情報処理学会論文誌 Vol.53 No.10 1–7 (Oct. 2012)

Android端末の盗難対策のためのページキャッシュ暗号化

福田直人[1] 光来健一[1]

受付日 2011年11月4日, 採録日 2011年12月1日

概要：スマートフォンやタブレットなどの Android 端末は盗難にあうリスクが高いため，ディスク暗号化により端末が盗まれた場合でもディスクからの情報漏洩を防いでいる．しかし，近年，コールドブート攻撃と呼ばれる攻撃手法が報告され，メモリからの情報漏洩が問題となってきている．Android OS ではディスク上のデータをメモリ上にページキャッシュとして保持しているため，コールドブート攻撃が行われるとディスクを暗号化していてもページキャッシュ上のデータを盗み見られてしまう．本稿では，メモリ上のページキャッシュを暗号化することで Android 端末の盗難時においても情報漏洩を防ぐためのシステム Cache-Crypt を提案する．Cache-Crypt はディスク上のファイルを読み込んだ時などにページキャッシュの内容を暗号化する．アプリがファイルにアクセスするときだけ OS がページキャッシュを復号し，アクセスが終わったら再び暗号化する．Cache-Crypt を用いることで情報漏洩の対象をアクセス中のページキャッシュのみに限定することができる．我々は Android OS に Cache-Crypt を実装し，ページキャッシュからの情報漏洩を防げることを確認した．

キーワード：Android，盗難対策，コールドブート攻撃，メモリ暗号化

Page Cache Encryption for Protecting Android Devices against Theft

Fukuda Naoto[1] Kourai Kenichi[1]

Received: November 4, 2011, Accepted: December 1, 2011

Abstract: Since Android devices such as smartphones and tablets are at high risk of theft, they prevent information leakage from disks by disk encryption in case of device theft. However, in recent years, cold boot attacks have been reported and information leakage from memory is being a major problem. Since the Android operating system maintains data on disks in memory as the page cache, data on the page cache can be stolen by cold boot attacks even if disks are encrypted. In this paper, we propose a system called Cache-Crypt for preventing information leakage in case of device theft by encrypting the page cache in memory. Cache-Crypt encrypts the page cache, e.g., when the operating system reads the operating system reads files on disk. The operating system decrypts the page cache only when apps access files, and encrypts it again when the acces is done. Cache-Crypt can limit to the range of information leakage only to the page cache being accessed. We have implemented Cache-Crypt in the Android operating system and confirmed that it could prevent information leakage from the page cache.

Keywords: Android, anti-theft, cold boot attack, memory encryption

1. はじめに

近年，スマートフォンやタブレットなどの Android 端末が急速に普及している．Android 端末のアクティベート数は 2013 年で 10 億台以上になり，今も増加し続けている．特に中国やインドなど新興国向けに安価な Android 端末が提供されているため，普及率も他の OS を搭載した端末に比べて非常に高い．Android 端末の普及や多機能化に伴って，Android 向けに数多くのアプリが提供されるようになっている．そのため，Android 端末は従来の携帯電話より多くの情報を保持しており，端末が盗難にあった際のリスクは高まっている．例えば Android 端末には，クレジットカード番号，ID やパスワード，その他の多くの個人情報が格納されていることが多い．加えて，Android 端末は

[1] 九州工業大学
Kyushu Institute of Technology

© 2012 Information Processing Society of Japan

ノート PC より小型軽量であるため，盗難にあうリスクも高い．盗難対策として，Android ではディスク暗号化の機能が提供されており，端末が盗まれた場合でもログインされなければディスクからの情報漏洩を防ぐことができる．

しかし，近年，コールドブート攻撃 [3] と呼ばれる攻撃手法が報告され，メモリからの情報漏洩が問題となってきている．コールドブート攻撃は，メモリを冷却した状態で端末をリセットし，別の OS を起動してメモリ上のデータを盗む攻撃である．Android 端末においてもコールドブート攻撃が報告されている [13]．Android OS では，ディスク上のデータをページキャッシュとしてメモリ上に保持することで，ファイルアクセスを高速化している．端末を盗まれてコールドブート攻撃が行われると，ページキャッシュ上のデータを盗み見られてしまう．その結果，ディスクが暗号化されていたとしてもディスク上の一部のデータがページキャッシュを通して漏洩してしまう危険がある．

