The Commercial Data Masking Facility (CDMF) data privacy algorithm

by D. B. Johnson
S. M. Matyas
A. V. Le
J. D. Wilkins

The Commercial Data Masking Facility (CDMF) algorithm defines a scrambling technique for data confidentiality that uses the Data Encryption Algorithm (DEA) as the underlying cryptographic algorithm, but weakens the overall cryptographic operation by defining a key-generation method that produces an effective 40-bit DEA key instead of the 56 bits required by the full-strength DEA. In general, products implementing the CDMF algorithm in an appropriate manner may be freely exported from the USA. The algorithm is thus intended as a drop-in replacement for the DEA in cryptographic products. Discussed in this paper are the design requirements, rationale, strength, and applications of the CDMF algorithm.

Introduction

A group of cryptographers at IBM designed the cryptographic algorithm now known as the USA Federal Data Encryption Standard (DES) [2], or the ANSI Data Encryption Algorithm [3]. The algorithm is a symmetric block cipher which uses a 64-bit key to encrypt a 64-bit input plaintext to produce a 64-bit output ciphertext. The 64-bit key contains 56 independent key bits which determine the exact cryptographic transformation and 8 bits which may be used as parity bits. The definition of the DEA has been public knowledge since 1977 and has undergone extensive public scrutiny. Its design has been extensively described, for example, in [4]. Currently, it is the most widely used commercial cryptographic algorithm. Its applications include protecting the privacy and integrity of a wide variety of information assets, including electronic funds transfers (EFTs) and the personal identification numbers (PINs) of automatic teller machines (ATMs).

RSA Data Security, Inc. has recently begun licensing code for a symmetric 8-byte block cipher algorithm, designated as RC2, and a symmetric stream cipher algorithm, designated as RC4 [5, 6]. The RC2 algorithm is characterized as suitable for use as a "drop-in" replacement for the DEA. RSA Data Security, Inc.

This paper is an expanded version of one presented in 1993 at the First ACM Conference on Communications and Computer Security [1]; portions are reproduced, with permission. Additionally, there is a pending patent application covering its subject matter; licensing information can be obtained from the Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, Connecticut 06904.

©Copyright 1994 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

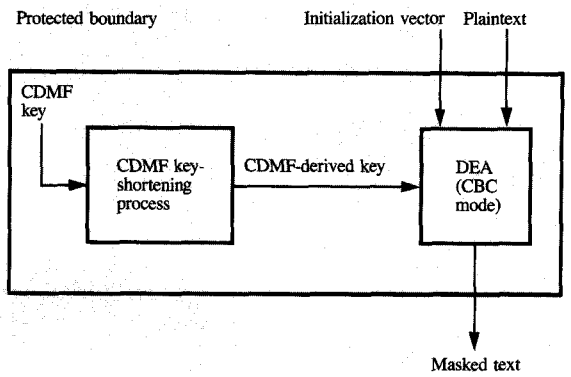

Figure 1

CDMF algorithm high-level process flow.

states that both cipher algorithms are immune to known cryptographic attacks [5]. However, public evaluation of the claimed strength of the algorithms is hampered because the algorithms are proprietary and their details have not been publicly disclosed. The RC2 and RC4 algorithms have a variable key size. When the keys are limited to 40 bits, products implementing these algorithms are generally exportable under U.S. Department of Commerce jurisdiction. See the section on advantages of the CDMF algorithm for a discussion comparing the CDMF algorithm and the RC2 algorithm.

Although widely implemented and a *de facto* international standard, the DEA is subject to government regulations limiting the foreign destinations to which DEA-based data privacy products can be shipped. Before the development of the CDMF algorithm, many users had no way to meet their increasing need for a publicly disclosed method to protect the privacy of information assets on communication lines. For example, see the paper by Higgins and Mashayeki [7] in which they see the need for a method of weakening the Data Encryption Algorithm. Offering the CDMF algorithm with a stated strength equivalent to 40 independent DEA-key bits meets the security requirements of many users. Products implementing the CDMF algorithm in an appropriate manner, although subject to U.S. Department of State jurisdiction, may be freely exported to any customer in most countries of the world.[1]

[1] A note on terminology: In IBM products, the use of the term *encryption* is reserved for strong algorithms, such as the DEA. Since the CDMF algorithm is not as resistant to key exhaustion as the DEA, we do not claim that the CDMF algorithm provides a form of data encryption, but rather that it provides a form of data hiding or data masking.

Design requirements

The design requirements for the CDMF algorithm are as follows:

1. It must provide data privacy protection.
2. In general, products appropriately implementing it must be freely exportable from the U.S.
3. It must have an easily understood strength against key exhaustion and must be extensively scrutinized and analyzed for possible weaknesses.
4. It must be capable of implementation on a wide range of products.
5. It must be possible to define its cryptographic services as an extension to the IBM Common Cryptographic Architecture (CCA).
6. Its external key length must be 64 bits, permitting its keys to be generated and distributed using existing DEA key generation and distribution methods.
7. It must be possible for a system to implement it and the DEA without undesirable side effects.

