Title

Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography

Permalink

https://escholarship.org/uc/item/80n3m1fj

Authors

Bellare, Mihir
Rogaway, Phillip

Publication Date

2000-03-06

Encode-then-encipher encryption:
How to exploit nonces or redundacy in plaintexts
for efficient cryptography

Mihir Bellare[*] Phillip Rogaway[†]

October 1999

Abstract

We investigate the following approach to symmetric encryption: first *encode* the message in some trivial way (eg., prepend a counter and append a checksum), and then *encipher* the encoded message. Here "encipher" means to apply a cipher (i.e. pseudorandom permutation) F_K, where K is the shared key. We show that if the encoding step incorporates a nonce (counter or randomness), in any way at all, then the resulting encryption scheme will be semantically secure. And we show that if the encoding step incorporates redundancy, in any form at all, then, as long as the receiver verifies the presence of this redundancy in the deciphered string, the resulting encryption scheme achieves message authenticity. The second result helps explain and justify the prevalent misunderstanding that encrypting messages which have redundancy is enough to guarantee message authenticity: the statement is actually true if "encrypting" is understood as "enciphering."

Encode-then-encipher encryption can be used to robustly and efficiently exploit structured message spaces. If one is presented with messages known *a priori* to contain something that behaves as a nonce, then privacy can be obtained with no increase in message length, and no knowledge of the structure of the message, simply by enciphering the message. Similarly, if one is presented with messages known *a priori* to contain adequate redundancy, then message authenticity can be obtained with no increase in message length, and no knowledge of the structure of the message, simply by enciphering the message.

Keywords: Ciphers, Encoding, Modes of Operation, Provable Security, Symmetric Encryption.

[*] Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-Mail: `mihir@cs.ucsd.edu` URL: `www-cse.ucsd.edu/users/mihir`.

[†] Dept. of Computer Science, Engineering II Bldg., One Shields Ave., University of California at Davis, Davis, CA 95616, USA. E-Mail: `rogaway@cs.ucdavis.edu` URL: `www.cs.ucdavis.edu/~rogaway/`

Contents

1 Introduction

Enciphering vs. encrypting. Many popular books on cryptography describe "encryption" as applying a key-indexed permutation F_K to the plaintext M, thereby obtaining the ciphertext $C = F_K(M)$. Yet, if the goal of encryption is privacy (as it is usually assumed to be), then our community has long since recognized that, being deterministic, realizing encryption in this way cannot possibly achieve the strong security guarantees that one would hope for (semantic security and beyond [5, 1]; for example, if the same message is encrypted twice an adversary should not be able to detect this). So from this point forward we'll avoid using the word "encryption" when what we have in mind is to apply a key-indexed permutation F_K.

A family of permutations $F = \{F_K\}$ will be called a *cipher*. Applying one of these functions F_K is *enciphering* (not encrypting). Applying F_K^{-1} is *deciphering* (not decrypting). In this paper, "good" for an enciphering method means approximating (in the usual ways [6]) a family of random permutations. On the other hand, "good" for an encryption scheme means achieving privacy properties at least as strong as semantic security. As indicated above, good enciphering never, by itself, makes for good encryption.

Despite the last statement, there seems to be a widespread belief that enciphering a message is, somehow, almost as good as encrypting it. When messages are somehow "structured," or the message space has "enough entropy," maybe enciphering does the job. Is there any scientific basis for such a belief?

In this paper we investigate the circumstances under which good enciphering really *does* make for good encryption. This leads us to introduce *encoding schemes* as a way to conceptualize what is happening when you encipher structured messages. Let us describe what are encoding schemes, and how they relate to enciphering.

Encode-then-encipher encryption. Start with a good cipher that operates on messages of any length at all. (In other words, F_K, for a random K, "looks like" a random length-preserving permutation.) Now to encrypt M, first "encode" it into some string M^*. The encoding might be really trivial—like prepending a counter, or appending some 0-bits, or maybe doing both. The encoding might even be the identity function. All that is demanded of an encoding method is that it does not "lose" information: you can "decode" M^* to recover M, and you can recognize when a string is and is not the encoding of some message. To encrypt message M under key K, encipher the encoded message M^* using F_K, yielding ciphertext $C = F_K(M^*)$. To decrypt a ciphertext C decipher it to find $M^* = F_K^{-1}(C)$, and then decode M^* to get either a message M or an indication that M^* is not the encoding of any message. We call this style of encryption "encode-then-encipher encryption." This is not a popular way to encrypt, though it is certainly a very natural paradigm.

Our results. In this paper we investigate how properties of the encoding scheme and the enciphering scheme can give rise to security properties of the resulting encryption scheme.

Suppose first that the encoding scheme adds in a *nonce*—usually a counter or a random value. The nonce can be added into the message in any way at all. All one needs is that the "collision probablity"—the chance that two encoded messages come out the same"—be small. We prove in Theorem 5.1 that enciphering such encodings provides *semantic security.*

Next we look at encoding scheme which result in encoded messages which have enough *redundancy.* This means that "most" strings M^* will be considered "bad." We prove in Theorem 6.1 that the resulting encryption scheme will now achieve *message authenticity.* It is as though the sender had sent a MAC along with his transmission. Interestingly, this theorem requires that the cipher be a *strong* pseudorandom permutation [6]. We show in Theorem 6.2 that an ordinary pseudorandom permutation won't do.

The actual results above are quantitative. The show how much privacy and authenticity is guaranteed as a function of (easily-calcuated) numbers associated to the encoding scheme, and as a function of the (quantified) security of the underlying cipher.

Justifying some old intuition. At some level it would seem to be folklore that enciphering strings which employ nonces or redundancy makes for good encryption. In the security literature one sees many statements to the effect that *we assume that messages to be encrypted employ adequate redundancy*, or *we avoid replay attacks by including a nonce in the message we encrypt.* Our results help formalize what such authors may have had in mind, since the statements above become meaningful and true when "encryption" means "enciphering" and when the roles of nonces and redundancy are formally defined.

