暗号モジュールを搭載したCPUによる効率的な暗号処理方法の提案と実装

金子 洋平† 齋藤 孝道‡

†明治大学大学院
214-8571 神奈川県川崎市多摩区東三田 1-1-1
ce36017@meiji.ac.jp

‡明治大学
214-8571 神奈川県川崎市多摩区東三田 1-1-1.
saito@cs.meiji.ac.jp

あらまし近年，モバイル端末において暗号処理を行う機会が増えている．この処理は汎用プロセッサにとって負荷が高く，特に，モバイル端末用のCPUでは，暗号処理速度の低下などの原因となる．この負荷を軽減する方法として，暗号処理に最適化されたハードウェアモジュールを搭載したプロセッサを利用し，暗号処理をオフロードする方法がある．そこで本論文では，モバイル端末上で暗号処理のオフロードを行うためにAM3358プロセッサを採用し，このプロセッサに搭載される暗号モジュールにオフロードを行うことに加え，CPUも同時に利用することで，モバイル端末における暗号処理速度の高速化を図り，その評価を測定した．

Proposal and Implementation of Efficient Cryptographic Operation Using Cryptographic Module on CPU

Yohei Kaneko† Takamichi Saito‡

†Graduate School of Meiji University
1-1-1, Higashimita, Tama-ku, Kawasaki-shi, Kanagawa 214-8571, JAPAN
ce36017@meiji.ac.jp
‡Meiji University
1-1-1, Higashimita, Tama-ku, Kawasaki-shi, Kanagawa 214-8571, JAPAN
saito@cs.meiji.ac.jp

Abstract In recent years, it is popular to utilize cryptographic processing on mobile computer. This kind of processing is high load to CPU. Especially, it is burden for CPU on mobile computer. In this paper, we propose to offload cryptographic processing between AM3358 processor and CPU simultaneously. We also evaluate the performance of our implementation.

1　はじめに

モバイル端末において，個人情報など秘匿性が求められる情報を保護するためにローカルストレージ上のファイルの暗号処理を行う機会が増えている．しかし，この処理は汎用的な処理を行うプロセッサコアにとって負担が大きく，暗号処理速度の低下などの原因となる．特にモバイル端末に搭載される CPU は，一般的なサーバ等に搭載される CPU よりも性能が低く，暗号処理のボトルネックとなる ．

そこで，暗号化・復号処理をオフロードする目的で，暗号処理を専門に行うハードウェアモジュール（以降，暗号モジュールと呼ぶ）をコア内に持つ CPU が数多く登場した [1][2]．また，その種の CPU における効率的な暗号化・復号処理のオフロード技術についての研究も行われている [3][4]．

Texas Instruments 社の AM3358 プロセッサ[5]は，ARM Cortex-A8 コアベースで，共通鍵暗号化方式やハッシュ処理などをサポートする暗号モジュールを搭載したシングルコア CPU である．本論文では，AM3358 プロセッサ用の OS として Linux を採用した．しかし，Linux は，直接暗号モジュールを利用するための仕組みが提供されていない．本論文では，OCF（OpenBSD Cryptographic Framework）[6] 及び AM3358 プロセッサ用の暗号モジュールのデバイスドライバを利用した．これを用いて，本論文では独自に，暗号処理を暗号モジュールにオフロードするだけではなく，適宜 CPU も同時に利用することで暗号処理の高速化を目指す．また，その評価を行った．

