Vol. 47 No. 8 情報処理学会論文誌 Aug. 2006

メモリカードのためのセキュアファイルシステムSASの提案と実装

川島 潤[†] 舩曵信生[†] 中西 透[†]
竹内順一[††] 石崎雅幸[††] ソウザ・アンドレ・カウダス[††]

メモリカードに代表される可搬性メモリは，屋外に持ち出されることの多い着脱可能な小型記録メディアである．そのためメモリカードなどでは，その盗難や紛失時にカードデータ全体が漏洩してしまうことから，カード内ファイルデータの暗号化だけでなく，ファイル管理情報の暗号化を行うことで安全性を高める必要がある．そこで本論文では，Linux をベースに，OS・アプリケーションの変更が不要，メタデータの暗号化が可能，アクセス可能なアプリケーションを指定可能，といった特徴を有するセキュアファイルシステム SAS（Storage Add-on Security）を提案する．本論文では，SAS の設計・実装の詳細説明を行ったうえで，ベンチマークソフトによる性能評価結果を報告する．

A Proposal and Implementation of a Secure Filesystem for Memorycard: SAS

Jun Kawashima,[†] Nobuo Funabiki,[†] Toru Nakanishi,[†]
Junichi Takeuchi,[††] Masayuki Ishizaki[††]
and Andre Caldas de Souza[††]

A memorycard has weakness in assuring the safety of its stored data when it is stolen or missing. Therefore, a secure filesystem that can encrypt the stored data is essential in preventing the risk from illegal users. Besides, the encryption of the metadata is desirable in this vulnerable memorycard to further strengthen the safety. In this paper, we present a design and an implementation of a secure filesystem named *SAS* (*Storage Add-on Security*) for a memorycard on Linux. SAS encrypts the metadata as well as the data, and can choose a security option to file by file, while the application programs accessible to SAS can be specified by system managers. SAS requires no change of the operating system when it is installed into PDA. The performance of SAS is investigated through a benchmark software.

1. はじめに

近年の携帯型の情報処理端末の普及には目を見張るものがある．その顕著な例が PDA（Portable Digital Assistance）である．PDA は手のひら大の電子機器で，情報管理ツールに加え，PC とのデータ共有，ネットワーク接続機能などを備えている．PDA の普及は，特に企業においてさかんであり，主な使用法としては，出先でのメール確認，顧客情報や在庫情報，財務記録の管理などがあげられる．

この PDA を含む情報処理端末の普及にともない，その記録媒体としてのメモリカードの需要が急増している．メモリカードは，記憶媒体としてフラッシュメモリを採用しているカード型の読み書き可能な外部記憶装置である．非常に小型で，データの読み書きにほとんど電力を消費しないため，PDA やノート PC などのバッテリ駆動のモバイル機器の記録メディアとして普及している．メモリカードは，反面，屋外に持ち出されることの多いモバイル機器の着脱可能な記録メディアであるため，盗難・紛失による情報漏洩が問題となっている．メモリカード上の，顧客情報などの個人情報の漏洩は，平成 17 年 4 月 1 日から施行されている個人情報保護法の影響もあり，今後さらに，企業にとっては深刻な問題となるとも考えられる．

そこで本論文では，メモリカードからの情報漏洩を防ぐために，ファイル管理情報（メタデータ）を含むカード内データの暗号化を行うセキュアファイルシステム SAS（Storage Add-on Security）の提案とその実装による評価を行う．SAS 実装上のプラットフォームは，メモリカードの主な使用対象である PDA とする．

本論文では，企業などの組織における PDA の使用

† 岡山大学
Okayama University
†† 株式会社バーズコミュニケーション
Ba-Z Communication Inc.

を前提としている．以後，PDA を管理する企業の情報部門の管理者をシステム管理者，PDA の使用者（一般社員）をユーザ，PDA 内の管理者権限ユーザを root ユーザと呼称する．また本論文が対象とする PDA としては，SHARP Zaurus（SL-C860）としている[15]．これは，SL-C860 では OS として Linux（OpenPDA: kernel-2.4.18 相当）を採用しているからである．Linux はオープンソースとして開発されている OS であり，そのソースコードは無償で公開されている．またデータの暗号化における従来研究の多くは，その動作環境を Linux としていることもあげられる[1),3),7),11),12),14]．

最後に，本論文の章構成を述べる．2 章では，PDA での暗号化システムに求められる要件について述べる．3 章では，ファイルシステムによる暗号化の利点について述べる．4 章では，既存のセキュアファイルシステムについて紹介する．5 章では，SAS の設計について述べる．6 章では，SAS の実装による評価について述べる．最後に 7 章で本研究のまとめを述べる．

2. PDA での暗号化システムの要件

PDA での暗号化システムに求められる要件として，以下の 5 点をあげる．

(1) メタデータの暗号化

個人情報保護法の施行により，保護すべき情報の対象が従来より厳しいものになると考えられる．そこで，暗号化の対象をメモリカード内のファイルデータ部分だけでなく，ファイル管理情報（メタデータ）にまで広げる必要がある．メタデータには，ファイル名，ファイルを生成したユーザの ID，生成時刻，変更時刻，サイズなどが含まれ，場合によってはデータの中身を推測できる情報が含まれている可能性がある．さらに，メタデータの暗号化は副次的な利点をもたらす．一般にファイルデータのみを暗号化した場合には，PDA に接続した時点でメモリカード内のファイルの存在が漏洩してしまう．しかし，メタデータも暗号化した場合，OS はファイルの存在を認識できず，通常「フォーマットエラー」を起こす．そのため不正ユーザは，ファイルの存在が分かる場合に比べ，よりメモリカードの中身に興味を持たないものと考えられる．

(2) 低負荷・高信頼の暗号方式

PDA などの携帯端末は，その小型さから PC に比べて性能が低いため，暗号化にかかるリソース（メモリ，処理時間）が少なく，かつ暗号強度の高い暗号方式が必要である．また，暗号方式は政府推奨暗号方式[16] の中から選択することが望ましいと考えられる．そこで本論文では AES（Advanced Encryption Standard）を採用している．

(3) 既存の OS・アプリケーションの変更が不要

暗号化システムの利用のために PDA 搭載の OS を変更（バージョンアップ）することは，一般ユーザにとって非常に敷居の高い作業である．特に，この作業により PDA 自体がメーカの保証外となる可能性が高く，OS の変更不要は必須の要件といえる．同様に，通常バイナリ形式で配布されるアプリケーションの変更不要も重要な要件である．

