriginalPlainTxt
DECRIPTION 96  Ciphertext: āÇrō°ōÅÄ ť dĀ≈m)y5o  Key:
DefaultCipherKey  Counter: 96 Offset: 12  Plaintext: zÍ-?ÅŽ-1wÉ4t*EY

ENCRYPTION 97  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 97 Offset: 124  Ciphertext: ?=ωXÎá8p;□D„?="ÃºYA

DECRIPTION 97  Ciphertext: ?=ωXÎá8p;□D„?="ÃºYA
Key: DefaultCipherKey  Counter: 97 Offset: 12
Plaintext:-ø/Ób× dictate ÜwbjΕ

ENCRYPTION 98  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 98 Offset: 12  Ciphertext: Ü| "|തfÃK®@!UVŞSi

DECRIPTION 98  Ciphertext: Ü| "|തfÃK®@!UVŞSi  Key:
DefaultCipherKey  Counter: 98 Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 99  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 99 Offset: 12  Ciphertext: (DÎ4h"Eœ?==ÄÇOQ©ÀA

DECRIPTION 99  Ciphertext: (DÎ4h"Eœ?==ÄÇOQ©ÀA  Key:
DefaultCipherKey  Counter: 99 Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 100  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 100  Offset: 12  Ciphertext: ûm-åÔîŒ¬-ōXÎpzüéac

DECRIPTION 100  Ciphertext: ûm-åÔîŒ¬-ōXÎpzüéac  Key:
DefaultCipherKey  Counter: 100  Offset: 12  Plaintext: OriginalPlainTxt

Cipher 1 was chosen 29 out of 100 iterations.
Cipher 1 was chosen at most 3 times in a row.
Randomized Algorithm with Multiple Ciphers and Hash and Varying Key

Ran for 100 iterations with 3 decrypting ciphers

**ENCRYPTION 1**
- Plaintext: OriginalPlainTxt
- Key: DefaultCipherKey
- Counter: 1
- Offset: 123
- Ciphertext: @!J$A|$l|f|U|EN-E³I-k

**DECRYPTION 1**
- Ciphertext: @!J$A|$l|f|U|EN-E³I-k
- Key: 67IiÇXơ%è ÇW-O
- Counter: 1
- Offset: 12
- Plaintext: ?jÈM$hB%P+Í工伤

**ENCRYPTION 2**
- Plaintext: OriginalPlainTxt
- Key: DefaultCipherKey
- Counter: 2
- Offset: 12
- Ciphertext: ñûzI艉 ü qEÔA€9I`a)

**DECRYPTION 2**
- Ciphertext: ñûzI艉 ü qEÔA€9I`a)
- Key: DefaultCipherKey
- Counter: 2
- Offset: 12
- Plaintext: ?ïËM<hB‰ÆP†Í½Ç

**ENCRYPTION 3**
- Plaintext: OriginalPlainTxt
- Key: DefaultCipherKey
- Counter: 3
- Offset: 124
- Ciphertext: C,FdAA.A院副院长•「n`[i

**DECRYPTION 3**
- Ciphertext: C,FdAA.A副院长•「n`[i
- Key: H…4ÅO/Xè_£%?bKk
- Counter: 3
- Offset: 12
- Plaintext: ÄxE „S.jûv‰~peƗ+eI

**ENCRYPTION 4**
- Plaintext: OriginalPlainTxt
- Key: DefaultCipherKey
- Counter: 4
- Offset: 123
- Ciphertext: C,FdAA.A副院长•「n`[i

**DECRYPTION 4**
- Ciphertext: C,FdAA.A副院长•「n`[i
- Key: ðÁÖÁ†NQ;O ÖD~4...O
- Counter: 4
- Offset: 12
- Plaintext: ÏO`:GmdDØo%"fI

**ENCRYPTION 5**
- Plaintext: OriginalPlainTxt
- Key: DefaultCipherKey
- Counter: 5
- Offset: 124
- Ciphertext: á®LØIf 2:oùÅÌ³ô[?ñ