Is the encoding process "real"? In some applications of encode-then-encipher encryption we imagine that the encoding step will be an ostensible part of encrypting: the piece of software which encrypts M will encode it first, and then encipher the encoded message. For example, the encryption engine might take in a message M, prepend a counter, append a checksum, and encipher the resulting string. But encode-then-encipher encryption is actually more interesting when the encoding and decoding operations do *not* occur within the customary boundary of the encryption engine. For example, the encryption software may be presented with an already-formatted IP packet M^*. Its payload is the message M one should get on decoding M^*, but the encryption software itself knowns nothing about where is the payload or how to extract it. Still, the encoding and decoding processes really did occur, albeit within a different piece of code. Finally, the encoding step may exist purely as a conceptualization. For example, if messages are supposed to be English-language sentences then the encoding step can be regarded as the the identity function on the space of proper English-language sentences, while the decoding function takes a string M^* and returns $M = M^*$ if it is English, or else an indication that this is not an English sentence. Probably this decoding operation can only performed by a human! Nonetheless, even in this case the language of encodings makes sense.

In general, the encoding of messages should be seen as a *model* for how the messages that we are enciphering might arise. This model is a more useful and general approach than trying to equip an unknown message space with a distribution. For example, a distrbution on messages can not handle ideas like inserting a counter into the message. The encoding/decoding model let's us naturally and generally discuss all the relevant properties about how messages might look.

Why encode-then-encipher? Encode-then-encipher encryption can be used to provide short ciphertexts with a high degree of independence on message-formatting conventions. It can be used to provide a convenient migration path for legacy protocols. Let us explain.

In various application, particularly in networking, a "packet format" will have been defined, where this packet format includes redundancy and/or nonces, but has no fields for cryptographic purposes (eg., fields for an IV or MAC). Now suppose a need arises to add in privacy or authenticity features. At the same time, there will often be a real-world constraint not to grow or re-define the packet format.

Using encode-then-encipher you probably do not have to. If packets are known to repeat rarely or not at all (eg., packets always contain a sequence number) then semantic security is automatically guranteed just by applying a good cipher. And if packet formats already include redundancy (which they typically do if for no other reason than to simplify parsing) then there may be no need to add in a separate MAC; once again, good enciphering (this time, with a *strong* pseudorandom permutation) is enough. And because it is irrelevant how and where the nonce and redundancy appeared in the packet format, privacy and authenticity will be retained, with no protocols changes at all, if packet formats should subsequently change in some details.

The result is that encode-then-encipher encryption would leave packet sizes alone (our ciphers are understood to be length-preserving), and they would leave packets looking identical (after deciphering) to the way they looked before. This allows for modular software changes with minimal code disruption. The code which enciphers as a way to encrypt doesn't know (or care) where is the sequence number (say) or what fields where can take on what values. Such indifference makes for robust and simple software, and thus an easier migration path for adding in security features.

Constructing variable-input-length ciphers. To encrypt with the encode-then-encipher approach you need to encipher strings which may be long or short, and whose lengths may vary from one enciphering to the next. The cipher should look like a random length-preserving permutation $\pi : \mathcal{M}^* \to \mathcal{M}^*$. This may sound just like a block cipher, but it is actually quite different, because the domain includes strings of different lengths. One construction is given in [3], and others are possible.

A notion of authenticity for encryption schemes. We briefly point out one final contribution of this paper, which is the notion of authenticity defined in Section 2. The usual way that message authenticity has been defined (eg., [2]) assumes that each message M is accompanied by a tag (the message authentication code) τ. The adversary wants to produce a hitherto unseen message M' and a valid tag τ' for it. But this setting does not apply to us, where the messages being authenticated are never made visible. In the new setting the adversary's goal is to get the receiver to accept as authentic a string C —with a possibly unknown "meaning" M— where the adversary has not already witnessed C. In fact what we need is a new notion (or measure) of security for a symmetric encryption scheme.

Several definitions of privacy for symmetric encryption schemes are given in [1]. Here we are suggesting a notion of authenticity for an encryption scheme. Namely, consider a symmetric encryption scheme in which the decryption algorithm is allowed to reject ciphertexts to indicate that they are unauthentic. We take the setting of [1] in which the adversary gets to see (via an oracle) ciphertexts of messages of her choice encrypted under a key K. We then say that she wins if she can produce a valid ciphertext (meaning one which the decryption function under K does not reject) which was never an output of the encryption oracle.

Version history. The first version of this paper dates to December 1998. This version, dating from October 1999, was submitted to Eurocrypt 2000 in November 1999.

2 Definitions

Notation and conventions. A *message space* $\mathcal{M}$ is a subset of $\{0,1\}^*$ for which $x \in \mathcal{M}$ implies that $x' \in \mathcal{M}$ for all x' of the same length of x. We also require the existence of an efficient (say linear time) algorithm to decide membership in $\mathcal{M}$. A *ciphertext space* $\mathcal{C}$ is a subset of $\{0,1\}^*$. A *key space* $\mathcal{K}$ is a set together with a probability measure on that set. Writing $K \leftarrow \mathcal{K}$ means to choose K at random according to this probability measure.

When we speak of the running time of an algorithm by convention we mean the actual running time plus the size of the length of the description of that algorithm.

Ciphers and PRFs. Let $\mathcal{K}$, $\mathcal{M}$ and $\mathcal{C}$ be a key space, message space, and ciphertext space. A *pseudorandom function* (PRF) is a collection of functions $F = \{F_K \mid K \in \mathcal{K}\}$, each $F_K : \mathcal{M} \to \mathcal{C}$ sharing the same domain $\mathcal{M}$ and range $\mathcal{C}$. We assume that $|F_K(M)| = \ell(|M|)$ depends only on $|M|$. We call ℓ the *length function* of the PRF.

A *cipher* is a PRF $F = \{F_K \mid K \in \mathcal{K}\}$ in which each $F_K : \mathcal{M} \to \mathcal{C}$ is one-to-one and onto. In this case, F_K^{-1} denotes the inverse of $F_K(\cdot)$. A cipher is *length-preserving* if $F_K(M) = |M|$ for all $K \in \mathcal{K}$ and $M \in \mathcal{M}$. For simplicity, all ciphers in this paper are assumed to be length-preserving. A *block-cipher* is a cipher with domain (and range) $\{0,1\}^n$. The number n is called the *block length.*

Let $\mathcal{M}$ be a message space and let ℓ be a length function. We define two "reference" PRFs:

$\mathsf{Rand}(\mathcal{M}, \ell)$ A random function ρ from this set is determined as follows: for each $M \in \mathcal{M}$ let $\rho(M)$ be a random string in $\{0,1\}^{\ell(|M|)}$.

$\mathsf{Perm}(\mathcal{M})$ A random function π from this set is determined as follows: for each number i such that $\mathcal{M}$ contains strings of length i, let π_i be a random permutation on $\{0,1\}^i$. Then define $\pi(M) = \pi_i(M)$, where $i = |M|$.