IBM's committed direction for cryptography and security is described in publications on the DEA-based Common Cryptographic Architecture [8, 9] and the RSA-based public key algorithm extension to the Common Cryptographic Architecture [10], which are significant parts of the IBM Security Architecture [11].

Implementations of the DEA and the CDMF algorithm are not necessarily mutually exclusive. For example, a financial institution could use the DEA when communicating with other financial institutions and use the CDMF algorithm when communicating with a manufacturer.

Algorithm definition

Since the CDMF algorithm can be considered a modification of the DEA, it was initially designated as green–DEA, DEA–light, DEA–junior, etc. To avoid the proliferation of designations, a single descriptive designation was needed. Since the term "encryption" implies a certain level of strength, for example, when used in "Data Encryption Algorithm," the term "masking" was chosen to indicate that the method was not as strong as the DEA. To emphasize its applicability to the commercial market, the Commercial Data Masking Facility (CDMF) designation was selected.

The market requirements described above resulted in the design of the algorithm. The high-level process flow for the design is shown in **Figure 1**. The model shows the component of a cryptographic system that provides the data masking and unmasking services to a calling program. The model assumes a protected boundary where intermediate results are not accessible to the caller of the masking or unmasking services. Note that the CDMF key

is assumed to be inside the protected boundary. In an actual product, a CDMF key is passed as input and must be protected when outside the protected boundary, for example, by encrypting it with a master key. For simplicity, the steps to recover the CDMF key are not shown, since means for protecting and recovering keys are well known and are not relevant to this discussion.

To mask data, the caller passes a 64-bit initialization vector and the arbitrary-length input data to the CDMF algorithm, which produces the output masked data under the control of a CDMF key. To unmask data, the caller passes the masked data to the algorithm, which recovers the original data under the control of a CDMF key. The entire masking process is composed of two processes: a "key-shortening" process and a standard DEA encryption process in cipher block chaining (CBC) mode. The initialization vector is Boolean exclusive-ORed with the first 64-bit input plaintext, exactly as called for in the CBC mode specification [12]. The CDMF key is first operated on by the "key-shortening" process to produce a CDMF-derived DEA key. The derived key is then passed directly to the DEA encryption process. Since the CDMF algorithm is intended to provide for the privacy of data, the CBC mode of the DEA is used. However, other modes of DEA encryption could also be used as required. It should also be clear that an implementation need only produce the CDMF-derived key once for each request and, if desired, could store the CDMF-derived key in an internal associative buffer for quick recovery and use by later requests.

A process flow diagram of the CDMF key-shortening process is shown in **Figure 2**. A CDMF key is input at the top of the diagram of the CDMF key-shortening process, and a derived key is the result. There are four subprocesses in the CDMF key-shortening process, each meeting one of four goals. The diagram shows the process flow for a generic solution applicable to the general situation being addressed by the CDMF concept, and also shows the process flow for the specific solution chosen for the CDMF algorithm. The first step, which is carried out for interoperability considerations, is to set every eighth bit to a constant, in this case binary zero. This step may be accomplished using a Boolean AND operation. The second step protects the value of the input CDMF key and is a simple cryptographic one-way function. That step is accomplished by encrypting the output of the first step with an arbitrarily determined constant DEA key K1 and then using a Boolean exclusive-OR operation on the result of the encryption and the result of the first step. The third step effectively shortens the derived key by setting 24 bits of the output of the second step to a constant. It may be accomplished using a Boolean AND operation. The fourth step scatters the value of the derived key throughout the key space of the DEA, thus eliminating the recognizable

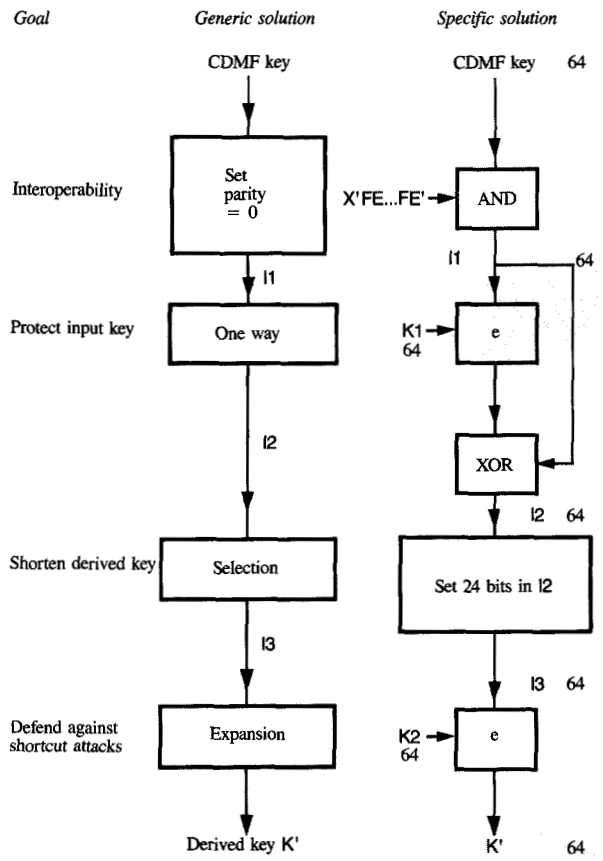

Figure 2

CDMF key-shortening process.

structure of 24 fixed key bits defined in the third step. That step is accomplished by encrypting the output of the third step with an arbitrarily determined constant DEA key K2. The constant key used in the fourth step is, of course, different from the constant key used in the second step.

