# Processing encrypted data — Source link

Niv Ahituv, Yeheskel Lapid, Seev Neumann

**Institutions:** Claremont Graduate University, Tel Aviv University

**Published on:** 01 Sep 1987 - Communications of The ACM (ACM)

**Topics:** Bus encryption and Encryption

*Management of Computing*

*Gordon Davis Editor*

# Processing Encrypted Data

NIV AHITUV, YEHESKEL LAPID, and SEEV NEUMANN

*ABSTRACT: A severe problem in the processing of encrypted data is that very often, in order to perform arithmetic operations on the data, one has to convert the data back to its nonencrypted origin before performing the required operations. This paper addresses the issue of processing data that have been encrypted while the data are in an encrypted mode. It develops a new approach for encryption models that can facilitate the processing of such data. The advantages of this approach are reviewed, and a basic algorithm is developed to prove the feasibility of the approach.*

## 1. INTRODUCTION

A severe problem associated with commercial encrypted data-processing systems is the difficulty of securing data while the data are being processed by the computing system [2–5]. This paper addresses this problem by developing a method that facilitates the processing of encrypted data without having to first decipher the data.

One of the weaknesses of an encrypted information system is that while being processed the classified data are in a plain (red[1]) state; that is, the data are deciphered before processing. At this state the data are highly exposed to unauthorized intrusion. Although there are some processes that do not require decipherment, particularly operations based on precise comparison and search, there is currently no secure system for processing encrypted data [2, 3, 5]. Other scholars [6] suggest the use of an algorithm based on homomorphic functions for processing encrypted data. Such functions enable the processing of the operation in an encrypted way (and even the decryption of the result), but are lacking in terms of the encryption strength (see Section 4).

The Department of Defense has invested considerable efforts in recent years in solving this problem by developing an operating system that could protect classified data that are being processed, and that could simultaneously process classified and unclassified jobs [4], but the results of these efforts have not yet been made public.

There is a high value to encrypted processing in classified and encrypted information systems. This capability provides such systems with the following significant advantages:

(1) strengthening of data security,
(2) considerable savings in computer time, and
(3) savings in the costs of handling part of the security problems of the operating system.

The following sections present a group of algorithms that facilitate processing of encrypted data. The proposed system will be presented in four steps; in each step a specific requirement is defined, and then an algorithm is developed to meet the requirement. As examples we will use banking transactions where the processing operation is an additive one, for example, updating the balance of a certain account by subtraction or addition.

The following notation will be used [1, p. 7]: *Plaintext* is the set of data before encryption. *Ciphertext* is the result of encryption.

The result of applying decryption to the ciphertext is to restore the plaintext. The terms *plaintext* and *ciphertext* are relative to a particular encipherment—it could easily happen that the plaintext is the result of a previous encryption, but for our purpose, since it enters into the encryption function, it is called the plaintext.

## 2. STEP A

### 2.1 Requirement and Analysis

The requirement is to design a system that can add an encrypted data element (last balance) to a plaintext (the updating transaction) without having to decipher the encrypted balance. In order to meet this requirement,

[1] Data are defined as "red" if access to the data by an unauthorized party can cause damage. Red data can be classified data that have not been encrypted, or classified data that have been ciphered and then deciphered.

we need an encryption function that takes a ciphertext ($C_1$), adds to it a plaintext ($P_2$), and produces a result that is identical to applying an encryption on the two plaintext items ($P_1$ and $P_2$). Alternatively, if we apply a decryption function on the sum ($C_1 + P_2$), we will get a plaintext result identical to deciphering the ciphertext corresponding to the plaintext sum ($P_1 + P_2$).

### 2.2 Development

Assume an encryption function $f_k$ and two plaintext items $P_1$ and $P_2$ (where $P_2$ updates $P_1$), so that $C_1 = f_k(P_1)$ is the ciphertext of $P_1$. To obey the requirement of Step A, the system must fulfill

$$f_k^{-1}[f_k(P_1) + (P_2)] = f_k^{-1}[f_k(P_1 + P_2)] = P_1 + P_2$$

or

$$f_k^{-1}[C_1 + C_2] = P_1 + P_2.$$

In order to satisfy this condition, the algorithm must satisfy

$$f_k(P_1, P_2) = f_k(P_1) + P_2 = C_1 + P_2$$

and symmetrically

$$f_k(P_1, P_2) = P_1 + f_k(P_2) = P_1 + C_2.$$

For example, if the encryption function $f_k$ employs a key and depends on $Z(k)$ so that