Thus $\rho \leftarrow \mathsf{Rand}(\cdot,\cdot)$ is used to choose a random function and $\pi \leftarrow \mathsf{Perm}(\cdot)$ is used to choose a random permutation. The arguments indicate the desired domain and range.

Security of PRFs and ciphers. We follow the formalization of [4], adapted to concrete security as in [2]. A *distinguisher* is a (possibly probabilistic) algorithm A with access to an oracle $\mathcal{O}$. Let A be a distinguisher and let $F = \{F_K \mid K \in \mathcal{K}\}$ be a PRF with key space $\mathcal{K}$ and length function ℓ. Then we let

$$\mathsf{Adv}_F^{\mathrm{prf}}(A) \;=\; \Pr[K \leftarrow \mathcal{K} : A^{F_K(\cdot)} = 1] - \Pr[\rho \leftarrow \mathsf{Rand}(\mathcal{M}, \ell) : A^{\rho(\cdot)} = 1]$$

denote the advantage of A in distinguishing F from a random function. We let

$$\mathsf{Adv}_F^{\mathrm{prp}}(A) \;=\; \Pr[K \leftarrow \mathcal{K} : A^{F_K(\cdot)} = 1] - \Pr[\pi \leftarrow \mathsf{Perm}(\mathcal{M}) : A^{\pi(\cdot)} = 1]$$

denote the advantage of A in distinguishing F from a random permutation. Define

$$\mathsf{Adv}_F^{\mathrm{prf}}(t, q, \mu) \;=\; \max_A\{\mathsf{Adv}_F^{\mathrm{prf}}(A)\} \qquad \text{and} \qquad \mathsf{Adv}_F^{\mathrm{prp}}(t, q, \mu) \;=\; \max_A\{\mathsf{Adv}_F^{\mathrm{prp}}(A)\}$$

where the maximum is taken over all adversaries which run in time at most t and ask at most q oracle queries, these queries totaling at most μ bits.

We also need the notion of a strong-PRP, as defined by Luby and Rackoff [6]. Here the distinguisher gets not only an oracle for the function, but also one for its inverse. Let $F = \{F_K \mid K \in \mathcal{K}\}$ be a PRP with key space $\mathcal{K}$ and length function ℓ. Then we let

$$\mathsf{Adv}_F^{\mathrm{sprp}}(A) \;=\; \Pr[K \leftarrow \mathcal{K} : A^{F_K(\cdot), F_K^{-1}(\cdot)} = 1] - \Pr[\pi \leftarrow \mathsf{Perm}(\mathcal{M}) : A^{\pi(\cdot), \pi^{-1}(\cdot)} = 1]$$

denote the advantage of A in distinguishing F from a random permutation. Define

$$\mathsf{Adv}_F^{\mathrm{sprp}}(t, q, \mu) \;=\; \max_A\{\mathsf{Adv}_F^{\mathrm{sprp}}(A)\}$$

where the maximum is taken over all adversaries which run in time at most t, make at most q queries to the two oracles combined, and all these queries together total at most μ bits.

Throughout, if the distinguisher inquires as to the value of oracle f at a point $M \notin \mathcal{M}$ then the oracle responds with the distinguished point $\perp$. Since we assume that there is a (simple) algorithm to decide membership in $\mathcal{M}$ there is in fact no point for the adversary to make such inquiries.

Encryption-or-authentication schemes. Fix a key space $\mathcal{K}$, a message space $\mathcal{M}$, and a ciphertext space $\mathcal{C}$. An *encryption-or-authentication scheme* is a triple of algorithms, denoted $\Pi = (\mathsf{Keygen}, \mathsf{Send}, \mathsf{Receive})$, as we now describe.

The key-generation algorithm Keygen is a probabilistic algorithm that produces a key $K \in \mathcal{K}$.

Algorithm Send can be either probabilistic or stateful. If probabilistic it takes a key $K \in \mathcal{K}$ and a message $M \in \{0,1\}^*$, and it flips some coins $r \in \{0,1\}^*$. The algorithm then returns $C = \mathsf{Send}_K(M, r)$. If stateful the algorithm takes a key $K \in \mathcal{K}$ and a message $M \in \{0,1\}^*$, and it uses its internal state $r \in \{0,1\}^*$. The algorithm then returns $C = \mathsf{Send}_K(M, r)$, and it may modify its internal state to some new state, r'. Either way, C can be either a binary string in $\mathcal{C}$

or the distinguished symbol $\perp$. The value $\perp$ is used if $M \notin \mathcal{M}$ or (if this is a stateful encryption scheme) the state r indicates that the message M can not be sent (when, for example, too many messages have already been sent).

Algorithm $\mathsf{Receive}$ takes $K \in \mathcal{K}$ and $C \in \{0,1\}^*$ and computes $M = \mathsf{Receive}_K(C)$ where M is either a string in $\mathcal{M}$ or the distinguished symbol $\perp$. A return value of $\perp$ is used to indicate that C is regarded as inauthentic. We call C *valid* if $\mathsf{Receive}_K(C) \in \mathcal{M}$ and we call C *invalid* if $\mathsf{Receive}_K(C) = \perp$.

We also permit applying Send_K to $(\perp, r)$, which results in a return value of $\perp$. Likewise, applying $\mathsf{Receive}_K$ to $\perp$ is permitted and this gives a return value of $\perp$.

We require the following: if $C = \mathsf{Send}_K(M, r)$ and $C \neq \perp$ then $\mathsf{Receive}_K(C) = M$.

When we think of the goal of Π as privacy, or a combination of privacy and message authenticity, then we call Π an *encryption scheme* and name its components by $\Pi = (\mathsf{Keygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$. When we think of the goal of Π as authenticating messages then we call Π an *authentication scheme* and we name its components by $\Pi = (\mathsf{Keygen}, \mathsf{Authenticate}, \mathsf{Recover})$. But we emphasize that there is no formal distinction between an encryption scheme and an authentication scheme under this formalization.

Privacy. Several formalizations for the security of a symmetric encryption scheme under chosen-plaintext attack were provided in [1] and compared in terms of concrete security. We will use one of these notions, namely "real-or-random" security.