**DECRYPTION 5**
- Ciphertext: á®LØIf 2:oùÅÌ³ô[?ñ
- Key: 4}vÇÇUmtB?Ž‘-i
- Counter: 5
- Offset: 12
- Plaintext: éôkåñim…+H/Š;tó°
ENCRYPTION 6
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 6
Offset: 124
Ciphertext: €jLÏ?%?'b¥VrÀ„\%

DECRYPTION 6
Ciphertext: €jLÏ?%?'b¥VrÀ„\%
Key: Ývâbrð6ðL?q7kù.ý
Counter: 6
Offset: 12
Plaintext: ßd€}?wûx*:0û

ENCRYPTION 7
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 7
Offset: 12
Ciphertext: ŸùN3'&Wý3;\+Wûš°-yM\-

DECRYPTION 7
Ciphertext: ŸùN3'&Wý3;\+Wûš°-yM\-
Key: DefaultCipherKey
Counter: 7
Offset: 12
Plaintext: OriginalPlainTxt

ENCRYPTION 8
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 8
Offset: 124
Ciphertext: ;Gt³î> ëa³-0±§AbçÔ

DECRYPTION 8
Ciphertext: ;Gt³î> ëa³-0±§AbçÔ
Key: VMû4Gçzì}DV?C|N
Counter: 8
Offset: 12
Plaintext: azü;"i%Kkê'{

ENCRYPTION 9
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 9
Offset: 124
Ciphertext: 'Îtýû |w×¹m'Ýýô:šN

DECRYPTION 9
Ciphertext: 'Îtýû |w×¹m'Ýýô:šN
Key: é=E|Qwâ|û"EC
Counter: 9
Offset: 12
Plaintext: ßyÜxwY,LaîÖÈë

ENCRYPTION 10
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 10
Offset: 124
Ciphertext: −7«BK±ê'|ùãç3'°+ô

DECRYPTION 10
Ciphertext: −7«BK±ê'|ùãç3'°+ô
Key: Ë? õ×ûBþ3b¹``
Counter: 10
Offset: 12
Plaintext: @IAM°çY~``\n
ENCRYPTION 11
Plaintext: OriginalPlainTxt
Key: DefaultCipherKey
Counter: 11
Offset: 123
Ciphertext: −7'BK±ê'|ùãç3'°+ô
DECRIPTION 11  Ciphertext: Ñ"BKîê'|üåçž³'°+¿  Key: 
ë˜(NiòfäBKî¬y[ð Counter: 11 Offset: 12 Plaintext: /...åå³²§èBë» Îr=r

ENCRYPTION 12  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey

Counter: 12 Offset: 123 Ciphertext: \ÁĂštEWu+ïG9žI¬«»
DECRIPTION 12 Ciphertext: "ÄâŒtEWu+×G9Î©»"
Key: 8k;Äfe±87xFTÝ]  Counter: 12 Offset: 12  Plaintext:
NÜ9ÁîKdÄN~ë19x

ENCRYPTION 13  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 13 Offset: 123  Ciphertext: È´%h¥fTéb±z$–ÄÔ/€

DECRYPTION 13  Ciphertext: È´%h¥fTéb±z$–ÄÔ/€  Key: 
=ÀâKUE:çLI~&}  Counter: 13 Offset: 12  Plaintext: µrS‡ŽÇ?ùc1Ôlµ0IG

ENCRYPTION 14  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 14 Offset: 123  Ciphertext: ë±08'a¶)="M~VÜ~µ7

DECRYPTION 14  Ciphertext: ë±08'a¶)="M~VÜ~µ7  Key: 
°××××6UÈôhØüj»p=  Counter: 14 Offset: 12  Plaintext: ŠSè–ä/äd  ±ÔL[B

ENCRYPTION 15  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 15 Offset: 12  Ciphertext: –C¶yÚYɿ³]\#ù8z~©9Éµ

DECRYPTION 15  Ciphertext: –C¶yÚYɿ³]\#ù8z~©9Éµ  Key: 
DefaultCipherKey  Counter: 15 Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 16  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 16 Offset: 123  Ciphertext: ë9è©åñì,0eÖf Δ}$:Â

DECRYPTION 16  Ciphertext: ë9è©åñì,0eÖf Δ}$:Â  Key: 
âYN£C/fÝbym[MúJ  Counter: 16 Offset: 12  Plaintext: mm,"Sà·¥ÚOœJ

