United States Patent [19]

Earnest

[11] **Patent Number: 4,888,798**

[45] **Date of Patent: Dec. 19, 1989**

[54] **MODULAR SOFTWARE SECURITY**

[75] Inventor: **Lester D. Earnest**, Los Altos Hills, Calif.

[73] Assignee: **OMS, Inc.**, Mobile, Ala.

[21] Appl. No.: **894,085**

[22] Filed: **Aug. 7, 1986**

Related U.S. Application Data

[63] Continuation of Ser. No. 725,254, Apr. 19, 1985, abandoned.

[51] **Int. Cl.**[4] **H04L 9/00**

[52] **U.S. Cl.** **380/4**; 380/25; 340/825.31; 340/825.34

[58] **Field of Search** 128/22.08, 22.09, 22.10, 128/22.14; 380/23–25, 28, 3, 4; 364/200, 900

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,962,539	6/1976	Ehrsam et al.	178/22.07
4,302,810	11/1981	Bouricius et al.	364/200
4,326,098	4/1982	Bouricius et al.	380/25
4,408,203	10/1983	Campbell	178/22.08
4,433,207	2/1984	Best	178/22.09
4,439,830	3/1984	Chuch	364/200
4,446,519	5/1984	Thomas	380/25
4,465,901	8/1984	Best	178/22.08
4,471,163	9/1984	Donald et al.	380/4
4,593,353	6/1986	Pickholtz	380/25
4,652,990	3/1987	Pailen et al.	364/200
4,720,860	1/1988	Weiss	380/23
4,723,284	2/1988	Munck et al.	380/25
4,759,062	7/1988	Traub et al.	380/25

OTHER PUBLICATIONS

Denning, *Cryptography and Data Security*, p. 25; (Addison–Wesley, 1982).

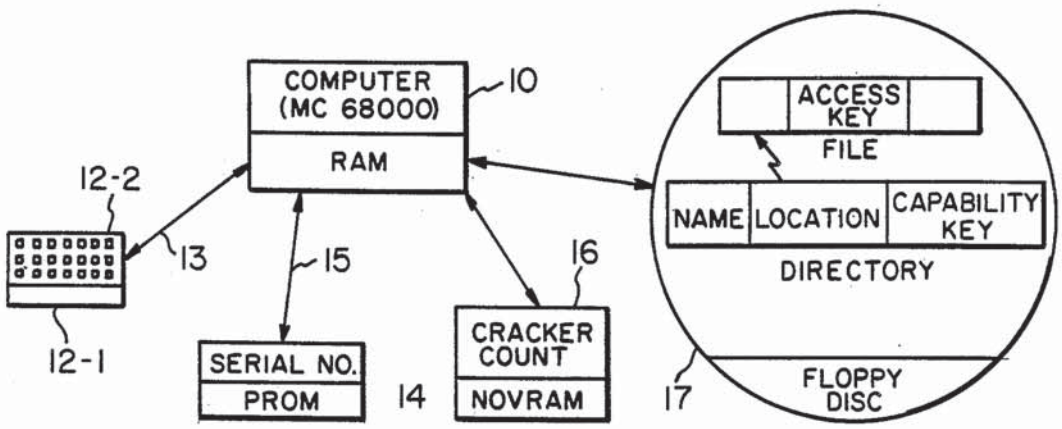
"Cipher Systems" by Beker and Piper, p. 180, 233–235, 1982.

Primary Examiner—Stephen C. Buczinski
Assistant Examiner—Bernarr Earl Gregory
Attorney, Agent, or Firm—Webb, Burden, Ziesenheim & Webb

[57] **ABSTRACT**

Disclosed is a computer method and apparatus that permits identical copies of encrypted computer software (including a number of software elements) to be distributed to many users while retaining central control over which elements are "unlocked", that is, are authorized for use by each user.