The idea is that an adversary cannot distinguish the encryption of text from the encryption of an equal-length string of garbage. For the formalization, let $\Pi = (\mathsf{Keygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be an encryption scheme and let A be an adversary with an encryption oracle. If the encryption scheme is probabilistic then fresh random choices are made for each query. If the encryption scheme is stateful then the state is properly initialized and then adjusted with each query. Define

$$\mathsf{Adv}_{\Pi}^{\mathrm{priv}}(A) \;=\; \Pr\left[K \leftarrow \mathsf{Keygen} : A^{\mathsf{Encrypt}_K(\cdot)} = 1\right] - \Pr\left[K \leftarrow \mathsf{Keygen} : A^{\mathsf{Encrypt}_K(\$^{|\cdot|})} = 1\right] .$$

In the first game, the oracle, given a message, returns its encryption under key K; in the second game the oracle, given a message, ignores it except to record its length n, and then returns the encryption of a random message of length n. The advantage of A is a measure of the adversary's ability to tell these two worlds apart. We define

$$\mathsf{Adv}_{\Pi}^{\mathrm{priv}}(t, q, \mu) \;=\; \max_{A}\{\mathsf{Adv}_{\Pi}^{\mathrm{priv}}(A)\}$$

where the maximum is over all adversaries who run in time at most t and ask at most q oracle queries, where these queries total at most μ bits.

Authenticity. A sender who holds a message M wants to send a string C to a receiver with whom he shares a key K. The sender wants to do this in such a way that the receiver can recover M from C and the receiver can be confident that C (and M) really does originate with the sender. That is, if C does not originate with the sender then the receiver should figure this out and C should be deemed a forgery.

To formalize this an adversary will be given a way to generate authenticated messages of her choice: $M_1 \mapsto C_1, M_2 \mapsto C_2, \ldots, M_q \mapsto C_q$. She will "win" if she computes a new string C (that is, $C \notin \{C_1, \ldots, C_q\}$) which would be deemed authentic by the receiver.

We remark that this notion of authenticity differs from a message authentication code (MAC) in several way. A MAC is a tag t which accompanies the message M which it authenticates. So appending a MAC —letting $C = (M, t)$— is *one way* to accomplish message authentication. Here the message M is recovered from the authenticated message C in a particularly trivial way; you don't even need the key to do this. But there is a deeper difference between a MAC and a general message authentication scheme. In formalizing the security of a MAC the adversary makes a number of queries to a MAC-generation oracle, with each query mapping the message M_i to its tag t_i. After that the adversary has to come up with a new message M and a tag t such that the receiver will deem (M, t) authentic. In particular, the adversary must "know" the message M that is being forged, insofar as the adversary outputs it along with t. In contrast, an adversary attacking message authentication scheme in the general sense we are defining wins *even if she does not know what is the message M which is being forged.* All that is required is that there *is* such a message underlying C—that is, the receiver will recover *something* in the message space $\mathcal{M}$ (and not an indication that C is bogus).

Formally, let $\Pi = (\mathsf{Keygen}, \mathsf{Authenticate}, \mathsf{Recover})$ be an authentication scheme and let A be an adversary who is given oracle access to $\mathsf{Authenticate}$. After interacting with that oracle the adversary outputs a string C. We say C is *new* if C was not the response to any earlier oracle query asked by A. Adversary A is said to be *successful* if C is new and valid, and we measure the probability of this:

$$\mathsf{Adv}^{\mathrm{auth}}_{\Pi}(A) \;=\; \Pr[\,K \leftarrow \mathsf{Keygen}\,;\, C \leftarrow A^{\mathsf{Authenticate}_K(\cdot)} \;:\; C \text{ is new and } \mathsf{Recover}_K(C) \neq \perp\,]\;.$$

The quality of Π in authenticating messages is measured by the function

$$\mathsf{Adv}^{\mathrm{auth}}_{\Pi}(t, q, \mu) \;=\; \max_{A}\{\mathsf{Adv}^{\mathrm{auth}}_{\Pi}(A)\}$$

where the maximum is over all adversaries who run in time at most t and make at most $q-1$ oracle calls, these totalling a most $\mu - |C|$ bits, where C is the length of A's output. For simplicity, we assume that an adversary A attacking the authenticity of Π will only output a string which is new.

3 Encoding Schemes

Syntax. Fix message spaces $\mathcal{M}, \mathcal{M}^*$. An *encoding scheme* ("on $\mathcal{M}$", or "from $\mathcal{M}$ to $\mathcal{M}^*$") is a pair of algorithms $\mathsf{S} = (\mathsf{Encode}, \mathsf{Decode})$ as we now describe.

Algorithm Encode can be either probabilistic or stateful, while Decode is neither. First assume that Encode is probabilistic (not stateful). Then each time Encode is called on an input $M \in \mathcal{M}$ the algorithm flips some coins, r, and returns a string $M^* = \mathsf{Encode}(M, r) \in \mathcal{M}^*$. We assume that for any string $M \in \mathcal{M}$ and any coins r, we have that $|\mathsf{Encode}(M, r)| = \ell(|M|)$ for some function ℓ, the "length function" of the encoding scheme.

Algorithm Decode takes as input $M^* \in \{0, 1\}^*$. It returns either a binary string $M \in \mathcal{M}$ or the distinguished symbol $\perp$. If $\mathsf{Decode}(M^*)$ is a binary string we say that M^* is *valid*, while we say that M^* is *invalid* if $\mathsf{Decode}(M^*) = \perp$. We demand that for any $M \in \mathcal{M}$ and any r, we have that $\mathsf{Decode}(\mathsf{Encode}(M, r)) = M$.

We allow that Encode and Decode be presented with any string at all, even ones outside of $\mathcal{M}$ and $\mathcal{M}^*$. If you try to encode a string $M \notin \mathcal{M}$ then the result is the distinguished value $\perp$. If you try to decode a string $M^* \notin \mathcal{M}^*$ then the result is the distinguished value $\perp$. We further establish the convention that you can encode or decode $\perp$, which once again returns $\perp$.

For simplicity in theorem statements we assume that Encode and Decode are efficiently computable, say in linear time.

Rare-collision encodings. Let $\mathsf{S} = (\mathsf{Encode}, \mathsf{Decode})$ be an encoding scheme and let $\ell(n)$ be its length function. Let $\epsilon(q) : \mathsf{N} \to \mathbf{R}$ be a function. We say that S is **ϵ-colliding** if for and any number q and any (computationally unbounded) adversary A who asks q queries the probability of some two of these queries receive the same valid response is at most $\epsilon(q)$:

$$\Pr[\,(M_1^*, \ldots, M_q^*) \leftarrow \mathsf{Responses}\ A^{\mathsf{Encode}(\cdot)} :\; \exists\, i < j \text{ s.t. } M_i^* \neq \perp,\, M_j^* \neq \perp, \text{ and } M_i^* = M_j^*\,] \leq \epsilon(q)\;.$$

We shall say that $\langle M_1^*, \ldots M_q^* \rangle$ "collide" if some pair of these strings are the same and are different from $\perp$. The reader may prefer to think of $M_1 = M_2 = \cdots = M_q$ since typically this would be the adversary's best strategy when trying to produce a collision (as $M \neq M'$ implies that their encodings, if valid, have to be different).

