セキュアファイルシステムの構築

宮崎 博、　鮫島 吉喜、　遠田 潤一
日立ソフトウェアエンジニアリング（株）

機密情報を含むファイルを不正な閲覧から守るためにはファイルの暗号化が必要である。しかし、既存のファイル暗号ツールは、ユーザがファイルの暗号・復号の操作を明示的に行う必要があるため、ユーザの負担が大きいだけでなく、暗号し忘れの恐れもある。そこで本論文では、ファイル I/O をフィルタリングして動的にファイルを暗号化するフィルタについて報告する。このフィルタの対象となったファイルシステムでは欠けていたアクセス制御の機能も付け加え、セキュアなファイルシステムを構築した。さらに試作したフィルタは、企業での使用も考慮に入れ、暗号化したファイルの緊急アクセス機能もサポートしている。

Implementation of Secure File System

Hiroshi Miyazaki,　Yoshiki Sameshima　and　Junichi Enda
Hitachi Software Engineering Co., Ltd.

File encryption is necessary to prevent files including classified information from being eavesdropped by unauthorized users. But most existing file encryption tools enforce file owners to order encipher or decipher operations of their files whenever they need to do so. These extra manipulations often irritate operators or cause to forget to encrypt important files after the finish of their uses. In order to solve these problems, the authors prototyped a secure file system with file I/O filter, which encrypts files designated by its user, controls file access which its target file system lacks and support emergency access to enciphered files with consideration of enterprise-uses.

1. はじめに

携帯端末の普及により、企業内のユーザが機密データを記憶装置内に保存した端末を持ち運ぶことが多くなると予想される。そのため、端末の盗難、紛失などによるデータ漏洩を防ぐ仕組みが必要となる。また、一台の端末を複数ユーザで共用する場合、端末内に保存したデータを他ユーザからの閲覧や改ざんから守る必要もある。

データ漏洩への対策の一つとしてファイルの暗号化が挙げられる。しかし、暗号化ファイルの編集のたびにファイルを復号してから編集し、保存後に暗号化し直すのでは、ユーザの負担が大きい上にユーザが機密ファイルを暗号化し忘れる恐れがある。これを防止するにはファイル I/O 時に動的に暗号処理を行う必要があるが、新たにファイルシステムを端末内に作成するのでは、企業ユーザにとって導入が困難となってしまう。

また、データの不正な閲覧や改ざんを防止するにはユーザ認証に基づいたアクセス制御が必要となるが、一部の OS ではユーザ認証やアクセス制御の機能が欠けている。そこでこれらの機能を導入する必要があるが、一方で企業ではデータの所有者が不在でも、そのデータに緊急にアクセスしたいという要求もある。

本論文では、以上の問題を解決するために構築したセキュアファイルシステムのプロトタイプについて報告する。本システムは、セキュリティ機能をファイル I/O フィルタとして実装すること

により、既存のファイルシステム上に構築可能となっている。プロトタイプはPC用OSのファイルシステム上に試作している[1]。以下では試作したプロトタイプについて、2章ではファイルI/Oフィルタによる動的な暗号処理であるアプリケーション透過ファイル暗号、3章では同様にフィルタで行われるユーザ及びグループごとのアクセス制御機能、4章では鍵情報の管理、5章では暗号化ファイルの緊急アクセス機能を述べ、6章では暗号化ファイルの入出力速度について簡単に報告する。

2. アプリケーション透過ファイル暗号

2.1 ファイルの暗号方式

ファイル暗号化の指定はユーザがディレクトリごとに行い、(以下､暗号ディレクトリと呼び、指定を行ったユーザは暗号ディレクトリの所有者と呼ぶ)、指定されたディレクトリ以下のすべてのファイルが暗号化される。ファイルI/Oフィルタは、ユーザが操作するアプリケーションからのファイルI/O要求を受け取ると、I/Oの種類、及び操作対象ファイルのパス名とユーザが指定済みの暗号ディレクトリのパス名との比較結果に応じて、暗号処理の有無を決めている。例えば暗号ディレクトリ内のファイルへの書き込みが要求された時は、一緒に渡される書き込みデータをそのファイルの暗号鍵で暗号化してからファイルに保存する。

また、ファイルの名前や更新日時はファイルの内容を類推する際の手がかりとなるため、暗号ディレクトリ以下のすべてのサブディレクトリやファイルの名前、作成日時、更新日時の情報も暗号化し、暗号ディレクトリにアクセス可能なユーザだけが見られるようにする必要があるが、プロトタイプではサポートしていない。