$$f_k(P) = Z(k) \times P,$$

we will obtain the requirement to fulfill

$$Z(k)[P_1 + P_2] = Z(k) \times P_1 + P_2 = P_1 + Z(k) \times P_2.$$

### 2.3 Description of the Algorithm

The proposed algorithm will perform the encryption by an addition operation between the keyword and the plaintext word. The operation of addition can be a "modulu 2 addition" or any other method used by the system, provided it uses a word size in a modulu $L$ that will prevent loss of data due to overflow. The first addition of the key to the plaintext amount ($P_1$) gives the first ciphertext balance ($C_1$). Thereafter, plaintext amounts ($P_2, P_3, \ldots, P_n$) can be added. Whenever the updated balance is needed, the decryption involves only the "subtraction" of the key from the ciphertext balance.

We use the following denotations:

| | |
|---|---|
| $X_i$ | key, |
| $Z_i$ | ciphertext update/word, |
| $P_i$ | plaintext amount, and |
| $C_i$ | previous ciphertext balance. |

Where $P_1$ the $n$th update of the system $[P_1 = P_i(n)]$, then the resulting word was

$$Z_i(n) = [x_i + P_i(n)] \bmod 2^L.$$

For the sake of simplicity, we will hereafter omit the modulu notation.

After the next update $[P_2 = P_i(n + 1)]$, the resulting word is

$$Z_i(n + 1) = [Z_i(n) + P_i(n + 1)]$$

or

$$Z_i(n + 1) = [Z_i(n) + P_2].$$

To decipher and get the current balance, we can at any stage subtract the key; for example, after $J$ updates we will perform

$$P_i(J) = Z_i(J) - X_i.$$

### 2.4 Limitations

The proposed algorithm has the following limitations:

(1) A one-time-only breaking of the key (e.g., by a plaintext–ciphertext pair) enables the decryption of all the data at any time.
(2) The periodic updates that are done in plaintext may increasingly provide information to a potential enemy.
(3) If ever the balance is discovered in an intermediary state (even without discovering the key), it is possible to trace all the updates.
(4) In order to perform a correct decryption, it is necessary to distinguish between data entering the system as plaintext and data entered as ciphertext: In other words, data have to be clearly marked.

## 3. STEP B

### 3.1 Requirement and Analysis

The second requirement is to design a system that can add encrypted data to other encrypted data without having to decipher both groups before the addition. In such a system, updates that have been encrypted at the source can be transmitted via terminals as ciphertext, so that the central computer will not unnecessarily store classified red data.

In order to satisfy this requirement, we seek an encryption function such that, if we add an encrypted data element $C_2$ to an encrypted element $C_1$, the result will be identical to applying the encryption function on the two data elements added in plaintext ($P_1 + P_2$). Alternatively, employing a decryption function on the encrypted result ($C_1 + C_2$) will generate the plaintext result identical to the one received had we predeciphered data that were encrypted from the two individual data elements.

### 3.2 Development

Similarly to Section 2.2, the system must fulfill

$$f_k^{-1}[f_k(P_1) + f_k(P_2)] = f_k^{-1}[f_k(P_1 + P_2)] = P_1 + P_2.$$

In order to satisfy this condition, the algorithm must perform the operation

$$f_k(P_1, P_2) = f_k(P_1) + f_k(P_2) = C_1 + C_2.$$

For the encryption function that employs a key, we will get

$$Z(k)[P_1 + P_2] + Z(k) \times P_1 + Z(k) \times P_2.$$

It can immediately be observed that the algorithm selected in Step A does not exactly satisfy the requirement of Step B, since each successive encryption keeps adding the value of the key; that is, the value accumulates with each update. This problem can be solved by recording the number of updates performed ($n$), and then during decryption, subtracting from the encrypted data not the key, but $n$ times the key:

$$P_i(n) = Z_i(n) - n \times X_i,$$

where $P_i(n)$ is the last updated balance.

### 3.3 Limitations

The limitation of plaintext updates ((2) in Section 2.4) has been (at least partially) solved, but a one-time discovery of the key (or of a plaintext–ciphertext pair) can still compromise the system. This means that it is still essential to frequently replace the key, not only for new data files, but also for the currently stored data.

