*Journal of Information Assurance and Security* 2 (2006) 41–50

# Conversion Functions for Symmetric Key Ciphers

Debra L. Cook and Angelos D. Keromytis

Department of Computer Science
Columbia University, mail code 0401
1214 Amsterdam Avenue
New York, NY 10027
{*dcook,angelos*}*@cs.columbia.edu*

*Abstract:* As a general design criterion, a symmetric key cipher should not be closed under functional composition due to the implications on the security of the cipher. However, there are scenarios in which this property is desirable and can be obtained without reducing the security of a cipher by increasing the computational workload of the cipher. We expand the idea of a symmetric key cipher being closed under functional composition to a more general scenario where there exists a function that converts the ciphertext resulting from encryption under a specific key to the ciphertext corresponding to encryption with another key. We show how to perform such a conversion without exposing the plaintext. We discuss the tradeoff between the computational workload and security, and the relationship between such conversions and proxy cryptography. We conclude with a discussion of some practical applications of our results.

*Keywords*: Symmetric Key Cipher Design, Conversion Function, Proxy Cryptography

## 1 Introduction

We expand the idea of a symmetric key cipher being closed under functional composition to a more general scenario in which there exists a function that converts the ciphertext resulting from encrypting with a specific key to the ciphertext corresponding to encrypting with another key. As a general design criterion, a symmetric key cipher should not be closed under functional composition due to the implications on the security of the cipher. However, there are scenarios in which this property is desirable. Two (sometimes conflicting) goals of such a cipher are to provide an efficient conversion between the encryption of data under two different keys and performing the conversion without exposing the unencrypted data, as occurs when decrypting data with one key then encrypting it with a second key.

Any situation involving multiple pairwise communication between entities with data encrypted using symmetric key ciphers can benefit from a symmetric key cipher that allows for efficient conversions between keys while maintaining the security of the cipher. Applications include virtual private network (VPN) gateways, file distribution systems using encrypted files, email and online chat programs. We show how to construct from any symmetric key cipher a cipher that allows an entity to convert ciphertexts between two keys without exposing the plaintext and how to develop a tradeoff between the computational workload required to perform the conversion and the security of the cipher. We describe practical applications of the results. We also discuss the relationship between such conversions and proxy cryptography.

The motivation for our work arises from a conversion problem in VPNs: Is it possible to define a symmetric key cipher that allows for converting the encryption of plaintext $P$, $E_{k1}(P)$, under key $k1$ to the encryption of the plaintext under another key $k2$, $E_{k2(}P)$, with fewer computations than what is required for decrypting with key $k1$ then encrypting with key $k2$? Consider the case of a VPN gateway transmitting data between users $A$ and $B$. The gateway shares $k1$ with $A$ and $k2$ with $B$. $A$ and $B$ do not share any key material. With existing symmetric key ciphers, the gateway must perform the conversion by decrypting with $k1$ then encrypting with $k2$. Specifically, $A$ computes $C1 = E_{k1}(P)$ and sends $C1$ to the gateway. The gateway computes $C2 = E_{k2}(D_{k1}(C1))$ and sends $C2$ to $B$ who computes $P = D_{k2}(C2)$. Is there a conversion function $F$ taking a key $kg$ such that

$$\text{(I)} \quad F_{kg}(E_{k1}(P)) = E_{k2}(P) \quad \forall P$$

where $kg$ depends on $k1$ and $k2$, and $F$ requires less work than applying both $E$ and $D$ with some acceptable tradeoffs? In this application, the goal is to decrease the conversion time. The gateway may have sufficient information to obtain $P$ and may or may not expose $P$ during the conversion. In some situations it is desirable for part of $P$ to be obtainable for inspection, such as when a firewall needs to examine the packet. The gateway may also need to modify parts of $P$, such as in application-aware network address translation (NAT). The existence of a function $F$ as shown in (I) has significant implications on the security of the cipher, which we will discuss.

We are also interested in conversions that prohibit the intermediate entity (the gateway in the example) from obtain-

Received December 5, 2005

ing the ciphertext in situations where there is no need for it to have access to the plaintext. This concept is known as proxy cryptography and is a subset of our general conversion concept. We consider proxy cryptography applied to symmetric key ciphers in order for the conversion to be applicable in situations involving larger quantities of data and faster processing than what can be supported with public key ciphers. Okamoto and Mambo introduced the notion of proxy cryptography [10]. This was further explored by Blaze, *et al.* in [2]. Prior work on proxy cryptography has almost exclusively been focused on public key ciphers. When considering encryption with public key ciphers, proxy cryptography allows for a public key to be used by a proxy to convert ciphertext received from one party into ciphertext that can be decrypted with another party's private key without the proxy being able to decrypt the data.

The contributions of our work consist of the following analysis regarding conversions. First, we extend the results in [9] concerning the security of a symmetric key cipher that is closed under functional composition to the more general scenario of a symmetric key cipher for which there exists a conversion function. Second, we introduce the concepts of *conversion crypto-systems* to refer to a symmetric key cipher for which a function exists that performs the conversion of $E_{k1}(P)$ to $E_{k2}(P)$ and *secure conversion crypto-system* to refer to a symmetric key cipher with a conversion function which allows converting between encryptions under different keys without exposing the plaintext. We define two classes of secure conversion crypto-systems, one in which the entity performing the conversion may have sufficient information to obtain the plaintext even though the conversion does not expose the plaintext, and one which is a proxy function in that the entity performing the conversion cannot obtain the plaintext. Third, we show how to construct secure conversion crypto-systems from any existing symmetric key cipher. We show that our conversion crypto-system constructions are optimal in terms of the order of computational work compared to the security of the underlying cipher utilized in the construction, and discuss tradeoffs between the workload and security in terms of the underlying cipher. Finally, we discuss possible applications of conversion and secure conversion crypto-systems.

**Paper Organization:** In Section 2 we define our notation and introduce the terms for conversion crypto-systems. In Section 3 we provide background information on proxy cryptography. In Section 4 we review the attacks from [9] on symmetric key ciphers which are closed under functional composition. In Section 5 we generalize the attacks from [9] to symmetric key ciphers for which conversion functions exist. In Sections 6 and 7 we present constructions and applications of conversion crypto-systems. Section 8 concludes the paper.

## 2 Conversion Definitions

In this section, we introduce and formally define the terms conversion function, secure conversion function, conversion crypto-system and secure conversion crypto-system for symmetric key ciphers. The following notation is used in our definitions.

