Scientific Research Publishing

Journal of Information Security, 2017, 8, 383-401
http://www.scirp.org/journal/jis
ISSN Online: 2153-1242
ISSN Print: 2153-1234

Ambiguous Multi-Symmetric Scheme and Applications

Richard Bassous[1], Ahmad Mansour[1], Roger Bassous[1], Huirong Fu[1], Ye Zhu[2], George Corser[3]

[1]Oakland University, Rochester, MI, USA
[2]Cleveland State University, Cleveland, OH, USA
[3]Saginaw Valley State University, Saginaw, MI, USA
Email: rbassous@oakland.edu, aamansour@oakland.edu, rbassou2@oakland.edu, fu@oakland.edu, y.zhu61@csuohio.edu, gpcorser@svsu.edu

How to cite this paper: Bassous, R., Mansour, A., Bassous, R., Fu, H., Zhu, Y. and Corser, G. (2017) Ambiguous Multi-Symmetric Scheme and Applications. *Journal of Information Security*, **8**, 383-401.
https://doi.org/10.4236/jis.2017.84024

Received: September 30, 2017
Accepted: October 28, 2017
Published: October 31, 2017

Abstract

This paper introduces and evaluates the performance of a novel cipher scheme, Ambiguous Multi-Symmetric Cryptography (AMSC), which conceals multiple coherent plain-texts in one cipher-text. The cipher-text can be decrypted by different keys to produce different plain-texts. Security analysis showed that AMSC is secure against cipher-text only and known plain-text attacks. AMSC has the following applications: 1) it can send multiple messages for multiple receivers through one cipher-text; 2) it can send one real message and multiple decoys for camouflage; and 3) it can send one real message to one receiver using parallel processing. Performance comparison with leading symmetric algorithms (DES, AES and RC6) demonstrated AMSC's efficiency in execution time.

Keywords

Deniable Encryption, Multi Encryption, Honey Encryption, Steganography, Symmetric Encryption, Multi Encoding

1. Introduction

Deniable encryption prevents attackers from knowing with certainty whether or not a particular sender or receiver can be linked to a specific plain-text message. This paper addresses the *deniable encryption* problem by proposing a new cipher scheme, Ambiguous Multi-Symmetric Cryptography (AMSC), which conceals multiple plain-texts, each with its own key, in one cipher-text. The deniable encryption problem is important because most encryption schemes are defenseless against an attacker once she possesses the key. Deniable encryption provides

an additional layer of protection. With multiple plain-texts concealed in one cipher-text, an attacker cannot be certain which plain-text is genuine even if she possesses the cipher-text and one or more of the keys.

Several recent efforts in the area of deniable encryption have demonstrated the possibility of hiding/protecting the sender or receiver from revealing the decryption key when force is used. Following the early work of Canetti *et al*. [1], a variety of methods for "deniable encryption" have already been presented, including, Kamouflage [2] and Honey Encryption [3]. These methods can protect against offline and brute force attacks on the encrypted data, as they provide multiple decoy coherent messages to fool the adversary. Unfortunately, they cannot be used for online secure communications.

Ideally, we want a deniable encryption scheme that: 1) Defends both communication parties against decryption key exposure. 2) Has good performance in both encryption and decryption. 3) Is secure against different attack models. For a), AMSC defends both sender and receiver by providing multiple decoy keys. As for b), AMSC has an initialization phase that speeds up the original encryption [4] tremendously. For c), we introduce a security operation in the encryption step that helps secure AMSC against Cipher-text only and Known plain-text attacks.

This problem is non-trivial due to the complexity of concealing multiple messages into one message. This problem can simply be solved by encrypting n messages and concatenating the sub cipher-texts into one cipher-text. However, this could lead to rubber-hose cryptanalysis [5] on the receiver if the adversary observes that sub cipher-texts and not the whole cipher-text is being decrypted. The adversary will continue to use force on the receiver to reveal more possible messages. Another problem with this approach happens if the adversary intercepts parts of the whole cipher-text. Theoretically, it could reveal one or more concatenated messages. While a partial cipher-text in AMSC does not reveal any message.

AMSC's applications include multicast messaging and broadcast encryption. One video channel could generate multiple unique channels for different receivers. A second application is to deny the correct plain-text and key from the adversary by providing decoys. The third application is to use parallel computing to split one message into chunks and encrypt it using AMSC. This is possible due to the independent encryption operations that can run on different cores in parallel. This allows for faster encryption of a single message.

The primary contributions of this paper are as follows. First, compared with [4], the encryption performance is enhanced by introducing a new algorithm. Second, the security is improved by introducing an extra operation in the encryption process, without affecting performance. Third, AMSC's performance is compared to leading symmetric algorithms like DES, AES and RC6. Fourth, the security analysis is researched in more detail, notably the security of keys used, in addition to introducing another probabilistic approach. Fifth, computa-

tional complexity analysis is performed on the new algorithm.

The remainder of this paper is organized as follows: Section 2 provides background and related work, and compares our scheme to others. Section 3 defines the scheme. Section 4 proposes a new algorithm and presents its applications. Section 5 studies the security and possible attacks and shows a probabilistic solution. Section 6 examines the time complexity. Section 7 shows the results of our experiments.

Finally, Section 8 concludes.

2. Background and Related Work

Canetti *et al*. [1] proposed a "Deniable Encryption", which is a theoretical approach to deny someone the original plain-text when they get hold of the cipher-text and the right decryption key. Assume that Bob sends an encrypted message to Alice, and later on, Trudy holds Bob hostage until Bob releases the key. The released key will provide a fake plain-text. Canetti distinguished between two models:

1) multi-distributional deniability, requires the users to know in advance which messages they might want to conceal, whereas

2) full deniability, allows the user to decide afterward

Canetti presented a sender deniable scheme, using this first model. They also constructed a receiver deniable scheme that requires an additional round of interaction, and a sender-receiver-deniable protocol that relies on third parties.