13 Claims, 1 Drawing Sheet



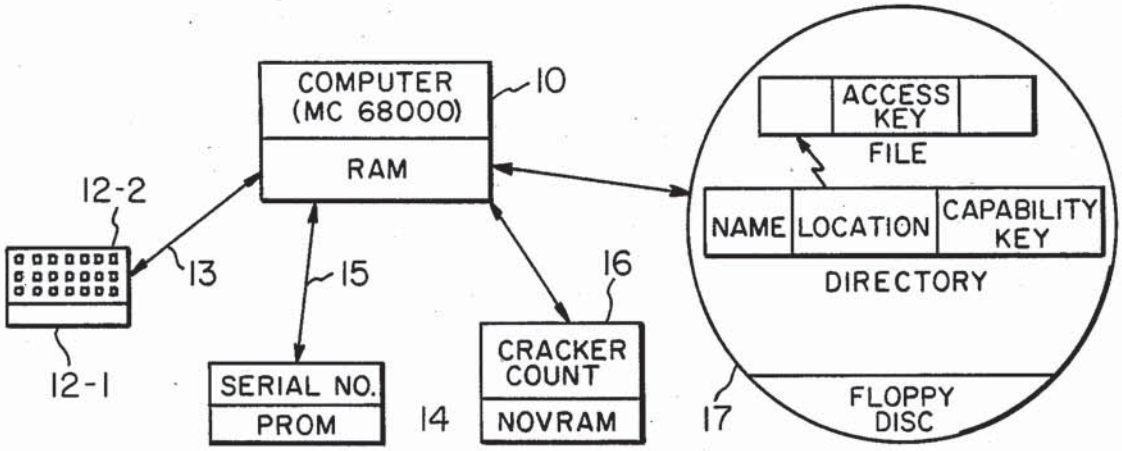


Fig. 1

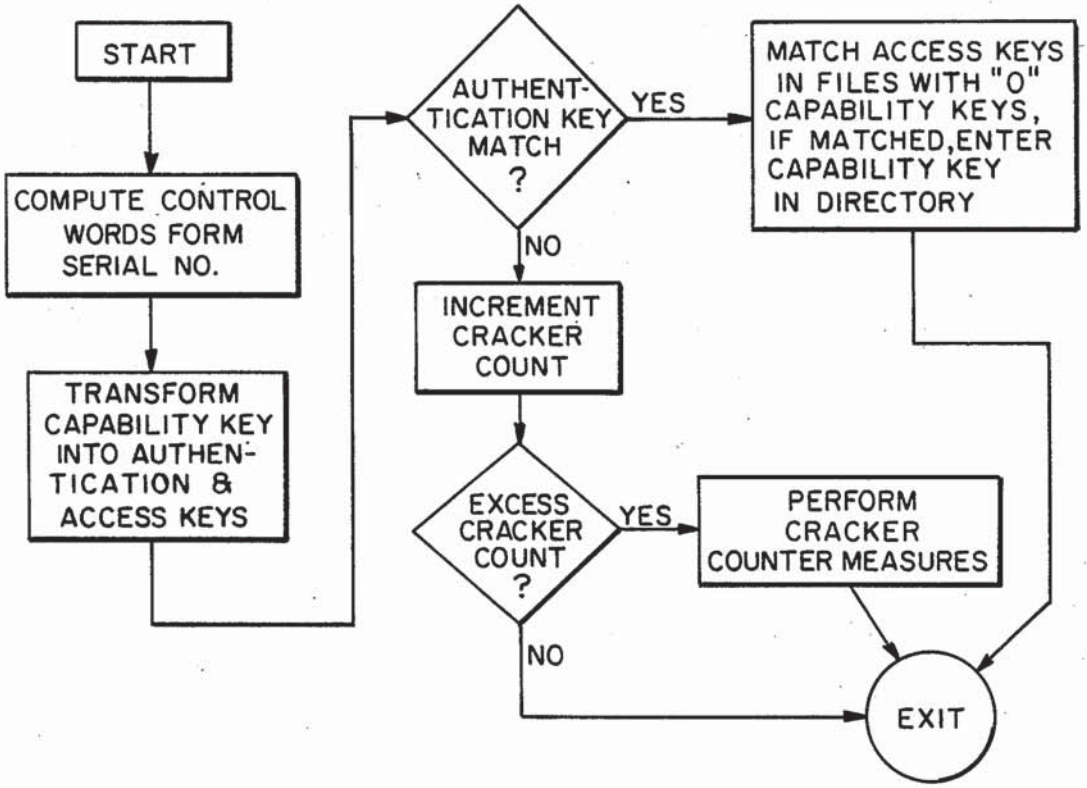


Fig. 2

MODULAR SOFTWARE SECURITY

BACKGROUND OF THE INVENTION

This application is a continuation of application 06/725,254, filed Apr. 19, 1985, now abandoned.

The present invention relates to computers and cryptosystems for use with computer software to prevent unauthorized use of the software on computers.

Frequently, a computer system is capable of receiving software which includes a number of different elements (sometimes called files, functions, modules, capabilities or options). Each element (file) may be segregated from the other elements so that the elements may be priced separately or otherwise separately controlled. The software distributor needs to deliver to each computer user those elements for which the computer user pays or is otherwise authorized. The software distributor wishes to prevent unauthorized users from having access to the elements for which the users have not paid or are otherwise unauthorized. Also, the distributor does not wish the user to be able to copy or otherwise share the software with other unauthorized users.