ENCRYPTION 17  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 17 Offset: 12  Ciphertext: ½0±óñ³"ky¹'s~`Žó-Këc

DECRYPTION 17  Ciphertext: ½0±óñ³"ky¹'s~`Žó-Këc  Key: 
DefaultCipherKey  Counter: 17 Offset: 12  Plaintext: OriginalPlainTxt
ENCRYPTION 18
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 18 Offset: 124 Ciphertext:  \[\text{cipher text}\]\n
DECRYPTION 18
Ciphertext:  \[\text{cipher text}\]\nKey:  \[\text{key}\]\nCounter: 18 Offset: 12 Plaintext:  \[\text{plaintext}\]\n
ENCRYPTION 19
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 19 Offset: 123 Ciphertext:  \[\text{cipher text}\]\n
DECRYPTION 19
Ciphertext:  \[\text{cipher text}\]\nKey:  \[\text{key}\]\nCounter: 19 Offset: 12 Plaintext:  \[\text{plaintext}\]\n
ENCRYPTION 20
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 20 Offset: 123 Ciphertext:  \[\text{cipher text}\]\n
DECRYPTION 20
Ciphertext:  \[\text{cipher text}\]\nKey:  \[\text{key}\]\nCounter: 20 Offset: 12 Plaintext:  \[\text{plaintext}\]\n
ENCRYPTION 21
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 21 Offset: 123 Ciphertext:  \[\text{cipher text}\]\n
DECRYPTION 21
Ciphertext:  \[\text{cipher text}\]\nKey:  \[\text{key}\]\nCounter: 21 Offset: 12 Plaintext:  \[\text{plaintext}\]\n
ENCRYPTION 22
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 22 Offset: 12 Ciphertext:  \[\text{cipher text}\]\n
DECRYPTION 22
Ciphertext:  \[\text{cipher text}\]\nKey:  \[\text{key}\]\nCounter: 22 Offset: 12 Plaintext:  \[\text{plaintext}\]\n
ENCRYPTION 23
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 23 Offset: 12 Ciphertext:  \[\text{cipher text}\]\n
DECRYPTION 23
Ciphertext:  \[\text{cipher text}\]\nKey:  \[\text{key}\]\nCounter: 23 Offset: 12 Plaintext:  \[\text{plaintext}\]\n
106
ENCRYPTION 24
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 24  Offset: 124  Ciphertext: !äš nØø..;3*8ÖR^1KÅ

DECRYPTION 24
Ciphertext: !äš nØø..;3*8ÖR^1KÅ  Key:
)8D;äÍ²YÉÇb1,zû  Counter: 24  Offset: 12  Plaintext: Dœ×!°/»Eœ;E-1â

ENCRYPTION 25
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 25  Offset: 123  Ciphertext: !äš/nØø..;3*8ÖR^1KÅ

DECRYPTION 25
Ciphertext: !äš/nØø..;3*8ÖR^1KÅ  Key:
àœëñr?fNëñj--ýf-  Counter: 25  Offset: 12  Plaintext: NB£vº»ãaÑÈ¢˜Ž

ENCRYPTION 26
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 26  Offset: 123  Ciphertext: ~ûáÖ1'g‡8ð!EC½†r"d

DECRYPTION 26
Ciphertext: ~ûáÖ1'g‡8ð!EC½†r"d  Key:
?FD‾tö[Jø:ÊŠ°7u©  Counter: 26  Offset: 12  Plaintext: 'I-[ÈæÅÉz”œ,?ø'

ENCRYPTION 27
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 27  Offset: 12  Ciphertext: -SœÉå?’›©,Jgz”á"ś

DECRYPTION 27
Ciphertext: -SœÉå?’›©,Jgz”á"ś  Key:
DefaultCipherKey  Counter: 27  Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 28
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 28  Offset: 124  Ciphertext: ::?ΔuQX-X©°d  DTÓ

DECRYPTION 28
Ciphertext: ::?ΔuQX-X©°d  DTÓ  Key:
àœëD®üu-!âm0oŠ™  Counter: 28  Offset: 12  Plaintext: ÏœBi>"u>eNά;Ý,B