In the following definition of the CDMF algorithm, all bits in a bit string are numbered from leftmost to rightmost as bit 1 to bit 64, eK(X) represents DEA encryption of X using key K, AND is the bitwise Boolean AND operation, XOR is the bitwise Boolean exclusive-OR operation, and := represents the assignment operation. The procedural definition of the CDMF algorithm is as follows:

1. Set parity bits.

   Zero the following bits in the input CDMF key:
   Bits 8, 16, 24, 32, 40, 48, 56, 64 of input CDMF key are set to zero.
   Call the result I1.

This may be accomplished by the following:
I1 := input-key AND X'FEFEFEFEFEFEFEFE'

2. One-way function.

   I2 := I1 XOR eK1(I1)
   where K2 is the fixed value X'C408B0540BA1E0AE'.

3. Selection function.

   Zero the following bits in I2:
   1, 2, 3, 4, 8, 16, 17, 18, 19, 20, 24, 32, 33, 34, 35, 36, 40, 48, 49, 50, 51, 52, 56, 64.
   Call the result I3.

   This may be accomplished by the following:
   I3 := I2 AND X'0EFE0EFE0EFE0EFE'

4. Expansion function.

   The derived key K' := eK2(I3)
   where K2 is the constant DEA key X'EF2C041CE6382FE6'.

5. Regular DEA invocation.

   The derived key K' is used internally as the key in a DEA invocation.

Design rationale

The CDMF key-shortening process consists of four steps or procedures, as follows: 1) zero parity bits, 2) one-way function, 3) selection function, and 4) expansion function. This section discusses the rationale for each.

Zeroing the parity bits on the input CDMF key helps to ensure interoperability. In the DEA [3], the definition of each eighth bit in the 64-bit key value states that each bit "may be used" as a parity bit, while the other 56 bits define a specific cryptographic transformation. In other words, this definition states that use of the parity bits is optional and implies that one system that conforms to the standard can set and check the parity bits, while another system can ignore the parity bits yet still conform to the standard. This obviously has implications regarding the interoperability between the two different, yet conforming, systems. Such a situation can lead to inconsistent results. (There can be advantages to ignoring parity bits besides the obvious one of improved performance because of less processing of the key value. For example, some applications have taken advantage of systems in which parity bits are ignored by declaring that an arbitrary random number is a key encrypted by another key-encrypting key, thereby eliminating the key-encryption processing step.)

For two systems to interoperate, the same CDMF-derived key must be produced from the same CDMF key everywhere. However, one system may set and test keys for odd parity, while another ignores key parity and sets the parity bits to some arbitrary values. In the CDMF, the CDMF-derived key is a function of all 64 bits of the input CDMF key, including the parity bits. However, only the 56 independent key bits in the input CDMF key, excluding parity bits, are used to determine the value of the CDMF-derived key. By first zeroing the parity bits on the input CDMF key, the algorithm ensures that the CDMF-derived key is the same on all systems, regardless whether parity is set on the input CDMF key.

The rationale for the one-way function involves some cryptanalytic considerations. An early notion was simply to set 16 of the 56 independent key bits in a DEA key to a constant, allowing the other 40 key bits to remain independent and variable. This solution would be acceptable provided that shortened keys and normal DEA keys could not be used interchangeably in a cryptographic system. Otherwise, certain security threats, discussed below, could be a concern. One way to ensure that keys are not used interchangeably is to build a system that uses only shortened keys or uses only DEA keys. However, it was recognized that a system with both DEA and the new algorithm would have to be supported and that such a cryptographic system must provide key separation between DEA keys and shortened keys. It is relatively simple to keep a key for a secret key algorithm, such as the DEA, separate from a key for a public key algorithm, such as RSA, because the natural sizes and components of the keys are very different, making it unlikely that a user would confuse a key used in one algorithm with another. Even if the user did this, the cryptographic system could easily be designed to ensure that such an incorrect usage would not succeed. However, the situation is very different when the keys under consideration have identical sizes and essentially similar components, as is the case with keys for the DEA and the CDMF algorithm, especially since the CDMF algorithm was designed to take advantage of existing DEA key generation and distribution methods. In this situation it was believed that the risk of either inadvertent or deliberate failure to maintain key separation was much higher. Therefore, it was deemed prudent to ensure key separation by imbedding a separation mechanism directly into the new algorithm definition, rather than depend on higher-level mechanisms.