(4) 暗号化システムにアクセス可能なアプリケーションの指定

メモリーカード内では暗号化されているデータも，暗号化システムによるその復号時には平文となるため，ファイルコピーや印刷などを許容するアプリケーションからのアクセスを許容した場合，データの不正利用が可能となる．これは，利用時にその利便性やモバイル性のために，通常はユーザ認証を行わない PDA では，特に深刻な問題となる．そこで暗号化システムにアクセス可能なアプリケーションを，あらかじめシステム管理者が指定しておくことでそのような問題を回避することが可能となる．すなわち，復号化されアプリケーションに取り込まれた段階で問題となる行為（ファイルコピー，印刷など）を実行できないアプリケーションのみを，あらかじめシステム管理者が暗号化システムにアクセス可能なものとして指定しておくことで，暗号化システムの信頼性を高めることが可能である．

(5) PDA の盗難時対策

PDA での盗難時には，PDA 内の HDD の取り出しなどにより，直接ファイルにアクセスされる恐れがある．特に，暗号化システムが認証に必要とする暗号鍵などが HDD に存在する場合に問題となる．そこで，HDD 中に保存する暗号鍵をユーザのログインパスワードなどで暗号化して保存する，暗号鍵を USB トークンなどの外部媒体に保存し HDD 内には保存しない，などの対策が必要となる．ただし本論文では，主にメモリカードの盗難，紛失に対するセキュリティの実装について述べており，PDA 本体の盗難については今後の課題としている．

3. ファイルシステムによる暗号化の採用理由

本章では，とりうる 5 種類のデータ暗号化手法についてその利点と欠点を比較することで，本論文の目的に合致した手法としてファイルシステムを採用した理由を明らかにする．また，各手法の OS 内での階層と，暗号化対象を**図 1** に示す．

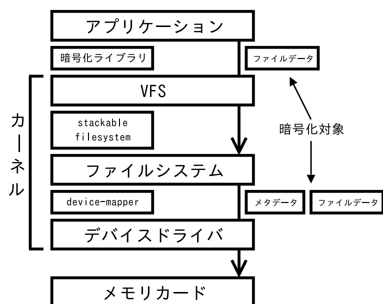


図 1　暗号化手法
Fig. 1　Overview of encryption methods.

(1)　アプリケーションでの暗号化ライブラリの利用
利点：アプリケーション・ライブラリで暗号化が完結するので，カーネルを変更する必要がない．他 OS への移植が容易である．既存のライブラリを流用することができる．
欠点：データ部分の暗号化は行えるが，メタデータの暗号化は行えない（*1）．各アプリケーションはライブラリを利用して暗号化を行うため，使用するすべてのアプリケーションに対し，本ライブラリを呼び出す変更を加える必要がある．
(2)　stackable filesystem の利用
stackable filesystem とは，既存のファイルシステム（Ext2，ReiserFS など）の上位層にラッパとして存在するファイルシステムである．後述するように，アプリケーションは VFS（Virtual FileSystem）を通じてファイルシステムに入出力を行う．stackable filesystem を用いると，通常の VFS → 既存のファイルシステムという呼び出しが，VFS → stackable filesystem → 既存ファイルシステムという呼び出しに変更される．
利点：アプリケーション，ファイルシステム，カーネルを変更することなく，ファイルデータを暗号化することが可能である．また，stackable filesystem を汎用的に作成する FiST（File System Translator）[13] を用いて実装することで，Solaris，Linux，FreeBSD 上で動作するコードを記述することが可能である．ファイルデータの暗号化のみを設計すればよく，ディレクトリ構造，メタデータなどは下位層である既存のファイルシステムにまかせればよい点である．
欠点：データ部分の暗号化は行えるが，メタデータの暗号化は行えない（*1）．
(3)　ファイルシステムへの暗号化機能の付加
利点：アプリケーションは透過的にファイルシステムの暗号化機能を利用することができるため，アプリケーションの変更が不要である．メタデータを暗号化することができる．ファイル情報を認識できるので，ファイルごとに柔軟なセキュリティをかけることができる．一般にファイルシステムの追加にはカーネルを変更する必要があるが，Linux では次章で述べる VFS 機構によりカーネルの変更を行うことなくファイルシステムを追加できる．
欠点：Linux に依存するため，他 OS への移植が困難である．
(4)　device-mapper ドライバの利用
device-mapper [5] は，ファイルシステムとデバイスドライバ（ブロックデバイス）の中間にラッパとして存在する層である．
利点：ファイルシステム，デバイスドライバを変更することなく，ファイルシステムからデバイスドライバへ渡されるデータを暗号化することが可能である．メタデータを暗号化することが可能である．
欠点：データブロックとしてのみデータを認識し，ファイルという粒度でデータを認識できないため，ファイル単位でのアクセス制限などは行うことができない．本論文で使用する Linux 2.4 kernel で利用するためには，カーネルの変更が必要である（*2）．
(5)　メモリカードデバイスドライバへの暗号化機能の付加
利点：アプリケーションとファイルシステムは，透過的にデバイスドライバの暗号化機能を利用できるため，アプリケーションとファイルシステムを変更する必要はない．メタデータを暗号化することができる．
欠点：多くのメモリカードデバイスドライバではソースコードが公開されていないので，ユーザ側で暗号化の変更を行うことができない（*3）．デバイスドライバごとに暗号機能の変更を加える必要がある．device-mapper と同様に，ファイルシステムに比べ単純なセキュリティしかかけることができない．

以上において，アプリケーションによる暗号化，stackable filesystem の利用では，2 章で掲げたメタデータ暗号化を，(*1）のように満たしていない．device-mapper ドライバの利用では，(*2）のようにカーネルの変更が必要である．また，デバイスドライバによる暗号化では（*3）のようにソースが公開されていない場合を考慮すると，現実的でない．そこで，本論文では Linux においてメモリカードの暗号化を目的に独自のファイルシステムを実装し，暗号化機能を付加する方法を採用することとした．

4. 既存のセキュアファイルシステム

本章では，従来の代表的な 5 種類のセキュアファイ

ルシステムを紹介する．その際，2 章で議論した特徴の中で，従来法のメタデータ暗号化の可能性，カーネル変更の不要性について述べる．また，最後に従来法と SAS との違いを述べる．