本稿では，メモリ上のページキャッシュを暗号化することで Android 端末の盗難時においても情報漏洩を防ぐためのシステム Cache-Crypt を提案する．Cache-Crypt はディスク上のファイルをメモリに読み込んだ際などにページキャッシュの内容を暗号化する．アプリがファイルにアクセスするときだけページキャッシュを復号し，アクセスが終わったら再び暗号化する．Cache-Crypt を用いることで情報漏洩の対象をアクセス中のページキャッシュのみに限定することができる．ページキャッシュの暗号化・復号化に用いる暗号鍵はメモリ上に置かず，ARMORED [8] の手法を用いて，通常使われないデバック用の CPU レジスタ上に置くことで保護する．この手法により，コールドブート攻撃によりメモリ上の暗号鍵を盗まれ，暗号化したページキャッシュを復号されてしまうのを防ぐ．

我々は Cache-Crypt を Android OS に実装し，ページキャッシュに対してメモリページ単位で暗号化フラグを管理するようにした．アプリが read システムコールを発行した時に，Cache-Crypt はページキャッシュ上のデータを復号してアプリに返す．write システムコールを発行した時には，Cache-Crypt は書き込むデータを暗号化してページキャッシュに書き込む．アプリが mmap システムコールを発行してファイルをメモリマップした時には，Cache-Crypt は最初のアクセス時にページキャッシュのデータを復号する．Android エミュレータを用いてメモリをダンプする実験を行い，ページキャッシュの内容が暗号化されていることを確認した．また，Nexus 7 を用いて Cache-Crypt のオーバヘッドを測定し，簡易な暗号方式の下ではオーバヘッドが許容範囲内であることを確認した．

以下，2 章では，ページキャッシュからの情報漏洩の危険性について述べる．3 章ではこの問題を解決する Cache-Crypt について述べ，4 章でその実装の詳細について述べる．5 章で Cache-Crypt を用いて行った実験について述べる．6 章で関連研究について述べ，7 章で本稿をまとめる．

2. ページキャッシュからの情報漏洩

Android では端末の盗難対策としてフルディスク暗号化が提供されている．フルディスク暗号化とは，ディスクのパーティション全体を暗号化することでデータを保護する仕組みである．Android ではアプリのデータが置かれたパーティションのみが暗号化される．Android のフルディスク暗号化には Linux の dm-crypt が用いられており，ディスクからの読み込み時にデータの復号が行われ，ディスクへの書き込み時にデータの暗号化が行われる．暗号化されたパーティションを復号するには，OS 起動時に PIN を入力する必要がある．OS は入力された PIN から暗号鍵を生成し，dm-crypt による暗号化・復号化に用いる．そのため，PIN を知られない限り，ディスクを取り外して別の端末に取り付けたとしてもディスク上の情報を盗み見られることはない．

しかし，近年，メモリから比較的容易に情報を盗み出せるコールドブート攻撃 [3] が報告されている．コールドブート攻撃は，メモリを冷却した状態で端末を強制リセットし，攻撃者自身の OS で起動してメモリ上に残っているデータを盗み見る攻撃である．揮発性メモリ上のデータは通常，端末のリセットにより電源供給が途絶えている間に次第に破壊されていくが，データが完全に消えるまでには少し時間がかかる．メモリを冷却することでメモリ上のデータが破壊されるのを遅らせることができる．この攻撃手法は端末を強制リセットするため，通常のシャットダウンとは異なり，OS がメモリ上の機密情報を消去することはできない．