ファイルの暗号化に使用するアルゴリズムにはブロック暗号を採用したが、アプリケーションからのランダムアクセスを考慮して、図1に示すようにファイルを暗号化している。暗号化する際はファイルを一定長ごとのレコードに区切り、それぞれのレコードごとに初期化ベクタを用意してCBCモードで暗号化している。レコードの末端のブロックまで暗号化したら、そこで暗号化を終えている。これにより、ファイル内の任意の位置への書き込みがあっても、毎回ファイルの終端まで再暗号化を行う必要がなくなりオーバヘッドが少なくなる。また、ファイル終端の暗号化の際にはパディングをせずにCFBモードで1バイトずつ暗号化し、ファイルサイズが暗号前と変わらないようにしている。

各レコードごとに使用する初期化ベクタの生成は、ファイルごとに異なる初期化ベクタの種とレコードインデックスを入力とする計算によって行う。暗号鍵はファイル内のすべてのレコードに共通なものを使っている。初期化ベクタの種とファイル暗号鍵はファイル新規作成時にランダムに生成され、ディレクトリごとに用意する鍵ファイル内に保存されている。

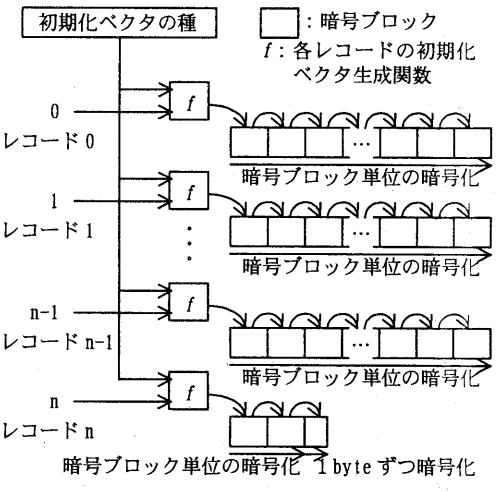

図1 ファイルの暗号方式

2.2 レコードサイズ

レコードサイズはファイルI/O時のパフォーマンスに大きく影響を与えるが、ファイルI/Oの高速化に最適なレコードサイズはアプリケーションからのファイルI/Oデータの大きさや記憶媒体のクラスタサイズなどに依存すると考えられる。

同一のアプリケーションでもI/Oデータのサイズは変わるだろうが、通常のアプリケーションは一定のフォーマットに従ってデータを保存しているので、I/Oデータのサイズには一定の傾向があると推測できる。実際、プロトタイプの試作にあたり観察したところ、ワードプロセッサや表計算のアプリケーションでは256byteやその倍数のI/Oデータサイズが多く見られた。また記憶媒体のクラスタサイズと同じか整数倍のレコードサイズにすれば、本来一つのクラスタ内に納まっていたI/Oが暗号化によって複数のクラスタをまたぐI/Oに変わってオーバヘッドを生じるようなことを防げると考えられる。したがって、レコードサイズを様々に変えてI/O速度を測定し、その最適なサイズを決める必要があるが、現在のところ未調査のままである。

3. アクセス制御

プロトタイプで対象としたPC用OSのユーザ認証は完全なものでなく、認証を回避することが可能である。そのため、プロトタイプにはパスワードによるユーザ認証機能を組み込んである。また、複数のユーザを登録可能であり、グループを定義してユーザをグループ単位で管理することもできる。

また、対象OSのファイルシステムではファイルの属性として読み取り専用の指定が可能であるが、誰でもその属性を変更できてしまう。そこで表1に示すアクセス権を定義し、暗号ディレクトリに設定できるようにした。例えば、ユーザが暗号ディレクトリに“R”（復号読み込み）のアクセス権を持っていると、I/Oフィルタはそのディレクトリ内の暗号化ファイル・サブディレクトリ名を復号して一覧を表示できる。また、暗号化ファイルを復号して読み込むこともできる。なお、“A”（アクセス権変更）のアクセス権を持つことができるのは暗号ディレクトリの所有者だけである。

暗号ディレクトリの所有者は他のユーザやグループにアクセス権を与えることもできる。これにより、暗号ディレクトリの共有を実現している。

4. 暗号鍵と設定情報の管理方法

ファイルの暗号鍵と初期化ベクタはファイルごとに異なるものを使用しており、対応する暗号化ファイルの名前と組にして暗号ディレクトリごとに用意する鍵ファイル内に保存している。保存の際にはファイル暗号鍵を鍵ファイルごとに異なるディレクトリ鍵で暗号化する。ディレクトリ鍵は、暗号ディレクトリの所有者やディレクトリへのアクセス権を与えられたユーザ、またはグループの暗号鍵を使って、与えられたアクセス権と一緒に暗号化して鍵ファイル中に保存する。これにより、ディレクトリへのアクセス権を持つユーザやグループが増えても、鍵ファイルのサイズはそれらの暗号鍵で暗号化されたディレクトリ鍵の分しか増えずにすみ、ファイル鍵の検索にオーバヘッドが生じないようになっている。図2に鍵ファ