One proposed scheme for denying symmetric encryption by Canetti would be to give n alternative messages to encrypt, and use n different keys, then construct the cipher-text as the concatenation of the encryption of all messages as shown in **Figure 1**. This is called a plan ahead scheme, where the i-th message is encrypted using the i-th key. One disadvantage of concatenation, is dealing with offsets at the recipient's side. If the cipher-text size changes, offsets have to change accordingly. All offsets have to be re-communicated from the sender to all receivers every time the cipher-text size changes, as changing the number or size of messages will affect the offsets. On the other hand, AMSC generates a variable cipher-text size without using offsets for sub cipher-texts. The only condition is that each key has to be bigger than its plain-text. The other advantage of AMSC over concatenation, is in intercepting the cipher-text. Assume that a concatenated cipher-text has 3 cipher-texts that are 50 bytes each. If the adversary gets hold of part of the cipher-text, say the last 70 bytes, then at least one key and one plain-text are exposed. Therefore, partial message eavesdrop could reveal a plain-text. In AMSC however, if this happens, the cipher-text would be

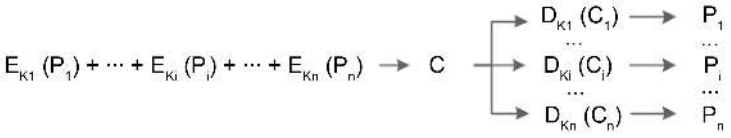

Figure 1. One scheme for plan ahead symmetric deniable encryption [1].

incomplete and would not reveal any information about any message. Concatenation could also lead to rubber-hose cryptanalysis [5], as the adversary might notice that a partial cipher-text was decrypted, and continue to use force to reveal more possible keys.

Kamouflage system [2] is used to store multiple decoys for each real password in a local password manager database like Firefox. Also a set of decoy master passwords (MB) on the database are generated. If the adversary cracks a decoy MB, they will get hold of decoy password sets. Kamouflage is only used to protect the local password manager of the stolen device.

Juels and Ristenpart [3] introduced "Honey Encryption" (HE). It's a method that creates plausible deniability for low min-entropy keys (like short passwords). HE generates a seed using a method called distribution-transforming encoder (DTE), from a message P, that belongs to a specific message space M (ex: credit card numbers). This seed is then encrypted by a conventional encryption algorithm. When an adversary tries to decrypt cipher-text C, plausible fake honey messages will be decrypted. Each application needs a construction of a different DTE. Ex: a DTE for RSA secret keys is different from a DTE for credit card numbers. Furthermore, HE is tightly coupled with password based encryption (PBE). HE security does not hold when the adversary has side information about the target message [3] (ex: if the adversary knows the public key in RSA, HE fails). Both "Kamouflage" and "Honey Encryption" protect against offline or online attacks by providing decoys. On the other hand, AMSC is used in secure communications. AMSC can send the same message with different interpretations to different receivers. In addition, AMSC can encrypt from a natural language message space, where HE for example, is focused on certain message spaces (ex: credit card numbers, RSA secret keys). Moreover, AMSC has the ability to deny encryption when force is used to reveal the keys, where both previous systems (Kamouflage and HE) cannot [3].

ONeill *et al*. [6] provided a public key solution, where both sender and receiver can use deniability without relying on any third party. The solution is based on Multi-distributional deniability. They defined a new term "bi-deniable encryption" which allows a sender in possession of the receiver's public key to communicate a message to the latter, confidentially. Additionally, if the parties are later coerced to reveal all their secret data namely, the coins used by the sender to encrypt her message and/or those used by the receiver to generate her key, bi-deniable encryption allows them to do so as if any desired message (possibly chosen as late as at the time of coercion) had been encrypted.

Sahai *et al*. [7] defined an identity based encryption (IBE), which allows sending an encrypted message to an identity without using a public key certificate. A user with a secret key K for the identity w is able to decrypt a cipher-text encrypted with the public key w' IFF w and w' are within a certain distance of each other by some metric. A document for example can be decrypted by a certain identity or group. They use biometric for IBE to generate

keys from a trusted authority, afterwords, distribute a master secret key using Shamir into multiple private components, one for each attribute in the user's identity, then only a subset of these private keys are necessary to decrypt the cipher-text.

A secret sharing scheme [8] follows a similar scheme as AMSC. It defines $A(k,n)$-threshold scheme as a method of sharing a secret S among a set of n participants in such a way that any k participants can compute the value of the secret, but no group of $k-1$ or fewer can do so. The Chinese remainder theorem can be used to construct the secret S like in Mignotte's [9] and Asmuth-Bloom's Schemes [10]. However, it differs, as the secret points to one message, and k shares are needed to solve it using CRT.

3. AMSC Scheme

Our scheme conceals various plain-texts into one cipher-text, hence the name "Multi-Symmetric". **Figure 2** shows the system model. The scheme can be defined in three steps: Let $P_1, P_2, \cdots, P_n$ be plain-texts, $K_1, K_2, \cdots, K_n$ be keys accordingly, then:

1) Alice exchanges a number of AMSC co-prime keys with Bob. For added security, Alice can also exchange X which is the multiplication of all keys.[1]

2) Alice generates cipher-text:

$$C = E_{AMSC}\left(\left[K_1, K_2, \cdots, K_n\right], \left[P_1, P_2, \cdots, P_n\right]\right).$$

3) Bob decrypts C using key K_i and gets P_i.

Table 1 shows a glossary of symbols used in this paper.

4. AMSC Algorithm

In this section, we present a new algorithm (AMSC v3) that satisfies the previous scheme. This algorithm enhances the performance of the two algorithms presented in [4] by introducing an initialization phase. This speeds up AMSC tremendously. Furthermore, the security is enhanced by adding an *XOR* operation to the encryption as a final step between the cipher-text and the multiplication of all keys. This strengthens the AMSC algorithm against the known plain-text attacks as explained in more detail in Section 5.1.2. The security and performance will be further discussed in Sections 5 and 7.

The AMSC algorithm is based on the Chinese Remainder theorem (CRT) [11] that is used to calculate the cipher-text.

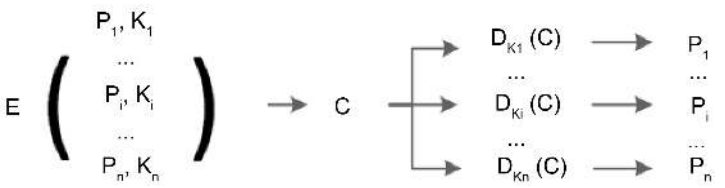

Figure 2. Ambiguous multi-symmetric scheme.

[1]Keys exchanges are out of scope of this paper.

Table 1. Glossary of symbols.