The ease with which computer software can be copied is both a great operational convenience and a great commercial weakness. Once a copy of software for standard computers has been delivered to a user in some form, it is usually a simple matter for the user to replicate and redistribute the software to other users whether or not the other users are authorized. Legal barriers, such as software licensing agreements and copyrights, offer some protection against unauthorized use, particularly for expensive software that has a limited number of users. Legal barriers offer less practical protection for low-priced software that runs on many machines such as personal computers.

In the environment where the software distributor also controls the design of the computer hardware on which the software will run, it is possible to erect barriers to unauthorized use by using encrypted software. For example, each computer may have an assigned identity key (typically, a unique serial number) that is accessible to the computer program and may have a built-in decryption mechanism which utilizes the identity key as a decryption key. When encrypted software is loaded into the computer, the software is automatically decrypted using the assigned computer identity key as the decryption key. In such a cryptosystem, any attempt to load encrypted software created for a different computer will fail because of the inability to decrypt the software. Key stream crypting in a cryptosystem is executed typically as follows.

Key-stream encryption uses the same process to either encrypt an unencrypted software element (file) or to decrypt an encrypted software element (file) to restore the encrypted element (file) to its original unencrypted form. The source data from the unencrypted software element (file) is treated as a data stream (that is, as a string of bits). Typically, the encryption process performs an "exclusive-or" operation between each bit of the source data stream and a corresponding bit from a key data stream. The key data stream is typically a random or pseudo-random sequence of bits. The result of this encryption operation is an encrypted data stream in which 0's and 1's occur with approximately equal probability.

Decryption of the encrypted data stream is accomplished by performing exactly the same "exclusive-or" operation on the encrypted data stream, using the same key data stream as was used during encryption. The second "exclusive-or" operation utilizing the same key data stream restores the encrypted data stream to its original unencrypted source data stream form. In other words, the "exclusive or" operation is its own inverse. In order for this key-stream cryptosystem to work, it is necessary to use identical key data streams in both the encryption and decryption processes.

By using key-stream crypting on software elements (files) stored on floppy disk or other media, the contents of those elements (files) are rendered unintelligible in the sense that a static analysis yields no information since the stored data looks like random bit patterns.

Pseudo-random number generators can be made to synthesize suitable key data streams. They have the nice property that they can generate a wide variety of such key streams under parametric control. In one system, for example, a simple additive process is used to generate the key-stream. Beginning with a "seed key" (a starting value for the "old key"), a "new key" is generated using 32-bit words by the following calculation of Eq.(1).

$$\text{"new key"} = (\text{"seed key"} + \text{"code"}) \text{ modulo } 32 \qquad \text{Eq.(1)}$$

In Eq.(1), the "code" is a constant in any given software version. By choosing different values of the "seed key" and "code" for different software versions, Eq.(1) generates a series of quite different "new keys".

Many other simple or elaborate key-stream generators are possible and have been described in well-known literature under the category "pseudo-random number generators."

While the above key-stream cryptosystem makes unauthorized use of software difficult, the cryptosystem creates a serious problem for the software distributor since the cryptosystem requires that the software be specifically encrypted for each machine on which it is to be run based on the identity key of that machine. Therefore, the software to be distributed is different for each computer so that the distributor must treat every user's computer differently and such treatment is obviously inefficient and expensive.

In view of the above limitations, there is a need for improved software cryptosystems which do not require a different encrypting of the software for each computer authorized to use the software.

SUMMARY OF THE INVENTION

The present invention is a computer method and apparatus that permits identical copies of encrypted computer software (including a number of software elements) to be distributed to many users while retaining central control over which elements are "unlocked", that is, are authorized for use by each user.

The computer software after distribution to a user is stored within the user's computer and may include both authorized and unauthorized elements. The user may "unlock" any one or more of the authorized elements by entering corresponding encrypted capability keys, usually one key for each authorized element. Each of the capability keys typically is a short string of alphanumeric characters entered through a keyboard or equivalent input device. While the capability keys may be different for each computer, the computer software including the encrypted elements are the same for all

computers so that the software can be copied and disseminated by a distributor without concern about whether or not use of any particular element is authorized.