For example, if the new algorithm were to set only 16 bits in the 56 independent bits of the key to a constant, permitting the other 40 key bits to remain variable, the input of a full-strength DEA key to the new algorithm would allow a key partition attack using only a small amount of known plaintext and matching ciphertext. (The details of the CDMF algorithm are assumed to be public knowledge.) First, an adversary passes the full-strength DEA key to the new algorithm and recovers the value of all bits in the key via a key exhaustion attack. This is done by considering the 40 variable bits in the key as a 40-bit

counter and checking all possible values, that is, from all bits being set to zero to all bits being set to one. This process takes a workfactor of 2^{40}, which is about a million million trials. Knowing these bits, the adversary then passes the key to a DEA invocation and exhausts on the remaining 16 bits in the key in a similar fashion. This latter attack has a workfactor of 2^{16}, which is about 64 thousand trials. The sum of these workfactors is still about 2^{40} or about a million million, which is substantially less than the workfactor of 2^{56} or about 72 quadrillion to exhaust a DEA key. Thus, the key partition attack is potentially devastating; the inclusion of a new shortened-key algorithm could possibly allow an attack on the existing DEA.

Another variant of the key partition attack, assuming that the new algorithm sets only some bits on the input key before passing the key to encipher and decipher functions, is that occasionally it would be true that those bits in the key would already be set. In other words, in rare instances (about one in 64000), a full-strength DEA key would conform to the pattern of the shorter new algorithm key. This would be undesirable. Consider the possibility of an insider adversary with access to the cryptographic service interface being able to substitute a known new algorithm key for a DEA key. An unsuspecting user would use this key thinking that it was a full 56-bit-strength DEA key, but the adversary would know that certain bits were set; thus, a key exhaustion attack would be much more feasible. Even more subtly, a user could generate a full 56-bit-strength DEA key, and an adversary could use the new algorithm encipher capability as a filter by determining whether a standard DEA encipher and a new algorithm encipher produced the same ciphertext given the same input. If the results were different, the adversary would know that that DEA key does not conform to the pattern of the alternate algorithm key (this would be the case most of the time). However, if the results were the same, the adversary would know that the apparently full-strength DEA key conformed by chance to the pattern of a new algorithm key; therefore, the key could be attacked by key exhaustion as if it were a new algorithm key.

Since a user could be the generator of the key and the filter processing run by an adversary without knowledge of the user, this attack has many interesting ramifications. For example, an adversary could write some trap code testing many full-length DEA keys to determine whether a full-length key ever conformed to the structure of a shortened key. This trap code could then run as a background task. The small cost of executing the trap code could be considered analogous to the purchase of a lottery ticket, because if the trap code ever detected the right conditions, the result would be like winning a lottery in that a supposedly full-length key could now be broken with much less work.

One way to perceive this problem is that the keyspace for full-strength DEA keys and the keyspace for new algorithm keys are not disjoint. After all, a DEA key or a new algorithm key are both just 64-bit binary numbers. If the values were equal, what could one do to tell them apart?

The solution to this apparent dilemma was to realize the essential asymmetry of the problem: Since a CDMF key has fewer effective key bits than a DEA key, an adversary would like to attack a DEA key through use of the CDMF algorithm, but the reverse is not a concern. Therefore, since the goal was to protect the value of the input key, the use of a cryptographically strong one-way function suggested itself. A strong one-way function is one for which it is easy to compute the output from the input but is considered computationally infeasible to invert; in other words, given a particular output of the one-way function, it is very difficult to find an input that will produce that particular output. Therefore, if the input key were passed through a one-way function, the value of the input key would be difficult for an adversary to determine, even if the adversary knew the value of the CDMF-derived key. A standard cryptographic one-way function is achieved by encrypting the input under a constant key and then exclusive-ORing (XORing) the resulting ciphertext with the input text to produce the result.[2]

The rationale for the selection function was to limit the variability of the resulting value so that it is easy to demonstrate the effective strength of the key both to customers and to organizations responsible for enforcing the regulations on cryptographic products. The selection function sets sixteen bits, which were variable up to this point, to a constant, leaving forty bits of variability.

The rationale for the expansion function was to reduce potential concerns regarding the possibility of the development of a shortcut method of attack because certain key bits are held constant. The term "shortcut" implies an attack that is quicker than exhaustion of the keyspace. While such a shortcut attack is currently unknown, the fear was that sometime in the future someone might be able to reduce the strength of the CDMF algorithm below its declared strength, possibly because of the pattern in the derived key. To mitigate this concern and mask any pattern in the derived key, this step takes advantage of the pseudorandomness property of the DEA and processes the shortened key by encrypting it

[2] It is interesting to examine the history of this one-way function. In 1979, while writing their book [4], Matyas and Meyer discussed different block chaining methods, and this one-way function was filed away, among others, in a "work-to-do" sheet. It was not until 1982 that Meyer recognized the importance and advantage of this particular function and caused it to be incorporated into an IBM cryptographic design proposal. It was subsequently also used as the kernel for the Modification Detection Code (MDC) algorithm [8, 13].

with a constant DEA key to produce the CDMF-derived key passed to the DEA invocation. This spreads the variability of the value produced in the selection step across the key space of the DEA, since a single bit change on the input produces a large difference in the output. This step helps ensure that an attacker is forced to determine the key by exhausting all possible keys.