4.1 CFS（Cryptographic FileSystem）

CFS[1] はユーザランド（アプリケーション・ライブラリレベル）で動作するファイルシステムである．カーネルに搭載されている NFS（NetworkFileSystem）インタフェースを利用することで，カーネルの変更なく CFS を追加することが可能となる．また，暗号処理の実体をユーザランドで実装しているため，システム全体をカーネルに依存しないで実装できる．

CFS の処理本体は，CFS デーモン（cfsd）と呼ばれる NFS サーバとして実装されており，ユーザはローカルホストの特定の領域を CFS 経由でマウントすることにより透過的なデータの暗号化，復号を行うことができる．データは，アプリケーションからカーネルを経由して CFS に渡され，そこで暗号化される．その後，再びカーネルを通して通常のファイルシステムに渡され，保存される．暗号化されたファイルをアプリケーションが呼び出す場合も同様にカーネルを介して CFS で復号が行われ，再びカーネルを介してアプリケーションに復号結果が渡される．

CFS の問題点は，ユーザランドからカーネルへのコンテキストスイッチが頻発することによる，処理時間の増大である．コンテキストスイッチでは，プロセスコンテキスト（レジスタやスタックなど）を切り替える必要があるため，その分の処理時間が余計に必要となる．また CFS ではファイル名など，メタデータの一部は暗号化されるが，ディレクトリ構造は暗号化されないので，不正利用者にファイルの存在が漏洩してしまう．

4.2 TCFS（Transparent Cryptographic FileSystem）

TCFS[3] は CFS と同様に，NFS のインタフェースを利用することで暗号化を行うセキュアファイルシステムである．CFS との違いは，暗号処理の実体をカーネル内で行う点である．よって，CFS の欠点であった処理時間の増大を克服している．しかし，カーネル内での処理を追加するために，TCFS の利用にはカーネルの変更を行う必要がある．また，メタデータについては，CFS と同様の問題点がある．

4.3 NCryptfs（New Cryptographic filesystem）

NCryptfs[12] は，stackable filesystem として設計されているセキュアファイルシステムである．NCryptfs の特徴は，アドホックなグループアクセス権限，暗号鍵の有効時間制限などである．NCryptfs の欠点は，カーネルの変更を必要としていることや，stackable filesystem が暗号化を行うのはファイルデータのみのため，ディレクトリ構成，メタデータなどの情報が漏洩してしまうことである．

4.4 eCryptfs

eCryptfs[7] は，NCryptfs の前身である Cryptfs[14] を基に設計されており，NCryptfs と同様 stackable filesystem として設計されるセキュアファイルシステムである．eCryptfs の特徴は，ファイルデータのヘッダに暗号化に関するメタデータを格納する点である．これにより，暗号化されたファイルを別のホスト（パーティション）にコピーするだけで，コピー先で eCryptfs を用いて復号を行うことが可能となる．メタデータがファイルデータに含まれない場合，暗号化されたファイルをコピーするだけでは，stackable filesystem が復号に必要とするメタデータはコピーされない．通常，セキュアファイルシステム上のファイルを別ホストにコピーする場合には一度平文に復号する必要があるが，暗号化したままコピー可能であれば，セキュリティの面からも暗号コストの面からも優れているといえる．また eCryptfs は NCryptfs と異なり，カーネルの変更を必要としない．しかし，Linux 2.6 kernel のみをサポートしており，本論文で使用する 2.4 kernel では利用できない．また stackable filesystem であるため，ディレクトリ構成，メタデータなどの情報は漏洩してしまう．

4.5 TrueCrypt

TrueCrypt[11] は，ループバックデバイスを用いてイメージファイルを仮想ディスクとして扱うセキュアファイルシステムである．TrueCrypt の特徴は，Linux だけでなく Windows でも動作する点である．これに対し前述のセキュアファイルシステムは，すべて Linux もしくは UNIX 系 OS でのみ動作する．TrueCrypt は Linux 上では device-mapper を用いて実装されている．TrueCrypt も eCryptfs 同様，Linux 2.6 kernel のみをサポートしており，本論文で使用する 2.4 kernel では利用できない．

上記手法において，ファイルシステムより上位の層における設計では，メタデータを暗号化できず，ファイルシステムより下位の層においてはファイルでのアクセス制限は行えない．これに対し提案する SAS では，ファイルシステム自体を扱うことでメタデータの暗号化とファイル単位でのアクセス制限をともに行うことが可能となる．**表 1** に従来手法と SAS の違いを

表 1 特徴比較
Table 1 Feature comparison.

	CFS	TCFS	NCryptfs	eCryptfs	TrueCrypt	SAS
メタデータ暗号化	—	—	—	—	√	√
OS・アプリケーション変更不要	√	—	—	√	√	√
アプリケーション制限	—	—	—	—	—	√
ファイル単位でのアクセス制限	√	√	√	√	—	√
Linux 2.4 kernel で利用可能	√	—	√	—	—	√

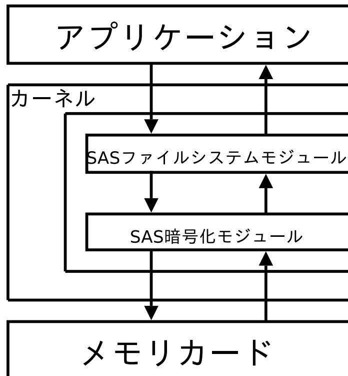


図 2 SAS システム
Fig. 2 Overview of SAS.

示す．

5. SAS の設計

本章では，提案システム SAS の設計について説明する．まず SAS の概要を述べ，次に SAS で用いた Linux の機能について述べる．その後，SAS の詳細を機能ごとに述べる．

5.1 SAS の構成

SAS は，2 つのカーネルモジュール（SAS ファイルシステムモジュール，SAS 暗号化モジュール）から構成される．このように暗号化部分を別モジュールとして実装することで，後に暗号化部分のみの変更を行うことが容易となる．**図 2** にアプリケーション，カーネル，提案システム，メモリカードの関係を示す．

5.2 カーネルモジュール

Linux におけるカーネルモジュールとは，カーネルを構成する要素（プロセス管理，メモリ管理，ファイルシステム管理，デバイス制御，ネットワーク管理など）を機能ごとに分割したプログラムの集合（モジュール）である．これには，デバイスドライバ，ファイルシステムなどがあげられる．

カーネルモジュールの利点として，機能ごとにコードを分割することで，カーネルが必要とするときのみ各モジュールを利用することが可能となり，メモリ使用量を最小限に抑えることができる．またカーネルに

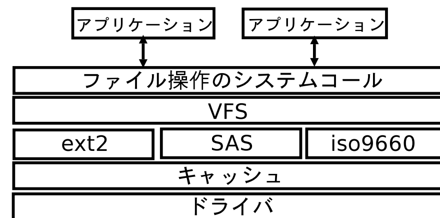


図 3 VFS の構成
Fig. 3 The VFS structure.