Android 端末においてもコールドブート攻撃が報告されている [13]．報告された攻撃手法では，端末全体を冷却した状態でバッテリを抜き差しして端末のリセットを行う．次に，USB 経由で PC から攻撃者のリカバリイメージをインストールし，そのリカバリイメージから起動する．リカバリイメージをインストールする際にブートローダをアンロックする必要がある場合でも，ディスク上のデータは消去されるが，メモリ上のデータは消去されない．リカバリイメージの OS はメモリ上のデータをダンプし，ディスク暗号化の鍵などを取り出すことができる．

Android OS は他の多くの OS と同様に，ファイルアクセスの高速化のためにディスク上のデータをメモリ上にキャッシュとして保持している．特に，ファイルの内容はページキャッシュと呼ばれるキャッシュに保持される．アプリがファイルを読み込む際には，まず，OS がディスク上のデータをメモリ上のページキャッシュに読み込む．そして，ページキャッシュ上のデータがアプリに返される．アプリが同じファイルにアクセスする時には，ディスクへのアクセスを行わず，ページキャッシュ上のデータを即座

© 2012 Information Processing Society of Japan

に返すことができる．

そのため，Android 端末がコールドブート攻撃を受けると，メモリ上のページキャッシュを盗み見られることになる．ページキャッシュ上にはディスクと同じデータ，もしくは，書き戻し前より新しいデータが置かれているため，フルディスク暗号化を行っていたとしてもディスクのデータの一部は漏洩してしまう．Android OS では空きメモリの多くはページキャッシュとして使われ，また，Android 端末のメモリ容量も増大してきているため，漏洩するデータ量も増大する傾向にある．

3. Cache-Crypt

本稿では，メモリ上のページキャッシュを暗号化することでコールドブート攻撃によるページキャッシュからの情報漏洩を防ぐシステム Cache-Crypt を提案する．

3.1 脅威モデル

本研究では端末を盗まれ，コールドブート攻撃によりページキャッシュ上のデータを盗み見られる攻撃を想定する．ただし，アプリのメモリなどからの情報漏洩は本研究の対象外とする．端末のディスクは暗号化されており，ディスクからの情報漏洩は起こらないものとする．また，端末に不正ログインされ，ログインした攻撃者に復号された後のディスクのデータを直接取得されることはないものとする．

3.2 Cache-Crypt

Cache-Crypt は OS がディスク上のファイルをメモリに読み込んだ際などにページキャッシュの内容を暗号化する．これを暗号化ページキャッシュと呼ぶ．アプリがアクセスする時だけ暗号化ページキャッシュの必要な領域を復号し，アクセスが終わったら再び暗号化する．Cache-Crypt を用いることでコールドブート攻撃による情報漏洩の対象を，端末がリセットされた瞬間にアプリがアクセス中であったページキャッシュのみに限定することができる．その他のページキャッシュについては暗号化されているため，メモリ全体を盗み見られても情報は漏洩しない．

アプリがファイルを読み込む際には，Cache-Crypt が暗号化ページキャッシュを復号し，そのデータをアプリのバッファにコピーする．アプリはバッファに格納された復号済みのデータを暗号化を意識せずに扱うことができる．読み込もうとしたファイルに対応するページキャッシュが存在しない時は，暗号化ディスクからページキャッシュ上にデータを読み込む．この時に，フルディスク暗号化と Cache-Crypt の暗号方式および暗号鍵を共通にしておくことにより，暗号化ディスク上のデータをそのままページキャッシュに読み込むことができる．これにより，暗号化ディスク上のデータを一旦復号して，再度，Cache-Crypt 用に暗号化し直す必要がなくなり，性能を向上させることができる．また，一時的に復号されたデータがメモリ上に置かれることもなくなり，セキュリティを向上させることができる．

アプリがファイルに書き込む際には，Cache-Crypt がアプリのバッファのデータを暗号化し，暗号化ページキャッシュに書き込む．アプリは暗号化を意識せずにバッファのデータを書き込むことができる．既存のファイルの一部を書き換える場合は，まず，暗号化ディスクから暗号化ページキャッシュにデータを読み込んでから上書きを行う．書き換えられた暗号化ページキャッシュ上のデータは，適切なタイミングで暗号化ディスクに書き戻される．この際に，フルディスク暗号化と Cache-Crypt を連携させることにより，ページキャッシュ上のデータをそのままディスクに書き込むことができる．