Security analysis of the algorithm

Some security-related goals of the CDMF algorithm are as follows:

1. Its strength is based on the difficulty of key exhaustion. The strength of a CDMF key is equivalent to 40 independent DEA-key bits.
2. A particular value of a CDMF key results in a pseudorandom selection from one of 2^{40} possible CDMF-derived keys.

By basing the kernel of the algorithm on the DEA, concerns relating to its inherent strength are addressed, since the DEA has proven itself strong against diverse cryptanalytic attacks. It is estimated that designing and validating the DEA (verifying that it had no inherent weaknesses to the best of the designers' knowledge) required about seventeen person-years (and some actual years). The DEA has demonstrated itself to be strong when challenged by many techniques that have defeated other supposedly strong cryptographic algorithms. Given our limited time and resources, it was suggested that the DEA be used as the basis for the new algorithm, with the addition that there be a preprocessor to the DEA process to shorten the effective length of the key.

A CDMF key has a variability of 2^{40}, or about a million million. In analyzing how to exhaust a CDMF key to recover the plaintext from ciphertext, the value of I2 calculated in step 3 of the CDMF algorithm can be considered to be a 40-bit counter, albeit with some extraneous constant bits scattered throughout the 64-bit value. With complete knowledge of the CDMF algorithm, with knowledge that it is being used in a specific situation, and with some small amount of plaintext with its corresponding masked text, testing whether a trial derived key will produce the masked text from the given plaintext takes two encryption steps (one to derive the key from the counter and the other to mask the plaintext) and a comparison. Assuming that the comparison step takes a negligible amount of time and the exhaustion attack utilizes software on a 16-MHZ microprocessor chip which is capable of 7000 DEA encryption steps per second, the expected time to find the derived key (that is, search half the key space) is calculated to be about 4.8 years.

Assuming the use of a very fast DEA chip capable of 4000000 DEA encryption steps per second, the expected time to find the derived key is calculated to be about three days. A study has recently been made by Wiener [13] in which construction is assumed of a special-purpose pipelined DEA chip capable of testing 50000000 DEA keys per second. Assuming the use of such a chip, the expected time to find the derived key is calculated to be about 5.8 hours. The study also proposes that a DES key search machine could be built using these chips for around $1500000. Using such a machine, the expected time to find the derived key is calculated to be about 364 milliseconds.

Hence, the capability of an adversary to find the derived key can be considered a matter of time and money. However, there are actions a user could take to mitigate the risk. For example, a user could perform selective masking (masking only secret data and avoiding masking of information that is not secret), since this would reduce the ability of an adversary to match masked text with its plaintext. The use of an appropriate data compression technique before masking would also complicate the adversary's task. Changing the CDMF key frequently would limit the scope of any exposure.

One type of possible refinement to a key exhaustion attack is to perform a time/memory tradeoff, that is, be willing to use some memory resources in an attempt to reduce the time required. One obvious way to do this is to build a dictionary of all CDMF-derived keys. This would have to be done only once and would eliminate the need for doing the encryption in the expansion step of the CDMF algorithm, thus reducing the number of encryptions needed to one per trial-derived key. Since there are over a million million CDMF-derived keys and each such key is 8 bytes long, such a dictionary would require about 8000 gigabytes (2^{43} bytes) of storage. This might be reduced by sorting and storing only portions of the keys, but the storage needed would still be around 2000 gigabytes. Such an attack is considered unlikely, since the cost in space does not appear worth halving the time.

Without knowledge that the masked text was produced using the CDMF algorithm rather than DEA, but still given the assumption of some masked text with known plaintext, the exhaustion workfactor of a CDMF key is the same as for a DEA key.

Note that if the CDMF algorithm definition is altered to use arbitrary secret values for the keys K1 and K2 used in steps 2 and 4 and the specific bits set to a constant in step 3, the key exhaustion attack mentioned above is not feasible. However, considerations of system interoperability, user acceptability, and Kerckhoff's principle[3] led us to decide on specific constants for these values in the CDMF definition.

[3] Kerckhoff's principle states that the security of a cipher system resides entirely in the secrecy of the key. That is, it is assumed that the adversary knows all the details of the cipher system, including the details of the algorithm, except for the secret key [15].

Note also that the complementary property of the DEA[4] is, in general, not a property of the CDMF algorithm. Since the complementary property of the DEA has been used to reduce the workfactor in certain types of key exhaustion attacks, the absence of this property in the CDMF algorithm is a positive attribute.