2　研究環境

2.1　AM335x プロセッサ

本論文で利用した AM3358 プロセッサのアーキテクチャを図 1 に示す．

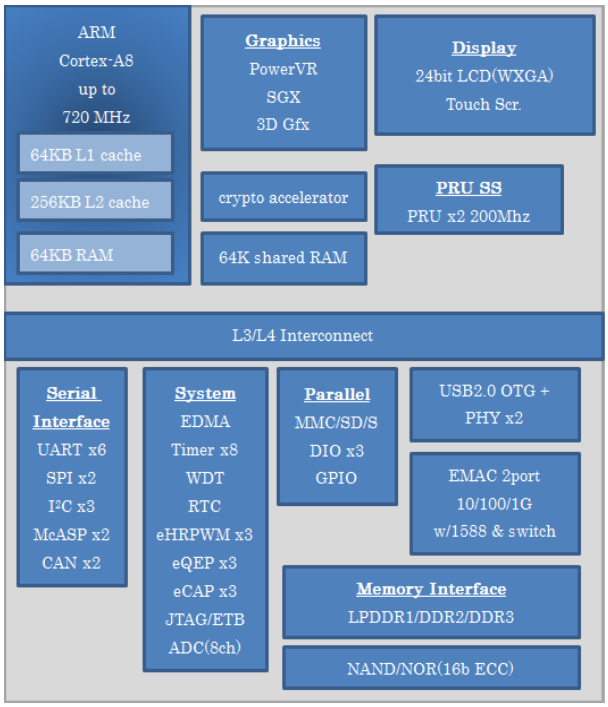

図 1：AM3358 プロセッサ

AM3358 プロセッサは，ARM Cortex-A8 アーキテクチャベースのシングルコアプロセッサである．NEON 命令に対応しており，キャッシュメモリは，L1 キャッシュはデータキャッシュが 32KB，命令キャッシュが 32KB で，いずれも 1 ビットエラー検出可能である．L2 キャシュは 256KB 搭載していて，ECC に対応している．また，動作周波数は，275MHz，500Mhz，600Mhz，720Mhz に対応している．DRAM は，LPDDR-400，DDR2-532，DDR3-606 に対応しており，本論文で AM3358 プロセッサの評価に使用した評価ボード TMDXEVM3358 では，DDR2 を 512MB 搭載している．稼働させることが可能な OS は，Linux，Android，Windows Embedded CE である．

暗号モジュールは，AM3358 プロセッサに 1 つ搭載されていて，共通鍵暗号化方式やハッシュ関数に対応しており，AES，SHA，MD5 のアルゴリズムと RNG（Random Number Generator）を利用可能である．

2.2 OCF

2.2.1 OCF の概要

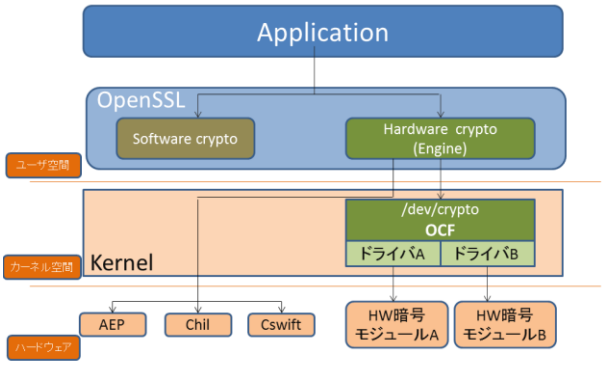

図 2:暗号処理専用モジュールの利用

OCF（図 2）とは，OpenSSL [7]，OpenSSH[8]などのアプリケーションから様々なハードウェアアクセラレータが提供する暗号処理機能を利用するための共通のインターフェースを提供する API と，ハードウェアアクセラレータのデバイスドライバから構成されたミドルウェアである．ただし，OCF は BSD や Linux のカーネルミドルウェアとして開発されているため，他の OS で利用するには，その環境に合わせた改修が必要となる．また，OCF がサポートしていない暗号モジュールに対しては，それに対応するデバイスドライバを用意する必要がある．

2.2.2 OCF の利用方法

OCF の利用について，まず，アプリケーションから OCF へのアクセス方法について説明する．アプリケーションはキャラクタ型のデバイスノードに対して，システムコールを発行することで，各システムコールに対応する OCF 内で定義された関数を呼び出すことができる．これは，標準的な file operations 構造体を利用しており，アプリケーションは open()，ioctl()システムコールを呼び出すことで，OCF へアクセスする．

暗号処理を，OCF を経由して暗号モジュールにオフロードする場合には，まず，アプリケーションと OCF 間で，後述するセッションを生成する．セッションの生成後，アプリケーションが OCF に対して，暗号処理の実行を指示することで，暗号モジュールの暗号処理機能を利用できる．暗号処理の終了後，アプリケーションと OCF 間のセッションを解放する．

2.2.3 OCF の制御

OCF を制御するためには，アプリケーションから制御リクエストを引数として ioctl()システムコールを/dev/crypto に対して発行する．その OCF への制御リクエストには，以下の 3 種類があり，次のように使い分ける．

CIOCGSESSION

アプリケーションと OCF 間でセッションの生成を行う際の制御リクエストである．セッションを生成すると，セッション ID を返り値としてアプリケーションへ渡す．このセッションとは，アプリケーションと OCF 間で暗号鍵や暗号化方式などの暗号情報と OCF 内の暗号情報を格納した構造体への識別子であるセッション ID を共有した状態である．