アプリがファイルをメモリマップしてアクセスする際には，Cache-Crypt がそのファイルに対応するページキャッシュを復号する．ファイルをメモリマップすると，ページキャッシュがアプリのアドレス空間にマップされ，アプリが OS を介さずに直接ページキャッシュにアクセスできるようになる．そのため，アプリがファイルをアンマップするまではページキャッシュを復号したままにする．メモリマップされたファイルが書き換えられた場合，Cache-Crypt はページキャッシュ上のデータの暗号化を行った上で暗号化ディスクに書き戻す．ファイルのアンマップ時には再びページキャッシュを暗号化する．

ページキャッシュの暗号化を行うための暗号鍵は既存研究の ARMORED [8] の手法を用いて保護する．暗号鍵は通常，メモリ上に置かれるが，コールドブート攻撃が行われるとメモリ上の暗号鍵も盗まれてしまう．その結果，暗号化したページキャッシュも復号されてしまうことになる．ARMORED では，CPU レジスタに暗号鍵を保持することによって，コールドブート攻撃を受けても暗号鍵が漏洩するのを防ぐことができる．

4. 実装

我々は Cache-Crypt を Android のカーネル 3.4 に実装した．現在の実装では，フルディスク暗号化の dm-crypt との連携は行わず，個別に暗号化・復号化を行っている．

4.1 暗号化フラグ

ページキャッシュの状態を管理するために，ページ構造体に暗号化フラグを追加した．ページ構造体はページキャッシュとして使われているメモリページを含め，すべてのページを管理するためのカーネルデータ構造である．Cache-Crypt は暗号化フラグを用いて，ページキャッシュが暗号化されているか，および，プロセスにメモリマップされているかを判断する．暗号化フラグの値は表 1 のいず

© 2012 Information Processing Society of Japan

表 1 ページキャッシュの状態を表すための暗号化フラグ

暗号化フラグ	ページキャッシュの状態
ENCRYPT	暗号化されている
DECRYPT	暗号化されていない
MEMORY_MAP	メモリマップされている

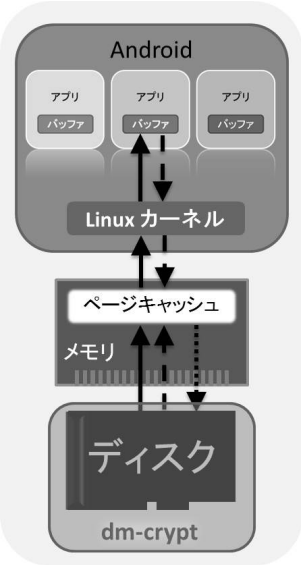

図 1 ファイルの読み書きの流れ

れかとなる．

4.2 ファイルの読み込み

図 1 に示すように、プロセスが read システムコール等を発行してファイルを読み込む時に，対応するページキャッシュが存在すればそのデータをプロセスから渡されたバッファにコピーする．この処理はファイルシステムに共通の VFS 層で行われる．その際に，ページキャッシュの暗号化フラグが ENCRYPT であればページキャッシュを復号し，暗号化フラグを DECRYPT にしてからデータをコピーする．コピーが終わるとデータを暗号化し，暗号化フラグを ENCRYPT にする．ただし，この暗号化は，暗号化フラグが MEMORY_MAP ではない，かつページキャッシュがファイルシステムのメタデータを保持していないという条件のもとで行う．メタデータかどうかは，ページキャッシュに対応するファイルの inode 番号が 0 かどうかで判断する．メタデータを暗号化するとカーネル内での処理を行うのが難しくなる．その上，メタデータに機密情報が含まれることは少ない．

一方，ファイルの読み込み時にページキャッシュが存在しない場合は，ディスクからページキャッシュにデータを読み込む．この際に，読み込んだデータを暗号化し，暗号化フラグを ENCRYPT にする．dm-crypt と連携して暗号化されたデータを直接ページキャッシュに読み込むことができれば，この暗号化は不要となる．