新しい機能を追加する場合にも，カーネル自体にあらかじめ機能を組み込んでおく必要がないため，新機能追加時にもカーネル自体の変更は必要ない．以上より SAS では，低リソース性とカーネル変更不要を実現するために，カーネルモジュールとして各機能を実装している．

5.3 VFS

VFS とは，様々なファイルシステムを統一的に扱うための仕組みであり，VFS はカーネルが提供する機能である（**図 3**）．具体的には，アプリケーションがデバイスにアクセスするために発行する標準的なシステムコールを，デバイスのファイルシステム特有の関数に結び付ける機能である．VFS により，アプリケーションはファイルシステム固有の形式を意識することなく，標準化された関数によりデバイスにアクセスできる．この VFS により，SAS のように新たなファイルシステムをカーネルに追加する場合にも，既存のアプリケーションを変更する必要がない．

5.4 SAS ファイルシステムモジュール

提案する SAS ファイルシステムモジュールは，通常のファイルシステムの機能に加え，後述する暗号化モジュールを呼び出す機構を持つ．

5.4.1 SAS ファイルシステムの実装

SAS ファイルシステムモジュールは，既存の Ext2 ファイルシステム[2)]に修正を加えることで実装を行っている．Ext2 に加えた修正は以下のとおりである．また以降では，SAS ファイルシステムの記述は具体的には SAS ファイルシステムモジュールを指すものとする．

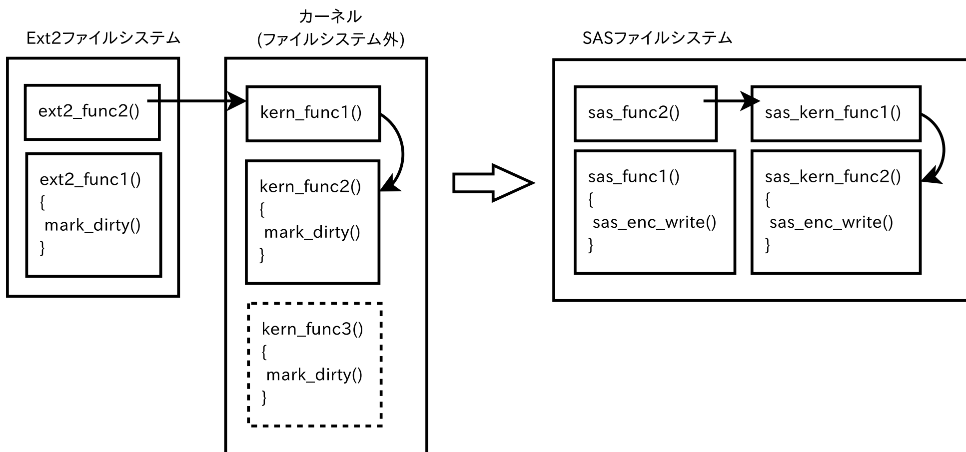


図 4 SAS の関数
Fig. 4 The functions of SAS.

（1） ライトバックからライトスルーへの変更

Ext2 は他のファイルシステムと同様に書き込み方式にライトバック方式を採用している．カーネル非依存性を実現するためには，書き込み方式をファイルシステムが直接書き込みを行うライトスルー方式へ変更する必要がある．以下に変更に要した手順を述べる．

ライトバック方式をライトスルー方式へ変更するために，ライトバックを引き起こす関数（mark_dirty() を呼び出す関数）を変更する必要がある．ここで，ライトバックを引き起こす関数とは，バッファキャッシュに汚れフラグを立てる関数（mark_dirty()）を呼び出す関数である．mark_dirty() を呼び出す関数の変更は，その関数が Ext2 ファイルシステム内に存在する場合と，Ext2 ファイルシステム外，すなわちカーネル内に存在する場合とで異なる．

まず，mark_dirty() を呼び出す関数が Ext2 ファイルシステム内に存在する場合について述べる（**図 4** の ext2_func1()）．SAS ファイルシステムではこの関数を sas_func1() と宣言し，mark_dirty() の代わりにライトスルーを行う関数（sas_enc_write()）を呼び出すよう変更を行う．sas_enc_write() とは，データの暗号化後，後述する write_block() を行う関数である．汚れフラグを立てず，その場で書き込みを行うためライトスルーとなる．

次に Ext2 ファイルシステム外で mark_dirty() を呼び出す関数が存在する場合について述べる．その関数は，Ext2 ファイルシステム経由で呼び出されるカーネル内の関数（図 4 の kern_func2()）と，経由されないで呼び出される関数（図 4 の kern_func3()）の 2 つに分けられる．どちらの関数内でも前述したように mark_dirty() を，sas_enc_write() に変更する必要がある．しかし，Ext2 ファイルシステム外，すなわちカーネル内を変更することは前述した制約に違反する．そこで，Ext2 ファイルシステム経由で呼び出される関数についてこの問題を解決する方法を説明する．Ext2 ファイルシステム外で mark_dirty() を呼び出す関数の処理（図 4 の kern_func1(), kern_func2()）を，すべて SAS ファイルシステム内で再実装する（sas_kern_func1(), sas_kern_func2()）．次に，SAS ファイルシステムで ext2_func2() に対応する sas_func2() を宣言し，kern_func1() を呼び出す代わりに，sas_kern_func1() を呼び出すよう変更を行う．同様に sas_kern_func1() は sas_kern_func2() を呼び出すよう変更する．また，sas_kern_func2() 内では，mark_dirty() を sas_enc_write() に変更する．

