2007

A Differential Power Analysis Resistant Randomized Algorithm using Multiple AES Ciphers

Richard Tran
San Jose State University

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

List of Tables

List of Figures

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.

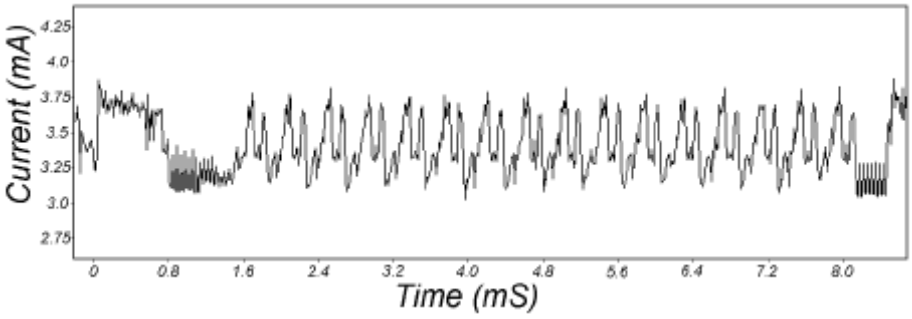

Figure 1. Simple Power Analysis Trace of DES [12]

Figure 1 clearly shows the 16 DES rounds in action.

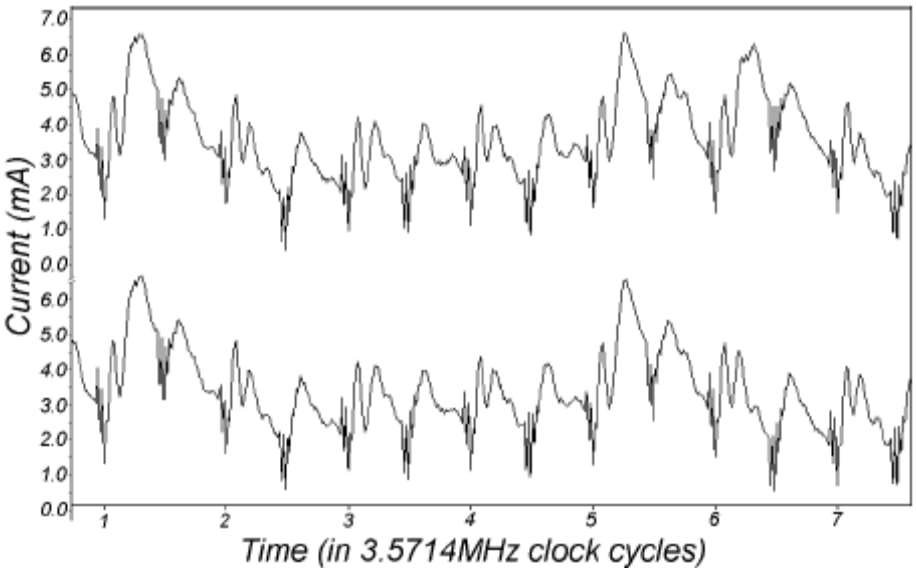

Figure 2. Higher Resolution View of DES Trace [12]

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]:

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

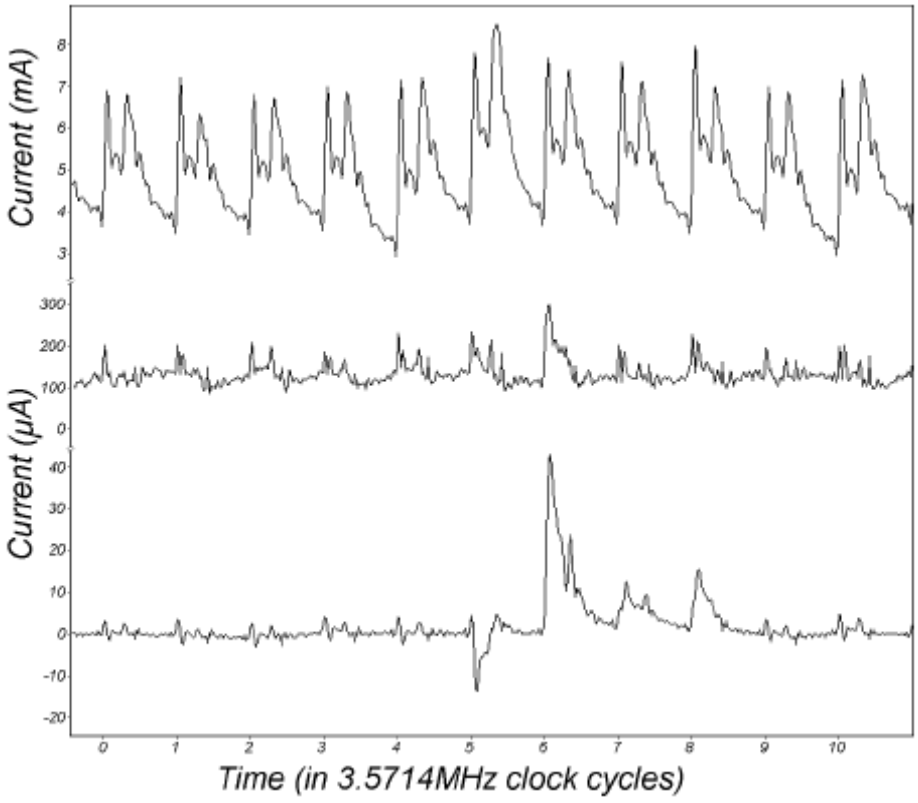

Figure 3. DPA Trace of DES [12]

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.

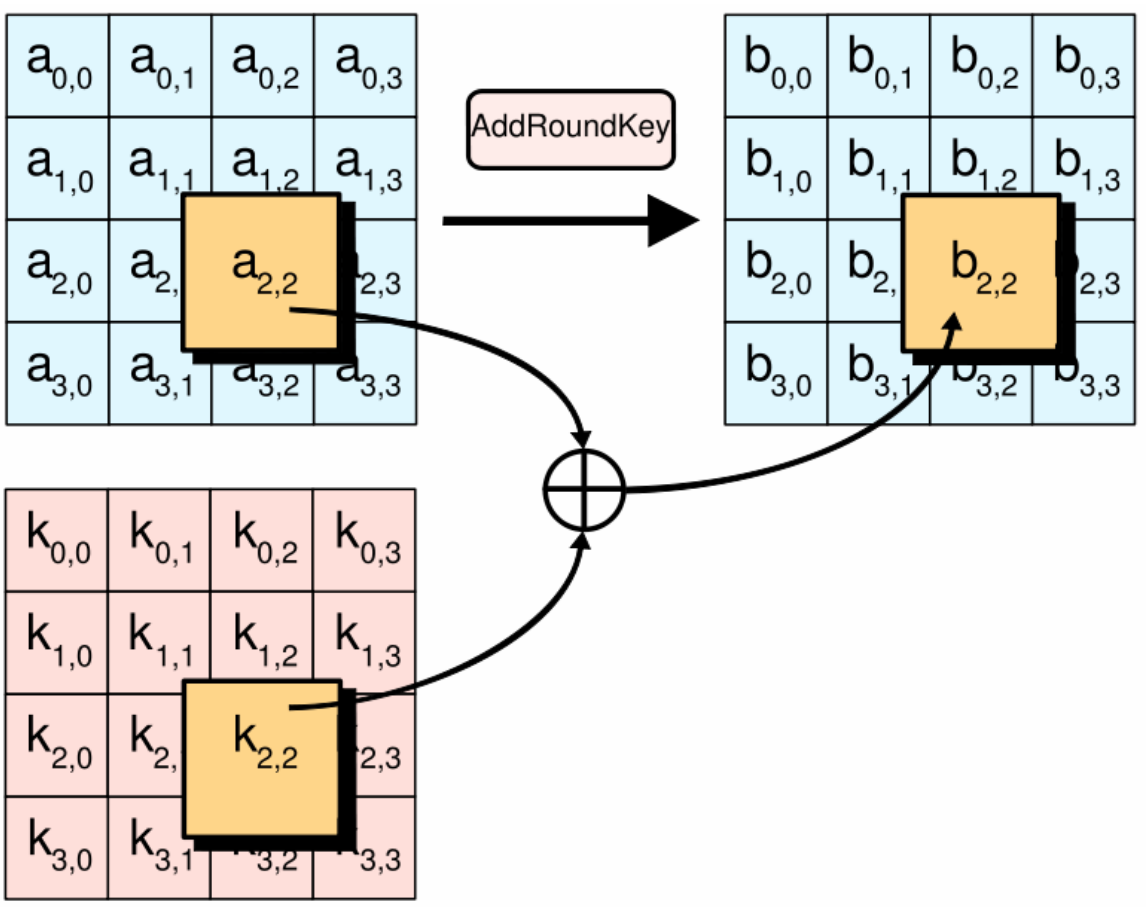

Figure 4. Add Round Key

Source: http://en.wikipedia.org/

4.1.2. Byte Substitution

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

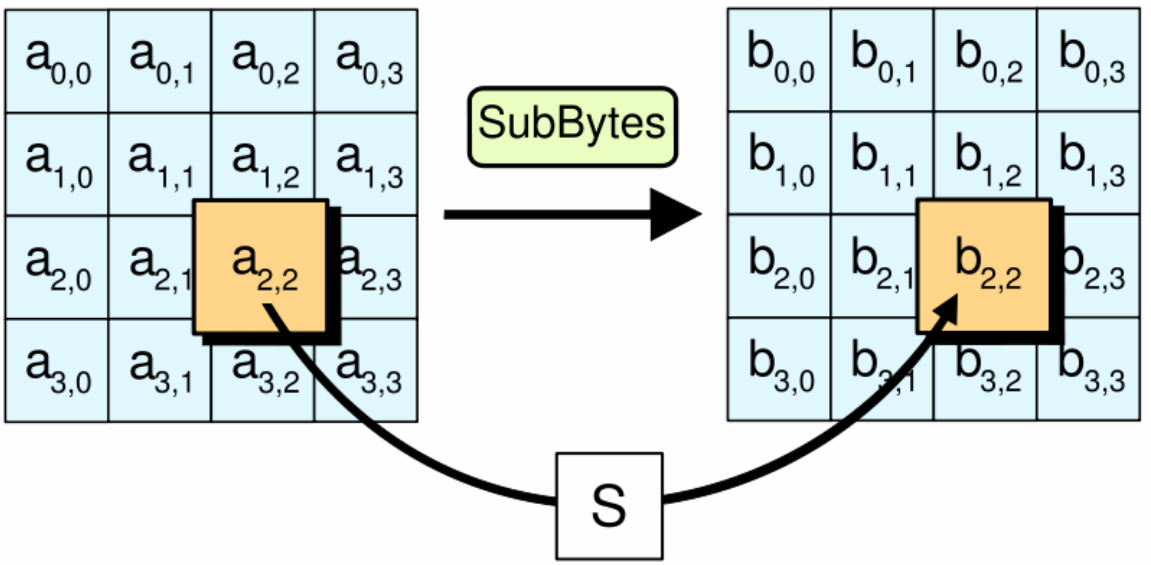

Figure 5. Byte Substitution

Source: http://en.wikipedia.org/

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.

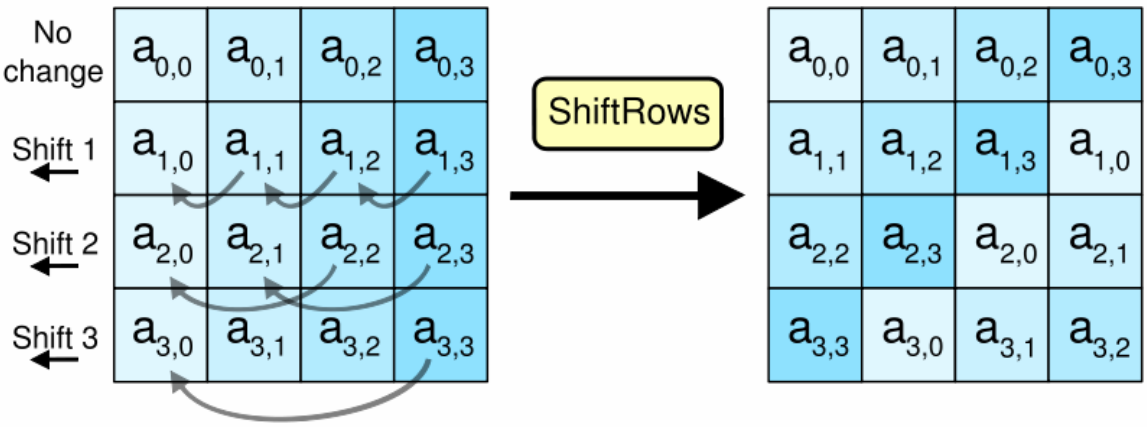

Figure 6. Shift Rows

Source: http://en.wikipedia.org/

4.1.4. Mix Columns

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

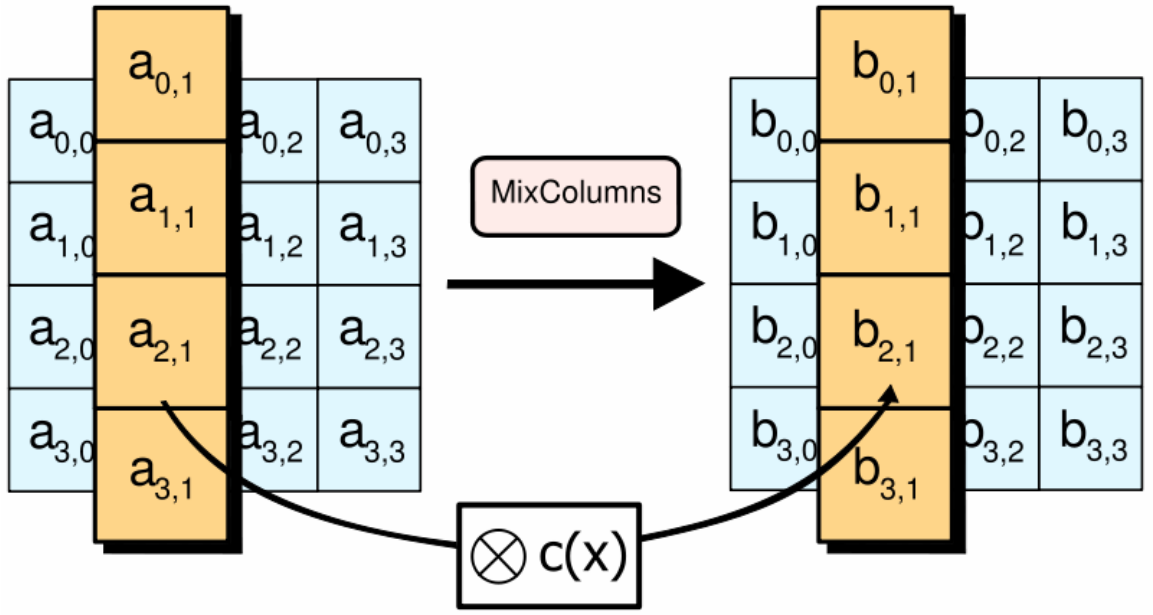

Figure 7. Mix Columns

Source: 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,

$$C(T, P) = (E(T * P) – E(T) * E(P)) / \sqrt{(Var(T) * Var(P))},$$

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 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).