表1　アクセス権の定義

アクセス権の種類	記号	アクセス権の定義	
		ディレクトリへのアクセス	ファイルへのアクセス
復号読込み	R	暗号化ファイル・サブディレクトリの名前を復号して一覧表示*	暗号化ファイルの読込み時に復号
暗号書込み	W	暗号化ファイル・サブディレクトリの新規作成・名前変更	暗号化ファイルの書き込み時に暗号化 暗号化ファイルの名前変更
暗号削除	D	暗号化ファイル・サブディレクトリの削除	暗号化ファイル削除
アクセス権変更	A	暗号ディレクトリのアクセス権の変更	―――

(*は未実装)

イル内に保存する各鍵の関係を示してある。図中の緊急アクセス用公開鍵については次章で述べる。なお、ユーザやグループの暗号鍵は別に用意する設定ファイル内に暗号化して保存している。この設定ファイルや鍵ファイルはユーザから見えなくするため、ファイル I/O フィルタがリスティングしないようになっている。

上記のディレクトリ鍵やアクセス権、ファイル鍵などの情報は、暗号化ファイルがオープンされた時にそのファイル名を使って鍵ファイルから検索されて読み出される。ここで、ファイルアクセスを行ったユーザやその所属するグループに対応するディレクトリ鍵の情報がなかったり、要求したファイル I/O を許可するアクセス権が割り当てられていなければ、そのファイル I/O は失敗する。

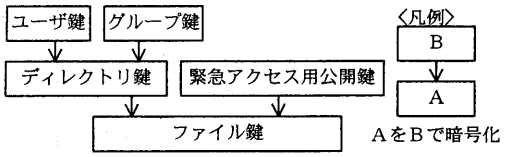

図 2　鍵ファイル内に保存する各鍵の関係

5. 暗号化ファイルへの緊急アクセス

暗号化ファイルへの緊急アクセスは Shamir の方式[2]を変形した (k, n) 閾値復号方式で実現している。この方式では復号鍵の復元の際に特定の誰かを必須にする、あるいは分割鍵の所有者をグループ化してそれぞれのグループ内で協力が成立しないと復元できないといった、柔軟な分割が行えるようになっている。なお、暗号方式には離散対数問題に基づいた公開鍵暗号を用いている。また、秘密鍵の分割には、${}_nC_k$ 個の n+1 変数一次式を利用する[3]。

以下、この (k, n) 閾値復号方式を適用する時の手順を示していく。

5.1 システムの初期化

図 3 に示したシステム初期化の様子に従って手順を説明していく。なお、図で示した例では、管理職 3 人のうち 2 人とシステム管理者 3 人のうち 2 人が協力すれば緊急アクセスを行えるものとしている。

(1)　システムパラメータの生成

まず、システムパラメータとして、素数 p, q (p=2q+1) と巡回群 Z^*_p 上の原始根 g を生成する。

(2)　緊急アクセス用の鍵の生成

次に緊急アクセス用の秘密鍵 x を Z_{p-1} の偶数から選び、公開鍵 $y = g^x \pmod p$ を計算する。

(3)　秘密鍵の分割

続いて、秘密鍵を分割するために、以下のような ${}_nC_k$ 個の n+1 変数一次式を生成して秘密情報 $x' = x/2$ を分割する。

$$a_{i0}x_0 + a_{i1}x_1 + \ldots + a_{in-1}x_{n-1} = x' \pmod q \qquad (0 \cdot I < {}_nC_k)$$