次に，Ext2 ファイルシステムを経由されずに呼び出される関数について説明する（図 4 の kern_func3()）．この関数は，SAS ファイルシステムから呼び出すよう変更できないため，通常ならば重大な障害となる．しかし，今回対象とするカーネルを調べた結果，mark_dirty() を呼び出すすべての関数は，Ext2 ファイルシステム経由で呼び出されているか，または Ext2 ファイルシステム内の動作でその呼び出しを避けられることが判明したため，この関数について考慮する必要はないことが明らかとなった．

以上よりすべての mark_dirty() は，SAS ファイルシステム内から呼ばれるよう変更され，さらに sas_enc_write() に変更された．ここで SAS の実装において，この処理が最も困難であったが，本実装方法は汎用性に乏しく，今後修正する必要があると考え

る．修正の方針としては，ファイルシステム内に専用のバッファキャッシュを作成することを検討している．この方針は実装が難しい半面，従来カーネルが管理していたバッファキャッシュをファイルシステム自身で管理できるため，ライトスルーを行う必要はなくライトバックとして実装可能である．よってカーネル非依存性が高まり，現在の実装より汎用性が確保できると考えられる．

(2) 暗号化に関する変更

ファイルデータとファイルシステム構造を秘匿するために，メモリカードやディスクへのアクセス時に暗号化・復号を行う．

最初に，データを復号して読み込むために変更した手順について述べる．まず，ファイルシステムがデバイスドライバに対して読み込みを要求する関数（read_block() と呼ぶ）を調べる．次に，read_block() を呼び出しているすべての関数を，カーネル，ファイルシステムの中から探し出す．そのうえでファイルシステム内の read_block() を呼び出す関数について，read_block() の直後に復号を行う関数を追加する．また，ライトスルーへの変更と同様に，ファイルシステム外で read_block() を呼び出す関数が存在する場合，その関数を呼び出す関数をすべてファイルシステム内で実装し，復号処理を追加する．

次に，データを暗号化して書き込むために変更した手順について述べる．まず，ファイルシステムがデバイスドライバに対して書き込みを要求する関数（write_block() と呼ぶ）を調べる．次に，write_block() を呼び出しているすべての関数を，カーネル，ファイルシステムの中から探し出す．そのうえでファイルシステム内の write_block() を呼び出す関数について，write_block() の直前に暗号化を行う関数を追加する．ライトスルーへの変更と同様に，ファイルシステム外で write_block() を呼び出す関数が存在する場合，その関数を呼び出す関数をすべてファイルシステム内で実装し，暗号処理を追加する．

5.5 SAS 暗号化モジュール

SAS 暗号化モジュールは前述の SAS ファイルシステムモジュールから呼ばれることで動作する．暗号方式としては AES（Advanced Encryption Standard）[4] を採用する．以下では，まず AES の概要を述べ，後にブロック暗号化モードについて述べる．

5.5.1 AES の概要

AES [10] とは，以下の特徴を持つ米国政府の次世代標準暗号化方式である．

- 128，192，256 bit のブロック長を持つ共通鍵ブロック暗号である．
- 128，192，256 bit の鍵が使用できる．
- 既知の暗号攻撃に対して耐性がある．
- 暗号・復号，鍵生成にかかる時間が少ない．
- 暗号・復号，鍵生成に使用するメモリが少ない．

5.5.2 ブロック暗号化モード

ブロック暗号方式では，同一の鍵と平文からは必ず同一の暗号文が生成される．そのため，ディスク暗号化など，サイズの大きなデータの暗号化に対して問題となる．これは繰り返し表れる部分を解析することなどで暗号文から平文の特徴が推測され，暗号解読の手がかりにされる可能性があるからである．

そこで，複数のブロックと乱数を組み合わせることで，同一の鍵と平文からも異なる暗号文を生成するブロック暗号化モード[6] が提案されている．ブロック暗号化モードには複数の方式があるが，SAS では実装が容易な CBC モード（Cipher Block Chaining Mode）を採用している．

CBC モードでは複数のブロックに対して処理が連鎖的に適用される．第 1 の平文ブロックに対し，乱数である初期ベクトル（IV：Initial Vector）と XOR をとり，第 1 の暗号ブロックとする．第 2 の平文ブロックに対し，第 1 の暗号ブロックと XOR をとり，第 2 の暗号ブロックとする．これを最後のブロックまで繰り返す．復号はこの逆の手法をとる．ただし，あるブロックの保存過程でビットエラーが生じた場合，このエラーが次々と次のブロックにエラーを波及させてしまう欠点を持っている．

5.5.3 暗号化モジュールの設計

SAS 暗号化モジュールは，暗号化用と復号用の 2 つの関数をファイルシステムに提供する．その際，AES と CBC の実装に関して GPL（General Public License）として公開されているものを利用している．その鍵長は 128 bit である．

処理としては，16 バイト（鍵長）を 1 ブロックとし，1,024 バイト（論理ブロックサイズ）単位で CBC モードを用いて変換している．この方法では，ファイルシステムが論理ブロックサイズ（=バッファキャッシュサイズ）単位でデバイスドライバに書き込み要求を行うため，効率が良いと考えられる．

CBC モードの IV としては，暗号鍵と論理ブロック番号に対し SHA-1（Secure Hash Algorithm 1）を適用したハッシュ値を用いる．SHA-1 は，認証やデジタル署名などに使われるハッシュ関数の 1 つである．前述したように IV は乱数でなければならず，しかも適用されるブロック群ごとに異なる必要がある．そのた

め，暗号鍵と論理ブロック番号のハッシュ値を利用することで，適用するブロック群ごとに異なる乱数が得られるようにしている．

以上より，実装した暗号・復号関数の引数は，暗号鍵，データ（変換前），データ（変換後），論理ブロック番号となる．

5.6 暗号鍵の管理

5.6.1 暗号鍵の保存方法

暗号鍵は，root ユーザのみ読み込み可能なファイルとして PDA 本体に保存する．root ユーザ権限の取得は OS のセキュリティシステムに依存するが，今回利用した Linux システムにおいてはパスワードを用いており，システムとしての暗号強度は root のパスワードに依存する．また本論文では，企業などの組織における PDA の使用を前提としており，PDA のユーザである一般社員は情報部門から提供された PDA を利用するため，情報部門で管理する root のパスワードを知らされていないものと仮定している．また root 権限がないと作業が行えないアプリケーション（コマンド）については，sudo コマンドなどを用いて root のパスワードなしで実行できるものと仮定をしている．PDA 本体の盗難時には，PDA 内部の HDD の取り出しなどにより暗号鍵が漏洩する可能性があり，この問題については今後，ネットワークから暗号鍵をダウンロードし復号を行う方法などにより解決を行う．