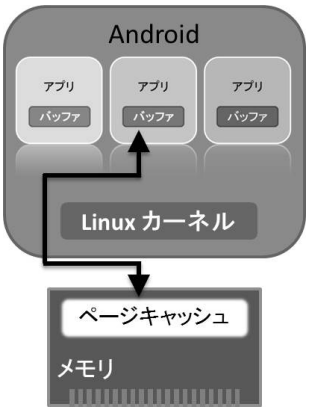

図 2 メモリマップされたファイルのアクセス

4.3 ファイルへの書き込み

プロセスが write システムコール等を発行してファイルに書き込む時は，プロセスから渡されたバッファからページキャッシュにデータをコピーする (図 1)．この処理も VFS 層で行われる．この際に，ページキャッシュの暗号化フラグが ENCRYPT であった場合は復号してからデータをコピーする．その後，ページキャッシュを暗号化し，暗号化フラグを ENCRYPT にする．この暗号化はファイルの読み込みと同様に，暗号化フラグが MEMORY_MAP ではない，かつメタデータを保持していないという条件の下で行う．

書き込み時にページキャッシュが存在しない場合には，DECRYPT の状態のページキャッシュを確保してから書き込みを行う．ディスク上に対応するファイルブロックが存在する場合には，4.2 節と同様にしてファイルをページキャッシュに読み込んでから書き込みを行う．

4.4 ファイルの書き戻し

ページキャッシュが書き換えられると，ディスク上のデータと同期するために適切なタイミングでページキャッシュ上のデータをディスクへ書き戻す (図 1)．ディスクへ書き戻す際は，ページキャッシュの暗号化フラグが ENCRYPT であった場合は復号して暗号化フラグを DECRYPT にする．ディスクへの書き戻しが完了するとコールバック関数が呼ばれるため，そこでページキャッシュを暗号化し，暗号化フラグを ENCRYPT にする．暗号化フラグが MEMORY_MAP の場合やメタデータを保持している場合にはこの暗号化は行わない．この処理は Android で標準的に用いられている ext4 内で行われているため，ファイルシステム依存となるが ext4 に実装した．

4.5 ファイルのメモリマップ

プロセスが mmap システムコールを発行してファイルをメモリマップした場合，ページキャッシュを暗号化していると正常な動作が行えなくなる．ファイルをメモリマップ

© 2012 Information Processing Society of Japan

すると，ディスク上のファイルがページキャッシュに読み込まれ，そのページキャッシュがプロセスのアドレス空間に直接マップされる．これによりメモリへのアクセスと同様にしてファイルにアクセスすることができ，高速なファイルアクセスやプロセス間でのデータ共有などができる．しかし，メモリマップされたページキャッシュは図 2 のように OS を介さずに直接アクセスされるため，アクセス時だけ復号するのは難しい．

そこで，Cache-Crypt では，メモリマップされたページキャッシュにプロセスが初めてアクセスした時にページキャッシュを復号する．mmap システムコールを実行した時点ではページキャッシュはまだマップされておらず，プロセスが最初にアクセスした時にページフォールトが発生する．そして，ファイルの読み込みおよびページキャッシュのマップが行われる．ページフォールトの際にページキャッシュの暗号化フラグが ENCRYPT であれば復号し，暗号化フラグを MEMORY_MAP にする．このページキャッシュはメモリマップが行なわれている間は復号されたままにする．ファイルがアンマップされたらページキャッシュを暗号化し，暗号化フラグを ENCRYPT にする．

4.6 暗号鍵の保護

Cache-Crypt ではコールドブート攻撃を受けても暗号鍵が漏洩しないように，既存研究の ARMORED [8] を用いて 128 ビットの暗号鍵をデバック用の CPU レジスタに格納する．デバック用のレジスタはデバック時以外は使用されないため，通常の実行には支障はない．コールドブート攻撃を行なう際は端末をリセットする必要があるが，CPU レジスタは端末のリセット時に初期化されるため，リセット後にレジスタから暗号鍵を取り出すことはできない．ARMORED ではメモリを使わずに AES の暗号処理を行うために，CPU の SIMD 拡張命令セットが提供するレジスタをメモリの代わりに用いる．これにより，暗号処理の途中結果がコールドブート攻撃により漏洩することも防ぐことができる．ARMORED は Linux カーネルのパッチとして提供されているが，Cache-Crypt ではまだ実装に利用していない．