- $P$ denotes plaintext.
- $C$ denotes ciphertext.
- $K$ denotes the key space for a symmetric key cipher.
- $|K|$ denotes the size of the key space, $K$.
- $k, ki$ denote keys. $i$ is any alphanumeric symbol.
- $|k|$ is the length of key $k$ in bits.
- $E, D$ denote the encryption and decryption functions of a symmetric key cipher $S$, respectively. When encryption is applied using a specific key, $k$, and plaintext, $P$, we write $E_k(P)$. When decryption is applied using a specific key, $k$, and ciphertext, $C$, we write $D_k(C)$. When $S$ is a block cipher, the lengths of $P$ and $C$ are the block size.
- $S = (E, D, K)$ is a symmetric key cipher with encryption function $E$, decryption function $D$ and key space $K$. This will be abbreviated as $S$.
- $Z$ is the set of all permutations on $b$ bits.
- $KZ$ refers to the set of keys by which to index $Z$. Viewing $Z$ as an enumerated set, the $i^{th}$ key in $KZ$ refers to the $i^{th}$ permutation in $Z$.
- $G = \{G_{kg}\}$ is a family of permutations on $b$ bits. A specific permutation in $G$ is indicated by $G_{kg}$, where $kg$ is a key used to index into $G$. $KG$ is the set of all $kg$ values. $|KG| = |G|$. $G_{kg}(X)$ is the result of the permutation $G_{kg}$ applied to a $b$ bit value $X$. The inverse of $G_{kg}$ will be written as $G_{kg}^{-1}$. Notice that $G \subseteq Z$ and $|KG| \leq |KZ|$. $|KG|$ can be thought of as an enumeration of the elements from $Z$ that form $G$.
- $F^G$ is a function that performs the permutations in $G$. $F^G$ is used to indicate the conversion function as defined in the definitions below. $F_{kg}^G(X)$ refers to $F^G$ using key $kg$ and operating on input $X$. The inverse of $F^G$ will be written as $F^{G^{-1}}$.

We now define (secure) conversion functions, (secure) conversion crypto-systems and related terms.

**Definition 1: Conversion Function**

Given a symmetric key cipher $S = (E, D, K)$ operating on $b$ bit inputs, the conversion for $S$ is the family of permutations $G = \{G_{kg}\}$ that converts $E_{k1}(P)$ to $E_{k2}(P)$ $\forall$ $P$ for any keys $k1, k2 \in K$. Specifically, $G_{kg}(E_{k1}(P)) = E_{k2}(P)$ $\forall$ $P$ where $kg$ is dependent on $k1$ and $k2$. The conversion function for $S$ is a function $F^G$ which takes key $kg$ and $b$ bit input $X$ and computes $G_{kg}(X)$. There is no restriction on whether the conversion function exposes $P$ during the conversion and on whether the entity performing the conversion has sufficient information to obtain $P$. $F^G$ is invertible because $G_{kg}$ is a permutation and thus is invertible. $(E_{k1}(P)) = F_{kg}^{G^{-1}}(E_{k2}(P))$ $\forall$ $P$.

$|KG| \geq |K|$. If not, then the existence of a $kg$ for every pair of keys $k1, k2 \in K$ would require at least one $kg$ to map a $k1$ to more than one $k2$. If $G$ is unknown, then the number of values to try to determine $KG$ may be more than $|K|$ because $S$ may be defined such that the length of the key is less than number of permutations on a $b$ bit block (which is true of symmetric key ciphers in practice). For example, if no information is known about $G$ other than the fact that $G$ exists, then any value in $KZ$ potentially maps to an element of $KG$. The number of keys to try in order to determine $KG$ is relevant when trying to attack a cipher for which a conversion function exists.

For any symmetric key cipher $S$ operating on $b$ bit inputs, the mapping of $E_{k1}(P)$ to $E_{k2}(P)$ is a permutation on $b$ bits. Therefore, a family of permutations, $G$, which corresponds to the conversion for $S$ exists. This family of permutations can always be created conceptually, if not practically, in the following manner: For every pair of keys, $(k1, k2) \in K$, create a table that contains the mapping of $E_{k1}(P)$ to $E_{k2}(P)$ of every $b$ bit $P$. Call the table $G_{kg}$ where $kg = k1||k2$. $G$ is the set of permutations defined by all the resulting tables. Define $F^G$ to be the function that performs the table lookups. $G$ corresponds to the $|K|^2$ tables. Regardless of the exact permutations in $G$, the representation of $G$ may be simplified at least slightly from the $|K|^2$ tables. For example, when $k1 = k2$, the resulting table corresponds to the identity function and there will be $|K|$ such tables. Also, the table for $G_{k2||k1}$ is the inverse of the table for $G_{k1||k2}$.

**Definition 2: Conversion Crypto-System**
A conversion crypto-system is a pair $(S, F^G)$ where $F^G$ is the conversion function for the symmetric key cipher $S$.

**Definition 3: Secure Conversion Function**
A secure conversion function is a conversion function that does not expose $P$ during the conversion. There is no restriction on whether or not the entity performing the conversion has sufficient information to obtain $P$. $\widehat{F^G}$ will indicate the conversion function $F^G$ is a secure conversion function.

**Definition 4: Secure Conversion Crypto-System**
A secure conversion crypto-system is a pair $(S, \widehat{F^G})$ where $\widehat{F^G}$ is the secure conversion function for the symmetric key cipher $S$.

**Definition 5: Proxy Function**
Given a symmetric key cipher $S = (E, D, K)$, a proxy function is a conversion function that does not expose $P$ during the conversion and does not provide the entity performing the conversion sufficient information to obtain $P$. A proxy function is a special case of a secure conversion function. $\widetilde{F^G}$ will indicate the conversion function $F^G$ is a proxy function.

**Definition 6: Proxy Crypto-System**
A proxy crypto-system is a pair $(S, \widetilde{F^G})$ where $\widetilde{F^G}$ is the proxy function for the symmetric key cipher $S$. Proxy crypto-systems are a subset of secure conversion crypto-systems.

**Definition 7: Conversion Entity and Converter**
Conversion entity and converter are used interchangeably to refer to the entity executing the conversion function $F^G$.

**Definition 8: Proxy**
A proxy is an entity executing $F^G$ in a proxy crypto-system. This is a special class of conversion entities.

**Definition 9: Effective Key Length**
The effective key length, $K_{eff}$, of a symmetric key cipher $S = (E, D, K)$ is a parameter defined in terms of the key length, $|k|$ for $k \in K$, which indicates the amount of work required to successfully attack $S$ compared to that of an exhaustive search over all keys. A cipher for which an exhaustive search of the key space is the best known attack requires $O(2^{|k|})$ work and has an effective key length of $|k|$. For example, if the key length is 128 bits, an exhaustive search over $|K|$ will require trying half of all keys ($2^{127}$ keys) on average. If an attack exists which requires trying $2^{63}$ keys on average, then $K_{eff} = 64$. In this case $S$'s effective key length is equivalent to that of a cipher with a 64 bit key on which an exhaustive search over all keys is the best attack even though $S$ uses 128 bit keys.

The conversion function, $F^G$, corresponding to $G$ can be constructed (trivially) by defining $F^G_{kg}(C)$ to be $E_{k_2}(D_{k_1}(C))$ to convert $E_{k_1}(P)$ to $E_{k_2}(X)$. In this case, $kg = k1||k2$ and $|KG| = 2^{2|k|}$ versus $|K| = 2^{|k|}$. Performing the conversion by decrypting with $k1$ then encrypting with $k2$ exposes the plaintext during the conversion. While both a secure conversion function and a proxy function exists for every $S$ (define all of the tables corresponding to the mapping and have $F^G$ perform table lookups), it should not be feasible to compute secure conversion and proxy functions for a symmetric key cipher used in practice by defining the tables due to the memory and/or computational resources required.