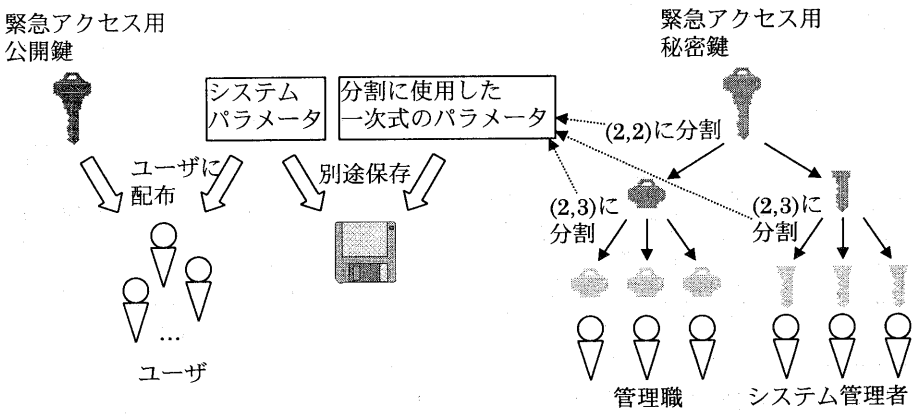

図 3　緊急アクセスの初期化

ただし、各式の $x_j(0<j<n)$ の係数 a_{ij} のうち $n-k$ 個の値は 0 であり、かつ係数が 0 でない x_j の組み合わせは各式において互いに異なるように係数 a_{ij} を選ぶものとする。そして $x_0, x_1, \ldots, x_{k-2}$ をランダムに選ぶと残りの $x_{k-1}, x_k, \ldots, x_{n-1}$ が決まるので、これらの x_j を分割鍵保有者に配布する。分割鍵保有者をグループ化するのであれば、同様の方法で x_j を分割し、グループ内のメンバに配ればよい。

本節の最初に述べた条件で緊急アクセスを行うには、最初に秘密鍵を(2,2)に分割し、さらにそれぞれを(2,3)に分割しなければならない。(2,2)に分割する場合には、例えば次のような ${}_2C_2=1$ 個の 3 変数一次式

$$2x_0 + 3x_1 = x' \pmod{q} \qquad \cdots \quad (A)$$

を生成し、x_0 をランダムに決めて x_1 を求める。さらに x_0 を(2,3)に分割するために次のような一次式

$$\begin{aligned} 1x_{01} + 1x_{02} + 0x_{03} &= x_0 \pmod{q} \\ 2x_{01} + 0x_{02} + 1x_{03} &= x_0 \pmod{q} \qquad \cdots \quad (B) \\ 0x_{01} + 2x_{02} - 1x_{03} &= x_0 \pmod{q} \end{aligned}$$

を生成して、x_{01} をランダムに決めて x_{02}, x_{03} を求め、管理職の 3 人に配る。x_1 についても(2,3)に分割する一次式を生成して 3 つに分割し、システム管理者に配ると、図に示した分割が行える。このような分割により、管理職の人間だけ、あるいはシステム管理者の人間だけで緊急アクセスを行うのを防止している。なお、配布する分割鍵は暗号化し、分割鍵所有者のパスワードでしかアクセスできないようにしている。

秘密鍵の分割に使用した一次式のパラメータは緊急アクセス時に必要となるので、システムパラメータと一緒に別途保存しておく。これらのパラメータを暗号化する必要はないが、改ざんを検知するために署名を入れている。

(4) システムパラメータと公開鍵の配布

システムパラメータ p, q, g と緊急アクセス用公開鍵 y はユーザに配布する。ユーザ側での処理については 5.2 節で述べる。

5.2 ユーザ側の緊急アクセス用の処理

まず、ユーザ側のファイル I/O フィルタでは、暗号化ファイルの新規作成時に Z_{p-1} から乱数 r を選ぶ。同時にファイル鍵 m をランダムに生成し、

$$c_1 = g^r \pmod{p}$$
$$c_2 = my^r \pmod{p}$$

を生成する。これらの c_1, c_2 を緊急アクセス用のデータとして各ディレクトリの鍵ファイル内に保存しておく。

5.3 緊急アクセスの実行

図 4 に示した、暗号化ファイルに緊急アクセスする際の様子に従って手順を説明していく。

まず、緊急アクセスしたい暗号化ファイルに対応する c_1 と c_2 を入手する。

次に緊急アクセス用の秘密鍵 x の分割に用いた一次式のうち、ファイル鍵の復元に用いる分割鍵 x_j の係数がすべて 0 でないものを選び、

$$d_j = c_1^{\wedge}(2a_{ij}x_j \bmod q)$$

を各分割鍵ごとに計算する。続いて、

$$\begin{aligned} c_2/\Pi d_j &= c_2 / c_1^{\wedge}(2\Sigma a_{ij}x_j \bmod q) \\ &= c_2 / c_1^{\wedge}(x) \\ &= m \pmod{p} \end{aligned}$$