Example 3.1 Encoding scheme Prepend-128-Random-Bits works as follows. The message space is $\mathcal{M} = \{0,1\}^*$. Function Encode takes an input M and outputs $r \parallel M$, where r is a sequence of 128 random bits. Function Decode takes an input M^* and behaves as follows. If M^* is at least 128 bits, then Decode outputs all but the first 128 bits of M^*. If M^* is less than 128 bits then $\mathsf{Decode}(M^*)$ outputs $\perp$. Then Prepend-128-Random-Bits is $C(q, 2^{128})$-colliding, where $C(q,m)$ denotes the probability of at least one collision in the experiment of throwing q balls, independently and at random, into m bins.

Collision-free encodings. For algorithm Encode to be stateful means that it maintains state across invocations. The initial value of that state is some fixed constant, r_0. Typically there will be a limit, N, on the number of times that Encode may be used. After that number of invocations Encode will return $\perp$ even when the inquiry is in $\mathcal{M}$. We require that for all messages M and all internal states r, if $\mathsf{Encode}(M,r)$ returns a binary string M^* then $\mathsf{Decode}(M^*) = M$. We emphasize that decoding is stateless.

Stateful encoding schemes are of interest because with them we can make an encoding scheme **collision free**, meaning 0-colliding, in the language above. Note that getting two $\perp$ values does *not* count as a collision. Here is an example.

Example 3.2 Encoding scheme Prepend-64-Bit-Counter works as follows. The message space is $\mathcal{M} = \{0,1\}^*$. A counter *ctr* is initialized to 0. The i-th message is encoded as follows. If $i \geq 2^{64}$ then the encoding is $\perp$. Otherwise the encoding is $M^* = \langle i \rangle \parallel M$, where $\langle i \rangle$ the number i written as a 64-bit binary string. Function Decode takes an input M^* and behaves as follows. If $|M^*| < 64$ then Decode returns $\perp$. Otherwise it returns M^* after having expunged the first 64-bits. Clearly Prepend-64-Bit-Counter is collision free: the counter guarantees that no two encodings can collide.

Sparse encodings. Let $\mathsf{S} = (\mathsf{Encode}, \mathsf{Decode})$ be an encoding scheme and let δ be a real number. We say that encoding scheme S is **δ-dense** if for all $n \in \mathsf{N}$,

$$\Pr[\, M^* \leftarrow \{0,1\}^n : \mathsf{Decode}(M^*) \in \{0,1\}^* \,] \leq \delta \; .$$

That is, for every message length, at most a δ-fraction of all strings of that length are valid (they decode to strings in $\mathcal{M}$). The rest are invalid encodings (they decode to $\perp$).

Example 3.3 The encoding scheme Prepend-32-Zeros works as follows. Let $\mathcal{M} = \{0,1\}^*$. Define $\mathsf{Encode}(M) = 0^{32} \parallel M$. Define $\mathsf{Decode}(M^*)$ to be M^* after stripping away its first 32 bits, assuming that M^* has at least 32 bits, and set $\mathsf{Decode}(M^*) = \perp$ otherwise. Then Prepend-32-Zeros is 2^{-32}-dense: a string is valid (it starts with 32 zeros) with probability at most 2^{-32}. Indeed the probability that a random string M^* is valid is exactly 2^{-32} if the length of M^* is at least 32 bits, while the probability is 0 if the length of M^* is less than 32 bits.

Example 3.4 Let the message space $\mathcal{M}$ be odd-parity-adjusted ASCII strings of length at least 50 bytes. This means that a message $M \in \mathcal{M}$ is a sequence of bytes $M = b_1 \parallel \cdots \parallel b_n$, for $n \geq 50$, where each b_i is a byte having its low 7 bits arbitrary and its high bit whatever is necessary so that

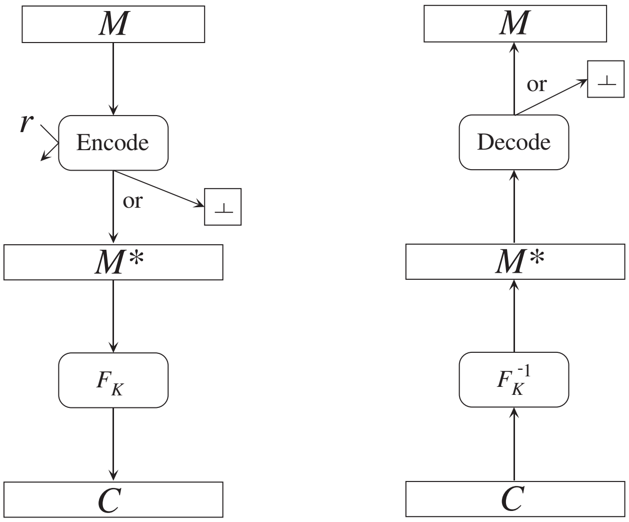

Figure 1: *Scheme $F \circ \mathsf{S}$: encrypting (left-hand side) and decrypting (right-hand side) using the encode-then-encipher paradigm. The plaintext is M, the ciphertext is C, the cipher is $F = \{F_K\}$, and the encoding scheme is* $\mathsf{S} = (\mathsf{Encode}, \mathsf{Decode})$.

the number of 1-bits in b_i will be odd. Encoding scheme Odd-Parity is defined as follows. Function Encode is the identity function. Function Decode checks that the bit length of its input is divisible by 8, that the input is at least 50 bytes, and that each byte has odd parity. If these conditions are satisfied then Decode returns its input. Otherwise it returns $\perp$. Then Odd-Parity is 2^{-50}-dense: a random string is valid with probability at most 2^{-50}. Indeed the probability that a random n-byte string is valid is 2^{-n} if $n \geq 50$, and 0 if $n < 50$ or if the input is not a byte string at all.

4 Enciphering Encoded Messages

Let $\mathsf{S} = (\mathsf{Encode}, \mathsf{Decode})$ be an encoding scheme from $\mathcal{M}$ to $\mathcal{M}^*$ and let $F = \{F_K : \mathcal{M}^* \to \mathcal{M}^*\}$ be a cipher with key space $\mathcal{K}$. Then we define the following encryption-or-authentication scheme $F \circ \mathsf{S} = (\mathsf{Keygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$:

(1) Keygen chooses a random key $K \leftarrow \mathcal{K}$ and outputs it.