## 3 Proxy Cryptography

The concept of conversion functions in general has not been discussed prior to our definition in [4]. However, two specific cases of conversion functions have been addressed previously. The first case is where the symmetric key cipher's encryption algorithm is the conversion function. The security implications of such as cipher has been analyzed under the concept of a block cipher which is closed under functional composition [9], which we review in Section 4. The second case is where the conversion function is a proxy function. The concept of a proxy function has been addressed under the topic of proxy cryptography, although almost always in the context of public key cryptography. We summarize the previous work on proxy cryptography here.

When discussed in terms of encryption, proxy cryptography refers to the concept of converting plaintext encrypted under one key to the encryption of the same plaintext under a second key without exposing the plaintext during the conversion. The entity which performs the conversion is referred to as the proxy and the function used to perform the conversion is referred to as the proxy function. The proxy does not have sufficient information, (*e.g.*, the appropriate keys) to obtain the plaintext. When applied to public key ciphers, proxy cryptography allows two parties to publish a key that the proxy will use to convert ciphertext received from one party into ciphertext that can be decrypted with the other party's private key without the proxy being able to decrypt the text. The concept of proxy cryptography is also relevant to signature schemes. If a proxy function exists for a signature scheme, an entity $A$ can sign for an entity $B$ by signing the data as itself then sending the signature to a proxy which transforms $A$'s signature into $B$'s signature.

The prior work that exists on proxy cryptography is focused on public key encryption and signature schemes. Okamoto, Usuda and Mambo introduced the notion of proxy signature schemes in [11]. Their work addressed the problem of an entity, $A$, delegating the signing of messages to a proxy without the proxy needing to know the secret component of the key $A$ uses for signing. Okamato and Mambo later expanded the concept to encryption with public key ciphers [10]. In addition to the general concept, they defined proxy schemes for El Gamal [5] and RSA [12]. Proxy cryptography was further explored by Blaze, Bleumer and Strauss in [2] under the term "atomic proxy cryptography". [2] discusses proxy cryptography as it relates to public key encryption, signature

schemes and identification schemes. In [2], Blaze, *et al.* also defined asymmetric and symmetric proxy cryptography. We note that this is not to be confused with proxy encryption using symmetric (secret key) and asymmetric (public key) ciphers. Given entities $A, B$ and the proxy, asymmetric proxy cryptography means the use of the proxy only works in one direction between $A$ and $B$. For example, $A$ can use the proxy to send messages to $B$ without the proxy being able to convert messages from $B$ into a form $A$ can decrypt. Symmetric proxy cryptography means the use of the proxy works in both directions between $A$ and $B$. [1] refers to asymmetric proxy cryptography as uni-directional proxy cryptography and symmetric proxy cryptography as bi-directional proxy cryptography.

The only place in which proxy cryptography for symmetric key ciphers has been discussed, albeit briefly, is in [8]. Ivan and Dodis formally defined in [8] the concepts of symmetric and asymmetric proxy cryptography from [2] in a manner that can be applied to either public or private key ciphers, although they only illustrate the concepts with public key ciphers. [8] does not analyze the notion of proxy cryptography in relation to private key ciphers. For example, it does not explore the applications of symmetric key proxy functions, does not discuss the workload and does not explore the relationships between the keys, workload and security of a symmetric key cipher with a proxy function, all of which we discuss within this paper for conversion functions. No prior work has considered the implications of a symmetric key cipher which has been defined in a manner that incorporates proxy cryptography, or more generally, conversions.

In [1], the use of proxy cryptography with public key ciphers was applied to the encryption and sharing of keys used to encrypt files in a file system. An entity, $A$, creates a file and encrypts it with a symmetric key cipher using secret key, $k_a$. $k_a$ is then encrypted using a public key cipher proxy encryption scheme. Let $k_{ap}$ be the key $A$ uses for the public key cipher and $X$ be the encryption of $k_a$ using $k_{ap}$. $A$ gives $X$ and a list of users who can access the file to an access control server, which functions as the proxy to convey $k_a$ to the users via the proxy encryption scheme. The access control server does not have sufficient information to decrypt $X$ and obtain $k_a$, but can only convert $X$ into data that an authorized user, $B$, on the list can decrypt to obtain $k_a$. Therefore, the access control server cannot obtain the contents of the files.

Proxy cryptography for public key ciphers has also been called "re-encryption" since it involves the proxy "re-encrypting" ciphertext received from the sending entity to create a ciphertext that the receiving entity can decrypt. [6] proposes efficient constructions of universal re-encryption schemes using El-Gamal.

# 4 Security Implications of a Symmetric Key Cipher that is Closed Under Functional Composition

## 4.1 Overview

Before describing conversion crypto-systems, we review why closure under functional composition is an undesirable property for symmetric key ciphers. A symmetric key cipher, $S = (E, D, K)$, that is closed under functional composition is an example of a conversion crypto-system where the conversion function is $S$'s encryption function, $E$, using key space $K$. Specifically, if $S = (E, D, K)$ is closed under functional composition, then $\forall\ k1, k2 \in K$, $\exists\ k3 \in K$ such that:

$$\text{(II)} \quad E_{k3}(E_{k1}(P)) = E_{k2}(P) \quad \forall\ P$$

Within the context of determining whether not DES [7] is a group, Kaliski, *et al.* proved in [9] that any symmetric key cipher with key length $|k|$ that is closed under functional composition is vulnerable to a known plaintext attack requiring $O(2^{|k|/2})$ work as opposed to $O(2^{|k|})$ work required for an exhaustive key search. We will write $O(2^{|k|/2})$ as $O(|K|^{\frac{1}{2}})$, where $K$ is the keyspace. Two methods of known plaintext attacks were described in [9], with a tradeoff between the memory and the time required to decrypt additional ciphertexts. We provide a brief summary of these attacks. They do not provide the actual secret key, but instead provide a series of keys which, when encrypting or decrypting with the keys in order, produce the same results as the secret key.

## 4.2 First Attack - Variation of Birthday Paradox

The first attack described in [9] produces a pair of keys, $(k1, k3)$, which can be used in place of $k2$ as indicated by (II). Let $S = (E, D, K)$ be closed under functional composition. Choose two sets of $r$ keys $KA = \{k_{a1}, k_{a2}, ...k_{ar}\}$ and $KB = \{k_{b1}, k_{b2}, ...k_{br}\}$ from $K$. For all pairs $(k_{ai}, k_{bj})$, $0 \leq i, j \leq r$, determine if (II) holds via a meet in the middle attack. Let $C = E_{k2}(P)$. Compute $E_{k_{ai}}(P)\ \forall\ k_{ai}$ and $D_{k_{bj}}(C)\ \forall\ k_{bj}$ and search for matches. Set $k1$ to the $k_{ai}$ and $k3$ to the $k_{bj}$ that produce the match. Test with additional plaintexts to ensure the match does not hold for only a specific $P$. This attack will produce a pair of keys that are equivalent to the single key $k2$ as opposed to finding $k2$. The key pair can be used to decrypt additional ciphertexts that have been encrypted with $k2$. Obviously if either $E_{k_{ai}}(P) = C$ or $D_{k_{bj}}(C) = P$ is found during the search, then $k2$ has been found. A match will be found in $O(|K|^{\frac{1}{2}})$ time and memory when using $r = O(|K|^{\frac{1}{2}})$ keys in $KA$ and $KB$ if $S$ is closed under functional composition.