Symbol	Description
C	Cipher-text
K_i	Encryption key i
Ka	Average size of all keys (1, ···, n) in bits
P_i	Plain-text block i
P_a	Average size of all plain-texts (1, ···, n) in bits
E	Encryption algorithm
D	Decryption algorithm
a_i	Unknown variable that is used in the formula to calculate cipher-text, $C = K_i a_i + P_i$
n	Number of plain-text(s) to be encrypted
v	Minimum number of steps to find all possible keys (if K_i s are primes), $v = C/\ln(C)$
X	The multiplications of all keys (1, ···, n)
r_i, s_i	The roots of the extended-GCD algorithm such that $r_i K_i + s_i X/K_i = 1$. s_i is the modular multiplicative inverse of X/K_i modulo K_i
GCD	The greatest common denominator
CRT	The Chinese remainder theorem
AES	The advanced encryption standard
DES	The data encryption standard
$RC6$	The Rivest cipher 6 algorithm

CRT Theorem: Suppose that $K_1, K_2, \cdots, K_n$ are pairwise relatively prime positive integers, and let $P_1, P_2, \cdots, P_n$ be integers. Then the system of congruences, $C \equiv P_i \pmod{K_i}$ for $1 \le i \le n$, has a unique solution modulo $X = K_1 * K_2 * \cdots * K_n$, which is given by:

$C \equiv P_1 X_1 s_a + P_2 X_2 s_2 + \cdots + P_n X_n s_n \pmod{X}$, where $X_i = X/K_i$ and $s_i \equiv (X_i)^{-1} \pmod{K_i}$ for $1 \le i \le n$.

The AMSC algorithm prepares $X_i * s_i, i = 1, \cdots, n$ from above in the initialization step (calculated once). Afterwords, the encryption multiplies all plain-texts $P_1, \cdots, P_n$ with the initialization values and calculates the cipher.

4.1. Initialization

The first part initializes AMSC values that are used in encryption. We calculate X (the multiplication of all keys) and a set of numbers $s_i * X/K_i$ for each key $K_i, i = 1, \cdots, n$.

These values are calculated only once and not per encryption. These are the steps needed to initialize:

1) Multiply keys $K_{i,\cdots,n}$ to get a number X.

2) Use the extended Euclidean algorithm to find the roots r, s for every key

K_i such that:

$$r_i(K_i) + s_i(X/K_i) = 1 \tag{1}$$

Algorithm 1 shows AMSC v3 Initialization.

Algorithm 1: AMSC v3 Initialization

input : $K_1..K_n$ keys corresponding to $P_1..P_n$, $K_i > P_i$ and $GCD(P_i, K_i) = 1$
output: AMSC_initial_values[$i = 0..n$]: when $i = 0$, then multiplications of all keys X, when $i > 0$, then $s_i * X/K_i$ for each key using the extended GCD

Initialize array AMSC_values to size $n + 1$;
$X = 1$;
for $i \leftarrow 1$ **to** n **do**
  $X* = K_i$
end
$AMSC\_values[0] = X$;
for $i \leftarrow 1$ **to** n **do**
  /* Per the extended Euclidean [11] roots r_i and s_i | $r_i * K_i + s_i * X/K_i = 1$ */
  $(GCD, r_i, s_i) = \text{ExtendedEuclidean}(K_i, X/K_i)$;
  $AMSC\_values[i] = s_i * X/K_i$;
end
return $AMSC\_values$;

4.2. Encryption

After initialization, subsequent cipher-texts are calculated by:

$$C = \sum_{i=1}^{n} P_i s_i \, X/K_i \tag{2}$$

where $s_i \, X/K_i$ is calculated in the initialization step. There is an option to *XOR* the final cipher-text C to X. This will make AMSC secure against known plain-text attacks as discussed in detail in Section 5.

$$C = \left(\sum_{i=1}^{n} P_i s_i \, X/K_i \right) \oplus X \tag{3}$$

Algorithm 2 shows the steps for AMSC v3 Encryption.

Algorithm 2: AMSC v3 Encryption

input : Array AMSC_initial_values: where AMSC_values[0] is the multiplications of all keys X and AMSC_values[i] is the s_i value calculated for each key using the extended GCD
$P_1..P_n$ the plain-text blocks to be encrypted
$K_1..K_n$ the keys corresponding to $P_1..P_n$ accordingly, $K_i > P_i$ and all K_is are co-primes
xorCipher: boolean flag if set then XOR the cipher-text with the multiplications of all keys X
output: Cipher-text C, where C can be decrypted using any key K_i to its corresponding plain-text P_i

$C = 0$;
for $i \leftarrow 1$ **to** n **do**
  $C+ = P_i * AMSC\_values[i]$;
end
if *xorCipher* **then**
  $C = (C \mod AMSC\_values[0]) \oplus AMSC\_values[0]$;
else
  $C = C \mod AMSC\_values[0]$;
end
return C ;

4.3. Decryption

The decryption simply takes the cipher-text C and mods it with the corresponding key K_i. If the *XOR* operation is used in the encryption to

strengthens the cipher, then the input for decryption needs X besides the cipher-text. One important note: If all keys are primes, and the receiver knows X and her key K_i, it is not possible to know the rest of the keys as this is a factorization problem. Algorithm 3 shows the steps for AMSC v3 Decryption.

Algorithm 3: AMSC v3 Decryption

input : Cipher-text C
Key K_i
Multiplications of all keys X
xorCipher: boolean flag if set then XOR the cipher-text with the multiplications of all keys X

output: Plain-text P_i

```
if xorCipher then
    P_i = (C ⊕ X) mod K_i;
else
    P_i = (C mod K_i);
end
return P_i ;
```

4.4. Example

Let $n=4$, $Pa=64$ bits and $Ka=65$ bits.

Assume we use prime keys (we can use co-primes as well):

$K_1=36893488147419103183$, $K_2=36893488147419103153$,

$K_3=36893488147419103117$, $K_4=36893488147419103091$ and plain-texts

$P_1=5407036729192671602$, $P_2=12217864333306969557$,

$P_3=9169178348075514855$, $P_4=8659079797496077286$.

Using AMSC with no XOR operations, we calculate cipher-text

$$C=1639418630032050050243577119286873823995 3 -7590007926788873589979804380708621632 9$$

5. Security Analysis

In this section, we evaluate our algorithm under a variety of security attack models, including a thorough study on prime and co-prime keys. Then, we present a probabilistic solution for the encryption process.