(2) $\mathsf{Encrypt}_K(M)$ sets $M^* \leftarrow \mathsf{Encode}(M)$, returns $\perp$ if $M^* = \perp$, and otherwise computes $C \leftarrow F_K(M^*)$ and returns that. Algorithm Encrypt is stateful if and only if Encode is. If Encode is stateful then the initial state for Encrypt is the initial state mandated by Encode, and Encrypt maintains the state needed by the encoding scheme.

(3) $\mathsf{Decrypt}_K(C)$ returns $\perp$ if $C \notin \mathcal{M}^*$, and otherwise computes $M^* \leftarrow F_K^{-1}(C)$, sets $M \leftarrow \mathsf{Decode}(M^*)$, and returns M.

For a pictorial representation, see Figure 4.

5 Privacy from Rare/Collision-Free Encodings

We show that encryption scheme $F \circ \mathsf{S}$ is private if encoding scheme S has rare or no collisions and F is a secure cipher, in the sense of being a good PRP. The following theorem makes this formal and quantitative.

Theorem 5.1 Let $\mathsf{S} = (\mathsf{Encode}, \mathsf{Decode})$ be an encoding scheme from $\mathcal{M}$ to $\mathcal{M}^*$ and let $F = \{F_K : \mathcal{M}^* \to \mathcal{M}^*\}$ be a cipher with key space $\mathcal{K}$. Suppose that S is ϵ-colliding. Then $F \circ \mathsf{S} = (\mathsf{Keygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ has security

$$\mathsf{Adv}^{\mathrm{priv}}_{F\circ\mathsf{S}}(t, q, \mu) \quad \leq \quad \mathsf{Adv}^{\mathrm{prf}}_{F}(t', q, \mu) + \epsilon(q),$$

where $t' = t + O(\mu)$.

Proof: Let B be an adversary attacking the privacy of $F \circ \mathsf{S}$. Let t be its running time, q the number of queries it makes, and μ the length of all its queries put together. plus the length of B's output. Our goal is to upper bound $\mathsf{Adv}^{\mathrm{priv}}_{F\circ\mathsf{S}}(B)$. To this end we introduce a couple of more algorithms and some associated probabilities.

Algorithm D is a distinguisher for F. It is given an oracle for a permutation $f \in \mathsf{Perm}(\mathcal{M}^*)$. It runs B. When B makes an oracle query M, distinguisher D computes $M^* \leftarrow \mathsf{Encode}(M)$ and $C \leftarrow f(M^*)$. It returns C to B as the answer to the query. When B terminates, D outputs whatever B outputs.

Algorithm A is a collision finding adversary for S. It is given oracle Encode. It picks a permutation f from $\mathsf{Perm}(\mathcal{M}^*)$ at random. (Or simulates such a permutation. The difference is technically immaterial since the running time of A is not restricted.) It then runs B. When B makes an oracle query M, algorithm A computes $M^* \leftarrow \mathsf{Encode}(M)$ and $C \leftarrow f(M^*)$. It returns C to B as the answer to the query. When B terminates, so does A.

We now define the following probabilities:

$$\begin{aligned}
p_1 &= \Pr[K \leftarrow \mathsf{Keygen} : B^{\mathsf{Encrypt}_K(\cdot)} = 1] \\
p_2 &= \Pr[K \leftarrow \mathsf{Keygen} : B^{\mathsf{Encrypt}_K(\$^{|\cdot|})} = 1] \\
p_3 &= \Pr[K \leftarrow \mathcal{K} : D^{F_K(\cdot)} = 1] \\
p_4 &= \Pr[\pi \leftarrow \mathsf{Perm}(\mathcal{M}^*) : D^{\pi(\cdot)} = 1] \\
p_5 &= \Pr[(M_1^*, \ldots, M_q^*) \leftarrow \mathsf{Responses}\ A^{\mathsf{Encode}(\cdot)} :\ \exists\, i < j \text{ s.t. } M_i^* = M_j^* \neq \bot] \,.
\end{aligned}$$

Note that $\mathsf{Adv}^{\mathrm{priv}}_{F\circ\mathsf{S}}(B) = p_1 - p_2$. To upper bound it we use the following claims.

Claim 1: $p_1 = p_3$.

Proof: This follows from the definitions of D and $F \circ \mathsf{S}$. □

Claim 2: $p_2 \geq p_4 - p_5$.

Proof: Let C be the event that there is a collision, meaning $\exists\ i < j$ s.t. $M_i^* = M_j^* \neq \bot$. Let experiment i be that underlying p_i for $i \in \{2, 4, 5\}$. Event C is defined in all of experiment 2,4 and 5. In all cases, it has the same probability, and conditioned on it not happening, the probability that D outputs 1 in experiment 1 equals the probability that B outputs 1 in experiment 4. Finally, the probability of C is p_5. □

Given these claims we have

$$\mathsf{Adv}^{\mathrm{priv}}_{F\circ\mathsf{S}}(B) = p_1 - p_2 \leq p_3 - (p_4 - p_5) = (p_3 - p_4) + p_5 \leq \mathsf{Adv}^{\mathrm{prp}}_{F}(D) + \epsilon(q) \,.$$

This concludes the proof of Theorem 5.1. ▌

6 Authenticity from Sparse Encodings

We show that $F \circ \mathsf{S}$ is an authenticated encryption scheme if encoding S adds adequate redundancy and F is a strong PRP. The following theorem makes this formal and quantitative. We remark that this result requires that the PRP be strong, which the previous result did not, and we subsequently show this extra requirement is necessary.

Theorem 6.1 Let $\mathsf{S} = (\mathsf{Encode}, \mathsf{Decode})$ be an encoding scheme from $\mathcal{M}$ to $\mathcal{M}^*$ and let $F = \{F_K : \mathcal{M}^* \to \mathcal{M}^*\}$ be a cipher with key space $\mathcal{K}$. Suppose that S is δ-dense and that $q \leq \frac{1}{2\delta}$. Then $F \circ \mathsf{S} = (\mathsf{Keygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ has security

$$\mathsf{Adv}^{\mathrm{auth}}_{F \circ \mathsf{S}}(t, q, \mu) \quad \leq \quad \mathsf{Adv}^{\mathrm{sprp}}_{F}(t', q, 2\mu) + 2\delta,$$

where $t' = t + O(\mu)$.