The result derives from a meet-in-the-middle variation of the Birthday Paradox using two samples $X$ and $Y$. If $X$ and $Y$ are of size $r$, and are drawn at random from $|K|$ elements with each element drawn independently with probability $\frac{1}{|K|}$, then there are $\binom{|K|}{r}$ ways to select $X$ and $\binom{|K|-r}{r}$ ways to select $Y$ such that $X \bigcap Y = \emptyset$ and $\binom{|K|}{r}^2$ ways to select $X$ and $Y$. The chance that $X$ and $Y$ do not intersect is:

$$\text{(III)}\ Pr(X \bigcap Y = \emptyset) = \frac{[(|K|)(|K|-1)...(|K|-2r+1)]}{[((|K|)(|K|-1)...(|K|-r+1))^2]}$$

If $r = \alpha(|K|^{\frac{1}{2}})$ for some constant $\alpha > 0$, then $Pr(X \bigcap Y = \emptyset) \approx e^{-3\alpha^2}$ for sufficiently large $|K|$. The cipher $S$ is transformed into this variation of the Birthday Paradox by using $KA$ and $KB$ as the two samples and defining intersection to mean there is a $k1 \in KA$ and a $k3 \in KB$ such that $E_{k3}(E_{k1}(P)) = E_{k2}(P)\ \forall\ P$. The probability of finding a $(k1, k3)$ is approximately $1 - e^{-3\alpha^2}$ and approaches 1 as $\alpha$ increases. The attack requires $O(|K|^{\frac{1}{2}})$ time and memory.

### 4.3 Second Attack - Cycling Attack

The second attack in [9] is referred to as a cycling attack. While it requires less memory than the first attack, it produces a series of keys that require $O(|K|^{\frac{1}{2}})$ time for decryption in contrast to the two keys produced by the first attack. This attack corresponds to taking a pseudorandom walk in the message space covered by the symmetric key cipher, with each step corresponding to encrypting (or decrypting) with another key. Encountering a cycle after a short number of steps is an indication that the cipher is likely to be closed under functional composition. Taking a large number of steps without encountering a cycle is an indication that the cipher is not closed under functional composition. For a cipher which is closed under functional composition, it is expected that a cycle will be encountered in $|K|^{\frac{1}{2}}$ steps.

Again let $S = (E, D, K)$ be closed under functional composition. Given the plaintext, ciphertext pair $(P, C)$ where $C = E_{k2}(P)$), the idea is to obtain some series of encryptions and decryptions that started with $P$ and $C$, respectively, and intersect. Starting with $L = P$ on the left and $R = C$ on the right, randomly pick a key, $k$ from $K$ and either encrypt the left side or decrypt the right side. Repeat, each time testing if the left and right sides are equal. We use $k \leftarrow K$ to indicate $k$ is randomly selected from $K$. This attack is summarized as follows:

Set:
$i = 0$;
$j = 0$;
$L = P$;
$R = C$;
while $(L \neq R)$ do {
$k \leftarrow K$;
if encrypting $L$\{
$k_{ai} \leftarrow k$;
$L \leftarrow E_{kai}(L)$;
$i = i + 1$;
}
else if decrypting $R$\{
$k_{bj} \leftarrow k$;
$R \leftarrow D_{kbj}(R)$;
$j = j + 1$;
}
}

A series of encryptions and decryption will be produced such that $E_{k_{ai}}(E_{k_{ai-1}}...(E_{k_{a2}}(E_{k_{a1}}(P)))) = D_{k_{bj}}(D_{k_{bj-1}}...(D_{k_{b2}}(D_{k_{b1}}(C))))$ for randomly chosen $k_{ai}, k_{bj} \in K$. The result can be verified by testing with a few additional plaintexts. An average of $|K|^{\frac{1}{2}}$ steps are needed before the two sides match when $S$ is closed under functional composition. Like the first attack, $k2$ is not found. However, the equivalent key for $k2$ that is produced this time is the series of $k_{ai}$'s and $k_{bj}$'s. To decrypt additional ciphertexts encrypted with $k2$, $D_{k_{a1}}(D_{k_{a2}}...(D_{k_{ai-1}}(D_{k_{ai}}(D_{k_{bj}}(D_{k_{bj-1}}$ ... $(D_{k_{b2}}(D_{k_{b1}}(C)))))))$ must be computed.

If the sequence of keys are saved, $O(|k| * |K|^{\frac{1}{2}})$ space is needed to store the keys. Only the current value of $L$ and $R$ need to be saved as opposed to all of intermediate results of the encryptions and decryptions. Time and space tradeoffs allow for the attack to occur in $O(1/w)$ space and $O(|K|^{(1+w)/2})$ time where $0 < w < 1$ [3].

## 5 Security Implications of a Symmetric Key Cipher with a Conversion Function

### 5.1 Overview

By definition, every symmetric key cipher, $S = (E, D, K)$, has a conversion function, $F^G$, as we mentioned in Section 2. Define $F^G_{kg}(X)$ to be the decryption of $X$ with key $k1$ followed by the encryption with key $k2$ when converting from $E_{k1}(P)$ to $E_{k2}(P)$ and $kg = k1||k2$. If this is the most efficient $F^G$ then the effective key length of $S$ is $|k|$, for $k \in K$ (assuming other types of attacks do not exist on $S$). Here the conversion function will not assist in any attack attempting to find the keys and/or recover plaintexts. We are concerned with the implications when a conversion function exists for $S$ that is more efficient to compute then encrypting and decrypting.

In the remainder of this section, we describe how the method used in the first attack from Section 4 can be used with a conversion function to attack a cipher and how the work required in the attack is related to the work of the conversion function. We then define how the probability of such an attack is related to the potential set of keys for the conversion function. In all cases, we deal with conversion functions in general, independent of whether or not the conversion function is also a secure conversion function or a proxy function.

### 5.2 Application of Birthday Paradox to Symmetric Key Ciphers with a Conversion Function

We now generalize the first attack from [9] to an attack on a symmetric key cipher for which a conversion function exists. Thus we prove why it is undesirable for a symmetric key cipher to be a conversion crypto-system if the computational and memory resources required of the conversion function are less than those required of decrypting then encrypting to perform the conversion. A symmetric key cipher which is closed under functional composition (which [9] addressed) is a special case of a symmetric key cipher for which a conversion function exists, specifically the case where the conversion function is the encryption function. In the following, we use the term "work" to refer to the computational and memory resources required.

**Lemma I:** *For a symmetric key cipher $S$ with keyspace $K$ and encryption function $E$, if there exists a function $F^G$ taking parameter $kg \in KG$, $KG$ is known and $|KG| = |K|$ such that $\forall k1, k2 \in K$, $\exists$ a $kg$ for which $F^G_{kg}(E_{k1}(P)) = E_{k2}(P)$ $\forall P$ and the work of $F^G_{kg}$ is $O$(work of $E$) then there exists a $O(|K|^{\frac{1}{2}})$ known plaintext attack on $S$.*

*Proof:* The cipher $S$ is transformed as in Section 4 into the variation of the Birthday Paradox by using $KA$ and $KB$ as the two samples and defining intersection to mean there is a $k_{ai}$ in $KA$ and a $k_{bj}$ in $KB$ such that $F^G_{k_{bj}}(E_{k_{ai}}(P)) = E_{k2}(P)$. The set $KB$ is selected from $KG$ instead of from $K$ as it was in the first attack in Section 4. Since $|KG| = |K|$, Equation (III) from Section 4 holds with $X = KA$ and $Y = KB$.