5.6.2 暗号鍵の設定方法

暗号鍵はファイルシステムのマウント時に，専用の mount プログラムにより sas ファイルシステムモジュールに渡される．ここで，mount プログラムは mount システムコールによりメモリカードをマウントするが，mount システムコールの引数は proc ファイルシステムを通じてユーザから閲覧可能である．したがって引数に暗号鍵そのものを渡すことはできない．そこで，本 mount プログラムは暗号鍵をスタックに積み，その先頭アドレスを mount システムコールの引数として指定し，ファイルシステムはそのアドレスから暗号鍵を取得する．mount プログラム終了後スタックは開放されるため，ユーザが mount システムコールの引数で指定されたアドレスから暗号鍵を取得することは困難である．

5.7 認証用プログラム

提案システム SAS にアクセス可能なアプリケーションをシステム管理者が指定可能とするために，SAS がアプリケーションを認証する機能を追加する．ユーザはシステム管理者が用意した認証用プログラムを利用して，アプリケーションを起動する必要がある．認証は 2 段階に分かれており，認証用プログラムによるアプリケーションの認証，SAS ファイルシステムモジュールによる認証用プログラムの認証である．以下，それぞれについて説明を行う．

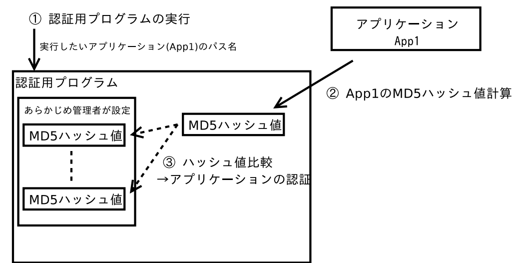


図 5 アプリケーション認証 1
Fig. 5 Application authentication 1.

5.7.1 アプリケーションの認証

システム管理者は，認証用プログラムによるアプリケーションの認証により，SAS にアクセス可能なアプリケーションを制限する．概要を**図 5** に示す．

（1） ユーザは認証用プログラムをコマンドライン上で実行する．その際，認証用プログラムの引数として実行したいアプリケーションの絶対パス名を渡す．

（2） 認証用プログラムは，あらかじめシステム管理者によって許可された複数のアプリケーションバイナリファイルの MD5 ハッシュ値を内部に持っており，引数として渡されたアプリケーションの MD5 ハッシュ値と比較することでアプリケーションの実行可否を判断する．

（3） ユーザが実行しようとするアプリケーションの MD5 ハッシュ値が認証プログラム内に存在しない場合，認証プログラムはエラーを返す．存在した場合は，次に記す認証用プログラムの認証を実行する．

5.7.2 認証用プログラムの認証

認証用プログラムから起動したアプリケーションのみを SAS にアクセス可能とする．そのためには，認証用プログラムが正規のものであることの認証と，アプリケーションが認証用プログラムから起動されたことを SAS が識別できる必要がある．また SAS では既存アプリケーションを変更することなく認証機能を付与するために，「プログラム中でオープンされたファイルオブジェクトは exec システムコール時に継承される」といった特徴を利用している．概要を**図 6** に示す．

（1） 認証用プログラムは SAS ファイルシステムのマウントポイントファイル（/mnt/card など）をオープンし，ioctl システムコールを SAS ファイルシステムモジュールに対し発行する．その際，ioctl システムコールの引数として認証情報を渡すことで正規

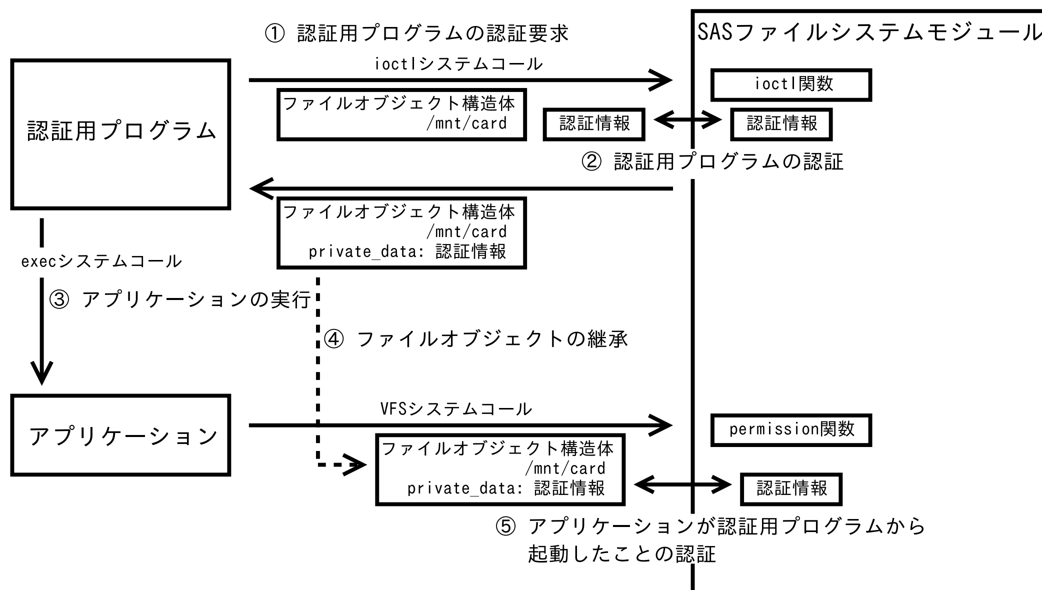


図 6 アプリケーション認証 2
Fig. 6 Application authentication 2.