## 4. STEP C: EXAMINATION OF AN APPROACH BASED ON A HOMOMORPHIC FUNCTION [6]

### 4.1 Requirement and Analysis

The third requirement is the same as in Step B, but the system now has to overcome the problem of one-time breaking of the key. In order to fulfill this requirement, we will part with the algorithms presented above and examine the use of homomorphic functions that satisfy the required conditions. These are linear functions that fulfill the additivity property and comply with the following rules:

$$f_k(A + B) = F_k(A) + F_k(B)$$

$$f_k(a \times A) = f_k(A) \times a = a \times f_k(A).$$

We will illustrate the idea of using such functions by enciphering blocks of 64 bits. Let

$\vec{P}$ be a plaintext vector (data block) comprised of bits of plaintext, and
$\vec{P} = (p_1, p_2, \ldots, p_{64})$,

where $p_i$ denotes the $i$th bit of the vector $\vec{P}$. Implementing the linear function $f^k$ will generate a block that is the vector $\vec{C}$, where

$$f_k(\vec{P}) = \vec{C} = (c_1, c_2, \ldots, c_{64}),$$

where $c_i$ denotes the $i$th bit of the vector $\vec{C}$. We will express the properties of the function with the notations defined above and obtain

$$f_k(\vec{P}_1 + \vec{P}_2) = f_k(\vec{P}_1) + f_k(\vec{P}_2) = \vec{C}_1 + \vec{C}_2$$

$$f_k(a \times \vec{P}) = a \times f_k(\vec{P}).$$

An updating process will be implemented as follows: Assume that vector $\vec{C}_1$ is updated by the updating vector $\vec{P}_2$:

(1) $\vec{P}_2$ is enciphered and results in $\vec{C}_2$: $f_k(\vec{P}_2) = f_k(0, 0 \cdots |$ updating section $| \cdots 0, 0) = \vec{C}_2$.

(2) The resulting $\vec{C}_2$ is added to $\vec{C}_1$, and the update process is completed. The normal decryption is performed by an inverse process.

### 4.2 Limitations

The main limitation of using homomorphic functions (a limitation not noted in [6]) is that such functions can, in principle, be broken relatively easily. In the example that we have used, the system can be broken by a dictionary of 64 words and by solving a set of linear equations of the same order. The dictionary will be comprised of 64 encrypted blocks (64-bit words), where in each block all bits but one are 0:

$$f_{k_1}(1, 0, 0 \cdots 0, 0) = a_1$$

$$f_{k_2}(0, 1, 0 \cdots 0, 0) = a_2$$

$$\vdots$$

$$f_{k_{64}}(0, 0, 0 \cdots 0, 1) = a_{64}.$$

Each encrypted block can be practically defined as a sum of values, according to the "1" bits in it. Assume that the enciphered block that we want to decipher is

$$f_k(\vec{P}_1) = (0, 1, 1, 0, 0, 1, 0 \cdots 0) = M_i.$$

It is easily verified that the blocks 2, 3, and 6 took part in constructing the encrypted block; therefore we can write the equation

$$a_2 + a_3 + a_6 = M_i$$

and similarly for all blocks. In this form, by defining a set of linear equations and their solution, the system can be broken. This is the reason why the use of homomorphic functions for encryption is not recommended.

## 5. STEP D

### 5.1 Requirement and Analysis

The last requirement is the same as those in Steps B and C, but we add an "acceptable" level of encryption strength. The idea is based on the function presented in Step B; however, the updating process will be implemented so that each update will be performed by a different key (a different section of the key). In this way, if a certain key is broken, it will perhaps provide the enemy with one-time, temporary information, but not with the ability to trace all the updates and decipher the final information.

### 5.2 Development

For illustrative purposes, we will develop a system of two keys (for one update). The system meets the following conditions:

(1) $f_{k_1,k_2}(P_1 + P_2) = f_{k_1}(P_1) + f_{k_2}(P_2) = C_1 + C_2$; and
(2) $Z(k_1, k_2)[P_1, P_2] = Z(k_1) \times P_1 + Z(k_2) \times P_2$.

The decryption will be performed by

$$\begin{aligned} &f^{-1}_{k_1,k_2}[f_{k_1}(P_1) + f_{k_2}(P_2)] \\ &\quad = f^{-1}_{k_1,k_2}[f_{k_1,k_2}(P_1 + P_2)] \\ &\quad = f^{-1}_{k_1}[f_{k_1}(P_1)] + f^{-1}_{k_2}[f_{k_2}(P_2)] \\ &\quad = P_1 + P_2. \end{aligned}$$

Assume a system where $(n - 1)$ updates have already been processed and are contained in $Z_1$ (which is the last updated plaintext $P_1$ in its encrypted form). The $n$th update, recorded as plaintext $P_2$, arrives now and appears in the form of $Z_2$ (after being enciphered at the location where the transaction took place or elsewhere). In order to perform the update (of $P_2$ into $P_1$), the following operations have to be carried out:

(1) $Z_3 = Z_1 + Z_2$: construction of a new block, which is the sum of the previous blocks.
(2) $X_3 = X_1(k_1) + X_2(k_2)$: construction of a new block.
(3) In order to decipher the ciphertext, we have to supply $X_3$ only and perform $P_3 = Z_3 - X_3$, where $Z_3$ is the last encrypted, updated text.