Lemma I implies that for any symmetric key cipher, $S = (E, D, K)$, with a conversion function, $F^G$, taking a key, $kg$, of the same length as the key length, $|k|$, of $S$, the effective key length of $S$ is equal to $\frac{1}{2}|k|$ when the work of $F^G$ is

equivalent to $E$. In this case, to obtain security comparable to an exhaustive search over all keys of length $|k|$, namely $O(2^{|k|})$ work, the key length of $S$ must be doubled to $2|k|$. Let $K'$ be the set of the longer keys such that for $k' \in K'$, $|k'| = 2|k|$. Then $|K'|^{\frac{1}{2}} = (2^{2|k|})^{\frac{1}{2}} = 2^{|k|} = |K|$ and the attack is $O(|K'|^{\frac{1}{2}}) = O(|K|)$. We point out that when the most efficient conversion function consists of decrypting with $k1$ and encrypting with $k2$, then $|kg| = 2|k|$ and the effective key length of $S$ is $|k|$.

It is possible for the effective key length of $S$ to be less than $\frac{1}{2}|k|$ if $F^G$ provides some speedup aside from the square root reduction in the number of keys to try to the extent that a brute force attack is $O(2^{(\frac{1}{2}|k|-w)})$ as opposed to $O(2^{(\frac{1}{2}|k|)})$ for some $w > 0$. As long as the work in determining the correct $kg$ and applying $F^G_{kg}$ is greater than or equal to the total work required in determining $k1$ and $k2$ then decrypting and encrypting, the effective key length of $S$ is $|k|$.

### 5.3 Attack Probability in Relation to Key Space

We now consider how the probability of finding the pair $(k1, kg)$ when $F^G$ is not known for $S = (E, D, K)$ ($F^G$ is not known aside from creating a complete mapping of all $E_{k1}$ to $E_{k2}$ $\forall k \in K$). Let $U$ be the potential set of permutations in $G$ and $KU$ be the corresponding set of keys (indices into $U$). $G \subseteq U$, $KG \subseteq KU$, $U \subseteq Z$ and $KU \subseteq KZ$, where $Z$ is the set of all permutations on $b$ bits.

We again choose two sets of keys, this time with one set created from $K$ and one set created from $KU$. Let $KA$ be a set of $r$ keys chosen from $K$ and let $KB$ be a set of $r$ keys chosen from $KU$. When $|KU| \geq |K|$, for any element in $KB$, there may or may not be an element in $K$ that we can combine with it to obtain a key equivalent to $k2$. Without loss of generality, choose $KA$ first. There are $r$ elements in $KU$ that can create a match with some element of $K$ and $\binom{|KU|-r}{r}$ ways to select $r$ elements from $KU$ such that no match is formed. Let $Pr[k1, k3]$ be the probability of finding a $k1$ from $KA$ and a $k3$ from $KB$, and let $Pr[\overline{(k1, k3)}]$ denote the probability of not finding such a pair.

(IV) $Pr[k1, k3] = 1 - Pr[\overline{(k_1, k_3)}]$

$$Pr[\overline{(k_1, k_3)}] =$$

$$\frac{\binom{|K|}{r}\binom{|KU|-r}{r}}{\binom{|KU|}{r}\binom{|K|}{r}}$$

$$\geq \frac{\binom{|K|}{r}\binom{|K|-r}{r}}{\binom{|K|}{r}^2}$$

$$\geq \frac{\binom{|K|-r}{r}}{\binom{|K|}{r}}$$

with equality holding when $|KU| = |K|$, and

$$Pr[(k1, k3)|\ |KU| \geq |K|] =$$

$$1 - Pr[\overline{(k_1, k_3)}|\ |KU| \geq |K|]$$

$$\leq 1 - Pr[\overline{(k_1, k_3)}|\ |KU| = |K|]$$

As the expected number of keys to try before finding a match increases from the $O(|K|^{\frac{1}{2}})$ obtained when $|KU| = |KG| = |K|$ (*e.g.* as the size of $KU$ increases), the probability of success decreases.

## 6 Conversion Crypto-Systems for Symmetric Key Ciphers

### 6.1 Secure Conversion Constructions

In this section, we define a general construction for symmetric key secure conversion crypto-systems. First, we consider all symmetric key ciphers and provide two variations for defining the keys in the general construction. The variations differ in the workload required of each entity and in which entities share key material. Second, we provide a variation that is restricted to stream ciphers and offers advantages over the general construction. Recall that a conversion function can be constructed for any symmetric key cipher, $S = (E, D, K)$, by defining:

(V) $F^G_{kg}(C) = E_{k2}(D_{k1}(C))\ \forall k1, k1 \in K$

with $kg = k1||k2$

to convert $E_{k1}(P)$ to $E_{k2}(P)$. This is not a secure system under our definition due to the fact that the plaintext is exposed during the conversion at the end of the decryption step.

Given a symmetric key cipher $S = (E, D, K)$, we define the following key format and function for use in our constructions:

- Let $kg = (k1, k2, flag_1, flag_2)$ for keys $k1, k2 \in K$ with $|k1| = |k2|$ and single bit values $flag_1, flag_2$. The $flag_i$ and $ki$ values will be used to denote whether to encrypt, decrypt or do nothing. A $flag_i$ value of 0 indicates to encrypt with key $ki$ and a value of 1 indicates to decrypt with key $ki$. If $ki$ is null, do nothing.
- Let $H = H_k(X)$ denote $H(E, D, X, kg)$ and be defined as applying $E$ or $D$, as indicated by $kg$, to $X$. From $kg$, $k1$ and $flag_1$ are used for the first application of $E$ or $D$, and $k2$ and $flag_2$ are used for the second application of $E$ or $D$. For example, given a plaintext $P$ and key $kg = (k1, k2, 0, 1)$, the conversion function will perform $D_{k2}(E_{k1}(P))$. $H$ itself can be considered to be a symmetric key cipher that uses $S$ as a building block.

Using the above key format, we define two general methods for creating a secure conversion crypto-system for any symmetric key cipher. In both cases the conversion function satisfies our definition of a proxy function. We also define a method for creating a secure conversion crypto-system that is specific to stream ciphers. We assume $S$ is secure in the sense that it has an effective key length of $|k|$ for $k \in K$ so, independent of the conversion functions we define, there is no practical attack on $S$.

### 6.2 General Methods

The following describes two ways of defining the keys using the 4-item tuple format and operations required of the sender, receiver and converter to create a secure conversion-crypto system.

**First Method:**

The first method requires each pair of the three entities to share some key material. We note that independent of our initial paper on conversion functions [4], [8] defined this method in a general scheme independent of the type of cipher (public key or private key) while illustrating it with a public key cipher, El Gamal. [8] neither explored the applications of the method when using symmetric key ciphers nor the implications, such as the workload required of the entities and the tradeoffs between workload, key length and security.