の認証用プログラムであることが証明可能となる．このマウントポイントファイルがオープンされるときにファイルオブジェクト構造体が生成される．ここで，認証情報には 128 bit の乱数値を用いており，システム管理者により認証用プログラム，SAS ファイルシステムモジュールのそれぞれに組み込まれているものとする．

（2） SAS ファイルシステムモジュールは，正規の認証用プログラムであることを認証後，オープンされたマウントポイントのファイルオブジェクト構造体中の private_data メンバに認証情報を格納する．認証に失敗した場合には，ioctl に対しエラーを返し，private_data メンバに認証情報は格納しない．ここで private_data メンバはカーネル内からのみ参照することが可能であり，root 権限を持ち，高度なスキルを持ったユーザがクラックのためにカーネルモジュールを作成する以外には改竄することはできない．

（3） 認証用プログラムはその後，指定されたアプリケーションを exec システムコールにより実行する．この際，マウントポイントファイルはオープンされた状態である．

（4） exec により起動されたアプリケーションは，マウントポイントファイルのファイルオブジェクトを継承している．そのため，アプリケーションがファイルシステムにアクセスする場合は，以下の手順でアクセスが制限可能となる．

（5） VFS ではファイルシステムにアクセスがあった場合，各ファイルシステムモジュールにおいて permission という構造体タグ名で登録された関数が呼ばれる．SAS ファイルシステムモジュールでは，この permission 関数中において，マウントポイントファイルオブジェクト中の private_data メンバを参照し，認証情報の有無に応じてアクセスを行ったアプリケーションが認証用プログラムから起動されたかどうかを識別する．アクセスが許可された場合，要求されたシステムコールを実行する．アクセスが拒否された場合は，Operation not permitted エラーを返す．

ここで，認証用プログラムから呼び出されるアプリケーションのみがシステムにアクセスできることと，アプリケーション自体には変更が行われない点に注意されたい．

認証用プログラムの問題として，アプリケーションの MD5 ハッシュ値の改竄により，任意のアプリケーションが実行されてしまう可能性がある．これに対しては認証用プログラムの改竄を監視する機能を，ユーザが改竄することが難しいカーネル内に設けることなどが考えらるが，これも今後の課題とする．

6. SAS の評価

本章では，提案システム SAS の評価について述べる．

6.1 システム性能の評価

6.1.1 評 価 方 法

まず，SAS の性能評価として，Ext2 に対する処理時間の 2 種類のオーバヘッドを測定する．ライトスルー

の採用によるライトバックに対するオーバヘッドと，暗号・復号処理にかかるオーバヘッドである．この処理時間の測定には，ファイルシステムの性能評価に一般に用いられる PostMark[8] を利用する．PostMark は，NetApp 社が実装/配布を行っているベンチマークソフトである．本ベンチマークでは一連の処理として，「ファイル作成」,「読み込み」,「追記」,「削除」を繰り返し行い，それぞれの処理内容について 1 秒ごとの平均回数を算出する．ベンチマークソフトへの入力設定項目と，出力項目は以下のとおりである．

- 入力設定項目
  処理の繰返し数，作成ファイルサイズ範囲.
- 出力項目
  「ファイル作成」,「読み込み」,「追記」,「削除」についての 1 秒ごとの平均回数，読み込み速度，書き込み速度.

6.1.2 評 価 結 果

評価に利用した Zaurus（SL-C860）の性能を**表 2** に記す.

処理の繰返し回数を 1,000 回，作成ファイルサイズの範囲を 500 B〜8 KB にしたときの測定結果を**表 3**, **表 4** に示す．ここで作成ファイルサイズについては，文献 9) の「UNIX システムに存在するファイルの約 85%が 8 KB より小さい」という記述を参考にしている.

表 3 は，ライトスルー方式とライトバック方式の比較である．ライトスルーの測定は，SAS システムから暗号化処理を除いた状態で行い，ライトバックの測定は，Ext2 を利用している．表 4 は，暗号化処理を行う場合と，行わない場合の比較である．どちらの場合も，SAS システム上で測定を行う．両者の違いは暗号化処理が加えられたか否かであり，どちらの場合もライトバック方式である.

表 2　評価環境
Table 2　Test environment.

CPU	XScale（PXA255 400 MHz）
メモリ	64 MB
メモリカード	SD カード（32 MB，平均転送速度 2 MB/s）
OS	OpenPDA 1.5.4

6.1.3 考 察

表 3 より，ライトスルーの読み込み・書き込み性能は，ライトバックに比べてその約 3 分の 1 である．これは，ライトスルー時のメモリカードへのアクセスが低速となるからである．ライトスルーでは，アプリケーションからの書き込み要求ごとに低速なメモリカードに対しトランザクションを開始する必要があり，これがボトルネックとなっている．一方ライトバックでは，カーネルが書き込み要求をキューにためることで，1 回のトランザクションで大量のデータを同時に書き込むことが可能となる.

次に表 4 より，暗号・復号化に要するオーバヘッドはほぼ存在しないことが分かる．これは，メモリカードへのアクセス処理時間に比べ，暗号・復号化に要する処理時間が小さいためである.

SAS ファイルシステムのベンチマークソフトによる性能評価の結果，ライトスルーの採用によりアクセス速度が従来の約 1/3 に低下することが分かる．しかし，SAS ファイルシステムを使用した状態で，メールクライアント，アドレス帳，メモ帳，音楽プレーヤなど PDA で日常使用するアプリケーションを動作させたところ，読み込み・書き込みとも違和感なく動作する．すなわち，アクセス速度の低下にもかかわらず，SAS ファイルシステムは実用的といえる.

6.2 システム規模の評価

6.2.1 評 価 結 果

次に，SAS の実装に要したコード量・モジュールサイズの評価として，Ext2 のコード量・モジュールサイズとの比較結果を**表 5** に示す．ここでコード量の目

表 3　ライトスルーとライトバックの比較
Table 3　Performance comparison with write-back and with write-through.

	作成（回/秒）	読み込み（回/秒）	追記（回/秒）	削除（回/秒）	読み込み（KB/sec）	書き込み（KB/sec）
write-through	4	1	1	6	6.4	13.7
write-back	12	5	5	16	17.3	36.9

表 4　暗号化ありと暗号化なしの比較
Table 4　Performance comparison with encryption and without encryption.

	作成（回/秒）	読み込み（回/秒）	追記（回/秒）	削除（回/秒）	読み込み（KB/sec）	書き込み（KB/sec）
encryption	4	1	1	6	6.2	13.1
non encryption	4	1	1	6	6.4	13.7

表 5 SAS のコード量とモジュールサイズ
Table 5 LOC and module sizes of SAS.