5. 実験

図 3 に示すように、Android エミュレータを用いてページキャッシュの暗号化を確認する実験を行った．次に，Android 端末の Nexus 7 (2013) を用いてファイルの読み書きにおける Cache-Crypt のオーバヘッドの測定を行った．Android のカスタムカーネルとして，Linux カーネル 3.4 に Cache-Crypt を実装したものを用いた．Android エミュレータには Android 4.4.2 の ADT に含まれるものを用い，カスタムカーネルを指定して起動した．Nexus 7 (2013) で

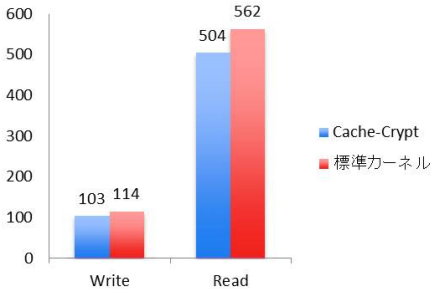

図 3 ベンチマーク結果

は，まず Nexus 7 用のファクトリイメージをダウンロードし，boot.img を abootimg コマンドでカーネルとそれ以外に分けた．次に，カーネルだけカスタムカーネルに入れ替えて abootimg コマンドで再び boot.img に戻した．そして，Nexus 7 をアンロックして boot.img を fastboot コマンドを使用して起動した．

5.1 ページキャッシュ暗号化の確認

Android のメモリ上のデータをダンプできるツールである Lime [12] を使用して，ページキャッシュ上のデータの確認を行った．まず，エディタアプリを用いて'A' の文字が大量に書かれたファイルを作成した．次に，Lime のカーネルモジュールを用いてメモリをダンプし，strings コマンドを用いてダンプファイルから文字データを抜き出した．その中からエディタで書き込んだ大量の'A' の文字を検索したが，大量の'A' の文字は発見されなかった．比較のために，Cache-Crypt を実装していない標準カーネルでも同様の実験を行った．その結果，大量の'A' の文字が発見された．以上より，Android 上のエディタで作成したファイルの内容が，ページキャッシュ上で暗号化されていることが確認できた．

5.2 オーバヘッドの測定

Cache-Crypt のオーバヘッドの測定には Benchmark というアプリを用いた．このアプリは，様々なベンチマークを提供しているが，本実験では空のファイルに 1MB の書き込みを行った後，同じファイルから 1MB の読み込みを行うベンチマークを利用した．Cache-Crypt を実装したカスタムカーネルと実装していない標準カーネルでそれぞれ 10 回ずつファイルの書き込みと読み込み速度を測定した．ただし，実装上の問題により，書き込みおよび書き戻しは暗号化していない．図 3 に書き込み速度と読み込み速度の平均値を示す．この結果より，ファイルの読み書きともに，Cache-Crypt によるオーバヘッドは 10 %程度であった．ただし，現在の実装では暗号方式に簡易な XOR を用いているため，暗号化のオーバヘッドは小さくなっている．ARMORED の手法を用いて CPU レジスタのみで AES の暗号化を行うと，このオーバヘッドは大きく増加すると考

© 2012 Information Processing Society of Japan

えられる．

6. 関連研究

暗号化ファイルシステムはディスクからの読み込み時に暗号化されたファイルを復号し，ディスクへの書き込み時にファイルを暗号化する．そのため，一般的にページキャッシュは暗号化されない．TransCrypt [11] では設定でページキャッシュを暗号化することもできると論文中で述べられているが，詳細については述べられておらず，メモリマップへの対処法も不明である．デバイスマッパを用いて実装し直された TransCrypt [11] ではページキャッシュの暗号化については考慮されていない．eCryptfs [6] は既存のファイルシステムの上に重ねて利用する暗号化ファイルシステムであり，既存のファイルシステムが用いるページキャッシュは暗号化される．しかし，eCryptfs が用いるページキャッシュは暗号化されない．EncFS [7] ではファイルをオープンしている間だけ独自のキャッシュが作られるが，それは暗号化されない．