We define the method as the following sets of keys and computations for the sending entity, receiving entity and converter:

(VI) **Method 1:**

*Keys:*

- $kA = (k_{ab}, k_a, 0, 0)$
- $kg = (k_a, k_b, 1, 0)$
- $kB = (k_b, k_{ab}, 1, 1)$

*Computations:*

- Sending Entity: Computes $C1 = H_{kA}(P)$ on plaintext $P$ and sends $C1$ to the converter.
- Converter: Computes $C2 = H_{kg}(C1)$ and sends $C2$ to the receiving entity.
- Receiving Entity: Computes $H_{kB}(C2)$ to obtain $P$.

Let $A$ be an entity encrypting data using a symmetric key cipher $S = (E, D, K)$ to send data to entity $B$. To send plaintext $P$ from $A$ to $B$ via a converter, $A$ computes $C1 = H_{kA}(P)$, which is the equivalent of the double encryption $E_{k_a}(E_{k_{ab}}(P))$, and sends $C1$ to the converter. The converter computes $C2 = H_{kg}(C1)$, which is the equivalent of $E_{k_b}(D_{k_a}(C1))$, and sends $C2$ to $B$. To obtain $P$, $B$ computes $H_{kB}(C2)$, which is the equivalent of the double decryption $D_{k_{ab}}(D_{k_b}(C2))$.

Let $F^G_{kg}(X) = H_{kg}(X)$. When $H$ is used in this manner, $(H, \widetilde{F^G})$ is a secure conversion crypto-system that is a proxy crypto-system. $F^G_{kg}(X)$ is a secure conversion function because the converter does not expose $P$ during the conversion. $F^G_{kg}(X)$ is also a proxy function because the converter does not have sufficient information by which to obtain $P$. The disadvantages of this method are that each pair of entities (A and B, A and the converter, and B and the converter) must share partial key material and each entity incurs two applications of the cipher. While having the converter decrypt and encrypt is no worse in terms of workload then what is required in the conversion defined in (V) and provides the benefit of being a proxy function, the receiving and sending entities must also incur two applications of the cipher compared to one application each when using the conversion in (V).

There are a couple options for how key material can be used in this method when communicating when either endpoint, $A$ or $B$, communicates with other entities. The key shared by an endpoint and the converter can be used for communication between the endpoint and multiple entities. For example, $A$ and the converter can use $k_a$ as their shared key when $A$ sends data to another entity, $B'$, through the converter. If $k_a$ is not public and $B$ intercepts the message to $B'$, $B$ cannot perform even one layer of the decryption. If $k_a$ is public, $B$ can perform the same decryption layer as the converter, but cannot obtain the plaintext unless it has the key, $k_{b'}$, belonging to $B'$ that is needed for the second layer of decryption. Likewise, $B$ and the converter can use $k_b$ when $B$ sends messages to entities other than $A$ through the converter.

If the keys shared between the endpoints and the converter are not public or used in multiple pairs, the key, $k_{ab}$, that shared between the endpoints can be used for communicating with multiple entities. For example, $A$ can use $k_{ab}$ with $B'$ if $B'$ uses a key $k_{b'} \neq k_b$ with the converter if $B'$ cannot obtain $k_b$ and if $B$ cannot obtain $k_{b'}$. If either $B'$ or $B$ intercept messages between the converter and the other that originated from $A$, neither can reverse the encryption performed by the converter. This "public" use of $k_{ab}$ has a downside in that if it is possible for an adversary to access memory within the converter as $F^G_{kg}(C1)$ is being computed and obtain $D_{k_a}(C1)$, then the adversary can compute $D_{k_{ab}}(D_{k_a}(C1))$ and obtain $P$.

**Second Method:**

The second method uses onion routing [13]. This method does not require any shared key material between the converter and $B$ in order for $B$ to receive messages from $A$.

(VII) **Method 2:**

*Keys:*

- $kA = (k_{ab}, k_a, 0, 0)$
- $kg = (k_a, null, 1, 0)$
- $kB = (k_{ab}, null, 1, 0)$

*Computations:*

- Sending Entity: Computes $C1 = H_{kA}(P)$ on plaintext $P$ and sends $C1$ to the converter.
- Converter: Computes $C2 = H_{kg}(C1)$ and sends $C2$ to the receiving entity.
- Receiving Entity: Computes $H_{kB}(C2)$ to obtain $P$.

Let $A$ and $B$ be defined as before. $A$ performs $H_{kA}(P)$ which is two encryptions to compute $C1 = E_{k_a}(E_{k_{ab}}(P))$. The converter performs $H_{kg}(C1)$ to produce $C2 = D_{k_a}(C1)$ and $B$ performs $H_{kB}(C2)$ to compute $P = D_{k_{ab}}(C2)$. Again we set $F^G_{kg}(X) = H_{kg}(X)$ and $(H, \widetilde{F^G})$ is a secure conversion crypto-system that is a proxy crypto-system.

In this method, only the sending entity incurs two applications of the cipher. The converter incurs one application of the cipher versus the two applications the converter performs in (V). Across the three entities, the total workload is equivalent to the workload of (V). In contrast to the first method, every pair of entities do not share key material. $A$ and the converter must share $k_a$. $A$ and $B$ must share $k_{ab}$.

Even though the first method imposes twice the workload on each entity compared to a single application of a cipher; whereas, the second method only increases the workload of one entity, the first method offers a potential advantage in how the key material is shared. While in both cases $A$ and $B$ must share some key material, in the first method no entity has the entire key of any other entity. In the second method, $A$ has the entire key used by each of the other two entities. $B$ must share some key material with $A$ in both methods. The second method does not require shared key material between $B$ and the converter for $B$ to receive messages, but they must share key material if $B$ is also sending messages.

The second method provides an advantage over both method 1 and the basic decrypt-encrypt approach in how the work is distributed across the three entities. The converter is likely to have a greater workload than either $A$ or $B$ because the converter has to process all of the pairwise sessions between the entities it serves; whereas, neither $A$ nor $B$ will likely be involved in every session. By decreasing the converter's work to one application of $S$, the overall number of simultaneous sessions supported by the converter will

likely increase in comparison to when the converter must perform two applications of $S$. In most applications, increasing $A$'s workload from the one encryption incurred in the basic decrypt-encrypt approach to two encryptions will have less of an impact on the number of simultaneous sessions supported than when the converter must perform an encryption and decryption.

## 6.3 Alternate Interpretation of Method 1

An alternative way of viewing the first method requires a symmetric key cipher, $S$, that has is structured as a series of rounds. This is typical of block ciphers used in practice. Let $r$ be the number of rounds in $S$. In the first method, let $E^{r1,r2}_{k1,k2}$ denote running $r1$ rounds of encryption using key $k1$ followed by $r2$ rounds of encryption using key $k2$. Let the keys be defined as before in Method 1. Each entity will compute two key expansions, one for each part of its key, use the first key expansion for a specified number of rounds and use the second key expansion for the remaining number of rounds. For example, $A$ will run $r1$ rounds of encryption with the expanded $k_{ab}$ and $r2 = r - r1$ rounds of encryption with the expanded $k_a$. The converter will run $r2$ rounds of decryption using the expanded $k_a$ and $r2$ rounds of encryption using the expanded $k_b$. $B$ will decrypt by running $r2$ rounds with the expanded $k_b$ and $r1$ rounds with the expanded $k_{ab}$. The converter requires a total of $2 * (r2)$ rounds. Setting $r1 = r2$ results in each entity running $r$ rounds. Specifically,

(VIII) **Alternative Version of Method 1:**
*Keys:*

- $kA = (k_{ab}, k_a, 0, 0)$
- $kg = (k_a, k_b, 1, 0)$
- $kB = (k_b, k_{ab}, 1, 1)$

*Computations:*

- $A$ computes $C1 = E^{r1,r2}_{k_{ab},k_a}(P)$.
- The converter computes $C2 = E^{r2}_{k_b}(D^{r2}_{k_a}(C1))$.
- $B$ computes $P = D^{r2,r1}_{k_b,k_{ab}}(C2)$.