Advantages of the algorithm

While comparable implementations of the CDMF algorithm are expected to be slower than implementations of the DEA or of RC2, there are some significant advantages to the CDMF algorithm, as listed below:

1. The primary advantage of the algorithm is that systems can be implemented using it that should, in general, be exportable; yet it is based on the strength of the DEA, which has stood the test of time.
2. The specification of the CDMF algorithm, like the DEA and unlike RC2, is public knowledge and therefore permits open scrutiny and analysis by cryptographers. Thus, the CDMF algorithm is aligned with the emphasis on open systems.
3. A new system that supports only data privacy methods using the CDMF algorithm can be set up so that it is interoperable with an existing system that supports only data privacy methods using the DEA. This can be accomplished by having the new CDMF system generate the keys and apply the key-shortening process (only) to the key that is being sent to the existing DEA-only system. On both systems the cryptographic transformations will then be equivalent, although the values of the keys will be different. Because the key *must* be shortened on the CDMF-based system, there is an inherent asymmetry to this solution. Note that it is possible to prove, in some sense, that a DEA key was actually derived from the CDMF key-shortening process. This may be achieved by performing a DEA decryption with the constant key used in the expansion procedure on the DEA key (as data) and determining whether the result conforms to the pattern for the counter after the selection function.
4. Existing methods of generating pseudorandom 64-bit DEA keys can be used to generate 64-bit CDMF keys.
5. Existing methods of distributing DEA keys can be used to distribute CDMF keys. The key distribution method can make use of DEA key-encrypting keys of either single or double length. This allows the key distribution mechanism to be strong and helps allay concerns that an adversary will be able to break a key-encrypting key, thereby exposing all of the keys encrypted under it. For example, methods found in ANSI X9.17 [15] or the IBM Common Cryptographic Architecture [7] may be used to distribute CDMF keys.

The CDMF algorithm and the IBM Common Cryptographic Architecture

With regard to integrating the CDMF algorithm into existing implementations conforming to the IBM Common Cryptographic Architecture, the following four configurations may be of interest:

1. A CDMF-only system in a homogeneous CDMF-only network.
2. A mixed DEA and CDMF system in a homogeneous DEA and CDMF network.
3. A CDMF-only system in a heterogeneous DEA and CDMF network.
4. A mixed DEA and CDMF system in a heterogeneous DEA and CDMF network.

The first configuration is the easiest to understand. In this case, because there was formerly no existing method to achieve data privacy, the Common Cryptographic Architecture encipher and decipher services must be modified to use (only) the CDMF algorithm. The existing services to generate or install keys into the system would remain unchanged. This system is intended to be exportable.

The second configuration is an extension of the first. In this case, both the DEA and the CDMF algorithms could be selected by the user as the method to use for data privacy. Under current guidelines, this system would not be intended to be freely exportable, but would be available for financial institution use. A financial institution could use the DEA when communicating with other financial institutions and could use the CDMF algorithm when communicating with nonfinancial institutions.

The third configuration assumes the requirement that a user will wish to be able to exchange masked text between a CDMF-only system and a DEA-only system; otherwise, the configuration simplifies to the first one. To meet this requirement, a new service, the CDMF key transform service, is needed to transform a CDMF key into a CDMF-derived DEA key. This service can only be assumed to exist on a new CDMF-only system, since DEA-only systems are assumed to be installed already in an existing network. This system would be intended to be exportable.

The use of this CDMF key transform service requires some intelligence. If both systems are CDMF-only systems or both systems are DEA-only systems, the CDMF key transform service should not be used. If one of the systems is a DEA-only system and the other is a CDMF-

[4] The complementary property of the DEA can be described as follows: Let $e_K(X) = Y$ denote DEA encryption of plaintext X using key K to produce ciphertext Y, and let ' indicate bitwise complementation, that is, inverting the value of each bit in a value. Then the complementary property of the DEA states that if $e_K(X) = Y$, it follows that $e_{K'}(X') = Y'$. Alternately, the complementary property can be stated as $e_K(X) = (e_{K'}(X'))'$; see [4].

only system, the key used on the CDMF-only system must not be transformed, while the key used on the DEA-only system must be processed by the CDMF key transform service. To aid in the determination of what should be done, either the key distribution application will be required to keep track of what is occurring, or enhancements will have to be made to products conforming to the IBM Common Cryptographic Architecture that allow an indication to be kept in the key token for a key-encrypting key (used to encrypt keys between one system and another) regarding whether the system for a particular key-encrypting key is a DEA-only system, a CDMF-only system, or whether the type of system has not been specified. Use of this information would permit the correct decision to be made transparent to the caller of the services.