ZIA [1] はハードウェアトークンが近くにある時だけファイルを復号するファイルシステムである．ZIA ではファイルを暗号化するのに用いた鍵をトークンと通信して復号し，復号した鍵を用いてファイルを復号する．端末が盗まれるなどしてトークンが近くにない状態になると，ZIA は復号された暗号鍵を破棄し，ページキャッシュを暗号化する．しかし，トークンを持ったユーザが端末から離れている間はメール受信などファイルシステムを使う処理をバックグラウンドで行うことができない．また，端末とトークンの両方が盗まれた場合，ZIA はファイルを保護できない．

Keypad [4] は盗まれやすい端末のためのファイルシステムである．Keypad では，端末が盗まれた後でアクセスされたファイルが分かり，また，それ以降のファイルアクセスを禁止することができる．Keypad はファイル単位で暗号化を行い，暗号鍵をサーバに保存する．そのため，ネットワークに接続されていなければファイルを復号できず，安全に暗号化を行うこともできない．もう一台の端末を用いる手法も提案されているが，対象がスマートフォンの場合にはそれ以外の端末を携帯するかどうかは疑問である．Keypad ではページキャッシュも暗号化されるようであるが，FUSE を用いて実装されており，ページキャッシュの扱いの詳細については不明である．

スワップ暗号化 [10] はディスクにスワップアウトされるページを暗号化し，スワップインされる時に復号する．短い期間だけ有効な暗号鍵を用いて暗号化を行うことにより，プロセスが終了した後はスワップアウトによってディスクに書き込まれたデータを復号することができなくなる．Cache-Crypt ではページキャッシュを暗号化しているため，スワップアウトされるページは暗号化済みである．

Cryptkeeper [9] はソフトウェアによる暗号化を行う仮想メモリマネージャである．Cryptkeeper はメモリを小さなワーキングセットとそれ以外の部分に分け，後者を暗号化する．暗号化されたページはアクセス禁止にされ，アクセスされてページフォールトが発生した時に復号される．暗号化されないページ数が上限を超える場合には，その中で最初に復号されたページが暗号化される．Cryptkeeper は，プロセスのメモリのみを暗号化し，カーネルのメモリは暗号化しないため，ページキャッシュは暗号化されない．また，暗号鍵はメモリ上に置かれ，コールドブート攻撃対策が行われていない．

Lacuna [2] は，プライベートセッションでアプリケーションを実行し，セッションが終わると実行に関する全てのメモリを消去するシステムである．Lacuna は，プロセス間通信が制限された VM 上でアプリケーションを実行する．外部デバイスとはエフェメラルチャネル経由で接続される．ハードウェアチャネルを用いる場合はゲスト OS のデバイスドライバが直接アクセスし，QEMU がエミュレーションする場合はホスト OS 内を通るデータを暗号化する．これにより，アプリケーションの終了時に消去すべきデータが存在する範囲を限定する．

Vanish [5] や CleanOS [14] は一定時間たつと機密データを暗号化して，暗号化に用いた鍵をインターネット上に格納する．CleanOS [14] は Android OS を拡張し，SDO と呼ばれる Java オブジェクトに機密データを格納する．独自の GC を用いて SDO のデータが頻繁に使われているかどうかを判定し，使われていなければ暗号化して鍵をクラウド上に格納する．そのため，ネットワークに接続されていない時は SDO を暗号化するまでの時間を長くしたり，暗号鍵を端末に保持させたりする必要があり，情報漏洩の危険性が増す．