5.1. Security Attack Models

5.1.1. Cipher-Text only Attack [12]

When one cipher-text is intercepted, a brute force attack [13] is one way to crack the encryption. AMSC can use prime or co-prime keys. Both will be analyzed.

1) Primes: The prime number theorem states that there are approximately $C/\ln(C)$ primes <= C. The size of the cipher-text depends on three factors: the average block size *Pa*, the number of blocks *n*, and the average key size *Ka*. **Table 2** shows how the three factors *n*, *Pa*, *Ka* affect the cipher-text size, along side the number of steps needed to find all prime keys.

2) Co-primes: For any cipher-text C, the number of sets of positive integers ≤ C in which two elements are co-primes lies between

$$2^{\Pi(C)} * e^{(1/2+O(1))*\sqrt{C}} \text{ and } 2^{\Pi(C)} * e^{(2+O(1))*\sqrt{C}} \quad (4)$$

by Theorem 3.3 of Cameron and Erdos [14]. $\Pi(C)$ is defined as the prime

Table 2. Three factors *n*, *Pa*, *Ka* that affect cipher-text size.

# of blocks (*n*)	average block size in bits (*Pa*)	Average key size in bits (*Ka*)	Cipher-text (*C*)	Min # of steps to find all prime keys ($C/\ln(C)$) (*v*)
16	8	9	2^{130}	2^{123}
8	16	17	2^{127}	2^{120}
4	32	33	2^{129}	2^{123}

counting function of *C*.

Nathanson [15] improved this in Theorem 2 as follows:

$$2^C - 2^{\lfloor C/2 \rfloor} - C * 2^{\lfloor C/3 \rfloor} <= F(C) <= 2^C - 2^{\lfloor C/2 \rfloor} \tag{5}$$

where $F(C)$ is the number of relatively prime subsets of $\{1,2,\cdots,n\}$.

Furthermore, Nathanson derived an approximation $F_n(C)$ for the number of n-elements sets of positive integers ≤ *C* in which two elements are co-primes:

$$\begin{aligned} &\binom{C}{n} - \binom{\lfloor C/2 \rfloor}{n} - C * \binom{\lfloor C/3 \rfloor}{n} \\ &<= F_n(C) <= \binom{C}{n} - \binom{\lfloor C/2 \rfloor}{n} \end{aligned} \tag{6}$$

Using Equation (6), we construct **Table 3** to approximate the number of co-prime sets based on the size of cipher-text *C* and the number of plain-text(s) *n*. **Table 4** shows the number of all co-prime subsets ≤ cipher-text *C*. It also shows the number of co-prime subsets if $C \oplus X$ is used.

To find all elements of the co-prime sets we can use different methods:

- *n* = 2: if we want to find all pair sets that are co-primes ≤ *C*, we can use the Farey sequence [16]. There exists an algorithm [17] to find all sets ≤ *C* in $O(C^2)$ time complexity.
- *n* = 3: if we want to find all triplet sets that are co-primes ≤ *C*, we can use the primitive Pythagorean triples [18]. One formula for finding all primitive triplets ≤ *C* is the Euclid's Formula. The Time complexity of this formula is $O(C * \log(C))$ [19].
- *n* > 3: In this case we can examine all subsets where *n* = 2 and chain them together to generate the subsets with the required *n*.

3) The *XOR* operation has been widely used in cryptography, especially in symmetric key cryptography [20] [21] [22]. The security of *XOR* mainly depends on the key strength, where it must be extremely difficult for the adversary to predict the key. In addition, the key and the message should have the same length to avoid repetition. With these two conditions, the brute force attack is the only possible attack that can be used to break the cipher-text [23] [24]. To break an encrypted message of size *n* bits, the adversary needs 2_n steps. This process is computationally infeasible even for small values of *n*.

In this work, we introduce an *XOR* between the cipher-text *C* with *X* (The multiplication of all keys). This is done to break the mathematical pattern. In

Table 3. The number of relatively prime subsets of $\{1,2,\cdots,C\}$ of cardinality n, where C is the AMSC cipher-text.

Cipher-text size	$n=2$	$n=3$	$n=10$	$n=20$
2^{64}	2^{127}	2^{190}	2^{619}	2^{1219}
2^{128}	2^{255}	2^{382}	2^{1259}	2^{2499}
2^{256}	2^{511}	2^{766}	2^{2539}	2^{2059}

Table 4. The number of all relatively prime subsets of $\{1,2,\cdots,C\}$ of any cardinality.

Cipher-text size	Lower bound for the number of co-prime subsets [15]	Lower bound for the number of co-prime subsets when C is *XORed* with X
2^{64}	$2^{4.16\times10^{17}}$	$2^{64}\times2^{4.16\times10^{17}}$
2^{128}	$2^{3.835\times10^{36}}$	$2^{128}\times2^{3.835\times10^{36}}$
2^{256}	$2^{6.525\times10^{74}}$	$2^{256}\times2^{6.525\times10^{74}}$

other words, if there is any kind of attacks that uses mathematical operations to break the cipher-text C and extract the keys, then it will be of no use after the *XOR* operation. Moreover, *XOR* defends against the known plain-text attacks as discussed in Section 5.1.2.

5.1.2. Known Plain-Text Attack [25]

In a classical attack, the adversary can examine one plain-text to its cipher-text and tries to reveal the key. In AMSC, however, the adversary has multiple inputs and one output. We will study two cases. One, where the adversary only knows one plain-text and the rest are unknown, and the second, we assume that all plain-texts are known going into the oracle.

1) one plain-text is known: The adversary does not know the total number of plain-texts n or their contents. The oracle generates the final cipher-text C. The adversary has to solve the equation: $P_i = C \bmod K_i$, where P_i and C are known. No one solution is possible. If keys are primes, then a possible prime factorization (computationally infeasible) of $C - P_i$ might reveal one possible key K_i.