Proof: Let B be an adversary attacking the authenticity of $F \circ \mathsf{S}$. Let t be its running time, $q - 1$ the number of queries it makes, and μ the total length of all its queries put together, and its final output. Our goal is to upper bound $\mathsf{Adv}^{\mathrm{auth}}_{F \circ \mathsf{S}}(B)$. To this end we introduce an algorithm D and some probabilities.

Algorithm D is a distinguisher for F. It is given two oracles: f and f^{-1}, where $f \in \mathsf{Perm}(\mathcal{M}^*)$ is a permutation. It runs B. When B makes an oracle query M, distinguisher D computes $M^* \leftarrow \mathsf{Encode}(M)$ and $C \leftarrow f(M^*)$. It returns C to B as the answer to the query. When B terminates, it outputs a ciphertext $\mathbf{C}$, which is supposed to its forgery. Algorithm D outputs 0 if $\mathbf{C} \notin \mathcal{M}^*$. Otherwise D computes $\mathbf{M}^* \leftarrow f^{-1}(\mathbf{C})$ (this is the one and only time it uses its f^{-1} oracle). Algorithm D then computes $\mathbf{M} \leftarrow \mathsf{Decode}(\mathbf{M}^*)$. If $\mathbf{M} = \perp$ then D outputs 0, else D outputs 1.

We now define the following probabilities:

$$\begin{aligned} p_1 &= \Pr[K \leftarrow \mathsf{Keygen}\ ;\ \mathbf{C} \leftarrow B^{\mathsf{Encrypt}_K(\cdot)} : \mathbf{C} \text{ is new and } \mathsf{Decrypt}_K(\mathbf{C}) \neq \perp] \\ p_2 &= \Pr[K \leftarrow \mathcal{K} : D^{F_K(\cdot), F_K^{-1}(\cdot)} = 1] \\ p_3 &= \Pr[\pi \leftarrow \mathsf{Perm}(\mathcal{M}^*) : D^{\pi(\cdot), \pi^{-1}(\cdot)} = 1] \end{aligned}$$

Note that $\mathsf{Adv}^{\mathrm{auth}}_{F \circ \mathsf{S}}(B) = p_1$. To upper bound it we use the following claims.

Claim 1: $p_1 = p_2$.

Proof: This follows from the definitions of D and $F \circ \mathsf{S}$. □

Claim 2: $p_3 \leq 2\delta$.

Proof: If the ciphertext $\mathbf{C}$ is new and $f = \pi$ is a random permutation, then $\mathbf{M}^* = f^{-1}(\mathbf{C})$ is random subject to not being in the set $\{\, f^{-1}(C_i) \ :\ i = 1, \ldots, q \,\}$ where $C_i = f(\mathsf{Encode}(M_i))$ is the response to the i-th query M_i of B. For any integer n let

$$g(n) \ = \ |\{\, M^* \in \{0,1\}^n \ :\ \mathsf{Decode}(M^*) \in \{0,1\}^* \,\}| \ .$$

We know $g(n)/2^n \leq \delta$ for all n. In the worst case, all the ciphertexts are of some common length n. The probability that $\mathsf{Decode}(\mathbf{M}^*) \neq \perp$ is at most $g(n)/(2^n - q)$. If $g(n) = 0$ then this probability is zero, and hence certainly at most 2δ, so suppose $g(n) \neq 0$. In that case $g(n) \geq 1$, so it must be that $2^n \geq 1/\delta$. But by assumption $q \leq 1/(2\delta)$. So $2^n - q \geq 1/\delta - 1/(2\delta) = 1/(2\delta)$. So $g(n)/(2^n - q) \leq 2\delta$. □

Given these claims we have

$$\mathsf{Adv}^{\mathrm{auth}}_{F \circ \mathsf{S}}(B) = p_1 = p_2 = (p_2 - p_3) + p_3 \leq \mathsf{Adv}^{\mathrm{prp}}_{F}(D) + 2\delta \ .$$

This conclues the proof of Theorem 6.1. ▌

We now discuss the necessity of the extra requirement on the PRP above, namely that it be strong. The following indicates that without this requirement, the authenticity does not hold. Using the bounds found in the proof, the slightly informal theorem statement is easily adapted to give a more precise (but less understandable) quantitative assertion.

Theorem 6.2 If there exists a secure PRP then there exists a secure PRP F (that is not a strong-PRP) and a δ-dense encoding scheme S for which the scheme $F \circ \mathsf{S}$ does not achieve authenticity.

Proof: Let $\mathcal{M} = \mathcal{K} = \{0,1\}^n$ and $\mathcal{M}^* = \{0,1\}^{2n}$. Let $\mathsf{S} = (\mathsf{Encode}, \mathsf{Decode})$ be the encoding scheme from $\mathcal{M}$ to $\mathcal{M}^*$ in which $\mathsf{Encode}(M) = M0^n$, namely the encoding function appends n zeros to the input. The Decode function on input a $2n$ bit string M^* outputs $\perp$ if the last n bits of M^* are not all zero, and otherwise outputs the last n bits of M^*. This encoding scheme is δ-dense where $\delta = 2^{-n}$.

Assume we are given a secure PRP $G = \{G_K : \mathcal{M}^* \to \mathcal{M}^*\}$ with key space $\mathcal{K}$. We construct a secure PRP $F = \{F_{K_1,K_2} : \mathcal{M}^* \to \mathcal{M}^*\}$ with key space $\mathcal{K} \times \mathcal{K}$. This will have the property that $F \circ \mathsf{S} = (\mathsf{Keygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ can be broken from the authenticity point of view.

We will design F so that for any key K we have $F_{K_1,K_2}(K_2 0^n) = 0^{2n}$. Let us first see why this furnishes the necessary example, and then see how to construct F from G.

The attack on $F \circ \mathsf{S}$ is as follows. The adversary makes no queries. It simply outputs $C = 0^{2n}$. Now notice that $\mathsf{Decode}(F^{-1}_{K_1,K_2}(C)) = \mathsf{Decode}(K_2 0^n) = K_2$ is not equal to $\perp$, so the adversary wins, with probability one. This shows the scheme is totally insecure.

Notice that F is definitely *not* a strong-PRP. If a distinguisher has oracles f, f^{-1}, it can call f^{-1} on 0^{2n}, and obtain a reply xy. It returns 1 if $y = 0^n$ and 0 otherwise. The probability it returns 1 is 1 if $f = F_{K_1,K_2}$, and 2^{-n} if f is a random permutation on $\mathcal{M}^*$, so its advantage is almost one.