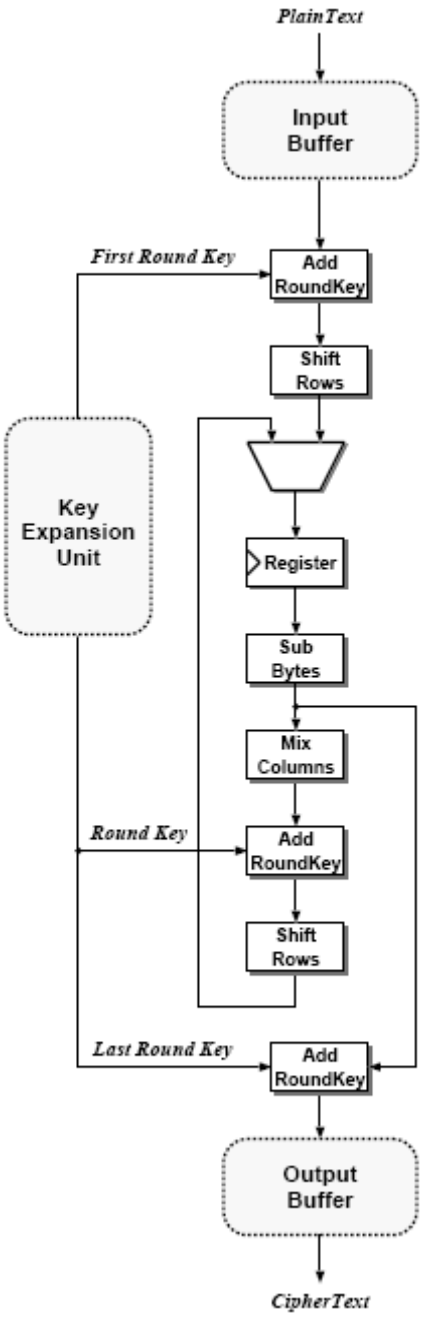

Figure 8. Fastcore Crypto-chip Block Diagram [20]

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, …, 10000 and j = 0, …, 2^L-1

The results are shown in figure 9 and 10.

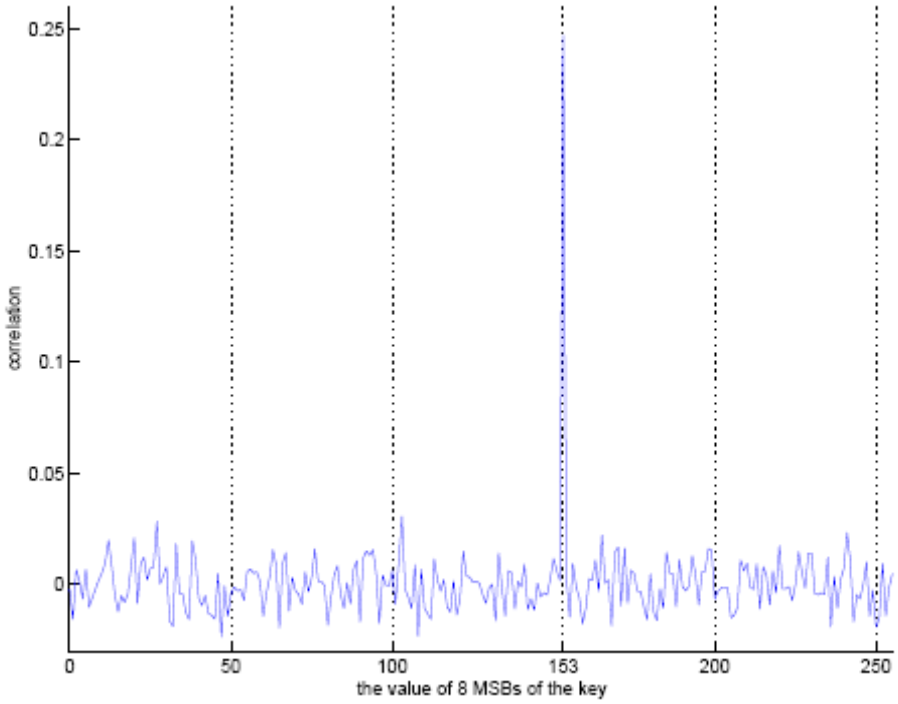

Figure 9. Correlation Between M_1 and M_4. [20]

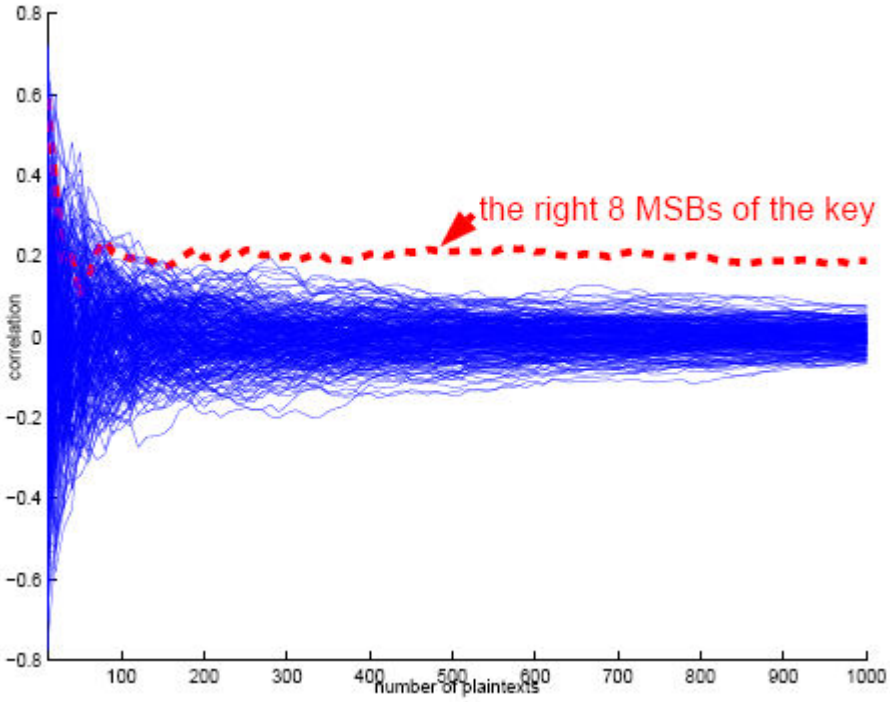

Figure 10. Correlation Between M_1 and M_4 For Different Measurements. [20]

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 Nx1000 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,…,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,…,N and j = 1,…,D.

The results are shown in Figure 11 and 12.

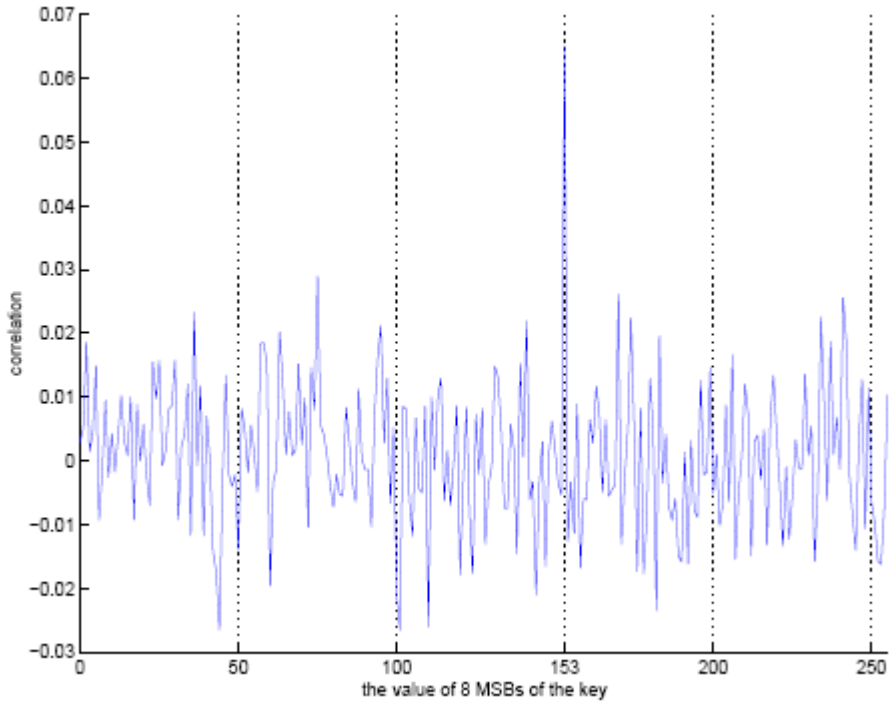

Figure 11. Correlation between M_4 and M_7. [20]

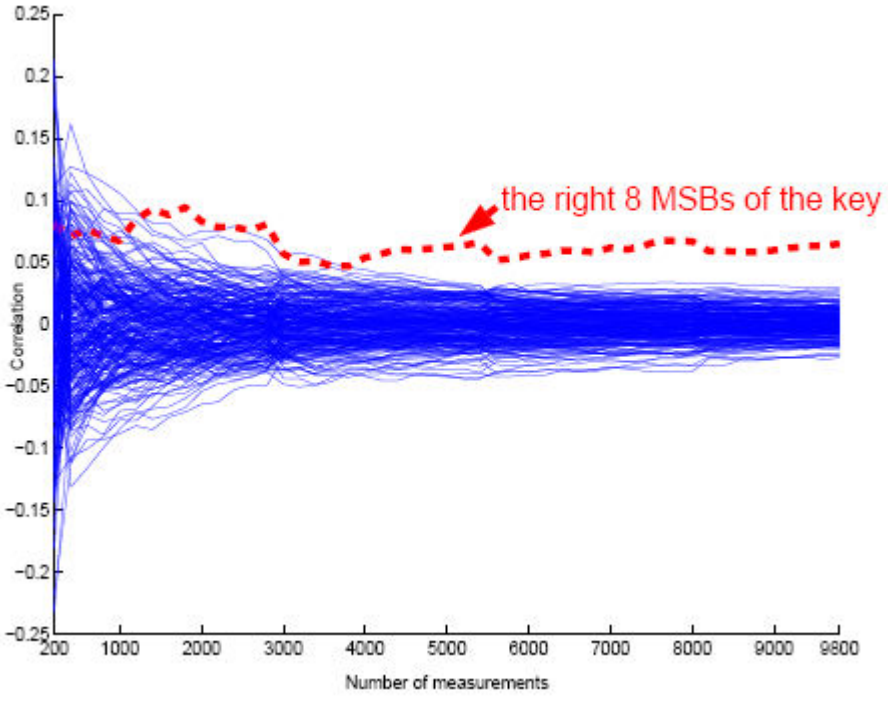

Figure 12. Correlation between M_4 and M_7 for different measurements. [20]

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.

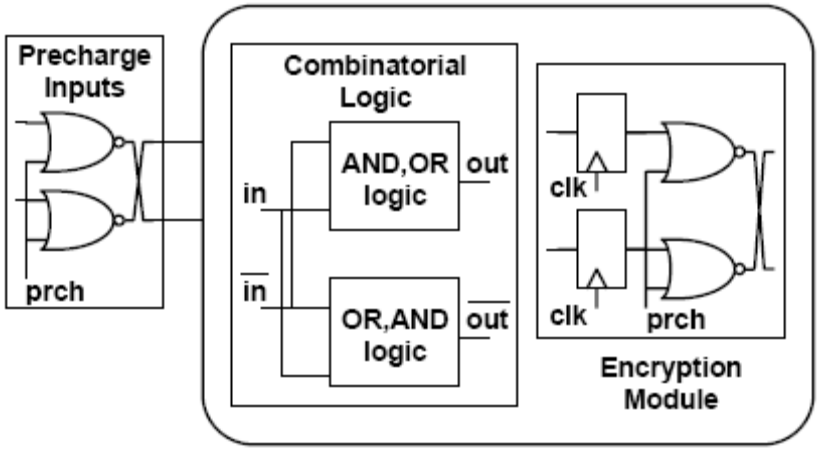

Figure 13. Wave Dynamic Differential Logic [23]

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].

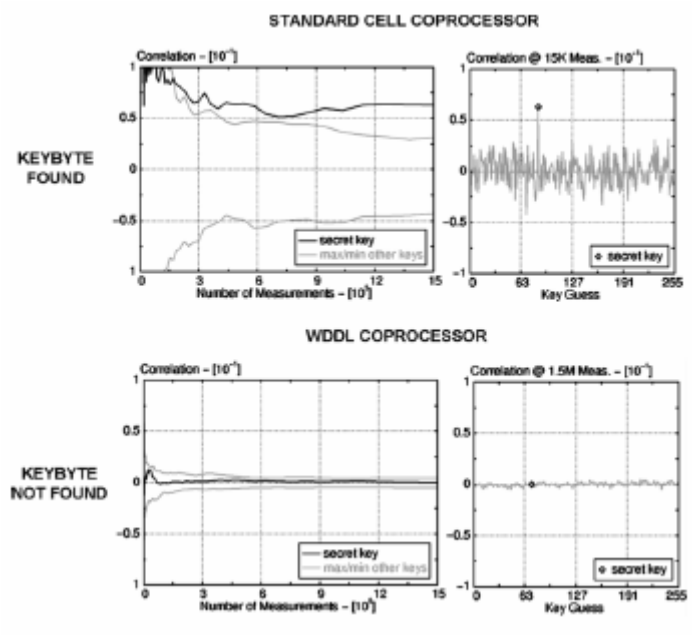

Figure 14. AES Power Graph with and without WDDL [10]

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.

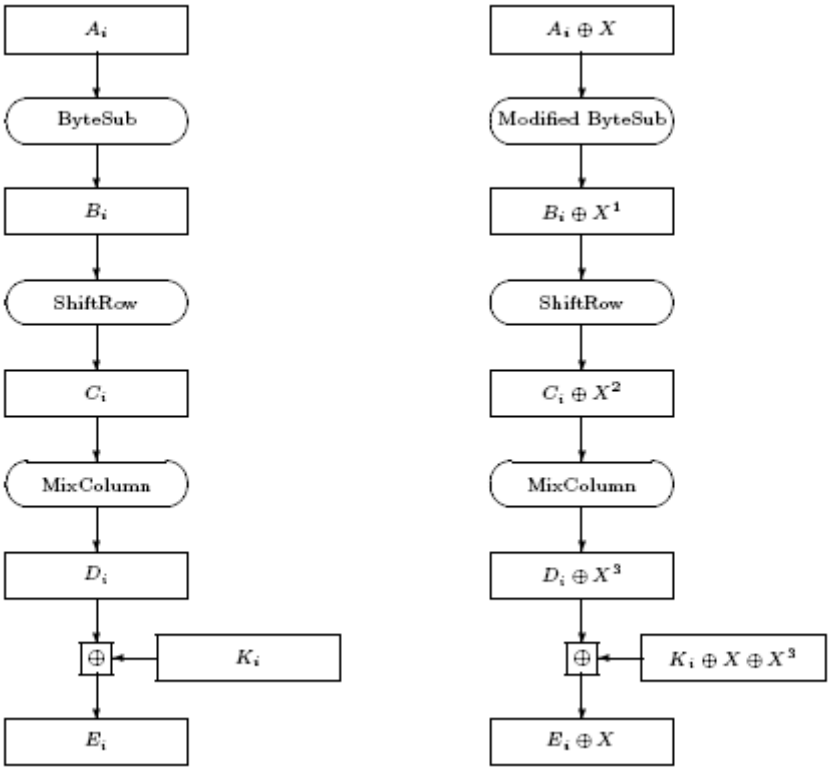

Figure 15. The i[th] Round of AES with and without Masking Countermeasure [9]

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).

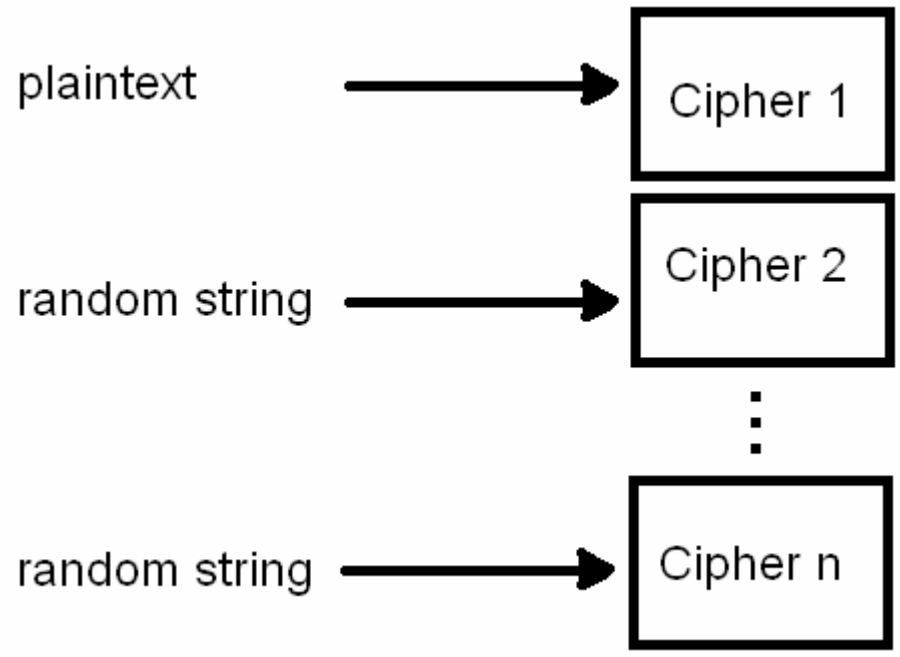

Figure 16. Initial Randomized Algorithm

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.