2) n plain-texts are known: The adversary examines n plain-texts and their cipher-texts for each encryption. We end up with n equations for each cipher:

$$\begin{aligned} C_1 &\equiv P_{i1} \left(\bmod K_i\right), i=1,\cdots,n \\ C_2 &\equiv P_{i2} \left(\bmod K_i\right), i=1,\cdots,n \\ &\vdots \\ C_z &\equiv P_{iz} \left(\bmod K_i\right), i=1,\cdots,n \end{aligned}$$

We know that:

K_i is a divisor of $\left(C_1 - P_{i1}, C_2 - P_{i2}, \cdots, C_z - P_{iz}\right)$

therefore:

$$K_i \left| GCD\left(C_1 - P_{i1}, C_2 - P_{i2}, \cdots, C_z - P_{iz}\right)\right. \tag{7}$$

where GCD is the greater common denominator. We have to find the *GCD* of z − 1 numbers which has a computational complexity of $O\left(z-1*\left(\log\left(C_1-P_{i1}\right)\right)\right)$.

As more ciphers are calculated and z approaches ∞, the *GCD* gets close to K_i. To mitigate this issue, we do $C \oplus X$ in the last step of encryption.

5.1.3. Chosen Plain-Text Attack (CPA) [26]

This case is very similar to Section 5.1.2. However, *XOR* can not help in this case because the adversary can feed their own plain-texts. The adversary can reveal X in such a simple way:

Let all plain-texts $P_{1,\cdots,n}=0$, then

$$C \oplus X = 0 \oplus X = X$$

Once X is known, subsequent oracle cipher-texts can be *XORed* with X to produce the original cipher-texts. Afterwords, the GCD can be used to reveal the keys as stated previously. We conclude that AMSC is not secure against this attack if the adversary chooses all plain-texts. We know that chosen plain-text attack and chosen cipher-text attack fail with all deterministic algorithms. Hence we have to use probabilistic approaches as discussed in Section 5.2.

5.1.4. Chosen Cipher-Text Attack (CCA) [27]

This attack happens when the adversary has access to the decryption oracle.

AMSC is not CCA immune in the current form. We can start with $C=1$ (**Table 5**) and keep doubling the cipher-text input until the output plain-text becomes smaller than the previous value. Example: Let $K_i=75$. When output $P=53$, we stop and calculate $K_i=C-P=128-53=75$.

To mitigate this, when we can add the *XOR* operation $C \oplus X$ at the end of encryption, then we would have two cases for X:

1) X is odd: The adversary can find the key by feeding the oracle $C=1$. The reason is:

$\left(C \oplus X\right)=X-1$. This is due to the add without carry in the *XOR* operation. Ex:

if $X=9=\left(1001\right)_2$ and $C=1$ then $\left(C \oplus X\right)=\left(1000\right)_2=X-1$.

We also know that:

$\left(X-1 \bmod K_i\right)=K_i-1$, since K_i is a divisor of X. Therefore:

Table 5. Example of chosen cipher-text attack, where $K_i=75$.

Cipher-text	Plain-text ($C \bmod K_i$)
1	1
2	2
4	4
8	8
16	16
32	32
64	64
128	53

$$P_i = (C \oplus X) \bmod K_i$$

$$P_i = (X - 1 \bmod K_i) = K_i - 1$$

$$K_i = P_i + 1$$

2) X is even: The previous case will not work. We can choose X to be even by having only one of the keys K_i as even. This will strengthens the security of AMSC against CCA attacks.

5.2. Deterministic vs. Probabilistic

AMSC is deterministic. We present two approaches to make AMSC probabilistic [28]:

- First approach: We construct:

$$C = C_0 + t * LCM(K_1, K_2, \cdots, K_n) \tag{8}$$

where C_0 is a base solution using CRT, t is any random integer and LCM is the least common multiplier of all keys. Note that

$LCM(K_1, K_2, \cdots, K_n) = K_1 * K_2 * \cdots * K_n / GCD(K_1, K_2, \cdots, K_n) = K_1 * K_2 * \cdots * K_n$

We can use variable t as a random Initialization vector (IV) to yield different cipher-texts:

$$C_1 = E_{\text{AMSC}}([K_1, K_2, \cdots, K_n], [P_1, P_2, \cdots, P_N])$$

$$C_2 = E_{\text{AMSC}}([K_1, K_2, \cdots, K_n], [P_1, P_2, \cdots, P_N])$$

$$C_i = E_{\text{AMSC}}([K_1, K_2, \cdots, K_n], [P_1, P_2, \cdots, P_N])$$

where $C_1 \neq C_2 \neq \cdots \neq C_i \ \ i = 1, \cdots, n$.

- Second approach: We define another probabilistic solution. Let K_r, P_r be a random key and a random plain-text accordingly. The cipher-text will become:

$$\left(\sum_{i=1}^{n} P_i * s_i * X / K_i\right) + P_r * s_r * X / K_r \tag{9}$$

The random key and plain-text can be re-generated for every encryption. In the encryption phase, we only need to calculate $P_r * s_r * X / K_r$ once, and then add it to the cipher-text as a final step. This random key will make the cipher-text probabilistic with a small increase in size. Ex: for n = 4, where average block size $P_a = 32$ and average key size $K_a = 33$. For a deterministic cipher-text, the average size is about 129 bits. For a probabilistic cipher-text using approach 2 by adding a random key, the size grows to about 163 bits, a difference of about 34 bits. For a probabilistic cipher-text using approach 1 by setting the random IV $t = 150$, the cipher-text grows to about 139 bits. For $t = 1500$ the cipher-text grows to about 143 bits.

We compare Equations (8) and (9), and examine the cipher-text size. When $n = z$, we calculate $X_z = \sum_{i=1}^{z} K_i$. When we add a random key K_r, n becomes $z + 1$. Thus we have:

$X_{z+1}/K_r = X_z$. Therefore, Equation (9) will have a smaller cipher-text size than the Equation (8) iff $Pr * Sr < t$.

6. Computational Complexity Analysis

We know that multiplication, division and modular operations take $O(d^2)$ [29], addition takes $O(d)$ [29], where d is the number of decimal digits of the largest operands. For base 10 number X, that would be $O\left(\left(\lfloor \log(X) \rfloor + 1\right)^2\right)$ and $O\left(\lfloor \log(X) \rfloor + 1\right)$ respectively.