ENCRYPTION 29
Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 29  Offset: 123  Ciphertext: ::?ΔuQX-X©°d  DTÓ
DECRIPTION 29  Ciphertext: ??ÉuOQX–X×©’êd  DTÓ  Key: gÈuďôÂÄ×…nÎLy×cÚ  Counter: 29 Offset: 12  Plaintext: Ïë$ Ïpɔ£×Œ¬´"'

ENCRIPTION 30  Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
        Counter: 30 Offset: 124  Ciphertext: ÔNCĨ?’â×hµqHãâ®œ^uœ^  

DECRIPTION 30  Ciphertext: ÔNCĨ?’â×hµqHãâ®œ^uœ^  Key: ;x–
        Counter: 30 Offset: 12  Plaintext: I·”éİ?Gjh:’kũ?”É

ENCRIPTION 31  Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
        Counter: 31 Offset: 124  Ciphertext: Éc~?Øjýřżœ‘µ 2!"  

DECRIPTION 31  Ciphertext: Éc~?Øjýřżœ‘µ 2!" Key: F(Èýy6âéý#{-Éóè  Counter: 31 Offset: 12  Plaintext: ykl3Mw~$oc)?£...$%

ENCRIPTION 32  Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
        Counter: 32 Offset: 124  Ciphertext: Ŷa;éDż 4Y¬;iÔ)¥P&Ý!2  

DECRIPTION 32  Ciphertext: Ŷa;éDż 4Y¬;iÔ)¥P&Ý!2  Key: 8µµ®A_ÉÈ  " : iPod  Counter: 32 Offset: 12  Plaintext: ë)(’îSÉÖÖRžô; ä

ENCRIPTION 33  Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
        Counter: 33 Offset: 124  Ciphertext: ”cf$0?¬óOžùô?d$?dp>^-  

DECRIPTION 33  Ciphertext: ”cf$0?¬óOžùô?d$?dp>^-  Key: ³Üdâ¬êàéé­ïï...  Counter: 33 Offset: 12  Plaintext: RNIC\4 3ä\ëi+–É

ENCRIPTION 34  Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
        Counter: 34 Offset: 12  Ciphertext: p[Åu?±B"íOSÜSåëëâëëÝY"  

DECRIPTION 34  Ciphertext: p[Åu?±B"íOSÜSåëëâëëÝY"  Key: DefaultCipherKey  Counter: 34 Offset: 12  Plaintext: OriginalPlainTxt

108
<table>
<thead>
<tr>
<th>Encryption Level</th>
<th>Plaintext:</th>
<th>Key:</th>
<th>Counter:</th>
<th>Offset:</th>
<th>Ciphertext:</th>
<th>Decryption Level</th>
<th>Ciphertext:</th>
<th>Key:</th>
<th>Counter:</th>
<th>Offset:</th>
<th>Plaintext:</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>OriginalPlainTxt</td>
<td>DefaultCipherKey</td>
<td>41</td>
<td>12</td>
<td>y□</td>
<td>s°?□°:- R-ÄêfyōX</td>
<td>41</td>
<td></td>
<td>Ciphertext:</td>
<td>DefaultCipherKey</td>
<td>Counter: 41</td>
</tr>
<tr>
<td>42</td>
<td>OriginalPlainTxt</td>
<td>DefaultCipherKey</td>
<td>42</td>
<td>12</td>
<td>2{Û&lt;}§}æÅët‹¹Òž2ð+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>OriginalPlainTxt</td>
<td>DefaultCipherKey</td>
<td>43</td>
<td>123</td>
<td>E)¥õÛÇØÆ?%íõÜŠ&amp;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>OriginalPlainTxt</td>
<td>DefaultCipherKey</td>
<td>44</td>
<td>124</td>
<td>‰j,ÄÆ¶ªÖ7Ÿ´¥O©?5³?`¶</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>OriginalPlainTxt</td>
<td>DefaultCipherKey</td>
<td>45</td>
<td>123</td>
<td>‰j,²Æ¶ªÖ7Ÿ´¥O©?5³?`¶</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 46               | OriginalPlainTxt | DefaultCipherKey | 46 | 123 | ],$Ý+??"immo2Ö-commercial
ENCRYPTION 52  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
      Counter: 52  Offset: 123  Ciphertext: .8Z+-åÜ|@» -ý√θ²→»v 
DECRYPTION 52  Ciphertext: .8Z+-åÜ|@» -ý√θ²→»v  Key: 
   Counter: 52  Offset: 12  Plaintext: óÄ×àYNsYö? 