This method is equivalent to using two applications of a reduced round version of the underlying cipher, $S$, thus the number of rounds chosen for each step must be large enough avoid making the reduced round version susceptible to other attacks. A tradeoff can be established between the workload and security by adjusting the number of rounds. To obtain an effective key length equal to $|k|$, the effective key length of the cipher $S$, set $r1 = r2 = r$. This results in the method being equivalent to Method 1. If the workload of each entity is set to that of $S$ ($r1 = r2 = r/2$ so the work is equivalent to a single encryption or decryption of $r$ rounds), Lemma I applies and the effective key length is $\frac{1}{2}|k|$.

We note that if $r1 = r2 = r/2$ and $S$'s key schedule creates keys corresponding to the following:

- $k1$ : The round keys for the first $r/2$ rounds are the same as in $k_{ab}$ and the round keys for the second $r/2$ rounds are the same as in $k_a$.
- $k2$: The round keys for the first $r/2$ rounds are the same as in $k_b$ and the round keys for the second $r/2$ rounds are the same as in $k_{ab}$.
- $k3$ : The round keys for the first $r/2$ rounds produce the same result as when decrypting $r/2$ rounds with $k_a$ and the round keys for the second $r/2$ rounds produce the same result as when encrypting with $k_b$.

Then $E_{k3}(E_{k1}(P)) = E_{k2}(P)$ and $S$ is closed under functional composition.

## 6.4 Stream Ciphers

We now define a third method that is a construction of a secure conversion crypto-system that is specific to stream ciphers. Let $S = (E, D, K)$ be a stream cipher (or a block cipher run in a stream cipher mode, for example, OFB or CTR) and $C = KS \oplus P$, where $KS$ is the key stream. Then, unlike the first two methods, it is not necessary for $A$ and $B$ to share any key material.

(IX) **Method 3:**
*Keys:*

- $kA = (k_a, null, 0, 0)$
- $kg = (k_b, k_a, 1, 1)$
- $kB = (k_b, null, 1, 0)$

*Computations:*

- Sending Entity: Computes $C1 = H_{kA}(P)$ on plaintext $P$ and sends $C1$ to the converter.
- Converter: Computes $C2 = H_{kg}(C1)$ and sends $C2$ to the receiving entity.
- Receiving Entity: Computes $H_{kB}(C2)$ to obtain $P$.

Let $KS_a$ and $KS_b$ denote the key streams produced by $E$ when using $k_a$ and $k_b$ respectively. $A$ computes $C1 = H_{kA}(P)$, which is the equivalent of $KS_a \oplus P$. The converter computes $C2 = H_{kg}(C1)$, which is the equivalent of $KS_a \oplus KS_b \oplus C1$. $B$ computes $H_{kB}(P)$, which is the equivalent to $KS_b \oplus C2$. The converter can compute $KS_a \oplus KS_b \oplus C1$ without exposing the plaintext by either computing $KS_b \oplus C1$ or $KS_a \oplus KS_b$ first.

There are several advantages to this construction. First, $A$ and $B$ do not share any key material. Second, $P$ is not revealed during the conversion. However, the converter does have the information needed to obtain $P$ and therefore can be used in applications where it needs to inspect $P$ (such as a firewall) and/or modify $P$ (such as application aware NAT). To obtain $P$, the converter performs the conversion by computing $KS_b \oplus (KS_a \oplus C)$, in which case this is the same as the basic method of decrypting then encrypting stated in (V). Third, it is not necessary that $A$ and $B$ incur any overhead related to utilizing $H$ in place of the underlying stream cipher. $H$ may be run only by the converter. $A$ and $B$ can just apply the stream cipher directly.

Let $F^G_{kg}(X) = H_{kg}(X)$, then $(H, \widehat{F^G})$ is a secure conversion crypto-system when the conversion is performed by as stated and not by computing $KS_a \oplus C$ then XORing the result with $KS_b$. The $F^G$ does not satisfy the definition of a proxy function because the converter has enough information to obtain the plaintext even though it does not do so by definition of the crypto-system. The converter is a proxy if it can only generate the combined keystream without knowing $kA$ and/or $kB$ and without knowing either of the individual keystreams, $KS_a$ or $KS_b$. This may be possible if, through the use of

an external device, such as a smart card or a secure crypto processor, which given $kA$ and $kB$ produces the keystream corresponding to $kg$, then it is possible to perform the conversion without the converter having sufficient information to obtain $P$. Another option is a hardware implementation that runs two instances of the key stream generator and XORs their outputs then makes the resulting key stream available for XORing with the data. The keys may be configurable in hardware and not accessible by any software applications on the converter.

### 6.5 Effective Key Length

When designing a conversion function we do not want to reduce the effective key length of the cipher. In our two general methods and the method stream cipher specific method, the effective key length is that of the cipher $S$. The work required of the converter in methods 1 and 3 is equal to that of encrypting and decrypting with the cipher, $S = (E, D, K)$. Assuming the work required when decrypting is equal to the work required when encryption, the workload of the converter is twice that of encryption. The length of the key, $kg$, used in the conversion is also twice that of $S$'s key length. The two flag bits in $kg$ can be considered to be known as part of the algorithm and do not count as key bits which an attacker will need to determine. Even though we formally define $H$ as the cipher in the crypto-system, we want the effective key length to be that of the underlying cipher $S$ since our purpose of defining the crypto-system is to provide a mechanism for a secure conversion (and proxy function) for $S$. By Lemma I, for the crypto-systems defined in the first and third methods, there exists an attack which is $\approx 2(|K|^2)^{\frac{1}{2}} = 2|K|$ and thus $K_{eff} = |k|$. Therefore, both methods 1 and 3 are the best possible in terms of security versus key length.

In the second method, the converter will need a key of length $2|k|$ to obtain the plaintext because of the double encryption by $A$. By the definition of method 2, we are using onion routing and not providing a function which decreases the amount of work needed to convert between keys $k1$ and $k2$. The actual computations that allow the conversion in method 2 can be viewed as being split between $A$ and the converter, with each performing one of the two applications of $S$ needed for the conversion. Thus the workload is no different than if the conversion was performed by decrypting with $k1$ then encrypting with $k2$.