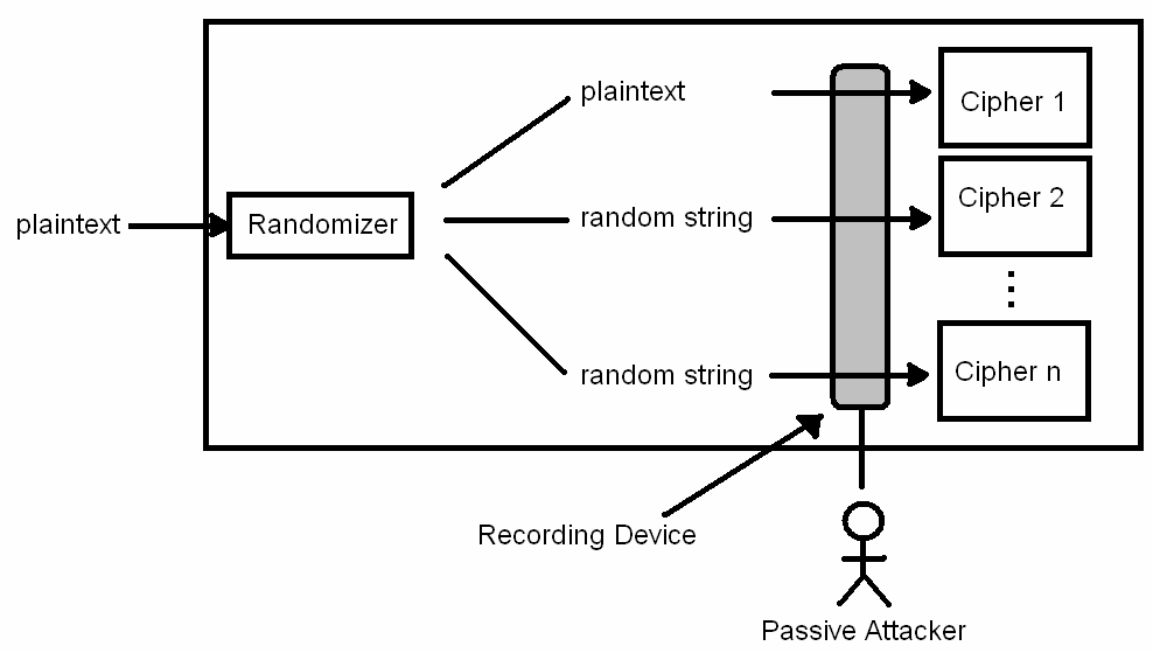

Figure 17. Flaw in Initial Algorithm

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.

$$H(IV, Counter + Offset_i) = One\ Time\ Pad,$$

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).

$$Message = (plaintext, Offset_i) \oplus one\ time\ pad$$

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.

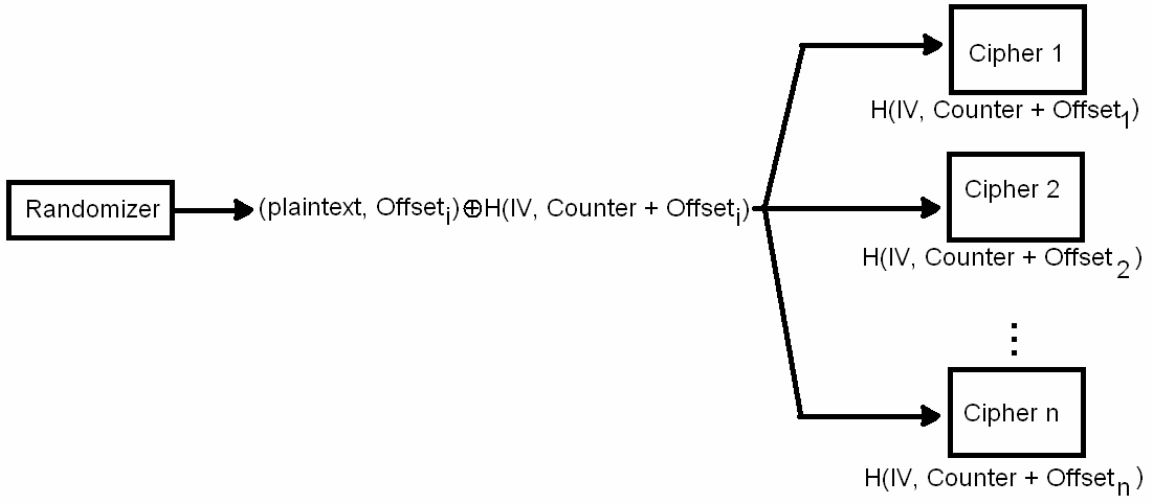

Figure 18. Revised Algorithm with Hash

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”

$Cipher_1$ 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:

stdEncryption	Performs the basic AES encryption algorithm.
stdDecryption	Performs the basic AES decryption algorithm
hashEncryption	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.
hashDecryption	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.
randomKeyEncryption	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.
randomKeyDecryption	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.

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: OÜ†HÞÔ›åÊ½â_Eb_f
DECRYPTION 1      Ciphertext: OÜ†HÞÔ›åÊ½â_Eb_f  Key: DefaultCipherKey
                  Plaintext: OriginalPlainTxt
      ...

ENCRYPTION N      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
                  Ciphertext: OÜ†HÞÔ›åÊ½â_Eb_f
DECRYPTION N      Ciphertext: OÜ†HÞÔ›åÊ½â_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

```
ENCRYPTION 1      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
                  Counter: 1  Offset: 423
                  Ciphertext: %j,žÆ¶ªÖ7Ÿ´¥O©?5³?`¶
DECRYPTION 1      Ciphertext: %j,žÆ¶ªÖ7Ÿ´¥O©?5³?`¶
                  Key: DefaultCipherKey   Counter: 1  Offset: 12
                  Plaintext: Þ«°&_¢.p_ä7™bZ„”
ENCRYPTION 2      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
                  Counter: 3  Offset: 72
                  Ciphertext: H®VE_rÐ__Ù_ÁHz)¹?Í]
DECRYPTION 2      Ciphertext: H®VE_rÐ__Ù_ÁHz)¹?Í]
                  Key: DefaultCipherKey   Counter: 3  Offset: 12
                  Plaintext: __•1ëãáønÃ×P_Â];
ENCRYPTION 3      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
                  Counter: 4  Offset: 423
                  Ciphertext: Âû[Ï?'Ý„ã¶__x?j1tõì¦
DECRYPTION 3      Ciphertext: Âû[Ï?'Ý„ã¶__x?j1tõì¦
                  Key: DefaultCipherKey   Counter: 4  Offset: 12
                  Plaintext: °_áz9óíÑtÅxE(Àf®
      ...
```

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.

```
ENCRYPTION 1      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
                  Counter: 1  Offset: 34
                  Ciphertext: ?_bcÀÐä+F?_Âà¶a_Š#¡j
DECRYPTION 1      Ciphertext: ?_bcÀÐä+F?_Âà¶a_Š#¡j
                  Key: é7IïÇXð^‰)¢W~Û^    Counter: 1  Offset: 12
                  Plaintext: 9VÁþ,í½_4Ì?IÎ(ƒè
ENCRYPTION 8      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
                  Counter: 8  Offset: 12
                  Ciphertext: VMÛ3_']|™Ã|Ÿ|À¦¹Õ!n·
DECRYPTION 8      Ciphertext: VMÛ3_']|™Ã|Ÿ|À¦¹Õ!n·
                  Key: DefaultCipherKey   Counter: 8  Offset: 12
                  Plaintext: OriginalPlainTxt
      ...
```

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.

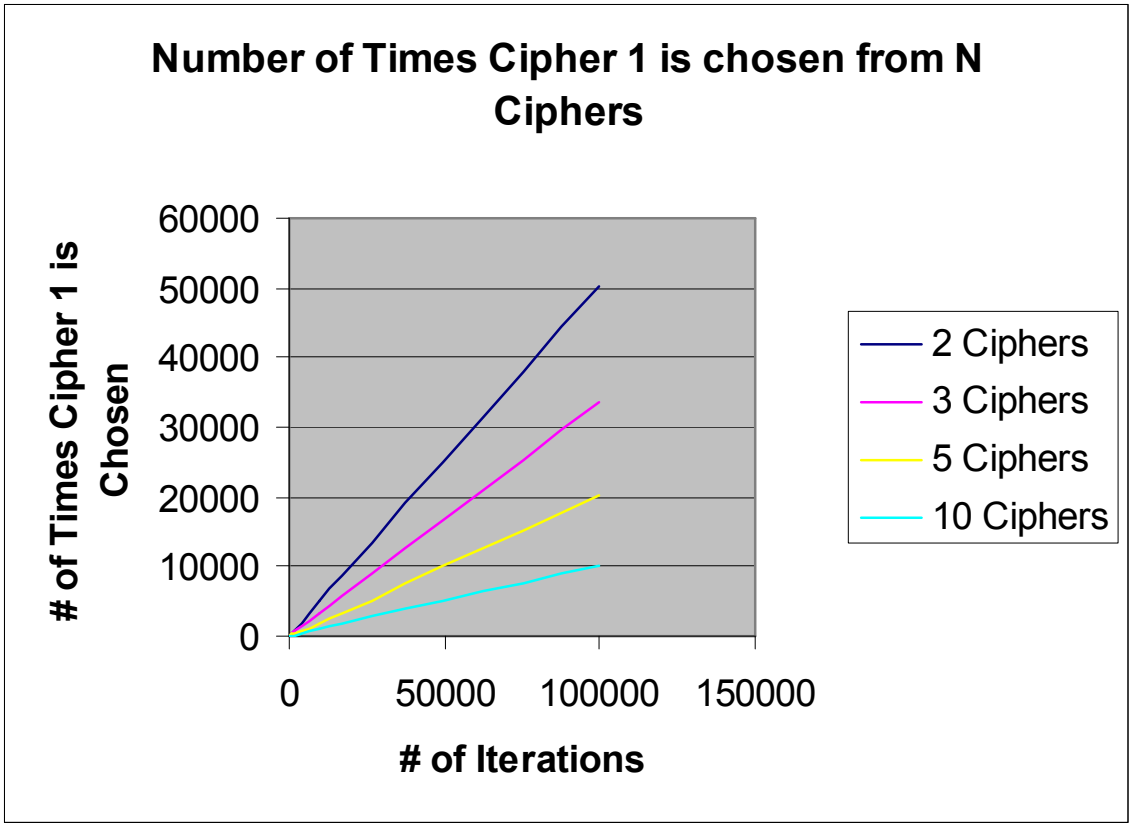

Figure 23. Number of Times Cipher 1 is chosen from N Ciphers in the Randomized Algorithm with Multiple Ciphers and Hash and Varying Key.

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.

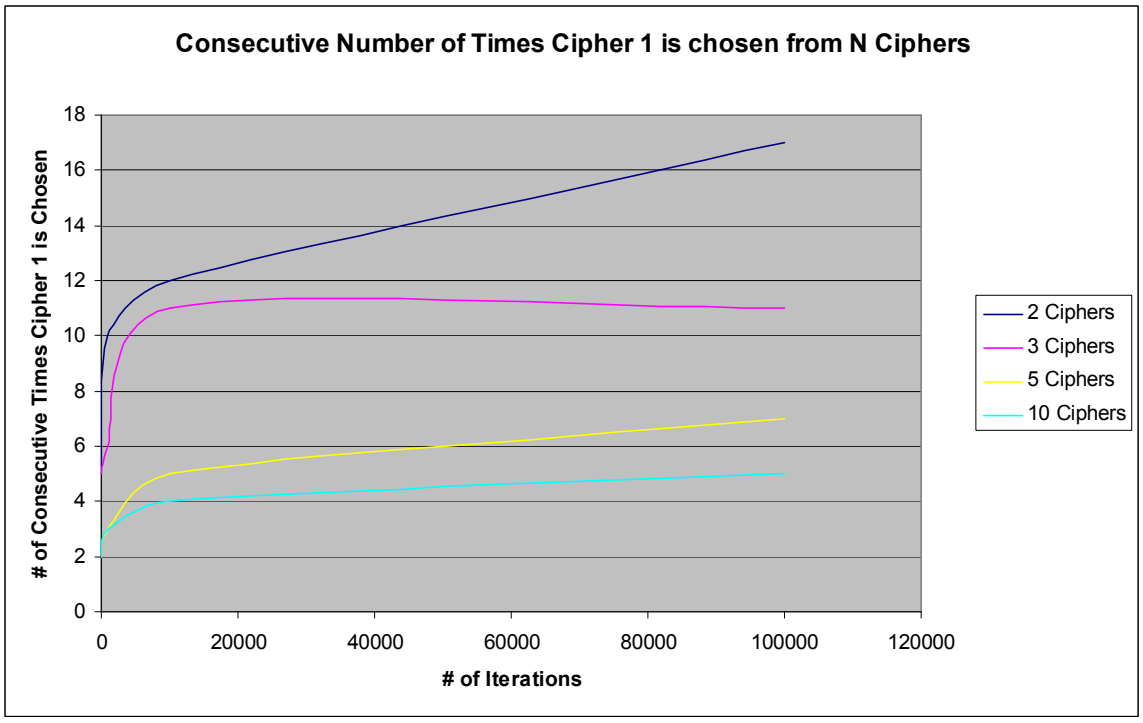

Figure 24. Consecutive Number of Times Cipher 1 is chosen from N Ciphers in the Randomized Algorithm with Multiple Ciphers and Hash and Varying Key.

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.