	コード量	モジュールサイズ（KB）
ファイルシステム	4,045	68
暗号化モジュール	481	29
SAS（合計）	4,526	97
Ext2	3,116	49

安として，LOC（Line of Code）を用いる．LOC とはソースコードから空白行，コメントアウト行を除いた行数である．

6.2.2 考　察

表 5 の，SAS ファイルシステムと Ext2 の比較結果では，LOC で 1.3 倍，モジュールサイズで 1.4 倍程度，SAS ファイルシステムが増加している．これは，ライトスルーを実現するために，Ext2 ではファイルシステム外で実装されていた関数を，SAS では追加的にファイルシステム内でも実装していることが原因である．しかし，Zaurus に搭載されているカーネルの本体サイズが 2.7 MB であることを考慮すると，この増加も実用上問題ないといえる．

SAS ファイルシステムは Ext2 に比べ約 900 行コード量が増加しているが，この内訳はカーネルから取り出して再定義した部分が約 500 行，アプリケーション認証に関する部分が約 300 行，デバッグ出力その他が約 100 行となっている．カーネルから取り出して再定義した関数は 21 個あり，そのほとんどが，バッファキャッシュを扱う関数が記述されている，カーネルソースディレクトリ下の fs/buffer.c 内に記述されているものである．

7. ま と め

本論文では，メモリカードを対象として，OS・アプリケーションの変更が不要，メタデータの暗号化が可能，アクセス可能なアプリケーションを指定可能なセキュアファイルシステム SAS（Storage Add-on Security）の設計・実装，および，評価について報告した．今後の課題は，ファイルごとの柔軟なセキュリティ設定，認証用プログラムの改竄防止，PDA の盗難時の対策として暗号鍵の安全な保管方法，PDA 本体のセキュリティの考慮，ライトスルーによるオーバヘッドの解消，適用可能 OS に関する汎用性の実現などである．

参 考 文 献

1) Blaze, M.: A cryptographic file system for Unix, *Proc. 1st ACM Conf. Computer and Communication Security*, pp. 9–16 (1993).
2) Card, R., Ts'o, T. and Tweedie, S.: Design and implementation of the second extended filesystem. http://web.mit.edu/tytso/www/linux/ext2intro.html
3) Cattaneo, G., Catuogno, L., Del Sorbo, A. and Persiano, P.: The design and implementation of a transparent cryptographic filesystem for UNIX, *Proc. Annual USENIX Tech. Conf. FREENIX Track*, pp.245–252 (2001).
4) Daemen, J. and Rijmen, V.: AES proposal: Rijndael. submission to NIST (1999). http://csrc.nist.gov/encryption/aes
5) Device-mapper. http://sourceware.org/dm/
6) Dworkin, M.: Recommendation for block cipher modes of operation, *Nist Special Publication 800-38a, National Inst. Standards and Technology* (2001).
7) Halcrow, M.A.: eCryptfs: An enterprise-class encrypted filesystem for Linux, *Proc. Linux Symp. Ottawa*, pp.201–218 (2005).
8) Katcher, J.: PostMark: A new filesystem benchmark, Technical report, TR3022, Network Appliance (1997).
9) Mullender, S.J. and Tanenbaum, A.S.: Immediate files, *Software Practice and Experience*, Vol.14, No.4, pp.365–368 (1984).
10) Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J. and Roback, E.: *Report on the development of the advanced encryption standard* (*AES*), National Inst. Standards and Technology (2000).
11) TrueCrypt home page. http://www.truecrypt.org
12) Wright, C.P., Martino, M. and Zadok, E.: NCryptfs: A secure and convenient cryptographic file system, *Proc. Annual USENIX Tech. Conf.*, pp.197–210 (2003).
13) Zadok, E. and Nieh, J.: FiST: A language for stackable file systems, *Proc. Annual USENIX Tech. Conf.* (2000).
14) Zadok, E., Adulescu, I.B. and Shender, A.: Cryptfs: A stackable vnode level encryption file system, Technical report, CUCS-021-98, Columbia University (1998).
15) Zaurus SL-C860. http://ezaurus.com/lineup/sl/index.html#slc860
16) 電子政府推奨暗号．http://www.soumu.go.jp/joho_tsusin/security/pdf/cryptrec_01.pdf

(平成 17 年 11 月 28 日受付)
(平成 18 年 6 月 1 日採録)

川島　　潤

昭和 56 年生．平成 17 年岡山大学大学院自然科学研究科電子情報システム工学専攻修士課程修了．現在，同大学院産業創成工学専攻博士後期課程．ファイルシステムの研究に従事．電子情報通信学会会員．

舩曳　信生（正会員）

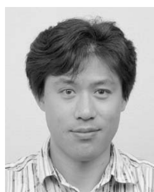
昭和 59 年東京大学工学部計数工学科卒業．同年住友金属工業（株）勤務．平成 3 年ケースウエスタンリザーブ大学大学院修士課程修了．平成 6 年大阪大学基礎工学部情報工学科講師．平成 7 年同助教授．平成 12 年カリフォルニア大学サンタバーバラ校客員研究員．平成 13 年岡山大学工学部通信ネットワーク工学科教授．平成 17 年同大学大学院自然科学研究科教授．博士（工学）．ネットワークプロトコル，セキュリティ，組合せ最適化，画像処理，教育工学等に関する研究に従事．IEEE，電子情報通信学会各会員．

中西　　透（正会員）

平成 7 年大阪大学大学院基礎工研究科博士前期課程修了．平成 10 年同大学院博士後期課程退学．同年岡山大学工学部情報工学科助手．平成 12 年同通信ネットワーク工学科助手．平成 15 年同講師．平 17 年同大学大学院自然科学研究科講師．平成 18 年同助教授．情報セキュリティ，ネットワークセキュリティに関する研究に従事．博士（工学）．電子情報通信学会会員．

竹内　順一

昭和 60 年岡山大学工学部電子工学科卒業．同年セイコーエプソン（株）入社，同年エー・アイ・ソフト（株）出向．平成 5 年エー・アイ・ソフト（株）退社．同年（株）バーズコミュニケーション入社．



石崎　雅幸

平成 5 年岡山大学教育学部小学校教員養成課程卒業．平成 7 年岡山大学大学院教育学研究科修了．同年岡村ゼミナール株式会社入社．平成 9 年学校法人朝日学園朝日塾小学校入社．平成 10 年（株）バーズコミュニケーション入社．

ソウザ・アンドレ・カウダス

平成 16 年岡山大学工学部卒業．同年（株）バーズコミュニケーション入社．