ENCRYPTION 53  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey 
      Counter: 53  Offset: 12  Ciphertext: 5úDé|j×  lίĐ>-»ó»øy 
DECRYPTION 53  Ciphertext: 5úDé|j×  lίĐ>-»ó»øy  Key: 
   DefaultCipherKey  Counter: 53  Offset: 12  Plaintext: OriginalPlainTxt 

ENCRYPTION 54  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey 
      Counter: 54  Offset: 123  Ciphertext: §üDé|j×  i‡üôÝ 
DECRYPTION 54  Ciphertext: §üDé|j×  i‡üôÝ  Key: 
   Ê´ËÑ^ÓM`"9  Counter: 54  Offset: 12  Plaintext: ÇD¸ðS;Âäy:êÊ 

ENCRYPTION 55  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey 
      Counter: 55  Offset: 12  Ciphertext: ³Ó;Œü½òcËa™˜p? ^b¼|ő 
DECRYPTION 55  Ciphertext: ³Ó;Œü½òcËa™˜p? ^b¼|ő  Key: 
   DefaultCipherKey  Counter: 55  Offset: 12  Plaintext: OriginalPlainTxt 

ENCRYPTION 56  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey 
      Counter: 56  Offset: 12  Ciphertext: ??*2BQãM...4°.Ol»ôºui 
DECRYPTION 56  Ciphertext: ??*2BQãM...4°.Ol»ôºui  Key: 
   DefaultCipherKey  Counter: 56  Offset: 12  Plaintext: OriginalPlainTxt 

ENCRYPTION 57  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey 
      Counter: 57  Offset: 123  Ciphertext: y:z?6åûYv?  OaÄ;5¬wO» 
DECRYPTION 57  Ciphertext: y:z?6åûYv?  OaÄ;5¬wO»  Key: 
   1)äœ?„g+Ä¨i;ä"  Counter: 57  Offset: 12  Plaintext: ìµ”¬ëm¬5ìSp}ä
ENCRYPTION 58  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
  Counter: 58  Offset: 123  Ciphertext: I ŶÄı|::?±Åm¢ÄDnĥÔ  Ő

DECRYPTION 58  Ciphertext: I ŶÄı|::?±Åm¢ÄDnĥÔ  Ő  Key:
  ^êßøb<¶§`¨®ZêÒ  Counter: 58  Offset: 12  Plaintext: ÿÂ³ÅHégúκ¬¬S"

ENCRYPTION 59  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
  Counter: 59  Offset: 12  Ciphertext: Ő„voÅ¬ý¥  ŐU«"¬¬d¬

DECRYPTION 59  Ciphertext: Ő„voÅ¬ý¥  ŐU«"¬¬d¬  Key:
  DefaultCipherKey  Counter: 59  Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 60  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
  Counter: 60  Offset: 12  Ciphertext: Ő„voÅ¬ý¥  ŐU«"¬¬d¬

DECRYPTION 60  Ciphertext: Ő„voÅ¬ý¥  ŐU«"¬¬d¬  Key:
  DefaultCipherKey  Counter: 60  Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 61  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
  Counter: 61  Offset: 123  Ciphertext: 0¶nÐ0ºÊÖ" YqÀ¥N

DECRYPTION 61  Ciphertext: 0¶nÐ0ºÊÖ" YqÀ¥N  Key:
  $?}ã3ÉÚ”mÕ'„øò  Counter: 61  Offset: 12  Plaintext: ÈÅ;ñ—ªPOn|óÜs>

ENCRYPTION 62  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
  Counter: 62  Offset: 124  Ciphertext: °gobyåťśÔY¬ŽD¢Â¬| U¬

DECRYPTION 62  Ciphertext: °gobyåťśÔY¬ŽD¢Â¬| U¬  Key:
  É%<,+ó.cvNö&MÎ  Counter: 62  Offset: 12  Plaintext: ÈÂÅ`ëëë.;ë×Șj?€

ENCRYPTION 63  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
  Counter: 63  Offset: 124  Ciphertext: ẺK\^N¬:*µ-Ø%×J®,ı3¬
ENCRYPTION 69  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 69 Offset: 123 Ciphertext: ŶŒŒ·Rp³j?8ó³öú²!