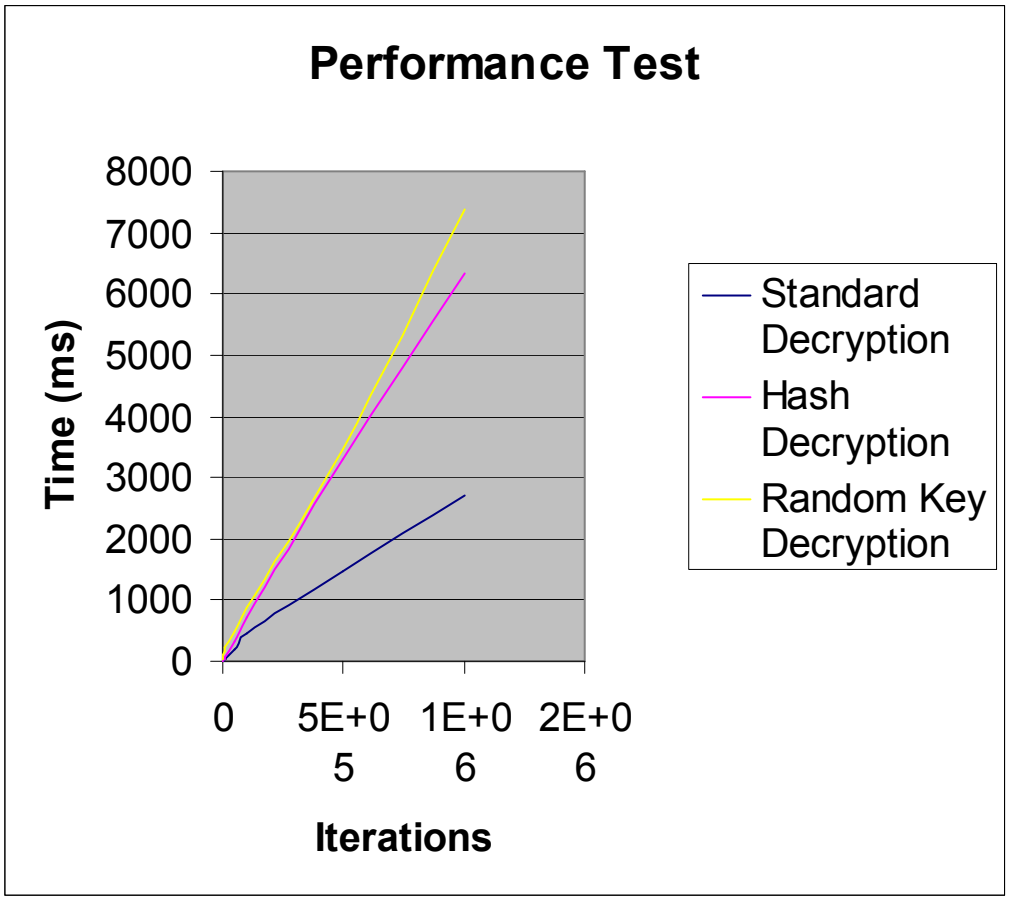

Figure 25. Performance Overhead Test

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);
```

```
            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 (counter-1 == 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
     * and Varying Key
     * @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);

   return output;
  }

/**
  * Concats 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
```

```
     * @param content to be hashed
     * @return 16 byte hash
     */
    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.");
```

```
    }

    /**
     * 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\u4906\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\u87E9\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 ^ s1;
        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
     */
    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 >>> 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       ) & 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.print("RK" + r + "=");
                  for (int i = 0; i < 4; i++) {
                        int w = rek[4*r + i];
                        System.out.print(" " + 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 |
            (pt[11] & 0xff)        ) ^ rek[2];
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    ];
ct[ 0] = (byte)(Se[(t0 >>> 24)       ] ^ (v >>> 24));
ct[ 1] = (byte)(Se[(t1 >>> 16) & 0xff] ^ (v >>> 16));
ct[ 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 |
            (ct[11] & 0xff)        ) ^ rdk[2];
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++) {
```

```
                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: OÜ†HÞÔ›åÊ½â□Eb□f
DECRYPTION 1      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f  Key: DefaultCipherKey
      Plaintext: OriginalPlainTxt

ENCRYPTION 2      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f
DECRYPTION 2      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f  Key: DefaultCipherKey
      Plaintext: OriginalPlainTxt

ENCRYPTION 3      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f
DECRYPTION 3      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f  Key: DefaultCipherKey
      Plaintext: OriginalPlainTxt

ENCRYPTION 4      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f
DECRYPTION 4      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f  Key: DefaultCipherKey
      Plaintext: OriginalPlainTxt

ENCRYPTION 5      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f
DECRYPTION 5      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f  Key: DefaultCipherKey
      Plaintext: OriginalPlainTxt
```

```
ENCRYPTION 6      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f
DECRYPTION 6      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f  Key: DefaultCipherKey
      Plaintext: OriginalPlainTxt

ENCRYPTION 7      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f
DECRYPTION 7      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f  Key: DefaultCipherKey
      Plaintext: OriginalPlainTxt

ENCRYPTION 8      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f
DECRYPTION 8      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f  Key: DefaultCipherKey
      Plaintext: OriginalPlainTxt

ENCRYPTION 9      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f
DECRYPTION 9      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f  Key: DefaultCipherKey
      Plaintext: OriginalPlainTxt

ENCRYPTION 10     Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f
DECRYPTION 10     Ciphertext: OÜ†HÞÔ›åÊ½â□Eb□f  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: Œ!J$ÃýÌlƒÚËÑ-É›□I—k
DECRYPTION 1      Ciphertext: Œ!J$ÃýÌlƒÚËÑ-É›□I—k     Key:
DefaultCipherKey  Counter: 1  Offset: 12  Plaintext: 3?¶ÕÜ7,ò¹□b,ywV

ENCRYPTION 2      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Counter: 2  Offset: 12  Ciphertext: ñûzI¾'ü qÊÙÂ€9□Iã)
DECRYPTION 2      Ciphertext: ñûzI¾'ü qÊÙÂ€9□Iã)      Key:
DefaultCipherKey  Counter: 2  Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 3      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Counter: 3  Offset: 123 Ciphertext: □¥fpZyà:égýÿ-KßG[0,M
DECRYPTION 3      Ciphertext: □¥fpZyà:égýÿ-KßG[0,M    Key:
DefaultCipherKey  Counter: 3  Offset: 12  Plaintext: £°□èìÄ„ìÅ‘ïåè□âO

ENCRYPTION 4      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Counter: 4  Offset: 123 Ciphertext: C¸FdÀA.ÄøswË¾□•□˜[
DECRYPTION 4      Ciphertext: C¸FdÀA.ÄøswË¾□•□˜[      Key:
DefaultCipherKey  Counter: 4  Offset: 12  Plaintext: J¡¼=?~u3ÄI¬<4□ËÓ