- In initialization, the first loop takes $n\left(\lfloor \log(X) \rfloor + 1\right)^2$ steps. In the second loop, the GCD takes $\left(\log(K_i) * \log(X/K_i)\right)$ [30], then a multiplications and a division. Overall, $n\left(\log(K_i) * \log(X/K_i) + 2\left(\lfloor \log(X) \rfloor + 1\right)^2\right)$. Therefore the time complexity is: $O\left(n\left(\lfloor \log(X) \rfloor + 1\right)^2\right)$, where n is the number of keys and X is the multiplication of keys.
- In encryption, we loop n times and do an addition and a multiplication of numbers close to X. Therefore, the overall time complexity for encryption is $O\left(n\left(\lfloor \log(X) \rfloor + 1\right)^2\right)$.
- To decrypt cipher-text C using key K_i, we have a time complexity of $O\left(\left(\lfloor \log(C) \rfloor + 1\right)^2\right)$, as the operation is $P_i = C \bmod K_i$.

Table 6 shows the time complexity for all versions of AMSC.[2]

7. Experimental Study Analysis

In this section, we evaluate the new algorithm AMSC v3 against different symmetric algorithms with different key sizes.

7.1. Experimental Setup

All experiments were done on an Intel Core *i7* 3610*QM* CPU with 8 GB memory. The AMSC core library and the symmetric algorithms comparison were implemented in .NET 4.0 using C# programming language. Each symmetric algorithm is measured in three different phases. The first, is initializing n cipher-text classes and creating n random keys that will be used for encryption/decryption. The second is encrypting n different random plain-texts using the previous keys accordingly, and then concatenating the sub cipher-texts. This gives us a fair comparison to AMSC. On the decryption side,

Table 6. Time complexity analysis of AMSC 1, 2 [4], and 3.

	Initialization	Encryption	Decryption
AMSC 1	NA	$O\left(nz\left(\lfloor \log(K_i) \rfloor + 1\right)^2 + nz^2\right)$	$O\left(\left(\lfloor \log(C) \rfloor + 1\right)^2\right)$
AMSC 2	NA	$O\left(n\left(\lfloor \log(X) \rfloor + 1\right)^2\right)$	$O\left(\left(\lfloor \log(C) \rfloor + 1\right)^2\right)$
AMSC 3	$O\left(n\left(\lfloor \log(X) \rfloor + 1\right)^2\right)$	$O\left(n\left(\lfloor \log(X) \rfloor + 1\right)^2\right)$	$O\left(\left(\lfloor \log(C) \rfloor + 1\right)^2\right)$

[2] z is the number of solutions that has to be intersected using AMSC v1.

we decrypt each sub-cipher-text by the its key to get back the original plain-text. We run each operation a total of 100,000 times and take the average. The total time for each operation is measured. For AES and DES, we used the built in .Net crypto libraries AESCryptoServiceProvider and DESCryptoServiceProvider respectively which are both Fips certified [31] libraries. As for RC6, we used Bouncy Castle's [32] crytpo library *v*1.7. Table 7 defines the legends that are used in the results.

7.2. Experimental Results

7.2.1. Initialization: AMSC, DES, AES and RC6

Figure 3 compares the execution time of AMSC's initialization to that of DES, AES and RC6. The initialization time for the symmetric algorithm includes initializing n cipher-texts objects with n random keys, and getting them ready for encryption or decryption.

Table 7. Experiment definitions.

Legend	Definition
AMSC number	AMSC with key size in bits
AMSC number $\oplus X$	AMSC, where final cipher-text is *XORed* with X
DES 64	DES symmetric algorithm with 64-bit key (56-bit + 8-bit for parity)
AES number	AES symmetric algorithm with number-bit key
RC6 number	RC6 symmetric algorithm with number-bit key

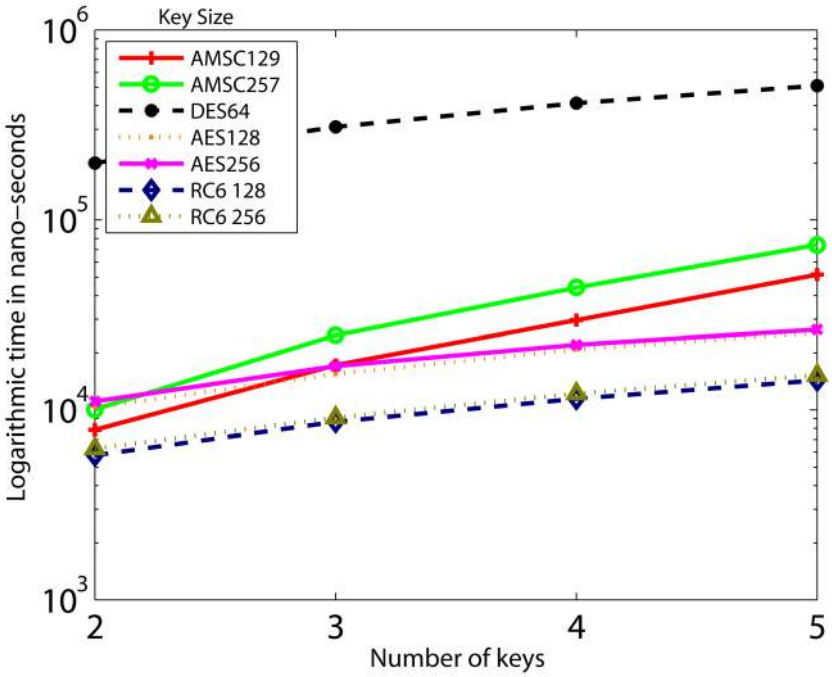

Figure 3. Initialization: AMSC with keys 129 and 257 bits, DES with key 64-bit, AES and RC6 with keys 128 and 256 bits.

DES uses 64-bit keys. AES and RC6 use both 128-bit and 256-bit keys. AS for AMSC, we pick 129-bit and 257-bit keys. These keys are very close to their counter part AES and RC6. Furthermore, they will be used in the encryption and decryption experiments. Recall that every AMSC key has to be greater than its plain-text block. In the case of DES, the plain-text block is 64-bit. AES and RC6, both use 128-bit plain-text block. Note that AMSC's initialization time grows linearly as n increases. Nonetheless, it still has smaller initialization time than DES.