The fourth configuration is the most complex. One goal of the design was to allow the system administrator to configure a system so that the system would appear to a user as if it were a system in configuration one, two, or three, yet continue to support the ability of a user to specify explicitly what type of processing should be done when that was appropriate. This objective attempts to give the user the best of both alternatives. A user might let the choices default and rational choices would be made "under the covers," but a knowledgeable user could exploit the functionality of the system as desired. This objective is met through the use of the "meta-default" concept whereby the system supports the system administrator specifying the default to be used if the user selects to use the default option. Of course, the system will select the default if the user has not specified which value to use. This chain of default specifications is very flexible in that it supports meeting the user's requirements. Since this system includes DEA-based data privacy, it is not intended to be freely exportable under current guidelines.

CDMF test cases

In addition to the standard suite of test cases used to test the DEA portion of the implementation [17], the following test cases can be used to help ensure, but obviously cannot guarantee, a correct implementation of the CDMF algorithm:

Test 1

clear CDMF key of X'FFFFFFFFFFFFFFFF'.
cleartext of X'0123456789ABCDEF'.
masked text is X'12CC8EE83C686380'.

Test 2

clear CDMF key of X'0000000000000000'.
cleartext of X'0123456789ABCDEF'.
masked text is X'79B1D72AD877D204'.

Test 3

clear CDMF key of X'0123456789103254'.
cleartext of X'0123456789ABCDEF'.
masked text is X'7D74922D74B12E13'.

These test cases may be run to ensure that the masked text is produced from the specified cleartext using the key designated for the CDMF masking. The same test cases may be run using CDMF unmasking to ensure that the cleartext is produced from the masked text in each case.

Concluding remarks

A CDMF version of Advanced Interactive Executive/370™ (AIX/370) code was made available by IBM late in 1992. An IBM satellite one-way broadcast project is currently using the CDMF algorithm, and a CDMF product has been developed in connection with the IBM Transaction Security System (TSS). The need to integrate the algorithm into the IBM Common Cryptographic Architecture has been identified and is being addressed. The CDMF algorithm is currently IBM's strategic exportable data masking method. As the benefits of the algorithm become known, we expect that its implementations will continue to expand.

Acknowledgments

We thank the following individuals for their input and insight into the derivation of the design of the CDMF algorithm: John Hevey (retired) of Secure Systems Strategy, IBM Poughkeepsie, NY; Mike Kelly and Paul Lubold of the Integrated Cryptographic Service Facility/MVS Design group, IBM Kingston, NY; Chris Holloway of the Consultancy Projects and Services group, IBM London; Dr. Don Coppersmith of IBM Research, Yorktown Heights, NY; "Abe" Abraham (retired) of the Transaction Security System Design group, IBM Charlotte, NC; and Russ Prymak (deceased), Bob Elander (retired), and Jim Randall of the Cryptography Center of Competence, IBM Manassas, VA. We also thank Dr. Carl H. Meyer (retired) of IBM Kingston for his incisive review comments.

Advanced Interactive Executive/370 is a trademark of International Business Machines Corporation.

References

1. Don B. Johnson, Stephen M. Matyas, An V. Le, and John D. Wilkins, "Design of the Commercial Data Masking Facility Data Privacy Algorithm," *Proceedings of the First ACM Conference on Communications and Computer Security*, ACM Press, Fairfax, VA, 1993.
2. FIPS 46-1, *Data Encryption Standard*, (USA) National Institute of Standards and Technology, Washington, DC, 1987.

3. American National Standard X3.92-1981, *Data Encryption Algorithm*, American National Standards Institute, New York, 1981.
4. Carl H. Meyer and Stephen M. Matyas, "Cryptography—A New Dimension in Computer Data Security," John Wiley & Sons, Inc., New York, 1982.
5. K. R. Stammberger, "The RC2 and RC4 Exportable Encryption Algorithms," RSA Data Security, Inc., 10 Twin Dolphin Dr., Redwood City, CA 94065, September 14, 1992.
6. G. H. Anthens, "Encrypted Data at Risk?," *Computerworld* **26,** No. 50, 6 (December 14, 1992).
7. J. Higgins and C. Mashayeki, "Provably Weak Cryptographic Systems," *Information Systems Security, Building Blocks to the Future (Proceedings of the 15th National Computer Security Conference),* National Institute of Standards and Technology, Gaithersburg, MD, 1993, pp. 523–530.
8. *Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference*, Order No. SC40-1675, 1990; available through IBM branch offices.
9. D. B. Johnson, G. M. Dolan, K. J. Kelly, A. V. Le, and S. M. Matyas, "Common Cryptographic Architecture Cryptographic Application Programming Interface," *IBM Syst. J.* **30,** No. 2, 130–150 (1991).
10. *Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference—Public Key Algorithm*, Order No. SC40-1676, 1993; available through IBM branch offices.
11. *IBM Security Architecture*, Order No. SC28-8135, 1993; available through IBM branch offices.
12. American National Standard X3.106-1983, *Modes of Encryption of the Data Encryption Algorithm*, American National Standards Institute, New York, 1983.
13. B. Brachtl, D. Coppersmith, M. M. Hyden, S. M. Matyas, C. H. Meyer, J. Oseas, S. Pilpel, and M. Schilling, "Data Authentication Using Modification Detection Codes Based on a Public One Way Encryption Function," U.S. Patent 4,908,861, March 13, 1990.
14. Michael J. Wiener, "Efficient DES Key Search," *Advances in Cryptology—CRYPTO '93* (Thirteenth Annual Cryptology Conference), Santa Barbara, CA, August 22–26, 1993, International Association for Cryptologic Research (paper distributed at conference).
15. *Contemporary Cryptology: The Science of Information Integrity*, Gustavus J. Simmons, Ed., ISBN 0-87942-277-7, IEEE Press, New York, 1992, p. 4.
16. American National Standard X9.17-1985, *Financial Institution Key Management (Wholesale)*, American National Standards Institute, New York, 1985.
17. NBS Special Publication 500-61, *Maintenance Testing for the Data Encryption Standard*, (USA) National Institute of Standards and Technology, Washington, DC.