ENCRYPTION 5      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Counter: 5  Offset: 123 Ciphertext: □Õ6ÝM °‘?iª□ý□ëH#□Ê
DECRYPTION 5      Ciphertext: □Õ6ÝM °‘?iª□ý□ëH#□Ê     Key:
DefaultCipherKey  Counter: 5  Offset: 12  Plaintext: □□ú,Òê±-□iŸ□-¤Lß
```

ENCRYPTION 6 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 6 Offset: 12 Ciphertext: ‹ÏváÄ□ð¬¼¦R□□ñT+.›<ù
DECRYPTION 6 Ciphertext: ‹ÏváÄ□ð¬¼¦R□□ñT+.›<ù Key:
DefaultCipherKey Counter: 6 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 7 Plaintext: OriginalPlainTxt 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<^^¨,ï^(ÏB

ENCRYPTION 8 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 8 Offset: 124 Ciphertext: ¡G†¤î› ëa³□¬Ô±§Åb□Ô†
DECRYPTION 8 Ciphertext: ¡G†¤î› ëa³□¬Ô±§Åb□Ô† Key:
DefaultCipherKey Counter: 8 Offset: 12 Plaintext: -¦¦—¶v□¥Ùð2@Ä□oâ

ENCRYPTION 9 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 9 Offset: 124 Ciphertext: ´Î†ýû□ ½×¹□□'Ýÿ□¢:š¬
DECRYPTION 9 Ciphertext: ´Î†ýû□ ½×¹□□'Ýÿ□¢:š¬ Key:
DefaultCipherKey Counter: 9 Offset: 12 Plaintext: x
®a6âÏJ□í□Ã&T|®

ENCRYPTION 10 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 10 Offset: 12 Ciphertext: <È?©s□¹è□□‹5¹"ÑÃüxƒ8
DECRYPTION 10 Ciphertext: <È?©s□¹è□□‹5¹"ÑÃüxƒ8 Key:
DefaultCipherKey Counter: 10 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 11 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 11 Offset: 123 Ciphertext: —7¨BK±ê'´|üãçž³'°+¿

DECRYPTION 11 Ciphertext: —7¨BK±ê‘´|üãçž³'°+¿ Key:
DefaultCipherKey Counter: 11 Offset: 12 Plaintext: Zoö.vpÜObe□bÁ+Ý

ENCRYPTION 12 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 12 Offset: 12 Ciphertext: 8k¡Ïw·'ŒY±*:z¦šZ□¿Oo
DECRYPTION 12 Ciphertext: 8k¡Ïw·'ŒY±*:z¦šZ□¿Oo Key:
DefaultCipherKey Counter: 12 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 13 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 13 Offset: 12 Ciphertext: ¤Æ□êë□”®••Î ðZ®UêDoÑ
DECRYPTION 13 Ciphertext: ¤Æ□êë□”®••Î ðZ®UêDoÑ Key:
DefaultCipherKey Counter: 13 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 14 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 14 Offset: 123 Ciphertext: h½08'a¶}=”¤M˜~VÜ–¼µ7
DECRYPTION 14 Ciphertext: h½08'a¶}=”¤M˜~VÜ–¼µ7 Key:
DefaultCipherKey Counter: 14 Offset: 12 Plaintext: ¶□□¿ÆQ□vÓ²þÄ«kß

ENCRYPTION 15 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 15 Offset: 124 Ciphertext: Æ9è¨‰ånî,0e0Í‡ õ□;:Á
DECRYPTION 15 Ciphertext: Æ9è¨‰ånî,0e0Í‡ õ□;:Á Key:
DefaultCipherKey Counter: 15 Offset: 12 Plaintext: b¢IÐ@X²×$Š&□üÓ,-

ENCRYPTION 16 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 16 Offset: 124 Ciphertext: °®8□õr¾Uj¢XE8±□□V°fC
DECRYPTION 16 Ciphertext: °®8□õr¾Uj¢XE8±□□V°fC Key:
DefaultCipherKey Counter: 16 Offset: 12 Plaintext: ðA7ÐõÙ¡rƒ□}□×Lz•

ENCRYPTION 17 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 17 Offset: 123 Ciphertext: °®8õr¾Uj¢XE8±□□V°fC
DECRYPTION 17 Ciphertext: °®8õr¾Uj¢XE8±□□V°fC Key:
DefaultCipherKey Counter: 17 Offset: 12 Plaintext: Hç□×Ä”/R6¬±6AÃó□

ENCRYPTION 18 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 18 Offset: 12 Ciphertext: µ¼Eõú`Ã¬Ô½ïèûËl5»ö`Ö
DECRYPTION 18 Ciphertext: µ¼Eõú`Ã¬Ô½ïèûËl5»ö`Ö Key:
DefaultCipherKey Counter: 18 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 19 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 19 Offset: 124 Ciphertext: \t□œ□¨›Æ{(eRª=Z?□—Àp
DECRYPTION 19 Ciphertext: \t□œ□¨›Æ{(eRª=Z?□—Àp Key:
DefaultCipherKey Counter: 19 Offset: 12 Plaintext: \□~À4³Œ~¸Åbëœ´

ENCRYPTION 20 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 20 Offset: 12 Ciphertext: ^ÂØ_□-^□ä□å□«eý?□t"
DECRYPTION 20 Ciphertext: ^ÂØ_□-^□ä□å□«eý?□t" Key:
DefaultCipherKey Counter: 20 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 21 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 21 Offset: 12 Ciphertext: D'›¤M□ø-6□Îã+’Éá¯0þ
DECRYPTION 21 Ciphertext: D'›¤M□ø-6□Îã+’Éá¯0þ Key:
DefaultCipherKey Counter: 21 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 22 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 22 Offset: 123 Ciphertext: ’ˆ»ÛÝT=†‰ß¦?—ÒÂ@□¼«*
DECRYPTION 22 Ciphertext: ’ˆ»ÛÝT=†‰ß¦?—ÒÂ@□¼«* Key:
DefaultCipherKey Counter: 22 Offset: 12 Plaintext: â□Ë^%‰B$›×¨)□à|ß

ENCRYPTION 23 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 23 Offset: 123 Ciphertext: þa†¡±½ ÿžõ*"Àöša„«l‘
DECRYPTION 23 Ciphertext: þa†¡±½ ÿžõ*"Àöša„«l‘ Key:
DefaultCipherKey Counter: 23 Offset: 12 Plaintext: Òi□yàL=à?5‘U,1”O

ENCRYPTION 24 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 24 Offset: 124 Ciphertext: !□š nØ□□.;3*ßÖR^ìK□A
DECRYPTION 24 Ciphertext: !□š nØ□□.;3*ßÖR^ìK□A Key:
DefaultCipherKey Counter: 24 Offset: 12 Plaintext: U-·ô£e§6-¿□□we

ENCRYPTION 25 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 25 Offset: 124 Ciphertext: ~ûá×1'g‡8ð!EC½†r□”;d
DECRYPTION 25 Ciphertext: ~ûá×1'g‡8ð!EC½†r□”;d Key:
DefaultCipherKey Counter: 25 Offset: 12 Plaintext: ð©□áµ%□,åïw,GAâ

ENCRYPTION 26 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 26 Offset: 12 Ciphertext: ?FD¶pšÂçª"À¯:‡+-^UgÈ
DECRYPTION 26 Ciphertext: ?FD¶pšÂçª"À¯:‡+-^UgÈ Key:
DefaultCipherKey Counter: 26 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 27 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 27 Offset: 124 Ciphertext: “ØÐ□Ü□VW?£□p.□Ž(YŽ·¶
DECRYPTION 27 Ciphertext: “ØÐ□Ü□VW?£□p.□Ž(YŽ·¶ Key:
DefaultCipherKey Counter: 27 Offset: 12 Plaintext: □Œ¸ê§†õÈrXW—ý□|å

ENCRYPTION 28 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 28 Offset: 124 Ciphertext: :?ÞÅuQX-X□©¨□d DTÓ

DECRYPTION 28 Ciphertext: :?ÞÅuQX-X□©¨□d DTÓ Key:
DefaultCipherKey Counter: 28 Offset: 12 Plaintext: þ□%□<□□JáüiÎãt□{

ENCRYPTION 29 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 29 Offset: 124 Ciphertext: ~‹os1Wé'Þ'ÿõÝheÒzÕY
DECRYPTION 29 Ciphertext: ~‹os1Wé'Þ'ÿõÝheÒzÕY Key:
DefaultCipherKey Counter: 29 Offset: 12 Plaintext: vÊª-$óïhÕI/~O□□□

ENCRYPTION 30 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 30 Offset: 124 Ciphertext: ÒNcÎ?'å›×hµqHàæß®uæ^
DECRYPTION 30 Ciphertext: ÒNcÎ?'å›×hµqHàæß®uæ^ Key:
DefaultCipherKey Counter: 30 Offset: 12 Plaintext: žB&Â=˜ ,^q'ÖÎ§ÓÉ

ENCRYPTION 31 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 31 Offset: 12 Ciphertext: F[+Õ ‡-ƒ@+c;h@?□Ó¤âŽ
DECRYPTION 31 Ciphertext: F[+Õ ‡-ƒ@+c;h@?□Ó¤âŽ Key:
DefaultCipherKey Counter: 31 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 32 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 32 Offset: 12 Ciphertext: 8□µ□wÚ3Rž□Ä, ´ž>eX,°
DECRYPTION 32 Ciphertext: 8□µ□wÚ3Rž□Ä, ´ž>eX,° Key:
DefaultCipherKey Counter: 32 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 33 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 33 Offset: 124 Ciphertext: "cƒšÛ¿□òÕzü□?§?dþ>^-
DECRYPTION 33 Ciphertext: "cƒšÛ¿□òÕzü□?§?dþ>^- Key:
DefaultCipherKey Counter: 33 Offset: 12 Plaintext: äwÁÚh□¯l¤;Hndö<□

ENCRYPTION 34 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 34 Offset: 124 Ciphertext: îTÒ¥¡^TÌY!ï□□lký□¡□O
DECRYPTION 34 Ciphertext: îTÒ¥¡^TÌY!ï□□lký□¡□O Key:
DefaultCipherKey Counter: 34 Offset: 12 Plaintext: □J/+£ó^Ä&«îëüÝÓ;

ENCRYPTION 35 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 35 Offset: 124 Ciphertext: ?*ï†pöiì□”ªt05à□‹·
DECRYPTION 35 Ciphertext: ?*ï†pöiì□”ªt05à□‹· Key:
DefaultCipherKey Counter: 35 Offset: 12 Plaintext: ^pHÊ™ãGgæ+l×YËçë

ENCRYPTION 36 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 36 Offset: 12 Ciphertext: ©Îp□æ□öwyïFäßÛÛù¬fs´
DECRYPTION 36 Ciphertext: ©Îp□æ□öwyïFäßÛÛù¬fs´ Key:
DefaultCipherKey Counter: 36 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 37 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 37 Offset: 12 Ciphertext: ž□^ãÑÁØ?ÝBêý Ì¤H□S©□
DECRYPTION 37 Ciphertext: ž□^ãÑÁØ?ÝBêý Ì¤H□S©□ Key:
DefaultCipherKey Counter: 37 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 38 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 38 Offset: 12 Ciphertext: r"ð□=þvz□□F§³—'¾{Ê˜®
DECRYPTION 38 Ciphertext: r"ð□=þvz□□F§³—'¾{Ê˜® Key:
DefaultCipherKey Counter: 38 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 39 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 39 Offset: 124 Ciphertext: Ä¹□?‹e„á□þþ□ª‰éæçg
DECRYPTION 39 Ciphertext: Ä¹□?‹e„á□þþ□ª‰éæçg Key:
DefaultCipherKey Counter: 39 Offset: 12 Plaintext: ÚÈ□□XQÌ±Ñ[□ÑÝ«é3

ENCRYPTION 40 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 40 Offset: 123 Ciphertext: Ä¹□Ž‹e„á□þþ□ª‰éæçg
DECRYPTION 40 Ciphertext: Ä¹□Ž‹e„á□þþ□ª‰éæçg Key:
DefaultCipherKey Counter: 40 Offset: 12 Plaintext: ø„"[‰ÊøÔLÇÞ □ã¶□

ENCRYPTION 41 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 41 Offset: 12 Ciphertext: ÿ□ s°Þ?□°:- R¬ÁèfyðX
DECRYPTION 41 Ciphertext: ÿ□ s°Þ?□°:- R¬ÁèfyðX Key:
DefaultCipherKey Counter: 41 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 42 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 42 Offset: 12 Ciphertext: 2{Û<}§]]æÅët‹¹Òž-ð+
DECRYPTION 42 Ciphertext: 2{Û<}§]]æÅët‹¹Òž-ð+ Key:
DefaultCipherKey Counter: 42 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 43 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 43 Offset: 124 Ciphertext: gQ□¿(?•Ý¨¼□z-«^ÑÇÏ?H
DECRYPTION 43 Ciphertext: gQ□¿(?•Ý¨¼□z-«^ÑÇÏ?H Key:
DefaultCipherKey Counter: 43 Offset: 12 Plaintext: Í,c9-¸òaK-N¥ÜÚAK

ENCRYPTION 44 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 44 Offset: 123 Ciphertext: gQ□¾(?•Ý¨¼□z-«^ÑÇÏ?H
DECRYPTION 44 Ciphertext: gQ□¾(?•Ý¨¼□z-«^ÑÇÏ?H Key:
DefaultCipherKey Counter: 44 Offset: 12 Plaintext: ü□ôR?□ø}óÙÏá?b|*

ENCRYPTION 45 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 45 Offset: 123 Ciphertext: ‰j,²Æ¶ªÖ7Ÿ´¥O©?5³?`¶

DECRYPTION 45 Ciphertext: %j,²Æ¶ªÖ7Ÿ´¥O©?5³?`¶ Key:
DefaultCipherKey Counter: 45 Offset: 12 Plaintext: @□F*al ”÷]r~1□

ENCRYPTION 46 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 46 Offset: 123 Ciphertext: ¸]$Ï÷?¢ªí□□³2Ó□“j□h
DECRYPTION 46 Ciphertext: ¸]$Ï÷?¢ªí□□³2Ó□“j□h Key:
DefaultCipherKey Counter: 46 Offset: 12 Plaintext: §-□4!□M3ùáŽýÝ–}[

ENCRYPTION 47 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 47 Offset: 124 Ciphertext: Âû[â?'Ý„ã¶□x?j1tõì¦
DECRYPTION 47 Ciphertext: Âû[â?'Ý„ã¶□x?j1tõì¦ Key:
DefaultCipherKey Counter: 47 Offset: 12 Plaintext: Çðç□²_ÿÒ¬□×Ûx®’Ý

ENCRYPTION 48 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 48 Offset: 123 Ciphertext: Âû[ã?'Ý„ã¶□x?j1tõì¦
DECRYPTION 48 Ciphertext: Âû[ã?'Ý„ã¶□x?j1tõì¦ Key:
DefaultCipherKey Counter: 48 Offset: 12 Plaintext: 7ä„5N□*íóc□ª□Â~?

ENCRYPTION 49 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 49 Offset: 123 Ciphertext: □à□)U<„Q□·¹¹j™€|D»êT
DECRYPTION 49 Ciphertext: □à□)U<„Q□·¹¹j™€|D»êT Key:
DefaultCipherKey Counter: 49 Offset: 12 Plaintext: A¨ùûe¶7,A¤ßkg—xy

ENCRYPTION 50 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 50 Offset: 124 Ciphertext: Uh)€□´¯ùÀ› ¨Á@\é]l?m
DECRYPTION 50 Ciphertext: Uh)€□´¯ùÀ› ¨Á@\é]l?m Key:
DefaultCipherKey Counter: 50 Offset: 12 Plaintext: ¿?ÓÕz□÷§Ä¿©]8°é

ENCRYPTION 51 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 51 Offset: 12 Ciphertext: T1?‹□í¹ñ□Õ¤Ž5(²?«œ•ã
DECRYPTION 51 Ciphertext: T1?‹□í¹ñ□Õ¤Ž5(²?«œ•ã Key:
DefaultCipherKey Counter: 51 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 52 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 52 Offset: 123 Ciphertext: ¸8Z□÷äÜ|@» ¬Ý□□®²-»v
DECRYPTION 52 Ciphertext: ¸8Z□÷äÜ|@» ¬Ý□□®²-»v Key:
DefaultCipherKey Counter: 52 Offset: 12 Plaintext: ÜÞ£kP˜À!!ÿ´'üžh

ENCRYPTION 53 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 53 Offset: 124 Ciphertext: § □¥è|›Ù”Ï>=Ãûi‡ü□ô
DECRYPTION 53 Ciphertext: § □¥è|›Ù”Ï>=Ãûi‡ü□ô Key:
DefaultCipherKey Counter: 53 Offset: 12 Plaintext: ÃxŒT˜±É#«<Ú¸Ë¥×¾

ENCRYPTION 54 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 54 Offset: 123 Ciphertext: § □¤è|›Ù”Ï>=Ãûi‡ü□ô
DECRYPTION 54 Ciphertext: § □¤è|›Ù”Ï>=Ãûi‡ü□ô Key:
DefaultCipherKey Counter: 54 Offset: 12 Plaintext: /A2Q□□Ž?□aáñI□7!

ENCRYPTION 55 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 55 Offset: 123 Ciphertext: †¸DCZ>:?¯ýåé’.¼öÎc¦
DECRYPTION 55 Ciphertext: †¸DCZ>:?¯ýåé’.¼öÎc¦ Key:
DefaultCipherKey Counter: 55 Offset: 12 Plaintext: &)?□]9U?×´^‡XR‘ü

ENCRYPTION 56 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 56 Offset: 12 Ciphertext: ??`2ßQæM…¼□ð.a1»ö°uí
DECRYPTION 56 Ciphertext: ??`2ßQæM…¼□ð.a1»ö°uí Key:
DefaultCipherKey Counter: 56 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 57 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 57 Offset: 124 Ciphertext: I ¼Â□|:²□±Äm¢ãDnûÒ Ò

DECRYPTION 57 Ciphertext: I ¼Â□|:²□±Äm¢ãDnûÒ Ò Key:

DefaultCipherKey Counter: 57 Offset: 12 Plaintext: ‘«®□^s~□¸Ä-8›˜°4

ENCRYPTION 58 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 58 Offset: 123 Ciphertext: I ¼Ã□|:²□±Äm¢ãDnûÒ Ò

DECRYPTION 58 Ciphertext: I ¼Ã□|:²□±Äm¢ãDnûÒ Ò Key:

DefaultCipherKey Counter: 58 Offset: 12 Plaintext: mÀí¯*÷□¬Þ3i—ü□e□

ENCRYPTION 59 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 59 Offset: 123 Ciphertext: Ç%Þ□^ùXzÔ^Skñ)Œh¬›

DECRYPTION 59 Ciphertext: Ç%Þ□^ùXzÔ^Skñ)Œh¬› Key:

DefaultCipherKey Counter: 59 Offset: 12 Plaintext: 6?/O¤ª□êsá0□ý-Tö

ENCRYPTION 60 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 60 Offset: 12 Ciphertext: ?Ç^›Ò□ØèÙ2´□äì□qø□ÙÏ

DECRYPTION 60 Ciphertext: ?Ç^›Ò□ØèÙ2´□äì□qø□ÙÏ Key:

DefaultCipherKey Counter: 60 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 61 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 61 Offset: 124 Ciphertext: ¯}²□à¡4ré□`·□°5NÆ□«_

DECRYPTION 61 Ciphertext: ¯}²□à¡4ré□`·□°5NÆ□«_ Key:

DefaultCipherKey Counter: 61 Offset: 12 Plaintext: æ#Þ□□®ØÌ@-É0SŠç^

ENCRYPTION 62 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 62 Offset: 123 Ciphertext: ¯}²□à¡4ré□`·□°5NÆ□«_

DECRYPTION 62 Ciphertext: ¯}²□à¡4ré□`·□°5NÆ□«_ Key:
DefaultCipherKey Counter: 62 Offset: 12 Plaintext: Šú·.IÎlz_tÇÛ□[é

ENCRYPTION 63 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 63 Offset: 12 Ciphertext: H®Vy□rÐ□□Ù-ÁHz)¹?Í]
DECRYPTION 63 Ciphertext: H®Vy□rÐ□□Ù-ÁHz)¹?Í] Key:
DefaultCipherKey Counter: 63 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 64 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 64 Offset: 123 Ciphertext: žK¼]Ñ—:*µ-Û%×J®,¼½˜□
DECRYPTION 64 Ciphertext: žK¼]Ñ—:*µ-Û%×J®,¼½˜□ Key:
DefaultCipherKey Counter: 64 Offset: 12 Plaintext: ÐÀè–Ý~£·€qk□·MÁ¬

ENCRYPTION 65 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 65 Offset: 12 Ciphertext: 8+õQw÷sYp0?¹x4GZ□?]¾
DECRYPTION 65 Ciphertext: 8+õQw÷sYp0?¹x4GZ□?]¾ Key:
DefaultCipherKey Counter: 65 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 66 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 66 Offset: 124 Ciphertext: □©ì{Xujr+Kñ¿Ò,éë®Iß¬
DECRYPTION 66 Ciphertext: □©ì{Xujr+Kñ¿Ò,éë®Iß¬ Key:
DefaultCipherKey Counter: 66 Offset: 12 Plaintext: /□?w¥%N²2Ý;¥0D□×

ENCRYPTION 67 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 67 Offset: 124 Ciphertext: î 0C¡|¶IXË #h"tÜ|□¬
DECRYPTION 67 Ciphertext: î 0C¡|¶IXË #h"tÜ|□¬ Key:
DefaultCipherKey Counter: 67 Offset: 12 Plaintext: Œ$Úï§n3□/œ|□xÓn

ENCRYPTION 68 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 68 Offset: 124 Ciphertext: ÔŒñ□›Pw□Ïj?8□ò³øú²!

DECRYPTION 68 Ciphertext: ÔŒñ□›Pw□Ïj?8□ò³øú²! Key:

DefaultCipherKey Counter: 68 Offset: 12 Plaintext: ø×W÷ÇL¸=+õ¾□'ý"³

ENCRYPTION 69 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 69 Offset: 123 Ciphertext: ÔŒñ□›Pw□Ïj?8□ò³øú²!

DECRYPTION 69 Ciphertext: ÔŒñ□›Pw□Ïj?8□ò³øú²! Key:

DefaultCipherKey Counter: 69 Offset: 12 Plaintext: o\Xž-…^÷|Wk‰ðÕ

ENCRYPTION 70 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 70 Offset: 124 Ciphertext: o□Íý ÓKðxÀAÒy6Ö(†¹¹£

DECRYPTION 70 Ciphertext: o□Íý ÓKðxÀAÒy6Ö(†¹¹£ Key:

DefaultCipherKey Counter: 70 Offset: 12 Plaintext: •Y^ž÷;Ç»„;]©ƒÍhL

ENCRYPTION 71 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 71 Offset: 12 Ciphertext: ."!„aþ§ŠßmÚ?—&o·3/I□

DECRYPTION 71 Ciphertext: ."!„aþ§ŠßmÚ?—&o·3/I□ Key:

DefaultCipherKey Counter: 71 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 72 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 72 Offset: 123 Ciphertext: T?k5□]í:□L^ÍD.È?æÖŒœ

DECRYPTION 72 Ciphertext: T?k5□]í:□L^ÍD.È?æÖŒœ Key:

DefaultCipherKey Counter: 72 Offset: 12 Plaintext: fË□|z-ý¢Ñy§¢ãÍ)\

ENCRYPTION 73 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 73 Offset: 124 Ciphertext: Ç□Òáˆ ÚTáj1Á□SÒ‡ûìÅ□{

DECRYPTION 73 Ciphertext: Ç□Òáˆ ÚTáj1Á□SÒ‡ûìÅ□{ Key:

DefaultCipherKey Counter: 73 Offset: 12 Plaintext: 'OÛ"□"□ý©=3,ûc…¨

ENCRYPTION 74 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 74 Offset: 123 Ciphertext: Ç□Òà^ÚTáj1Á□SÒ‡ûìÅ□{
DECRYPTION 74 Ciphertext: Ç□Òà^ÚTáj1Á□SÒ‡ûìÅ□{ Key:
DefaultCipherKey Counter: 74 Offset: 12 Plaintext: ë®□Ý³-z)!(?´□□pq

ENCRYPTION 75 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 75 Offset: 124 Ciphertext: 7□È3xÎN1®uÀ+¬{~úŠ?ã□
DECRYPTION 75 Ciphertext: 7□È3xÎN1®uÀ+¬{~úŠ?ã□ Key:
DefaultCipherKey Counter: 75 Offset: 12 Plaintext: ø)q¡ŽUGQe MñÔ□ã

ENCRYPTION 76 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 76 Offset: 12 Ciphertext: ³¶ÐÂüjVÁKÍõ"èóJ°-¤+À
DECRYPTION 76 Ciphertext: ³¶ÐÂüjVÁKÍõ"èóJ°-¤+À Key:
DefaultCipherKey Counter: 76 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 77 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 77 Offset: 123 Ciphertext: ‘ÆDÙÞ□ÂÝ□ùÁ_jƒI57§
DECRYPTION 77 Ciphertext: ‘ÆDÙÞ□ÂÝ□ùÁ_jƒI57§ Key:
DefaultCipherKey Counter: 77 Offset: 12 Plaintext: w91Ëp?¹¢¹©¶B (

ENCRYPTION 78 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 78 Offset: 124 Ciphertext: #ß□²l□‡·¸¿ŠÑ(^{?¯□t?
DECRYPTION 78 Ciphertext: #ß□²l□‡·¸¿ŠÑ(^{?¯□t? Key:
DefaultCipherKey Counter: 78 Offset: 12 Plaintext: àÊûÂO□A(-ì?·Wƒ2□

ENCRYPTION 79 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 79 Offset: 123 Ciphertext: #ß□±l□‡·¸¿ŠÑ(^{?¯□t?

DECRYPTION 79 Ciphertext: #ß□±l□‡·¸¿ŠÑ(^{?¯□t? Key:
DefaultCipherKey Counter: 79 Offset: 12 Plaintext: À2’ïÌ?Ù48€ay□;B

ENCRYPTION 80 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 80 Offset: 123 Ciphertext: £V?-ìŠ*ü□m m□×©ŠšÜV
DECRYPTION 80 Ciphertext: £V?-ìŠ*ü□m m□×©ŠšÜV Key:
DefaultCipherKey Counter: 80 Offset: 12 Plaintext: 7‰□ÁŠ{FíÑg“ŠUöõ0

ENCRYPTION 81 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 81 Offset: 12 Ciphertext: ¹Ä1uö□·mWð□ ñÞ±JZAð^
DECRYPTION 81 Ciphertext: ¹Ä1uö□·mWð□ ñÞ±JZAð^ Key:
DefaultCipherKey Counter: 81 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 82 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 82 Offset: 124 Ciphertext: %³□zjo,cÎ(u□,Á5^ì□Å€
DECRYPTION 82 Ciphertext: %³□zjo,cÎ(u□,Á5^ì□Å€ Key:
DefaultCipherKey Counter: 82 Offset: 12 Plaintext: få)»å?—hcEeˆ)xXA

ENCRYPTION 83 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 83 Offset: 123 Ciphertext: %³□yjo,cÎ(u□,Á5^ì□Å€
DECRYPTION 83 Ciphertext: %³□yjo,cÎ(u□,Á5^ì□Å€ Key:
DefaultCipherKey Counter: 83 Offset: 12 Plaintext: ’s¶uï°-□Ÿ□Â□P™Ò¦

ENCRYPTION 84 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 84 Offset: 124 Ciphertext: Å=«?Šá-‹R9Y? î□□?˜JÜ
DECRYPTION 84 Ciphertext: Å=«?Šá-‹R9Y? î□□?˜JÜ Key:
DefaultCipherKey Counter: 84 Offset: 12 Plaintext: 7□Ë□ˆœû M@¬ □LÍ

ENCRYPTION 85 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 85 Offset: 123 Ciphertext: Å=«—Šá-‹R9Y? î□□?˜JÜ
DECRYPTION 85 Ciphertext: Å=«—Šá-‹R9Y? î□□?˜JÜ Key:
DefaultCipherKey Counter: 85 Offset: 12 Plaintext: gnj□”FJGvD□r·□~¯

ENCRYPTION 86 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 86 Offset: 12 Ciphertext: ÛVŒ¸”Š¥Añ´u™ÿ£ß¿3Äµ
DECRYPTION 86 Ciphertext: ÛVŒ¸”Š¥Añ´u™ÿ£ß¿3Äµ Key:
DefaultCipherKey Counter: 86 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 87 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 87 Offset: 124 Ciphertext: VJ<¶□-°¨¨"Å°,œ¼íÑ½-□
DECRYPTION 87 Ciphertext: VJ<¶□-°¨¨"Å°,œ¼íÑ½-□ Key:
DefaultCipherKey Counter: 87 Offset: 12 Plaintext: ¿□fÖ¦««|ž0^ç!Ñ

ENCRYPTION 88 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 88 Offset: 123 Ciphertext: VJ<·□-°¨¨"Å°,œ¼íÑ½-□
DECRYPTION 88 Ciphertext: VJ<·□-°¨¨"Å°,œ¼íÑ½-□ Key:
DefaultCipherKey Counter: 88 Offset: 12 Plaintext: žœû□2™□ÉíÞ·□·Ö"

ENCRYPTION 89 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 89 Offset: 124 Ciphertext: -¨cÞbtåÎ?Öp:0ûûÚ□;4¸
DECRYPTION 89 Ciphertext: -¨cÞbtåÎ?Öp:0ûûÚ□;4¸ Key:
DefaultCipherKey Counter: 89 Offset: 12 Plaintext: □R5ye□Ô¢ì,í*Z7Ió

ENCRYPTION 90 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 90 Offset: 12 Ciphertext: ó□‹ ¼Î□¨÷³©uúÏ*ƒþ?á
DECRYPTION 90 Ciphertext: ó□‹ ¼Î□¨÷³©uúÏ*ƒþ?á Key:
DefaultCipherKey Counter: 90 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 91 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 91 Offset: 123 Ciphertext: ƒ¼%TÌ`£Fà^EØ-á2{òŒß?

DECRYPTION 91 Ciphertext: ƒ¼%TÌ`£Fà^EØ-á2{òŒß? Key:

DefaultCipherKey Counter: 91 Offset: 12 Plaintext: □-Œõt¼õÉ´ÕwK„PòN

ENCRYPTION 92 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 92 Offset: 123 Ciphertext: UÆ□ã□□?ð søMŽ·bN-2¡j

DECRYPTION 92 Ciphertext: UÆ□ã□□?ð søMŽ·bN-2¡j Key:

DefaultCipherKey Counter: 92 Offset: 12 Plaintext: f½<Ç©§>-©9+"?Xžé

ENCRYPTION 93 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 93 Offset: 123 Ciphertext: □□XÐLÚÞÄ¢1s\ nÆ?'Æ)†

DECRYPTION 93 Ciphertext: □□XÐLÚÞÄ¢1s\ nÆ?'Æ)† Key:

DefaultCipherKey Counter: 93 Offset: 12 Plaintext: …ÀÄ^<,□òè9'{eH□

ENCRYPTION 94 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 94 Offset: 124 Ciphertext: -Wá)â‹g<„G¯.□ëÒ□Ïì□Ü

DECRYPTION 94 Ciphertext: -Wá)â‹g<„G¯.□ëÒ□Ïì□Ü Key:

DefaultCipherKey Counter: 94 Offset: 12 Plaintext: sÍ}GÚ€□2q'UÉ0±åÖ

ENCRYPTION 95 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 95 Offset: 123 Ciphertext: -Wá*â‹g<„G¯.□ëÒ□Ïì□Ü

DECRYPTION 95 Ciphertext: -Wá*â‹g<„G¯.□ëÒ□Ïì□Ü Key:

DefaultCipherKey Counter: 95 Offset: 12 Plaintext: •´¾Í□?—¦¢n2^¹6ê

ENCRYPTION 96 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 96 Offset: 123 Ciphertext: õÇrÕ°□ôÂR² dÃ¤□¼y□5o

DECRYPTION 96 Ciphertext: õÇrÕ°□ôÂR² dÃ¤□¼y□5o Key:
DefaultCipherKey Counter: 96 Offset: 12 Plaintext: zÍ-?åŽ-lw□È±*EY

ENCRYPTION 97 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 97 Offset: 124 Ciphertext: ?=¾XÎá8p¡□Þ„?¤□À'ŸÀ

DECRYPTION 97 Ciphertext: ?=¾XÎá8p¡□Þ„?¤□À'ŸÀ
Key: DefaultCipherKey Counter: 97 Offset: 12
Plaintext:¬□ø/Ûb×□¨Ù¾ŸL8)Ë

ENCRYPTION 98 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 98 Offset: 12 Ciphertext: Ü| ®" ¦‡ƒÃK□@Ð!UV□Ší
DECRYPTION 98 Ciphertext: Ü| ®" ¦‡ƒÃK□@Ð!UV□Ší Key:
DefaultCipherKey Counter: 98 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 99 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 99 Offset: 12 Ciphertext: {Ðî¾4h"Èø?□ÂCØNQ□Åå
DECRYPTION 99 Ciphertext: {Ðî¾4h"Èø?□ÂCØNQ□Åå Key:
DefaultCipherKey Counter: 99 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 100 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 100 Offset: 12 Ciphertext: ›m~àÔ±øË□-öXÎpzüéaç
DECRYPTION 100 Ciphertext: ›m~àÔ±øË□-öXÎpzüéaç 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: Œ!J$ÃýÌlƒÚËÑ-É›□I—k
DECRYPTION 1      Ciphertext: Œ!J$ÃýÌlƒÚËÑ-É›□I—k
      Key:é7IïÇXð^‰)¢W~Û^     Counter: 1  Offset: 12  Plaintext:
?ïËM<hB‰Æ□P†Í½Ç

ENCRYPTION 2      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Counter: 2  Offset: 12  Ciphertext: ñûzI¾'ü qÊÙÂ€9□Iã)
DECRYPTION 2      Ciphertext: ñûzI¾'ü qÊÙÂ€9□Iã)      Key:
DefaultCipherKey  Counter: 2  Offset: 12  Plaintext: OriginalPlainTxt

ENCRYPTION 3      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Counter: 3  Offset: 124 Ciphertext: C¸FdÀA.ÄøswË¾□•□˜[
DECRYPTION 3      Ciphertext: C¸FdÀA.ÄøswË¾□•□˜[      Key:
H…4À0/Xè_É‰?bK□k  Counter: 3  Offset: 12  Plaintext: Ã‹„SjÚv¶~peÌ‡e□³

ENCRYPTION 4      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Counter: 4  Offset: 123 Ciphertext: C¸FdÀA.ÄøswË¾□•□˜[
DECRYPTION 4      Ciphertext: C¸FdÀA.ÄøswË¾□•□˜[      Key:
úÁÖÄ†Nð;Û ÓÐ~4…Û  Counter: 4  Offset: 12  Plaintext: □®˜.ŒmdD□öô‰`”fI

ENCRYPTION 5      Plaintext: OriginalPlainTxt   Key: DefaultCipherKey
      Counter: 5  Offset: 124 Ciphertext: á□%L®Ï£ 2;oüÅL¹□ó[²ñ
DECRYPTION 5      Ciphertext: á□%L®Ï£ 2;oüÅL¹□ó[²ñ    Key:
4}vGÇGÙMtß?Ž'¬í   Counter: 5  Offset: 12  Plaintext: éökæñïm…±H/Š¦tÓ°
```