## 7 Applications

In existing applications that convert ciphertext via the basic mode of decrypting then encrypting, whether or not the plaintext is accessible temporarily during the conversion (in part due to intermediate results being written to memory) depends on the application and implementation. With a secure conversion crypto-system, it is not a concern if intermediate results are written to temporary files or insecure memory during the conversion because it is still encrypted under one key. Applications where a conversion that is faster than decrypting and encrypting would be valuable include VPN gateways and cases where an entity needs to distribute data or communicate with multiple users without establishing pairwise keys or sharing the same key with multiple entities. Examples of this latter case include file distribution systems, email and online chat.

In cases where the converter is trusted or requires access to the plaintext, a basic conversion crypto-system in which the converter can access the plaintext is beneficial over performing the conversion by decrypting then encrypting only if the conversion function is more computationally efficient then decrypting and encrypting. For the converter to be trusted, in must be ensured that the converter does not permit access to the plaintext by any process or for any reason other than that required by the conversion process. For example, there must not be a threat of malware on the converter accessing memory to which the plaintext is written as an intermediate result during the conversion. However, as we showed, any gains in efficiency that a conversion crypto-system offers over decrypting then encrypting imply a decrease in security. Even without efficiency gains, a conversion crypto-system is useful if the application involves a converter which must serve as a secure converter or proxy in some cases while requiring access to the plaintext in other cases. This can be accomplished by defining the entities according to one of the methods which provides a secure conversion or proxy and providing the full key material used by the sending entity to the converter when needed to allow the converter access to the plaintext. Scenarios where a converter may need to inspect packets include gateways providing firewall and/or application-aware NAT functionality. In the case of a firewall, packet inspection is needed and the plaintext may be altered if malicious content must be removed. In application-aware NAT, IP addresses embedded in the application's data are replaced. For example, in VoIP services, NAT may be performed by the service provider on the IP addresses of the caller and called party contained within the VoIP protocol (such as SIP).

A scenario where the converter does not need to inspect packets is a file system in which the files are encrypted under one key and sent to a requesting user encrypted with the user's key. In such an application, a proxy function is useful even if it incurs overhead not present in the basic decrypting then encrypting approach because it ensures the file contents are not obtained by unauthorized users from the converter. Given that the converter may be using the same resources as the users who store and access the files, the chances of malicious activity can be higher than for a converter executing on a system to which no users have access, thus increasing the need for a proxy function.

The main disadvantage of our two general methods in Section 6 is that the sending entity, $A$, and receiving entity, $B$, must share key material, which results in implementation issues. If the converter needs to inspect some of the packets (thus a basic conversion crypto-system is needed instead of a secure conversion crypto-system), either the converter must establish the key material shared between $A$ and $B$, or $A$ and $B$ must establish shared key material on their own and then send the necessary key material to the converter. If the converter must be secure or act as a proxy, then if $A$ and $B$ can establish a shared key, it can be argued there is no need for a conversion entity unless it is used in preventing traffic analysis since now the converter does not inspect or alter the data but just converts it to ciphertext which $B$ can decrypt. In such cases it is not advisable for the converter to establish the key material and send it to $A$ and $B$ because then the converter has all of the key material and can obtain the plaintext even though it may not do so if the algorithm is

applied as specified. The methods are useful in cases where the keys are established by an entity external to $A$, $B$ and the converter that never has access to the ciphertext in any form. This allows keys to be established such that no entity has all of the key material. Our method specific to stream ciphers avoids the need for sender and receiver to share key material, instead requiring only that they each share a key with the converter. However, the implementation must be carefully designed if the method is to be used as a proxy function instead of a conversion or secure conversion function.

## 8 Conclusions

We introduced the concept of (secure) conversion functions. We have shown that for any symmetric key cipher $S$ with key space $K$, if there exists a conversion function requiring a key whose length is the same as the length of the elements of $K$ and whose work is $O(S)$ then there exists an $O(|K|^{1/2})$ attack. Our work poses the question of whether or not an actual symmetric key cipher can be designed with a conversion function (other than decrypt and encrypt) such that a variable tradeoff between workload and security (effective key length) can be set per application.

We provided methods for constructing a secure conversion crypto-system from any symmetric key cipher that can convert text between the ciphertext corresponding to encryption under one key to that corresponding to encryption under a second key without exposing the plaintext during the conversion. The work of the resulting system is twice that of the underlying cipher with an effective key length of that of the underlying cipher. If the underlying cipher consists of multiple rounds, the conversion crypto-system can be defined such that the number of rounds performed by the sender, converter and receiver vary according to the desired level of security and workload. If the underlying cipher is a stream cipher, the conversion can be implemented in a manner requiring no additional work on the sending and receiving entities, and either allowing or disallowing the converter access to the plaintext. The security is unchanged from that of the basic stream cipher with the added benefit that the plaintext is not exposed during the conversion.

## References

[1] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, February 2005.

[2] M. Blaze, G. Bleumer, and M. Strauss. Atomic Proxy Cryptography and Protocol Divertibility. In *Proceedings of EUROCRYPT '98, LNCS 1403, Springer-Verlag*, May 1998.

[3] A. Chandra. Efficient Compilation of Linear Recursive Programs. Technical Report STAN-CS-72-282, Stanford University, April 1972.

[4] D. Cook and A. Keromytis. Conversion and Proxy Functions for Symmetric Key Ciphers. In *IEEE International Conference on Information Technology: Coding and Computing (ITCC), Information Assurance and Security Track*, pages 662–667, April 2005.

[5] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[6] P. Fairbrother. An Improved Construction for Universal Re-encryption. In *Workshop in Privacy Enhancing Technologies*, May 2004.

[7] FIPS 46-3. Data Encryption Standard (DES), 1999.

[8] A. Ivan and Y. Dodis. Proxy Cryptography Revisted. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, February 2003.

[9] B. Kaliski, R. Rivest, and A. Sherman. Is the Data Encryption Standard a Group? *Journal of Cryptology*, pages 3–36, 1988.

[10] E. Okamoto and M. Mambo. Proxy Cryptosystems: Delegation of Power to Decrypt Ciphertexts. *IEICE Trans. Fund. Eletronic Communications and Comp Sci. E80-A/1*, pages 54–63, 1997.

[11] E. Okamoto, K. Usuda, and M. Mambo. Proxy Cryptosystems: Delegation of Power to Decrypt Ciphertex ts. *IEICE Trans. Fund. Eletronic Communications and Comp Sci. E79 -A/9*, pages 1338–1354, September 1996.

[12] RSA Laboratories. *PKCS #1: RSA Encryption Standard*, version 1.5 edition, November 1993.

[13] US Navy Research Laboratories. Onion Routing. http://www.onion-router.net, 1998.

## Author Biographies

**Debra Cook** is a Ph.D. student in Computer Science Department at Columbia University in New York. She is completing her doctorate in 2006. Her research interests are in in applied cryptography and security. She has a B.S. and M.S.E. in mathematical sciences from the Johns Hopkins University in Baltimore, Maryland and a M.S. in computer science from Columbia University. After graduating from Johns Hopkins, she was a senior technical staff member at Bell Labs and AT&T Labs before pursuing her Ph.D.

**Angelos Keromytis** is an Associate Professor of Computer Science at Columbia University. He received his Masters and PhD from the University of Pennsylvania, and his Bachelors (all in Computer Science) from the University of Crete, in Greece. His research interests include network and system survivability, authorization and access control, and large-scale systems security. A full CV can be found at http://www.cs.columbia.edu/~angelos/cv.html