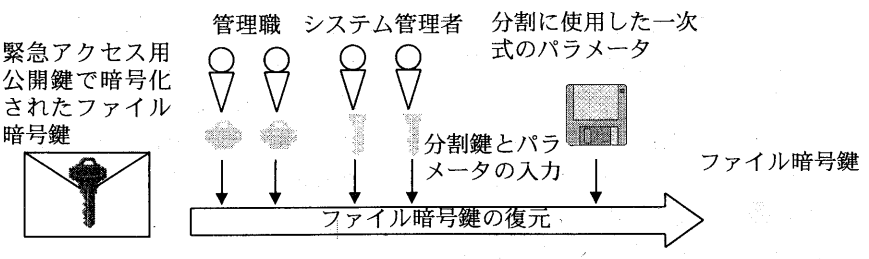

図 4 緊急アクセス用ツール内での動作

を計算してファイル鍵 m を得ることができる。

分割鍵保有者のグループ化のために分割鍵をさらに分割した場合でも、各グループにおいて同様の方法で一次式を選び、最初の分割に使った一次式の係数も利用すると d_j が計算でき、最終的にファイル鍵 m を得られる。

図 4 の例の場合では、管理職とシステム管理者がそれぞれ 2 人づつ集まって暗号化ファイルのファイル鍵を復元し、暗号化ファイルを復号しなければならない。復元に使用する一次式は、管理職側では x_{01} と x_{03} が復元に参加しているとすると、対応する係数がどちらも 0 でない式(B)を選ぶ必要がある。ここで、それぞれの係数を a'_{11}、a'_{13} と表すことにし、式(A)の x_0 の係数 a_{00} を用いると、d_j と同様の方法で x_{01} と x_{03} に対応する、

$$d_{01} = c_1^{\wedge}(2a_{00}a'_{11}x_{01} \bmod q)$$

$$d_{03} = c_1^{\wedge}(2a_{00}a'_{13}x_{03} \bmod q)$$

が生成できる。これらの値から、

$$\begin{aligned} d_0 &= c_1^{\wedge}(2a_{00}x_0 \bmod q) \\ &= c_1^{\wedge}(2a_{00}a'_{11}x_{01} + 2a_{00}a'_{13}x_{03} \bmod q) \\ &= d_{01}d_{03} \end{aligned}$$

を求めることができる。同様にシステム管理者側からも d_1 を得られるので、$c_2/\Pi d_j$ よりファイル鍵 m を復元でき、この m を使って暗号化ファイルに緊急アクセスできる。

6. 性能評価

暗号アルゴリズムに鍵長128ビットのSAFER[4]、初期化ベクタ生成関数にMD5を使い、暗号レコードサイズを512バイトとして速度評価を行った。速度評価の際はOSのキャッシュ機能を無効にし、1 メガバイト長のファイルのランダム位置からランダム長のデータを読み書きした。

ファイル I/O フィルタ組み込み時の入出力速度は、ファイルのレコード化によるオーバヘッド削減により、非組み込み時に比べて暗号処理時間の分、遅くなっただけである。アプリケーションでファイルを編集している際の入出力長は短いので、実用上、暗号処理によるオーバヘッドは問題のない程度である。

7. おわりに

ファイルの暗号化とアクセス制御を行うセキュアファイルシステムの構築について述べた。試作したプロトタイプはファイル I/O フィルタとして実装してあり、既存のファイルシステム上でファイルの動的な暗号化とアクセス制御を行うことができる。

また、Shamir の方式を変形した (k, n) 閾値復号方式を組み込んであり、暗号化ファイルの緊急アクセスにも対応している。この方式では秘密鍵の柔軟な分割が可能となっており、復号鍵復元の際に分割鍵の所有者のうち特定の者を必須にしたりグループ化したりすることができる。

試作したプロトタイプの動的な暗号処理によるオーバヘッドは、ファイルを一定長のレコードに区切った効果により、問題のないレベルに抑えている。

謝辞

本研究を進めるに当たり、適切な助言を頂いた横浜国立大学大学院 工学研究科 松本勉 助教授に感謝致します。

参考文献

[1] 宮崎 博, 鮫島 吉喜, ファイルシステム組み込み暗号ソフト, 第 57 回情処全国大会, 5Q-8, 1998.

[2] A. Shamir, How to Share a Secret, Communications of the ACM, Vol.22, No.11, pp.612-613, 1979.

[3] 遠田 潤一, 宮崎 博, 鮫島 吉喜, 分割鍵所有者のグループ化に対応した(k,n)閾値復号方式, 信学会総合大会, A-7-19, 1999.

[4] J. L. Massey, SAFER K-64: A Byte Oriented Block-Ciphering Algorithm, Fast Software Encryption, Cambridge Security Workshop Proceedings, 1995.