ENCRYPTION 6 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 6 Offset: 124 Ciphertext: €j□LÏ¶?□?'Þ¹VrÁ□„×%.

DECRYPTION 6 Ciphertext: €j□LÏ¶?□?'Þ¹VrÁ□„×%.

Key:‹ÏväbrÉôÈL¶7kù.Ÿ Counter: 6 Offset: 12 Plaintext:

□d€]?¼j□×íª:Û„□¹

ENCRYPTION 7 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 7 Offset: 12 Ciphertext: ÝúÑ³'&Wý3¦‡Wûš°¬y™M¯

DECRYPTION 7 Ciphertext: ÝúÑ³'&Wý3¦‡Wûš°¬y™M¯ Key:

DefaultCipherKey Counter: 7 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 8 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 8 Offset: 124 Ciphertext: ¡G†¤î› ëa³□¬Ô±§Åb□Ô†

DECRYPTION 8 Ciphertext: ¡G†¤î› ëa³□¬Ô±§Åb□Ô† Key:

VMÛ4G□çz¶}D¥?C|Ñ Counter: 8 Offset: 12 Plaintext: az□;ÿ"□i%Kk□&'(

ENCRYPTION 9 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 9 Offset: 124 Ciphertext: ´Î†ýû□ ½×¹□□'Ýÿ□¢:š¬

DECRYPTION 9 Ciphertext: ´Î†ýû□ ½×¹□□'Ýÿ□¢:š¬ Key:

é÷ÆQ|oæF}□Úü‟ÆC Counter: 9 Offset: 12 Plaintext: □¥ÜVx½Y,L¤Î□0ÉË□

ENCRYPTION 10 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 10 Offset: 124 Ciphertext: —7«BK±ê'´|üãçž³'°+¿

DECRYPTION 10 Ciphertext: —7«BK±ê'´|üãçž³'°+¿ Key:

<È? Ö×□ÐsŸ3ß¹□'^ Counter: 10 Offset: 12 Plaintext: @ìAM°çŸˆ"ŽnD□>6

ENCRYPTION 11 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 11 Offset: 123 Ciphertext: —7¨BK±ê'´|üãçž³'°+¿

DECRYPTION 11 Ciphertext: —7¨BK±ê‘´|üãçž³'°+¿ Key:

ë˜(Ñiòƒä BKî¯ÿ[ô Counter: 11 Offset: 12 Plaintext: /…åá3ž$èBë» Ïr=r

ENCRYPTION 12 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 12 Offset: 123 Ciphertext: \¨Ã□□tEWu÷□G9žI¯®»□

DECRYPTION 12 Ciphertext: \¨Ã□□tEWu÷□G9žI¯®»□

Key: 8k¡Ä‡e±ß°□xFTÝ] Counter: 12 Offset: 12 Plaintext: ÑÜ9ÃìâKdÅN˜èl9x□

ENCRYPTION 13 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 13 Offset: 123 Ciphertext: È´%□‡h£Téb□z□$—ÃÔ/€
DECRYPTION 13 Ciphertext: È´%□‡h£Téb□z□$—ÃÔ/€ Key: ¤Æ□æKcUE:çLI¯&}· Counter: 13 Offset: 12 Plaintext: µrS‡ŽÇ?ù¢lÔlµ0îG

ENCRYPTION 14 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 14 Offset: 123 Ciphertext: h½08'a¶}=”¤M˜~VÜ-¼µ7
DECRYPTION 14 Ciphertext: h½08'a¶}=”¤M˜~VÜ-¼µ7 Key: °□□x¾6UÈóh@ú¦jp¤ Counter: 14 Offset: 12 Plaintext: ŠSè-□ì/äd□ □óL[ß

ENCRYPTION 15 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 15 Offset: 12 Ciphertext: -C□ÿÙŸ›¹]#□‡r□v~Œ9Ë½
DECRYPTION 15 Ciphertext: -C□ÿÙŸ›¹]#□‡r□v~Œ9Ë½ Key: DefaultCipherKey Counter: 15 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 16 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 16 Offset: 123 Ciphertext: Æ9è©‰ånî,0e0Í‡ õ□;:Á
DECRYPTION 16 Ciphertext: Æ9è©‰ånî,0e0Í‡ õ□;:Á Key: □ÝÑ$C/ƒŸ‰yœ[MúJ Counter: 16 Offset: 12 Plaintext: mm,½!Sã·□¥û□O‹J

ENCRYPTION 17 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 17 Offset: 12 Ciphertext: ¾0¤óñì"«ý¹s¬`Žó-K□c
DECRYPTION 17 Ciphertext: ¾0¤óñì"«ý¹s¬`Žó-K□c Key: DefaultCipherKey Counter: 17 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 18 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 18 Offset: 124 Ciphertext: □□„oÆ‘Ýq\6}>ˆ„¸S®²ú

DECRYPTION 18 Ciphertext: □□„oÆ‘Ýq\6}>ˆ„¸S®²ú Key:

µ¼Eäit1vŽ)þ”r° Counter: 18 Offset: 12 Plaintext: €$□Öyõià?0O□Í±M[

ENCRYPTION 19 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 19 Offset: 123 Ciphertext: □□‡oÆ‘Ýq\6}>ˆ„¸S®²ú

DECRYPTION 19 Ciphertext: □□‡oÆ‘Ýq\6}>ˆ„¸S®²ú Key:

□•gH€_7ÑãæHWGì Counter: 19 Offset: 12 Plaintext: yB□Ê¶:;%pXÑ¿¤¨-m

ENCRYPTION 20 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 20 Offset: 123 Ciphertext: \t□?□¨›Æ{(eRª=Z?□—Àp

DECRYPTION 20 Ciphertext: \t□?□¨›Æ{(eRª=Z?□—Àp Key:

^ÂØL:À~ñaØ□zrfõ Counter: 20 Offset: 12 Plaintext: Ê"í¯^µ□ïä±fc%ðÓ

ENCRYPTION 21 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 21 Offset: 123 Ciphertext: "□¶—mÝ0Ë¢R1zZ±Ø³-‘□?