DECRYPTION 69  Ciphertext: ŶŒŒ·Rp³j?8ó³öú²!  Key: 
4ú!ã!uôIUSm“IMt  Counter: 69 Offset: 12 Plaintext: dû>¬NÔÉÜ颖ýZá

ENCRYPTION 70  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 70 Offset: 12 Ciphertext: Ŷ·tB¥7-;©I0^K^?n-ûð

DECRYPTION 70  Ciphertext: Ŷ·tB¥7-;©I0^K^?n-ûð  Key: DefaultCipherKey
Counter: 70 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 71  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 71 Offset: 123 Ciphertext: ôóIp ŶKôxÃÀóôy6Ô(†`ë

DECRYPTION 71  Ciphertext: ôóIp ŶKôxÃÀóôy6Ô(†`ë  Key: ."!Áü"Ad)›?«V[M[d  Counter: 71 Offset: 12 Plaintext: b™|¿k)VÚ°œ²

ENCRYPTION 72  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 72 Offset: 12 Ciphertext: ŶÅŽ®B©<°`ëú&%[M

DECRYPTION 72  Ciphertext: ŶÅŽ®B©<°`ëú&%[M  Key: DefaultCipherKey  Counter: 72 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 73  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 73 Offset: 12 Ciphertext: ŸTv4~Ý4Ýø7Å³ê(namedTQ`

DECRYPTION 73  Ciphertext: ŸTv4~Ý4Ýø7Å³ê(namedTQ`  Key: DefaultCipherKey  Counter: 73 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 74  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 74 Offset: 124 Ciphertext: Ŷv« Bö,“G.Öw”>,¨öpí

DECRYPTION 74  Ciphertext: Ŷv« Bö,“G.Öw”>,¨öpí  Key: Ŷ,çYa®Î,î×Aóx  Counter: 74 Offset: 12 Plaintext: ìñ“ÖS×ÎÝgRnàù
ENCRYPTION 75  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
Counter: 75 Offset: 123 Ciphertext: Ožv" Bðª"G.Öw", "Öp  
DECRIPTION 75  Ciphertext: Ožv" Bðª"G.Öw", "Öp  Key:  
Áv:Í9án+c*Ôtbť O Counter: 75 Offset: 12 Plaintext: F4 ţ,C-x?6â¿1ã"

ENCRYPTION 76  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
Counter: 76 Offset: 124 Ciphertext: 'ÉDDßÁÝüÁ_jfI57$  
DECRIPTION 76  Ciphertext: 'ÉDDßÁÝüÁ_jfI57$  Key:  
³¶Ð‰•нÇ"N;èê9;  Counter: 76 Offset: 12 Plaintext: ?p->âo%Ä0 ">ŽJ

ENCRYPTION 77  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
DECRIPTION 77  Ciphertext: ]p)OtÖyâ*.wâC“Çhd¿  Key:  
æA‡áÖV®¶g×1¶°,  Counter: 77 Offset: 12 Plaintext: H©C2g2!8âª.'4

ENCRYPTION 78  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
Counter: 78 Offset: 12 Ciphertext: YQ@??E ÇqfÝ8@IÔâ?~$  
DECRIPTION 78  Ciphertext: YQ@??E ÇqfÝ8@IÔâ?~$  Key:  
DefaultCipherKey  Counter: 78 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 79  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
Counter: 79 Offset: 123 Ciphertext: #8ù±lù+.¿SN(">{"?t?  
DECRIPTION 79  Ciphertext: #8ù±lù+.¿SN(">{"?t?  Key:  
-Ýµ4 ü\~çÔ(1ØšüÊ  Counter: 79 Offset: 12 Plaintext: ←Gùë  ø"ªô£R-0’