The user's computer system has interlocks that permit only those elements that have been "unlocked" by entry of a capability key to function in the computer. At the time of the initial delivery of the software, the user is given capability keys for only those elements that the user has purchased or for which the user is otherwise authorized. If the user later obtains additional capability keys corresponding to newly authorized elements, the user is then able to access the corresponding newly authorized elements. Typically, the newly authorized elements are already stored in the user's computer so that there is no need for the user to again receive the newly authorized software elements. The user need only receive the capability keys for those elements.

The computer software is typically distributed in different versions where each version has a different key-stream generator and encrypts software elements differently.

The present invention offers a substantial barrier to casual software piracy while facilitating software updates. Key-stream crypting (encryption and decryption) is used in the security system as a "first line of defense" together with capability keys which provide modular control of access to software elements.

In one embodiment, the number of uses of a software element authorized by a capability key is limited. When the limit for one capability key is reached, a user is required to enter another capability key to obtain authorization for additional uses of the same software element.

In a specific embodiment, the modular software security system permits modular control of access to software elements using specific "keys", including a "capability" key, an "identity" key, and a "stored" key.

The stored key is a key which is identical to the transformed key resulting from the transform of the capability key and the identity key. In one typical system, the stored key has two components, namely, an "authentication" key and an "access" key.

When the user is authorized to use a particular software element, as evidenced by the user having the corresponding capability key, the user enters the capability key into the user's computer system. The user's computer system transforms the capability key using the identity key to form a transformed key. The transformed key is then compared with the stored key and if a match occurs, that match indicates that access to the protected software element is authorized. Each "capability" key is provided to the user by the software distributor and controls access to a particular software element.

The "identity" key is a number for identifying a particular computer, user or other thing. When used to identify a computer, the identity key is permanently stored in the user's computer and is unique to the user's computer and therefore can function to assure that the authorized software element is authorized only for the computer having the proper identity key. The identity key transforms the capability key.

The "authentication" key is an arbitrary number that is compiled into the firmware of the computer system. The authentication key is used to check the validity of capability keys that are entered. All systems that use a given firmware version typically have the same authentication key, but the authentication key may be changed between versions.

Each software element in the system has an associated "access" key which must be matched with a portion of the transformed capability key to obtain permission to access the file. Typically, the access key is interleaved with the file data stored in the software element in a way that depends on the firmware version.

At the time of delivery of the computer system, the system has a stored key including an authentication key (software version dependent) an access key (one for each element), and a machine identity key (different for each computer). When the computer system is started, the system decrypts and downloads the computer software including certain main operating programs from the floppy disk or other program source using the same key-stream decryption process for all software. The computer software on the disk has a directory element that associates software element names with disk addresses and capability keys. Generally, there is one capability key per software element. Each software element stores an access key, generally in a number of distributed locations that are software version-dependent. Any elements with access keys of "0" are considered to be "unprotected" and no capability key is needed to access them.

In its initial configuration (before entry of any capability keys), the capability keys of all elements are set to a pre-establish value, for example, "0". No element with a capability key "0" can be accessed until a non-zero capability key has been computed and stored in the corresponding capability key field of the element directory except for unprotected elements which, as indicated by a "0" access key, require no capability key. Initially, therefore, before entry of any capability key, only the unprotected elements are accessible. Additional software elements having non-zero access keys can be "unlocked" by a capability key unlocking process.

In order to initialize the system and to unlock a protected element using the unlocking process, a capability key for the protected element is entered into the system. The entered capability key is transformed with the system identity key to form a transformed key. The transformed key is compared with the stored key. Typically, a portion of the transformed key is compared with the authentication key stored in firmware to verify the entered capability key's authenticity. If an authentication match occurs, the match resulting from entry of the correct capability key is used to "unlock" protected elements provided also that access is authorized. If an authentication key match does not occur, a special "cracker countermeasure" procedure can be performed.

If the authentication key is matched correctly, each element that has a "0" in its capability key field in the element directory is examined to determine which elements also have an access key matching another portion of the transformed capability key. Wherever an access key match is found, the capability key (not the transformed capability key) is stored into the capability key field of that element in the element directory. When the capability key is thus stored in the directory, the protected element has been unlocked and is available for use. In order to access the protected element, however, the capability key must be entered before each access.

Thereafter, whenever access to a software element is requested, the capability key stored (if any) in the ele-

ment directory for that element is transformed in the same way as when the capability key was entered into the system. Portions of the transformed key derived from the stored capability key are compared with the stored authentication key and the stored access key.

If either the stored authentication key or the stored access key does not match the corresponding portion of the transformed key, the failure to match indicates that access to the element is unauthorized. Access is unauthorized, for example, if the element has been reproduced from a system with a different version of the firmware or if a floppy disk storing the element has been moved from another system to the present system. In either case, the system typically resets all capability key fields in the element directory to "0". The user must then reenter non-zero capability keys into the system in order to gain access to any protected element.