7.2.2. Encryption: AMSC, DES, AES and RC6

For symmetric algorithms we encrypt n plain-texts using n keys for the n cipher-text objects that were initialized, and then concatenate all the sub cipher-texts into one final cipher-text. This makes it fair to compare against AMSC. Figure 4(a) compares the execution time of AMSC encryption to that of

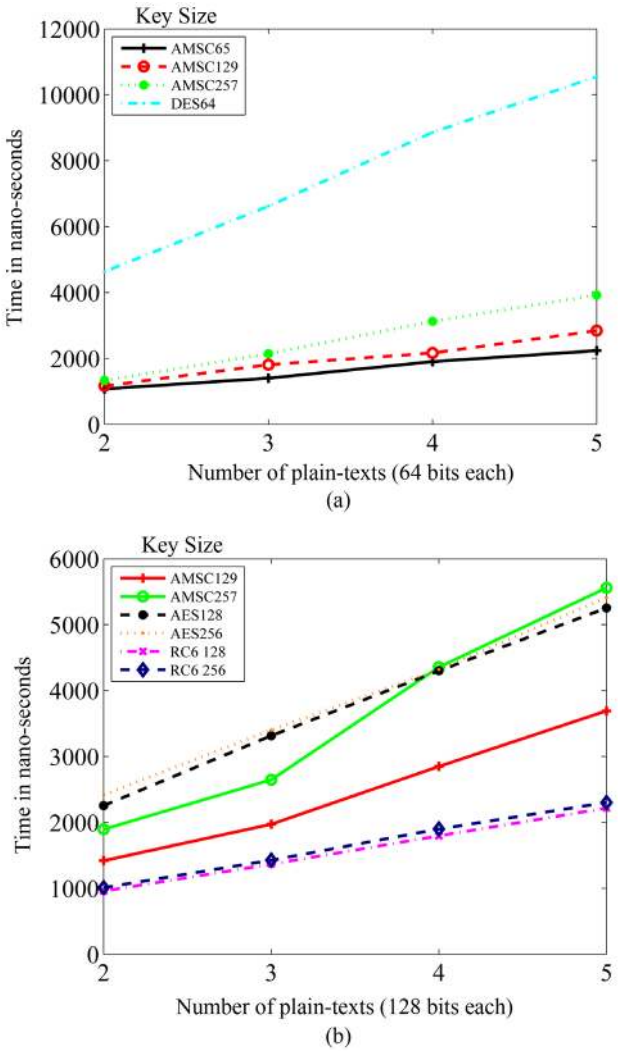

Figure 4. Encryption execution time. (a) AMSC and DES with 64-bit plain-text block and different key sizes. (b) AMSC, AES and RC6 with 128-bit plain-text block and different key sizes.

DES using a 64-bit block size with three different size keys for AMSC. Note that AMSC's encryption time is significantly less than that of DES. Furthermore, Figure 4(b) uses 128-bit block size to compare AMSC with AES and RC6. For AES, AMSC 129-bit beats AES 128-bit keys. AMSC 257-bit has better performance until about 4 plain-texts. This is due to the multiplication of large numbers as n increases. As for RC6, AMSC is slower.

7.2.3. Decryption: AMSC, DES, AES and RC6

The total AMSC time to decrypt the same cipher-text into n plain-text messages using n keys is measured. For the symmetric algorithms, n sub cipher-texts are decrypted and time is measured. Figure 5(a) shows that AMSC is faster than DES. Figure 5(b) shows that AMSC 129-bit beats both AES 128-bit and AES 256-bit. However, AMSC 257-bit is slower than AES due to the time it takes to divide the large cipher-text by each key.

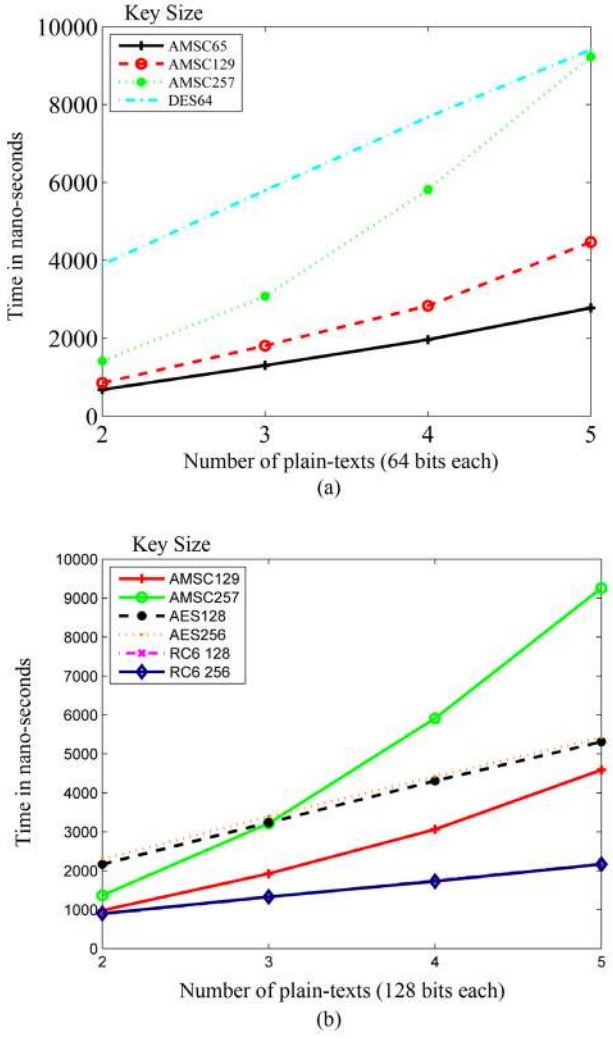

Figure 5. Decryption execution time. (a) AMSC and DES with 64-bit plain-text block and different key sizes. (b) AMSC, AES and RC6 with 128-bit plain-text block and different key sizes.

8. Conclusions

Deniable encryption offers an additional layer of protection for senders and receivers, who may be forced to give up encryption keys, or who may find it advantageous to have multiple plain-texts in one cipher-text. This paper showed that a novel system, ASMC, conceals multiple plain-texts in one cipher-text and performs competitively with more mainstream encryption techniques.

This paper showed that AMSC is a method for multi-key encoding and deniable encryption that withstands COA and KPA security attacks. AMSC's performance in initialization is faster than DES 64-bit but a little slower than AES. In Encryption, however, AMSC 129-bit is about 42% faster than AES 128-bit. On the decryption side, AMSC 129-bit is about 110% faster than DES 64-bit and 16% faster than AES 128-bit for 5 plain-texts.

Our future work in this area includes applying parallel computing to AMSC. We also like to explore different applications of AMSC in TV and other broadcasts.