DECRYPTION 21 Ciphertext: "□¶—mÝ0Ë¢R1zZ±Ø³-‘□? Key:

D‘›°Àâœ+)pÕ¤Í"˜ Counter: 21 Offset: 12 Plaintext: ¸“Ãƒ_JåFô×S“6 ÂJ

ENCRYPTION 22 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 22 Offset: 12 Ciphertext: Qÿ¥„-##Ùâq□Å□□«„\ûÁç

DECRYPTION 22 Ciphertext: Qÿ¥„-##Ùâq□Å□□«„\ûÁç Key:

DefaultCipherKey Counter: 22 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 23 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 23 Offset: 12 Ciphertext: ?buÀÐä+F?□Âà¶a□Š#¡j

DECRYPTION 23 Ciphertext: ?buÀÐä+F?□Âà¶a□Š#¡j Key:

DefaultCipherKey Counter: 23 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 24 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 24 Offset: 124 Ciphertext: !□š nØ□□.;3*ßÖR^ìK□A

DECRYPTION 24 Ciphertext: !□š nØ□□.;3*ßÖR^ìK□A Key:

)8Ð;äÎ²YËÇbî,z□□ Counter: 24 Offset: 12 Plaintext: D□□×¦°/»Eæ□¡Ë–ï@

ENCRYPTION 25 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 25 Offset: 123 Ciphertext: !□š/nØ□□.;3*ßÖR^ìK□A

DECRYPTION 25 Ciphertext: !□š/nØ□□.;3*ßÖR^ìK□A Key:

á»ëñr?fÑèñj--ýf- Counter: 25 Offset: 12 Plaintext: NB£vó*»ãa□ÑÈ¢˜□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[Jð:ÉŠ□7u® Counter: 26 Offset: 12 Plaintext: ¹[I–□¾ÅÊz„o‹¸?û¹

ENCRYPTION 27 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 27 Offset: 12 Ciphertext: -S□Éâ?’›□?,Jgz□”â'ª\

DECRYPTION 27 Ciphertext: -S□Éâ?’›□?,Jgz□”â'ª\ Key:

DefaultCipherKey Counter: 27 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 28 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 28 Offset: 124 Ciphertext: :?ÞÅuQX–X□©¨□d DTÓ

DECRYPTION 28 Ciphertext: :?ÞÅuQX–X□©¨□d DTÓ Key:

áè¾½¿ž□¬!ãm0oŠ¼□ Counter: 28 Offset: 12 Plaintext: Ì□oBi>”u>eNå;Ï,ß

ENCRYPTION 29 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 29 Offset: 123 Ciphertext: :?ÞÂuQX–X□©¨□d DTÓ

DECRYPTION 29 Ciphertext: :?ÞÂuQX-X□©¨□d DTÓ Key:
g€u□òâÄ×…nÎLy×cÙ Counter: 29 Offset: 12 Plaintext: Òê$ Ïp‡á†E®□´Ö"

ENCRYPTION 30 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 30 Offset: 124 Ciphertext: ÒNcÎ?'å›×hµqHàæß®uæ^
DECRYPTION 30 Ciphertext: ÒNcÎ?'å›×hµqHàæß®uæ^ Key: ;x-
0á□®Ñ„€†¾□E□ Counter: 30 Offset: 12 Plaintext: I·™¤Í?GjH:'kú?"Ê

ENCRYPTION 31 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 31 Offset: 124 Ciphertext: □¢ ŽG~?Øjÿfžœ·¸µ Z!"
DECRYPTION 31 Ciphertext: □¢ ŽG~?Øjÿfžœ·¸µ Z!" Key:
F[+ËžÿøÞ¢ýr□-Æðè Counter: 31 Offset: 12 Plaintext: ykl3Mw^$oc)?Æ…š½

ENCRYPTION 32 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 32 Offset: 124 Ciphertext: Ÿa¦êÐ½ ½¥□¡îÒ}$P&Y!2
DECRYPTION 32 Ciphertext: Ÿa¦êÐ½ ½¥□¡îÒ}$P&Y!2 Key: 8□µ□@Ã_ÉÊ
|" :?Ö Counter: 32 Offset: 12 Plaintext: ž)(³±$ËÖ□ÕRžò¡ ä

ENCRYPTION 33 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 33 Offset: 124 Ciphertext: "cƒšÛ¿□òÕzü□?§?dþ>^-
DECRYPTION 33 Ciphertext: "cƒšÛ¿□òÕzü□?§?dþ>^- Key:
¿Ú!d□□ð"êé¶ÏÎ…. Counter: 33 Offset: 12 Plaintext: RNIC\¼ □ž\□□1+-Ë

ENCRYPTION 34 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 34 Offset: 12 Ciphertext: p[Äú?‡B"ïÖSÜš□□ëë™Y°
DECRYPTION 34 Ciphertext: p[Äú?‡B"ïÖSÜš□□ëë™Y° Key:
DefaultCipherKey Counter: 34 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 35 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 35 Offset: 123 Ciphertext: îTÒ¦¡^TÌY!ï□□lký□¡□O

DECRYPTION 35 Ciphertext: îTÒ¦¡^TÌY!ï□□lký□¡□O Key:

à^3™ÈtŠ„\□/Ô□P®□ Counter: 35 Offset: 12 Plaintext: p□Bûé‹□$° □Ñjög

ENCRYPTION 36 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 36 Offset: 123 Ciphertext: ?*ï‡pöiì□”ªt05à□‹·

DECRYPTION 36 Ciphertext: ?*ï‡pöiì□”ªt05à□‹· Key:

©Îp?§;Ý□□f9åé□aÒ Counter: 36 Offset: 12 Plaintext: x÷□Þþ_æ‡¶%¢kªÂ’î

ENCRYPTION 37 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 37 Offset: 12 Ciphertext: ž□^ãÑÁØ?ÝBêý Ì¤H□S©□

DECRYPTION 37 Ciphertext: ž□^ãÑÁØ?ÝBêý Ì¤H□S©□ Key:

DefaultCipherKey Counter: 37 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 38 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 38 Offset: 124 Ciphertext: ¹‹G?öWÁð•áê}□Ëp‘?Û‰

DECRYPTION 38 Ciphertext: ¹‹G?öWÁð•áê}□Ëp‘?Û‰ Key:

r"ð2ÆÆÝBy*Å¢>¨ŠÈ Counter: 38 Offset: 12 Plaintext: Ëj"(Ì-□¢¹¯”i\□ÍÔ

ENCRYPTION 39 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 39 Offset: 124 Ciphertext: Ä¹□?‹e„á□þþ□ª‰éæçg

DECRYPTION 39 Ciphertext: Ä¹□?‹e„á□þþ□ª‰éæçg Key:

RÍÐ□îT'F□ÝÔþ> Counter: 39 Offset: 12 Plaintext: iö□f\ ;Ëß.”?æa¯q

ENCRYPTION 40 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 40 Offset: 123 Ciphertext: Ä¹□Ž‹e„á□þþ□ª‰éæçg

DECRYPTION 40 Ciphertext: Ä¹□Ž‹e„á□þþ□ª‰éæçg Key:

Œ%Ú¼œ‰,□½□À—o³ Counter: 40 Offset: 12 Plaintext: HüÑxOÄ$òàò÷UŽ2À

ENCRYPTION 41 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 41 Offset: 12 Ciphertext: ÿ□ s°Þ?□°:- R¬ÁèfyðX

DECRYPTION 41 Ciphertext: ÿ□ s°Þ?□°:- R¬ÁèfyðX Key:

DefaultCipherKey Counter: 41 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 42 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 42 Offset: 12 Ciphertext: 2{Û<}§]]æÅët‹¹Òž-ð+

DECRYPTION 42 Ciphertext: 2{Û<}§]]æÅët‹¹Òž-ð+ Key:

DefaultCipherKey Counter: 42 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 43 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 43 Offset: 123 Ciphertext: E)]¥õÛÇÃ·□Æ?%íõÜ□Š&

DECRYPTION 43 Ciphertext: E)]¥õÛÇÃ·□Æ?%íõÜ□Š& Key:

•Mä¢výÙ™÷□Ñ.T? Counter: 43 Offset: 12 Plaintext: ‘åÔýŽH¿Ì?12Bsž}|

ENCRYPTION 44 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 44 Offset: 124 Ciphertext: ‰j,µÆ¶ªÖ7Ÿ´¥O©?5³?`¶

DECRYPTION 44 Ciphertext: ‰j,µÆ¶ªÖ7Ÿ´¥O©?5³?`¶ Key:

Â+YÛÖ™‘¾iö´□\¹□ Counter: 44 Offset: 12 Plaintext: ˜&2™ê”i[□OÆx-Q□□

ENCRYPTION 45 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 45 Offset: 123 Ciphertext: ‰j,²Æ¶ªÖ7Ÿ´¥O©?5³?`¶

DECRYPTION 45 Ciphertext: ‰j,²Æ¶ªÖ7Ÿ´¥O©?5³?`¶ Key:

□Z□Ç□0ò¯KÏàÍ)□»o Counter: 45 Offset: 12 Plaintext: |/šðD²@òÄ6cØz’□

ENCRYPTION 46 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 46 Offset: 123 Ciphertext: ¸]$Ï÷?¢ªí□□³2Ó□“j□h

DECRYPTION 46 Ciphertext: ¸]$Ï÷?¢ªí□□³2Ó□"j□h Key: æO‡š-
?ifYA_'Hg° Counter: 46 Offset: 12 Plaintext: CÁ›mãƒà□¸:â[LF¬

ENCRYPTION 47 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 47 Offset: 124 Ciphertext: Âû[â?'Ý„ã¶□x?j1tõì¦
DECRYPTION 47 Ciphertext: Âû[â?'Ý„ã¶□x?j1tõì¦ Key:
"®½Àq□&Ã4Ùk#ëü Counter: 47 Offset: 12 Plaintext: Ÿ½%□AŒsä…Dª»ÒÙù™

ENCRYPTION 48 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 48 Offset: 123 Ciphertext: Âû[ã?'Ý„ã¶□x?j1tõì¦
DECRYPTION 48 Ciphertext: Âû[ã?'Ý„ã¶□x?j1tõì¦ Key:
Jï´¸÷□½$ÙD_-□?' Counter: 48 Offset: 12 Plaintext: -š-nú2Ý¯g®gl□fu□

ENCRYPTION 49 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 49 Offset: 12 Ciphertext: êav‹¥½ðóñ°GsPÂ7Æ ûf4
DECRYPTION 49 Ciphertext: êav‹¥½ðóñ°GsPÂ7Æ ûf4 Key:
DefaultCipherKey Counter: 49 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 50 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 50 Offset: 123 Ciphertext: □ò9ŒJ.¿õæ§□?S□i÷È□ãœ
DECRYPTION 50 Ciphertext: □ò9ŒJ.¿õæ§□?S□i÷È□ãœ Key:
§±;»/lTg÷J□Íó[p Counter: 50 Offset: 12 Plaintext: {Í—¬£¸`5‰!Ö□âžÓû

ENCRYPTION 51 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 51 Offset: 12 Ciphertext: T1?‹□í¹ñ□Õ¤Ž5(²?«œ•ã
DECRYPTION 51 Ciphertext: T1?‹□í¹ñ□Õ¤Ž5(²?«œ•ã Key:
DefaultCipherKey Counter: 51 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 52 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 52 Offset: 123 Ciphertext: ¸8Z□÷äÜ|@» ¬Ý□□®²-»v

DECRYPTION 52 Ciphertext: ¸8Z□÷äÜ|@» ¬Ý□□®²-»v Key:

□□%!|àé]8Ê{¸0Z·õ Counter: 52 Offset: 12 Plaintext: õA□ ×□¥Ns¥¢?]?

ENCRYPTION 53 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 53 Offset: 12 Ciphertext: 5ú□Dé|j¤ 1ïÐ›~.ò»oÝ

DECRYPTION 53 Ciphertext: 5ú□Dé|j¤ 1ïÐ›~.ò»oÝ Key:

DefaultCipherKey Counter: 53 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 54 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 54 Offset: 123 Ciphertext: § □¤è|›Ù”Ï>=Ãûi‡ü□ô

DECRYPTION 54 Ciphertext: § □¤è|›Ù”Ï>=Ãûi‡ü□ô Key:

Ê´ËÑ^ÓM□`”□□□□9 Counter: 54 Offset: 12 Plaintext: ÇD□¸ðS;Âä□y:êÊ□Þ

ENCRYPTION 55 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 55 Offset: 12 Ciphertext: ³Ó;Œü□½òcËa™˜p?^b¼|□

DECRYPTION 55 Ciphertext: ³Ó;Œü□½òcËa™˜p?^b¼|□ Key:

DefaultCipherKey Counter: 55 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 56 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 56 Offset: 12 Ciphertext: ??`2ßQæM…¼□ð.a1»ö°uí

DECRYPTION 56 Ciphertext: ??`2ßQæM…¼□ð.a1»ö°uí Key:

DefaultCipherKey Counter: 56 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 57 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 57 Offset: 123 Ciphertext: y:z?6æüýV? OaÁ¡5¯wO»

DECRYPTION 57 Ciphertext: y:z?6æüýV? OaÁ¡5¯wO» Key:

1)aú□?„g÷Äˆï¦æí” Counter: 57 Offset: 12 Plaintext: □µ’¯_ém□-5ïSµ}ä

ENCRYPTION 58 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 58 Offset: 123 Ciphertext: I ¼Ã□|:²□±Äm¢ãDnûÒ Ò

DECRYPTION 58 Ciphertext: I ¼Ã□|:²□±Äm¢ãDnûÒ Ò Key:

ˆëßø□b<¶§ ¨®ZˆÒ□ Counter: 58 Offset: 12 Plaintext: ï¿A³ÃHég□□«-□S\¤

ENCRYPTION 59 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 59 Offset: 12 Ciphertext: □„voÀ□□ý÷ □U«□□~—d÷

DECRYPTION 59 Ciphertext: □„voÀ□□ý÷ □U«□□~—d÷ Key:

DefaultCipherKey Counter: 59 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 60 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 60 Offset: 12 Ciphertext: ?Çˆ›Ò□ØèÙ2´□äì□qø□ÙÏ

DECRYPTION 60 Ciphertext: ?Çˆ›Ò□ØèÙ2´□äì□qø□ÙÏ Key:

DefaultCipherKey Counter: 60 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 61 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 61 Offset: 123 Ciphertext: 0¶n□Ð0□°ÊÃ□□-"YqÀ¥N

DECRYPTION 61 Ciphertext: 0¶n□Ð0□°ÊÃ□□-"YqÀ¥N Key:

$?}ã3ÉÚ”ùÖ¼„□ø□ò Counter: 61 Offset: 12 Plaintext: ÈÅ;ñ—ªPO□n□|óÜs>

ENCRYPTION 62 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 62 Offset: 124 Ciphertext: °9òbÿåt□□Ý-žD¿Â□| U□

DECRYPTION 62 Ciphertext: °9òbÿåt□□Ý-žD¿Â□| U□ Key:

É‰‹,‡□þðT□ÏÑð&MÎ Counter: 62 Offset: 12 Plaintext: □ÈÂÂê€Œ¿·ê×ŠÌj?€

ENCRYPTION 63 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 63 Offset: 124 Ciphertext: žK¼\Ñ—:*µ-Û%×J®,¼½˜□

DECRYPTION 63 Ciphertext: žK¼\Ñ—:*µ-Û%×J®,¼½˜□ Key:
H®VGÄ…$,ÇË¥È¯OF Counter: 63 Offset: 12 Plaintext: %®dRZ™'1Oø□~±‡É@

ENCRYPTION 64 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 64 Offset: 124 Ciphertext: ?Ã"ÛÒ□¬ûy˜□¹Ñ□"Û•ÆÙ
DECRYPTION 64 Ciphertext: ?Ã"ÛÒ□¬ûy˜□¹Ñ□"Û•ÆÙ Key:
Ö□Q¥r™.ÌA¼"5É^9R Counter: 64 Offset: 12 Plaintext: □M'}è?»†"j vää¤h

ENCRYPTION 65 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 65 Offset: 124 Ciphertext: |LÁ□3?G□´³□•YHåLuè>□
DECRYPTION 65 Ciphertext: |LÁ□3?G□´³□•YHåLuè>□ Key:
8+õ□®ä□\²‰¥FYïOØ Counter: 65 Offset: 12 Plaintext: ªhGœvvÅê>8¤ücÀ€Æ

ENCRYPTION 66 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 66 Offset: 12 Ciphertext: ëìÆ□¤0@□`ë1:pb5|Ý2?Ê
DECRYPTION 66 Ciphertext: ëìÆ□¤0@□`ë1:pb5|Ý2?Ê Key:
DefaultCipherKey Counter: 66 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 67 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 67 Offset: 123 Ciphertext: □©ìxXujr+Kñ¿Ò,éë®Iß¬
DECRYPTION 67 Ciphertext: □©ìxXujr+Kñ¿Ò,éë®Iß¬ Key:
+|…¥@Eô□«EÚ□z□¬ Counter: 67 Offset: 12 Plaintext: k?Pœã›b□wkÃ□□-1ç

ENCRYPTION 68 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 68 Offset: 124 Ciphertext: ÔŒñ□›Pw□Ïj?8□ò³øú²!
DECRYPTION 68 Ciphertext: ÔŒñ□›Pw□Ïj?8□ò³øú²! Key:
ÛÈ3o¡¾~×=ÜÜÙñwp□ Counter: 68 Offset: 12 Plaintext: □RñÄÑB•S0Ú□¡Y€?