The transformation of the capability key can be done in many different ways. The important criterion is that the capability key transformation be reversible so that, given a stored key (typically including an authentication key and an access key) and a system identity key, a capability key can be computed having the property that, when transformed, it will match these keys.

Additional objects and features of the invention will appear from the following description in which the preferred embodiments of the invention have been set forth in detail in conjunction with the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 depicts a cryptosystem for use in connection with or a computer.

FIG. 2 depicts a flow chart which represents the operation of the FIG. 1 cryptosystem in order to enter a capability key.

DETAILED DESCRIPTION

Encrypted Capability Keys

A practical realization of the modular capability-key cryptosystem embodied in a printing system is shown in FIG. 1. The printing system employs a general purpose processor and a number of different computer software elements which represent different capabilities of the system. Typical hardware elements used in the printing system are as follows.

A general purpose computer **10** with integral random-access memory (RAM) **11** to store computer software (including a number of protected and/or unprotected software elements) and to store data is employed. A Motorola MC68000 microcomputer with 256k bytes of MOS RAM memory is suitable for computer **10**.

An input device **12** for entering an alphanumeric string representing the capability key is employed. A display terminal **12-2** and keyboard **12-1** connected to standard RS232C port **13** of computer **10** is suitable for input device **10**.

A permanent, machine-readable memory device **14** is employed to store the serial number identity key and other firmware. A PROM (programmable read-only memory) physically and electronically attached to a backplane **15** of the computer **10** is suitable for permanent memory device **14**.

A small amount of permanent, alterable memory **16** is employed to keep track of the "cracker count" which is the total number of erroneous capability keys that have been entered. NOVRAM (non-volitile RAM) storage associated with the computer is suitable for memory **16**.

Software file storage **17** is employed for storing software elements in encrypted form. Storage **17** typically includes a floppy disk which stores programs (including a number of protected software elements such as font files), other data, and an element directory in encrypted form.

Key-stream crypting is used to render all software elements stored in storage **17** unintelligible until key-stream decrypting is performed. The key-stream crypting is the same for all software elements of a particular version of the software and is the same for all computers (printers) of a particular release; but the crypting may be different for other versions of the software and other releases of the computer. The function of the key-stream cryptosystem is to make static analysis of software elements difficult, but such a key-stream cryptosystem is not necessary for the "unlocking" of protected software elements by use of capability keys. The unlocking of protected software elements employs capability keys and requires that the capability keys be transformed. One convenient capability-key transformation that will be used in one embodiment of the system is now described.

The computer system's identity key, I_k, (for example, a 32-bit serial number) is read from the system firmware stored in PROM memory **14** of FIG. **1**. The identity key, I_k, is then transformed with the capability key, C_k, entered through keyboard **12** to form a transformed capability key, T_k, as follows:

$$I_k * C_k = T_k \qquad \text{Eq.(2)}$$

In Eq.(2), the symbol "*" designates a reversible transform which typically is performed in a number of steps. First, the serial number identity key, I_k, is transformed by one or more arithmetic or logical operations into two 32-bit control numbers, B_1 and B_2. Typically, these transformations of I_k are version-dependent. For example, the transformations consist of multiplying I_k by two different constants, J_1 and J_2 (where J_1 and J_2 may be different for different versions), as follows:

$$(I_k)(J_1) = B_1 \qquad \text{Eq.(3)}$$

$$(I_k)(J_2) = B_2 \qquad \text{Eq.(4)}$$

The transformation of the capability key, C_k, then further proceeds in two steps. The first control number, B_1, is transformed with the numeric capability key, C_k, to yield another 32-bit number, called the intermediate transform, X_k, as follows:

$$B_1 * C_k = X_k \qquad \text{Eq.(5)}$$

In Eq.(5), the transform is typically an EXCLUSIVE-OR operation as follows:

$$B_1 \oplus C_k = X_k \qquad \text{Eq.(6)}$$

The EXCLUSIVE-OR operation indicated by the symbol "$\oplus$" in Eq.(6) is bit-by-bit between each pair of correspondings bits of B_1 and C_k and yields for corresponding bits of X_k a "1" if the corresponding bits of B_1 and C_k are different and "0" if they are the same.

The second control number, B_2, is used to transform the intermediate transform, X_k, of Eq.(6) to form the transformed capability key, T_k, as follows:

$$B_2 * X_k = T_k \qquad \text{Eq.(7)}$$

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.