To complete the proof we need to show how to construct F from G so that it preserves the PRPness and invertibility of G–

```
Algorithm $F_{K_1,K_2}(xy)$ (Here $|x| = |y| = |K_1| = |K_2| = n$)
  If $xy = K_2 0^n$ then return $0^{2n}$
  Else If $G_{K_1}(xy) = 0^{2n}$ then return $G_{K_1}(K_2 0^n)$
    Else return $G_{K_1}(xy)$
```

We need to show that F is a family of permutations (meaning each F_{K_1,K_2} is invertible) and also that F is almost as secure as G in the PRP sense. To see the first, note that the following works as the inverse function–

$$F^{-1}_{K_1,K_2}(wz) \;=\; \begin{cases} K_2 0^n & \text{if } wz = 0^{2n} \\ G^{-1}_{K_1}(0^{2n}) & \text{if } wz = G_{K_1}(K_2 0^n) \\ G^{-1}_{K_1}(wz) & \text{otherwise.} \end{cases}$$

That F is still a secure PRP is because an adversary given an oracle f is unlikely to be able to touch f in any of the "bad" places, since one of them would correspond to knowing a part of the key, and another to inverting a random permutation on a fixed point. More precisely we claim that for any $q \leq 2^n/4$ and any t, μ we have

$$\mathsf{Adv}^{\mathrm{prp}}_F(t, q, \mu) \leq \mathsf{Adv}^{\mathrm{prp}}_G(t + O(n), q, \mu) + \frac{4q}{2^n} \,. \tag{1}$$

To see this, let D_F be a distinguisher for F that makes q queries, runs in time t, and queries a total of μ bits. We define a distinguisher D_G for G. The code is below and explanations follow–

```
**Algorithm** $D_G^g$ (Here $g$: $\mathcal{M}^* \to \mathcal{M}^*$)
    $K_2 \leftarrow \{0,1\}^n$
    **For** $i = 1, \ldots, q$ **do**
        $D_F \to x_i y_i$
        $w_i z_i \leftarrow g(x_i y_i)$
        **If** $x_i = K_2$ and $y_i = 0^n$ **then return** 1
        **Else If** $w_i z_i = 0^{2n}$ **then return** 1
        **Else** $D_F \leftarrow w_i z_i$
    $D_F \leftarrow b$
    **return** $b$
```

Distinguisher D_G first picks a string K_2 at random to play the role of the second part of the key for F. It then starts running D_F. The notation $D_F \to x_i y_i$ means D_F makes query $x_i y_i$. D_G must answer it. It answers by the value of g on this point, except for two special cases, in each of which D_G halts and returns 1. If D_G did not halt in the query process it obtains the final decision bit b of D_F, and returns the same.

We claim that

$$\begin{aligned} \Pr[K \leftarrow \mathcal{K} : D_G^{G_K(\cdot)} = 1] &\ \geq\ \Pr[(K_1, K_2) \leftarrow \mathcal{K} \times \mathcal{K} : D_F^{F_{K_1,K_2}(\cdot)} = 1] \qquad (2) \\ \Pr[\pi \leftarrow \mathsf{Perm}(\mathcal{M}^*) : D_G^{\pi(\cdot)} = 1] &\ \leq\ \Pr[\pi \leftarrow \mathsf{Perm}(\mathcal{M}^*) : D_F^{\pi(\cdot)} = 1] + \frac{2q}{2^n} \ . \qquad (3) \end{aligned}$$

We will argue this below. Assuming it, subtraction gives us

$$\mathsf{Adv}_F^{\mathrm{prp}}(D_F) \leq \mathsf{Adv}_G^{\mathrm{prp}}(D_G) + \frac{4q}{2^n} \ ,$$

which implies Equation (1).

Equation (2) is true because D_G simulates D_F except in some cases, but in those cases it returns 1, so its probability of returning 1 can only increase with respect to that of D_F. On the other hand, the difference

$$\Pr[\pi \leftarrow \mathsf{Perm}(\mathcal{M}^*) : D_G^{\pi(\cdot)} = 1] - \Pr[\pi \leftarrow \mathsf{Perm}(\mathcal{M}^*) : D_F^{\pi(\cdot)} = 1]$$

is bounded above by the probability that one of the special cases occurs when running D_G with a random permutation π. Each time, two of the remaining inputs could cause one of the special cases to happen, so the chance is bounded above by

$$\sum_{i=0}^{q-1} \frac{2}{2^n - 2i} \ \leq\ \frac{2q}{2^n/2} \ ,$$

the last inequality exploiting the assumption that $q \leq 2^n/4$. This gives us Equation (3) and completes the proof. ▌

Acknowledgments

Mihir Bellare was supported in part by NSF CAREER AWARD CCR-9624439 and a Packard Foundation Fellowship in Science and Engineering. Phillip Rogaway was supported in part under NSF CAREER Award CCR-962540, and under MICRO grants 97-150 and 98-129, funded by RSA Data Security, Inc.. Much of Phil's work on this paper was carried out while on sabbatical at Chiang Mai University, Thailand, hosted by the Computer Service Center, under Prof. Krisorn Jittorntrum and Prof. Darunee Smawatakul.

References

[1] M. Bellare, A. Desai, E. Jokipii and P. Rogaway, "A concrete security treatment of symmetric encryption." *Proceedings of the* 38th *Symposium on Foundations of Computer Science*, IEEE, 1997.

[2] M. Bellare, J. Kilian and P. Rogaway, "On the security of cipher block chaining." *Advances in Cryptology – Crypto* 94 *Proceedings*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.

[3] M. Bellare and P. Rogaway, "On the construction of variable-input-length ciphers." *Proceedings of the 6th Workshop on Fast Software Encryption*, Ed. L. Knudsen, 1999.

[4] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions." *Journal of the ACM*, Vol. 33, No. 4, 210–217, (1986).

[5] S. Goldwasser and S. Micali, "Probabilistic encryption." *Journal of Computer and System Sciences* **28**, 270-299, April 1984.

[6] M. Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions." *SIAM J. Computing,* Vol. 17, No. 2, April 1988.

[7] R. Rivest, "All-or-nothing encryption and the package transform." Fast Software Encryption '97, Springer-Verlag (1997).

[8] C. Shannon, "Communication theory of secrecy systems." Bell Systems Technical Journal, 28(4), 656–715 (1949).