ENCRYPTION 80  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
Counter: 80 Offset: 124 Ciphertext: ðW?³R•á"Wµzâ@FµÂöN
ENCRYPTION 86  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
Counter: 86 Offset: 12 Ciphertext: ŶYÃ¬wzm1yB¿3Åµ  
DECRYPTION 86  Ciphertext: ŶYÃ¬wzm1yB¿3Åµ  Key: DefaultCipherKey  
Counter: 86 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 87  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
Counter: 87 Offset: 12 Ciphertext: Ŷ2!ë...B§ðIÀ=´D{ äÖaD  
DECRYPTION 87  Ciphertext: Ŷ2!ë...B§ðIÀ=´D{ äÖaD  Key: DefaultCipherKey  
Counter: 87 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 88  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
Counter: 88 Offset: 123 Ciphertext: VJ<·-°''œ¹ìû,îýR©uúãî2  
DECRYPTION 88  Ciphertext: VJ<·-°''œ¹ìû,îýR©uúãî2  Key: ^pÍ‾e%û'T'Vi2¬  
Counter: 88 Offset: 12 Plaintext: iWÔÍ²AÔOªF"3u

ENCRYPTION 89  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
Counter: 89 Offset: 124 Ciphertext: -cßbtâ€’Öp:0ûO¼;4,  
DECRYPTION 89  Ciphertext: -cßbtâ€’Öp:0ûO¼;4,  Key: á‡éag9›LüùÖ†J  
Counter: 89 Offset: 12 Plaintext: i$*ýÈAYÜ16;'?êB*

ENCRYPTION 90  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
Counter: 90 Offset: 12 Ciphertext: ö榨 ¼Í°+3ÔúY*fp?á  
DECRYPTION 90  Ciphertext: ö榨 ¼Í°+3ÔúY*fp?á  Key: DefaultCipherKey  
Counter: 90 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 91  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey  
Counter: 91 Offset: 124 Ciphertext: UEÁÀâ?Ó søMZ·bN-2;j  
DECRYPTION 91  Ciphertext: UEÁÀâ?Ó søMZ·bN-2;j  Key: Qì3nãÀ`ù.œfê?...  
Counter: 91 Offset: 12 Plaintext: Ø?„T¼æû8^D¶(ÝÇ!
ENCRYPTION 92  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 92 Offset: 123 Ciphertext: ÙÈâ®?º söMŽ·bN·j

DECRYPTION 92  Ciphertext: ÙÈâ®?º söMŽ·bN·j  Key: Ë?pâ·@ýLŽ·Ép
Counter: 92 Offset: 12 Plaintext: Ñ–Ôæ8V>q0`Pąż

ENCRYPTION 93  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 93 Offset: 12 Ciphertext: '¥È?hy+ð¬®'óº®ë?È

DECRYPTION 93  Ciphertext: '¥È?hy+ð¬®'óº®ë?È  Key: DefaultCipherKey
Counter: 93 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 94  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 94 Offset: 124 Ciphertext: -Wá)â×g<„G¬.êÖÜi Ü

DECRYPTION 94  Ciphertext: -Wá)â×g<„G¬.êÖÜi Ü  Key: pÝ\üí;7|ïëv=Åu
Counter: 94 Offset: 12 Plaintext: Æ¬·%ãë=îb-ØU

ENCRYPTION 95  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 95 Offset: 123 Ciphertext: -Wá*â×g<„G¬.êÖÜi Ü

DECRYPTION 95  Ciphertext: -Wá*â×g<„G¬.êÖÜi Ü  Key: F®†¸R‡Œ'ɧErCý
Counter: 95 Offset: 12 Plaintext: AÙ&âû£èMFˆU|$Á/

ENCRYPTION 96  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 96 Offset: 12 Ciphertext: BQ#Ø??YÅY}bbzâµP]

DECRYPTION 96  Ciphertext: BQ#Ø??YÅY}bbzâµP]  Key: DefaultCipherKey
Counter: 96 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 97  Plaintext: OriginalPlainTxt  Key: DefaultCipherKey
Counter: 97 Offset: 124 Ciphertext: ?=êX Îâ®p;íà"?nÁ`YA
Cipher 1 was chosen 33 out of 100 iterations.

Cipher 1 was chosen at most 4 times in a row.
References