Acknowledgements

The authors would like to thank Dr. Kruk for giving feedback. This research work is partially supported by the National Science Foundation under Grants CNS-1338105, CNS-1343141, CNS-1460897, DGE-1623713. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Canetti, R., Dwork, C., Naor, M. and Ostrovsky, R. (1997) Deniable Encryption. In: Advances in *Cryptology CRYPTO'97*, Springer, Berlin, 90-104. https://doi.org/10.1007/BFb0052229

[2] Bojinov, H., Bursztein, E., Boyen, X. and Boneh, D. (2010) Kamouage: Loss-Resistant Password Management. In: *Computer Security-ESORICS* 2010, Springer, Berlin, 286-302.

[3] Juels, A. and Ristenpart, T. (2014) Honey Encryption: Security beyond the Brute-Force Bound. In: *Advances in Cryptology-EUROCRYPT* 2014, Springer, Berlin, 293-310. https://doi.org/10.1007/978-3-642-55220-5_17

[4] Bassous, R., Bassous, R., Fu, H. and Zhu, Y. (2015) Ambiguous Multi-Symmetric Cryptography. In: *Communications (ICC)*, 2015 *IEEE International Conference on*, 7394-7399.

[5] Schneier, B. (1996) Applied Cryptography. 2nd Edition, John Wiley & Sons, Hoboken, New Jersey.

[6] ONeill, A., Peikert, C. and Waters, B. (2011) Bi-Deniable Public-Key Encryption. In: *Annual Cryptology Conference*, Springer, Berlin, 525-542.

[7] Sahai, A. and Waters, B. (2005) Fuzzy identity-based encryption. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, Berlin, 457-473. https://doi.org/10.1007/11426639_27

[8] Shamir, A. (1979) How to Share a Secret. *Communications of the ACM*, **22**,

612-613. https://doi.org/10.1145/359168.359176

[9] Mignotte, M. (1983) How to Share a Secret. In: *Cryptography*, Springer, Berlin, 371-375. https://doi.org/10.1007/3-540-39466-4_27

[10] Asmuth, C. and Bloom, J. (1983) A Modular Approach to Key Safe-Guarding. *IEEE Transactions on Information Theory*, **29**, 208-210. https://doi.org/10.1109/TIT.1983.1056651

[11] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. (2001) Introduction to Algorithms. 2nd Edition, MIT Press and McGraw-Hill.

[12] Biryukov, A. and Kushilevitz, E. (1998) From Differential Cryptanalysis to Ciphertext-Only Attacks. In: *Advances in Cryptology CRYPTO*'98, Springer, Berlin, 72-88. https://doi.org/10.1007/BFb0055721

[13] Reynard, R. (1997) Secret Code Breaker II: A Cryptanalyst's Handbook. Vol. 2, Smith & Daniel.

[14] Mollin, R.A. (1990) Number Theory. Proceedings of the 1st Conference of the Canadian Number Theory Association, Banff, 17-27 April 1988, Vol. 1.

[15] Nathanson, M.B. (2007) Affne Invariants, Relatively Prime Sets, and a Phi Function for Subsets of {1, 2,..., n}. *Integers*, **7**, A1.

[16] Norman Routledge (2008) Computing Farey Series. The Mathematical Gazette, 55-62.

[17] Stack Exchange (2013) Generating All Co-Prime Pairs within Limits. http://math.stackexchange.com/questions/422830/generating-all-coprime-pairs-within-limits

[18] Berggren, B. (1934) Pytagoreiska trianglar. *Elementa*: *Tidskrift för elementär matematik*, *fysik och kemi*, **17**, 129-139. (in Swedish)

[19] Stackoverflow (2013) Proof: Pythagorean Triple Algorithm Is Faster by Euclid's Formula? http://stackoverflow.com/questions/18294496/proof-pythagorean-triple-algorithm-is-faster-by-euclids-formula

[20] Bellare, M., Krovetz, T. and Rogaway, P. (1998) Luby-Rackoff Back-Wards: Increasing Security by Making Block Ciphers Non-Invertible. In: *Advances in Cryptology Eurocrypt*'98, Springer, Berlin, 266-280. https://doi.org/10.1007/BFb0054132

[21] Hall, C., Wagner, D., Kelsey, J. and Schneier, B. (1998) Building Prfs from Prps. In: *Advances in Cryptology Crypto*'98, Springer, Berlin, 370-389. https://doi.org/10.1007/BFb0055742

[22] Lucks, S. (2000) The Sum of Prps Is a Secure Prf. In: *Advances in Cryptology Eurocrypt* 2000, Springer, Berlin, 470-484. https://doi.org/10.1007/3-540-45539-6_34

[23] Paar, C. and Pelzl, J. (2009) Understanding Cryptography: A Textbook for Students and Practitioners. Springer Science & Business Media, Berlin.

[24] Menezes, A.J., Van Oorschot, P.C. and Vanstone, S.A. (1996) Handbook of Applied Cryptography. CRC Press.

[25] Singh, S. (2000) The Code Book: The Secret History of Codes and Codebreaking. Fourth Estate, London.

[26] Matsui, M. (1994) Linear Cryptanalysis Method for Des Cipher. In: *Advances in Cryptology Eurocrypt*'93, Springer, Berlin, 386-397. https://doi.org/10.1007/3-540-48285-7_33

[27] Cramer, R. and Shoup, V. (1998) A Practical Public Key Cryptosystem Provably

Secure against Adaptive Chosen Ciphertext Attack. In: *Advances in Cryptology CRYPTO*'98, Springer, Berlin, 13-25. https://doi.org/10.1007/BFb0055717

[28] Goldwasser, S. and Micali, S. (1984) Probabilistic Encryption. *Journal of Computer and System Sciences*, **28**, 270-299.

[29] Wikipedia. Computational Complexity of Mathematical Operations.

[30] Math Stack Exchange (2014) What Is the Time Complexity of Euclid's Algorithm? http://math.stackexchange.com/questions/258596/what-is-the-time-complexity-of-euclids-algorithm-upper-bound-lower-bound-and-a

[31] National Institute of Standards and Technology (2002) Security Requirements for Cryptographic Modules.

[32] Bouncy Castle (2011) The Legion of the Bouncy Castle. http://www.bouncycastle.org/csharp/