Received March 26, 1993; accepted for publication October 25, 1993

Don B. Johnson *IBM Federal Systems Company, 9500 Godwin Drive, Manassas, Virginia 22110 (dbj@vnet.ibm.com).* Mr. Johnson received a B.A. in mathematics from Oakland University, Rochester, Michigan, and an M.S. in computer science from Union College, Schenectady, New York. He is one of the designers of the DEA-based Common Cryptographic Architecture and the public key extension to that architecture. Mr. Johnson has received an IBM Outstanding Innovation Award for his contributions to the Common Cryptographic Architecture and an IBM FSC President's Patent Award for two "one-rated" patents dealing with public key cryptography. He holds fifteen patents dealing with cryptography and has achieved the seventh plateau in the IBM Invention Achievement Award program. He is a co-author of 14 published papers, has achieved the third plateau in the IBM Manassas Author Recognition program, and has won an award for an article being chosen as one of the best from IBM Manassas in 1991. Currently, Mr. Johnson is an Advisory Programmer in the Secure Products and Systems Department at the IBM FSC Manassas facility.

Stephen M. Matyas *IBM Federal Systems Company, 9500 Godwin Drive, Manassas, Virginia 22110 (SMATYAS at MANVM2).* Dr. Matyas is a former member of the Cryptography Competency Center at the IBM Kingston Development Laboratory. He is currently manager of the Secure Products and Systems Department at the IBM FSC Manassas facility. He has participated in the design and development of all major IBM cryptographic products, has played a leading role in the design of the IBM Common Cryptographic Architecture, and originated the control vector concept incorporated in the Common Cryptographic Architecture. Dr. Matyas is currently an IBM Senior Technical Staff Member. He holds 35 patents and has published numerous papers covering aspects of cryptographic system design. He is the co-author of an award-winning book entitled *Cryptography—A New Dimension in Computer Data Security*, and is a contributing author to the *Encyclopedia of Science and Technology* and to *Telecommunications in the U.S.—Trends and Policies*. Dr. Matyas received a B.S. degree in mathematics from Western Michigan University and a Ph.D. degree in computer science from the University of Iowa. He received an IBM Outstanding Innovation Award and an IBM FSC President's Patent Award; he has achieved the nineteenth plateau in the IBM Invention Achievement Award program.

An V. Le *IBM Federal Systems Company, 9500 Godwin Drive, Manassas, Virginia 22110 (ANLE at MANVM2).* Mr. Le is an Advisory Engineer in the Cryptographic Center of Competence in the IBM Manassas Laboratory. He received a master's degree in electrical engineering from the University of Utah, Salt Lake City, in 1982. Mr. Le joined IBM in 1983 at Boca Raton, Florida, where he worked as a computer designer. In 1986, he transferred to the Cryptographic Center of Competence in Manassas, and has since been working in the areas of cryptographic algorithms and architectures. He is a co-developer of the Common Cryptographic Architecture (CCA) implemented in several IBM products. He is also a key designer of the Public Key Extension to the CCA implemented in the IBM Transaction Security System. Mr. Le holds fourteen issued patents, several patents on file, and has published more than twenty technical papers and technical disclosures in the area of computer design and cryptography. He has received seven IBM Invention Achievement Awards, several informal awards, three IBM FSC Manassas Publication Plateau Awards, and an IBM FSC President's Patent Award for two patents on key management for public-key cryptography.

John D. Wilkins *IBM Federal Systems Company, 9500 Godwin Drive, Manassas, Virginia 22110 (jwilkins@vnet.ibm.com).* Mr. Wilkins is an Advisory Programmer in the Secure Products and Systems Department in the IBM FSC Manassas facility. He received a B.S. degree in mathematics and computer science from Bucknell University in 1980. Prior to joining IBM in 1986, Mr. Wilkins was a lead software engineer on a prominent national intelligence program. He holds various patents in the field of cryptography, has achieved the first plateau in the FSC Manassas Author Recognition program, and has received an IBM FSC President's Patent Award for two patents related to public key cryptography.