ENCRYPTION 69 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 69 Offset: 123 Ciphertext: ÔŒñ□›Pw□Ïj?8□ò³øú²!
DECRYPTION 69 Ciphertext: ÔŒñ□›Pw□Ïj?8□ò³øú²! Key:
4□!ä!uõîU$m”íMt Counter: 69 Offset: 12 Plaintext: dß#>¬Ñ0ÊÜ□ÛyÝ□□ã

ENCRYPTION 70 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 70 Offset: 12 Ciphertext: ;ž;:tB½7-¡©I0K^?ñ-ûÞ
DECRYPTION 70 Ciphertext: ;ž;:tB½7-¡©I0K^?ñ-ûÞ Key:
DefaultCipherKey Counter: 70 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 71 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 71 Offset: 123 Ciphertext: o□Íþ ÓKðxÀAÒy6Ö(†¹¹£
DECRYPTION 71 Ciphertext: o□Íþ ÓKðxÀAÒy6Ö(†¹¹£ Key:
."!Â□¹Ad]›?«vM[d Counter: 71 Offset: 12 Plaintext: b¼□|¿Kâ?VÚ□°□œ²

ENCRYPTION 72 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 72 Offset: 12 Ciphertext: ?C□¦ÀŸž©ß<°`□□ú&¦‰[M
DECRYPTION 72 Ciphertext: ?C□¦ÀŸž©ß<°`□□ú&¦‰[M Key:
DefaultCipherKey Counter: 72 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 73 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 73 Offset: 12 Ciphertext: ×yÒ4˜¥T4Ÿáø7á3™è□TQ¼
DECRYPTION 73 Ciphertext: ×yÒ4˜¥T4Ÿáø7á3™è□TQ¼ Key:
DefaultCipherKey Counter: 73 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 74 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 74 Offset: 124 Ciphertext: Ožv« Bðª„G.Òw¨›,°Öp□
DECRYPTION 74 Ciphertext: Ožv« Bðª„G.Òw¨›,°Öp□ Key:
´,<□Ya°®ì,¤×□Aóx Counter: 74 Offset: 12 Plaintext: ½ï1"M□§□□ŒŸgRnãù

ENCRYPTION 75 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 75 Offset: 123 Ciphertext: Ožv¨ Bðª„G.Òw¨›,°Öp□

DECRYPTION 75 Ciphertext: Ožv¨ Bðª„G.Òw¨›,°Öp□ Key:

åv:□Ñ9ån÷cª@tb†Ò Counter: 75 Offset: 12 Plaintext: F4 ý,C¬x?6□±Íã„

ENCRYPTION 76 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 76 Offset: 124 Ciphertext: ‘ÆDÞÞ□ÂÝ□ùÁ_jƒI57§

DECRYPTION 76 Ciphertext: ‘ÆDÞÞ□ÂÝ□ùÁ_jƒI57§ Key:

³¶Ð‰•□nÇ"N¨¦èÆ9¦ Counter: 76 Offset: 12 Plaintext: ?Þ->åo%Å0□ ”>ŽJ

ENCRYPTION 77 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 77 Offset: 124 Ciphertext:]p}□Òöyâ*.□v□C˜Ç□h¿

DECRYPTION 77 Ciphertext:]p}□Òöyâ*.□v□C˜Ç□h¿ Key:

æA‡áÖV®¶g×□1¶°, Counter: 77 Offset: 12 Plaintext: H©C2ðg2!□8□□‘□´4

ENCRYPTION 78 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 78 Offset: 12 Ciphertext: YQ@□□?Æ ÇqƒÝ8¢I®å?¬§

DECRYPTION 78 Ciphertext: YQ@□□?Æ ÇqƒÝ8¢I®å?¬§ Key:

DefaultCipherKey Counter: 78 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 79 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 79 Offset: 123 Ciphertext: #ß□±l□‡·¸¿ŠÑ(^{?¯□t?

DECRYPTION 79 Ciphertext: #ß□±l□‡·¸¿ŠÑ(^{?¯□t? Key:

-Ýµ4 ù\~□Ò(îÚðüÊ Counter: 79 Offset: 12 Plaintext: ¯□G□õ □^‡□£R-0′

ENCRYPTION 80 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 80 Offset: 124 Ciphertext: □W?□R‹□□å“Wµ□□@FµÁóN

DECRYPTION 80 Ciphertext: □W?□R‹□□å‟Wµ□□@FµÁóN Key: ÉÊSÅ□□
wÖ GÒI*@ Counter: 80 Offset: 12 Plaintext: 5i¹öQ²ó³…Ð-?□È_r

ENCRYPTION 81 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 81 Offset: 12 Ciphertext: ¹Ä1uö□·mWð□ ñÞ±JZAð^
DECRYPTION 81 Ciphertext: ¹Ä1uö□·mWð□ ñÞ±JZAð^ Key:
DefaultCipherKey Counter: 81 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 82 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 82 Offset: 12 Ciphertext: :ƒ'‟u_¡ŠçÞúÜÐ¬(½?žÞÍ
DECRYPTION 82 Ciphertext: :ƒ'‟u_¡ŠçÞúÜÐ¬(½?žÞÍ Key:
DefaultCipherKey Counter: 82 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 83 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 83 Offset: 12 Ciphertext: ót Ð¼¨¦ÊQ□HÚnsO?7ÂÆ^
DECRYPTION 83 Ciphertext: ót Ð¼¨¦ÊQ□HÚnsO?7ÂÆ^ Key:
DefaultCipherKey Counter: 83 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 84 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 84 Offset: 12 Ciphertext: *¾þÔebxÏÞèO·XvWs Mþ£
DECRYPTION 84 Ciphertext: *¾þÔebxÏÞèO·XvWs Mþ£ Key:
DefaultCipherKey Counter: 84 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 85 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 85 Offset: 123 Ciphertext: Å=«—Šá-‹R9Y? î□□?˜JÜ
DECRYPTION 85 Ciphertext: Å=«—Šá-‹R9Y? î□□?˜JÜ Key:
&□•S>~¤áã0□yãÆ Counter: 85 Offset: 12 Plaintext: 2cWÇ³iÃ4»¼ó@ú¢QX

ENCRYPTION 86 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 86 Offset: 12 Ciphertext: ÛVŒ¸”Š¥Añ´u™ÿ£ß¿3Äµ
DECRYPTION 86 Ciphertext: ÛVŒ¸”Š¥Añ´u™ÿ£ß¿3Äµ Key:
DefaultCipherKey Counter: 86 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 87 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 87 Offset: 12 Ciphertext: Êž!ë…B§õIÃ=´Ð[äÓaÐ
DECRYPTION 87 Ciphertext: Êž!ë…B§õIÃ=´Ð[äÓaÐ Key:
DefaultCipherKey Counter: 87 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 88 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 88 Offset: 123 Ciphertext: VJ<·□–°¨¨“Å°,œ¼íÑ½–□
DECRYPTION 88 Ciphertext: VJ<·□–°¨¨“Å°,œ¼íÑ½–□ Key:
^Ï¸Í¯e□%û,T‘Vi–¬ Counter: 88 Offset: 12 Plaintext: iWÒ□3Í²A□ð*□F“3u

ENCRYPTION 89 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 89 Offset: 124 Ciphertext: -¨cÞbtåÎ?Öp:0ûûÚ□;4¸
DECRYPTION 89 Ciphertext: -¨cÞbtåÎ?Öp:0ûûÚ□;4¸ Key:
á□éäg9›Lî□ó¤†J Counter: 89 Offset: 12 Plaintext: □$*ýÈA¥Ül6'Î?□Bª

ENCRYPTION 90 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 90 Offset: 12 Ciphertext: ó□‹ ¼Î□¨÷³©uúÏ*ƒþ?á
DECRYPTION 90 Ciphertext: ó□‹ ¼Î□¨÷³©uúÏ*ƒþ?á Key:
DefaultCipherKey Counter: 90 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 91 Plaintext: OriginalPlainTxt Key: DefaultCipherKey
Counter: 91 Offset: 124 Ciphertext: UÆ□â□□?ð søMŽ·bN-2¡j
DECRYPTION 91 Ciphertext: UÆ□â□□?ð søMŽ·bN-2¡j Key:
Qì3nã□Á‘ù™.□fê?… Counter: 91 Offset: 12 Plaintext: õ?„T&¼æûš^D¶(YÇ!

ENCRYPTION 92 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 92 Offset: 123 Ciphertext: UÆ□ã□□?ð søMŽ·bN-2¡j

DECRYPTION 92 Ciphertext: UÆ□ã□□?ð søMŽ·bN-2¡j Key:

È?\p¤□ã·@ÿ□LŽ-Êþ Counter: 92 Offset: 12 Plaintext: Ñ□-Ô¤8V›*¶O‘PÂJ□

ENCRYPTION 93 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 93 Offset: 12 Ciphertext: '¥Œ?hy+õ□›□'ô0□¼□?È

DECRYPTION 93 Ciphertext: '¥Œ?hy+õ□›□'ô0□¼□?È Key:

DefaultCipherKey Counter: 93 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 94 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 94 Offset: 124 Ciphertext: -Wá)â‹g<„G¯.□ëÒ□Ïì□Ü

DECRYPTION 94 Ciphertext: -Wá)â‹g<„G¯.□ëÒ□Ïì□Ü Key:

p¥\□Uí¡7|%ëïv¤Âu Counter: 94 Offset: 12 Plaintext: ß¬□˜%□«□b-□ØU□]+

ENCRYPTION 95 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 95 Offset: 123 Ciphertext: -Wá*â‹g<„G¯.□ëÒ□Ïì□Ü

DECRYPTION 95 Ciphertext: -Wá*â‹g<„G¯.□ëÒ□Ïì□Ü Key:

F®†¸R‡Œ□'É§ErC□ý Counter: 95 Offset: 12 Plaintext: AÙ□&âû£èMFˆU|$Á/

ENCRYPTION 96 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 96 Offset: 12 Ciphertext: ßQ#Õ??¥ÂŸ}bbzë□µP]¦

DECRYPTION 96 Ciphertext: ßQ#Õ??¥ÂŸ}bbzë□µP]¦ Key:

DefaultCipherKey Counter: 96 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 97 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 97 Offset: 124 Ciphertext: ?=¾XÎá8p¡□Þ„?¤□À’ŸÀ

DECRYPTION 97 Ciphertext: ?=¾XÎá8p¡□Þ„?¤□À'ŸÀ

Key: □!°?<¤Ñs—‰□w<□ôÂ Counter: 97 Offset: 12 Plaintext:

H6‰u□)]ïRÿ~¸èÇ°=

ENCRYPTION 98 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 98 Offset: 123 Ciphertext: ?=¾YÎá8p¡□Þ„?¤□À'ŸÀ

DECRYPTION 98 Ciphertext: ?=¾YÎá8p¡□Þ„?¤□À'ŸÀ

Key: Ü| Ï]□ĐšŠmÃI□r˜‹ Counter: 98 Offset: 12 Plaintext:

íûí,0s ¡@ï™¦¸/‹□

ENCRYPTION 99 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 99 Offset: 12 Ciphertext: {Đî¾4h”Èø?□ÂCØNQ□Åå

DECRYPTION 99 Ciphertext: {Đî¾4h”Èø?□ÂCØNQ□Åå Key:

DefaultCipherKey Counter: 99 Offset: 12 Plaintext: OriginalPlainTxt

ENCRYPTION 100 Plaintext: OriginalPlainTxt Key: DefaultCipherKey

Counter: 100 Offset: 124 Ciphertext: ,CžˆcŸ□£$tÆ□ÂMîÂÿ?□

DECRYPTION 100 Ciphertext: ,CžˆcŸ□£$tÆ□ÂMîÂÿ?□ Key:

›m~ƒÍùm½□·'f¹‹s? Counter: 100 Offset: 12 Plaintext:

äÂy/€~ð□)A#qÚà

Cipher 1 was chosen 33 out of 100 iterations.

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

References

[1] Akkar, M. and Giraud, C. (2001). An Implementation of DES and AES, Secure against Some Attacks. *Lecture Notes in Computer Science, Vol. 2161*, 309-318.

[2] Anderson, R. and Biham, E. (1996). Tiger: A Fast New Hash Function. *Fast Software Encryption, Vol. 1039*, 89-97.

[3] Barretto, P. (2001). Rijndael.java. Retrieved Nov 2, 2007, from http://www.cs.ucdavis.edu/~rogaway/ocb/ocb-java/Rijndael.java

[4] Biham, E. and Shamir, A. (1997). Differential Fault Analysis of Secret Key Cryptosystems. *Lecture Notes in Computer Science, Vol. 1294*, 513-525.

[5] Chau, R. S. (2004). Intel's Breakthrough in High-K Gate Dielectric Drives Moore's Law Well into the Future. *Technology@Intel Magazine*, 1-7.

[6] Daemen, J. and Rijmen V. (1999). *The Rijndael Block Cipher*. Retrieved May 22, 2007, from http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf

[7] Diffie, W. and Hellman, M. E. (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory, IT-22, n. 6*, 644-654.

[8] Golic, J. D. (2007). Techniques for Random Masking in Hardware. *IEEE Transactions on Circuits and Systems-I: Regular Papers, 54,* 291-300.

[9] Golic, J. D. and Tymen C. (2002). Multiplicative Masking and Power Analysis of AES. *Lecture Notes in Computer Science, Vol. 2523*, 198-212.

[10] Hwang, D. D., Tiri, K., Hodjat, A., Lai, B., Yang, S., Schaumont, P., & Verbauwhede, I. (2006). AES-Based Security Coprocessor IC in 0.18-µm CMOS With Resisitance to Differential Power Analysis Side-Channel Attacks. *IEEE Journal of Solid-State Circuits, 41,* 781-791.

[11] Kocher, P. C. (1996). Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems. *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, 104-113.

[12] Kocher, P. C., Jaffe, J. and Jun, B. (1999). Differential Power Analysis. *Lecture notes in Computer Science, 1666*¸ 388-397.

[13] McGrew, D. and Viega, J. (2004). The Galois/Counter Mode of Operation (GCM). *Submission to NIST Modes of Operation Process.*

[14] *NACSIM 5000 Tempest Fundamentals (Public Version)* (2000). Retrieved May 20, 2007, from http://cryptome.sabotage.org/nacsim-5000.htm

[15] National Institute of Standards and Technology. (1999). *Data Encryption Standard (DES)* (FIPS PUB 46-3).

[16] National Institute of Standards and Technology. (2001). *Advanced Encryption Standard* (FIPS 197).

[17] National Institute of Standards and Technology. (2002). *Secure Hash Standard (SHS)* (FIPS 180-2).

[18] National Institute of Standards and Technology. (2004). *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher* (Special Publication 800-67). Gaithersburg: Barker, W.

[19] National Security Agency. (1995). *Capstone (MYK-80) Specifications* (NSA R21 Informal Technical Report R21-TECH-30-95).

[20] Ors, S. B., Gurkaynak, F., Oswald, E. and Preneel, B. (2004). Power-Analysis Attack on an ASIC AES implementation. *Proceedings of the International Conference on Information Technology: Coding and Computing, Volume 2*, 546.

[21] Rivest, R. L., Shamir, A., and Adleman, L. M. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM, 21*, 120-126.

[22] Satyanarayana, H. (2004). aes_crypto_core. Retrieved August 29, 2007, from http://opencores.mirrorgeek.com/projects.cgi/web/aes_crypto_core/

[23] Tiri, K. & Verbauwhede, I. (2004). A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. *Proceedings of the Conference on Design, Automation and Test in Europe, 1*, 10246.

[24] Trichina, E., De Seta, D. and Germani L. (2002). Simplified Adaptive Multiplicative Masking for AES. *Lecture Notes in Computer Science, Vol. 2523*, 187-197.

[25] van Eck, W. (1985). Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk?. *Computers & Security, 4*, 269-286.