Note that to decipher it is not necessary to have all the keys. In the case where the keys are known, we can perform

$$P_3 = Z_3 - X_1(k_1) - X_2(k_2).$$

If a special need arises to increase the system strength above what is suggested here, we can perform a permutation on the blocks by interchanging their relative locations, but this is not essential.

### 5.3 Limitations

(1) There is a need to generate and use a sufficiently long random key. With contemporary technologies, this problem can be easily overcome.
(2) The protocol between the central computer and the terminals becomes more complicated and awkward.
(3) It becomes necessary to organize and handle the relatively complex subject of key management.

## 6. CONCLUSIONS AND IDEAS FOR FURTHER RESEARCH

This paper develops the idea of processing data that are in an enciphered state. The paper then suggests and analyzes the possibilities for using special algorithms that facilitate such front-end processing.

The approach presented in this paper is based on a one-time key as an ideal, or on a pseudorandom key that operates in a stream method and not in a block method. This may indicate the following interface between the proposed approach and the widespread data encryption standard (DES) algorithm: The DES algorithm operates in two possible alternatives certified by the National Bureau of Standards. The first, and more prevalent alternative, is based on the processing of blocks. The proposed approach is not suitable for this alternative. The second alternative can be implemented by a different electronic design of the DES chip or by a software change (the latter has been adequately discussed in the relevant literature). Under this alternative, the DES algorithm starts to operate on streams; thus it is then immediately applicable to the approach proposed in this paper.

The algorithms that are developed in this paper are basic and exploratory. Their exposition is intended to provoke ideas for further research. Some suggestions for future research are as follows:

- greater specification of the practical difficulties in processing encrypted data,
- the actual implementation of algorithm operations in code or in typical I/O sequences,
- identification of increases in complications for the protocol between the central computer and the terminals, and
- identification of increases in complications for the protocol between the central computer and the terminals, and
- the interface between the approach proposed in this paper and public-key cryptography.

A successful and more complete treatment of the ideas presented in this paper and the suggested further research may remedy one of the main trouble spots of every encrypted information system.

**REFERENCES**

1. Davis, D.W., and Price, W.L. *Security for Computer Networks.* Wiley, New York, 1984.
2. Denning, D.E., and Denning, P.J. Data security. *ACM Comput. Surv. 11*, 3 (Sept. 1979), 227–249.
3. Diffie, W. Cryptographic technology: Fifteen years forecast. In *Secure Communications and Asymetric Cryptosystems*, AAAS Selected Symposia, vol. 69, G.J. Simmons, Ed. Westview Press, Boulder, Colo., 1982, pp. 38–57.
4. Jacobson, R.B. Secure operating systems. *Comput. Fraud Secur. Bull.* (Sept. 1982), 6–10.
5. Popek, G.J., and Kline, C.S. Encryption and secure computer networks. *ACM Comput. Surv. 11*, 4 (Dec. 1979), 331–356.
6. Rivest, R.L., Adleman, L., and Dertouzos, M.L. On data banks and privacy homomorphisms. In *Foundation of Secure Computation*, R.A. DeMillo, D.P. Dobkin, A.K. Jones, and R.J. Lipton, Eds. Academic Press, New York, 1982, pp. 169–179.

**CR Categories and Subject Descriptors:** E.3 [**Data**]: Data Encryption—*data encryption standard (DES); public key cryptosystems*
**General Terms:** Algorithms, Security
**Additional Key Words and Phrases:** Data encryption, data security, security

Received 9/85; revised 2/86; accepted 11/86

Authors' Present Addresses: Niv Ahituv, Yeheskel Lapid, and Seev Neumann, Faculty of Management, Graduate School of Business, Tel Aviv University, Ramat-Aviv, Tel Aviv 69978, Israel; Niv Ahituv and Seev Neumann are also at Programs in Information Science, The Claremont Graduate School, 130 East Ninth Street, Claremont, CA 91711-6190.