7. まとめ

本稿では，ページキャッシュを暗号化することで Android 端末の盗難時においても情報漏洩を防ぐシステム Cache-Crypt を提案した．Cache-Crypt では，OS がディスクのデータを読み書きする際に作られるページキャッシュを暗号化し，アプリがアクセスする際にだけ復号する．これにより，コールドブート攻撃による情報漏洩の対象をアプリがアクセス中のページキャッシュに限定することができる．暗号化に用いる暗号鍵は ARMORED の手法を用いて CPU レジスタに格納することでコールドブート攻撃から保護する．Cache-Crypt を Android OS に実装し，メモリ上のページキャッシュが暗号化されていることを確認した．さらに，ファイルの読み書きにおける Cache-Crypt のオーバヘッド簡易な暗号方式を用いた場合に 10 %程度であることを確認した．

現在の実装では，暗号方式に簡易な XOR を用いているため，今後は AES を用いて Cache-Crypt を実装し，オー

© 2012 Information Processing Society of Japan

バヘッドを測定する．また，現在のところ，Cache-Crypt とフルディスク暗号化の連携を行えていないため，暗号化ディスクの読み書きの際に同じ暗号方式を用いていない．Cache-Crypt とフルディスク暗号化に同じ暗号方式および同じ暗号鍵を用いてディスクの読み書きの際の復号化と暗号化の回数を減らすことを含めて，Cache-Crypt のオーバヘッドの削減をすることが今後の課題である．

参考文献

[1] M. D. Corner and B. D. Noble. Zero Interaction Authentication. In *Proc. of the ACM Annual International Conference on Mobile Computing and Networking*, 2002.

[2] A. M. Dunn, M. Z. Lee, S. Jana, S. Kim, M. Silberstein. Y. Xu, V. Shmatikow, and E. Witchel. Eternal Sunshine of the Spotless Machine: Protecting Privacy with Ephemeral Channels. In *Proc. of the 10th USENIX Symposium on Operating Systems Design and Implementation*, 2012.

[3] J. Halderman et al. Lest We Remember: Cold Boot Attacks on Encryption Keys. In *Proc. USENIX Security Symposium*, 2008.

[4] R. Geambasu, J. P. John, S. D. Gribble, T. Kohno, and H. M. Levy. Keypad: An auditing file system for theft-prone devices. In *Proc. of the ACM European Conference on Computer Systems (EuroSys)*, 2011.

[5] R. Geambasu, T. Kohno, A. levy, and H. M. Levy. Vanish: Increasing data privacy with self-destructing data. In *Proc. of USENIX Security*, 2009.

[6] M. A. Halcrow. eCryptfs: An Enterprise-class Encrypted Filesystem for Linux. In *Proc. of the Linux Symposium*, pages 2495–2503, 2005.

[7] Taylor Hornby. EncFS Encrypted Filesystem. `https://defuse.ca/audits/encfs.htm`, 2 2014.

[8] J. Götzfried and T. Müller. ARMORED CPU-bound Encryption for Android-driven ARM Devices. In *Proc. of the 8th ARES Conference (ARES 2013)*, 2013.

[9] P. A. H. Peterson. Cryptkeeper: Improving security with encrypted RAM. In *Proc. of the IEEE International Conference on Technologies for Homeland Security (HST)*, 2010.

[10] N. Provos. Encrypting Virtual Memory. In *Proc. of the 9th USENIX Security Symposium*, 2000.

[11] S. Sharma. TransCrypt: Design of a Secure and Transparent Encrypting File System. Master's thesis, Indian Institute of Technology Kanpur, 2006.

[12] Joe Sylve. LiME - Linux Memory Extractor. `https://code.google.com/p/lime-forensics/`, 2012.

[13] T. Müller and M. Spreitzenbarth and F. C. Freiling. FROST – Forensic Recovery of Scrambled Telephones. In *Proc. of The 11th International Conference on Applied Cryptography and Network Security (ACNS)*, 2013.

[14] Y. Tang, P. Ames, S. Bhamidipati, A. Bijlani, R. Geambasu, and N. Sarda. CleanOS: Limiting Mobile Data Exposure with Idle Eviction. In *Proc. of the 10th Symposium on Operating Systems Design and Implementation*, 2012.

© 2012 Information Processing Society of Japan