CIOCCRYPT

暗号処理をアプリケーションから暗号モジュールにオフロードする際の制御リクエストである．この制御命令を発行すると，まず，アプリケーションから OCF にセッション ID，IV と平文を転送する．OCF 側では，セッション ID により，暗号情報を格納した構造体を取得する．その後，暗号モジュールに暗号処理をオフロードする．

CIOCFSESSION

アプリケーションと OCF 間のセッションを解放する際の制御リクエストである．セッションの解放とは，暗号情報を格納した構造体の削除とセッション ID の削除を行うことである．

2.3 OpenSSL

OpenSSL(図 2)は，SSL や TLS だけでなく，証明書の発行といった PKI（Public Key Infrastructure）関連の処理や公開鍵暗号化

方式や共通鍵暗号化方式などを容易なインターフェースで利用可能としたAPIライブラリを含むツールキットである．特に，共通鍵暗号化方式とハッシュ関数については，共通のインターフェースでの利用を可能とするEVP APIを提供している．

また，OpenSSLは，AEP，Chilや Cswiftをはじめとした様々なハードウェアアクセラレータに対応しており，アプリケーションからそれらを利用するためにENGINE APIを提供している．

2.3.1 OpenSSLからOCF利用の手続き

OpenSSLからOCFを利用するには，ENGINE APIに用意されたオブジェクト（以降，ENGINEオブジェクトと呼ぶ）を利用し，図 3 に示す所定の手続きを行う必要がある．

```
1.ENGINE *e;
2.ENGINE_load_cryptodev();
3.if(!(e=ENGINE_by_id("cryptodev")));
4.        /*エラー処理*/
5.else if(!ENGINE_set_default(e,ENGINE_METHOD_ALL));
6.        /*エラー処理*/
```

図 3：OpenSSLからのオフロードの手続き

図 3 の 2 行目でENGINE_ load_ cryptodev関数を呼び出すと，この関数内で，OCF に対応する ENGINE オブジェクトを生成し，それらをENGINE オブジェクト専用のリスト（以降，ENGINEリストと呼ぶ）に登録する．3行目では，暗号処理モジュールの識別子を引数として指定してENGINE_ by_ id関数を呼び出し，引数に対応したENGINE オブジェクトを ENGINE リストから取得する．ここでは，引数に指定した″cryptodev″に対応する ENGINE オブジェクトを取得している．5行目のENGINE_ set_ default関数を呼び出すことで，暗号処理モジュールが対応している暗号アルゴリズムを ENGINE オブジェクトに登録する．

これらを利用することで，アプリケーションがEVP API を用いて暗号処理を実行する際に，OCF に暗号処理を実行させることができる．

2.3.2 OpenSSL暗号処理の受け渡し

図4に，OpenSSLからEVP APIを用いてOCFに暗号処理を受け渡す一連の処理を示す．図4のEVP_ Encryptinit関数，EVP_ Ecrypt・Update/Final関数，EVP_ CIPHER_ CTX_ Cleanup関数は，それぞれOCF内部のCIOCGSESSION，CIOCCRYPT，CIOCFSESSION制御リクエストに対応している．

```
1.EVP_CIPHER_CTX ctx;
2.EVP_Encryptinit(&ctx, EVP_des_cbc(), key, iv);
3.
4.EVP_EncryptUpdate(&ctx, output, &outlen, input, len);
5.EVP_EncryptFinal(&ctx, outbuf+outlen, &tmplen);
6.
7.EVP_CIPHER_CTX_cleanup(&ctx);
```

図 4：ENGINEを利用した暗号処理

OpenSSL が OCF に暗号処理を受け渡すには，まず，EVP_EncryptInit 関数を呼び出す．この関数内で CIOCGSESSION 制御リクエストが発行され，暗号方式，鍵などの暗号情報をOCFに受け渡され，OCFとのセッションを生成される．次に，EVP_ EncryptUpdate 関数と EVP_EncryptFinal 関数が呼び出され，暗号処理が行われる．この関数内で CIOCCRYPT 制御リクエストが発行され，OCF に暗号処理を受け渡される．暗号処理が終わると，EVP_ CIPHER_ CTX_Cleanup 関数を呼び出される．この関数内でCIOCFSESSION 制御リクエストを発行することで，OCFとのセッションが解放される．

3 提案システム

3.1 提案システムの概要

暗号処理自体は，Texas Instruments社から提供されているAES，SHA，MD5のOCFと暗号モジュール用のデバイスドライバを組み込み，OpenSSLライブラリから暗号処理をオフロ

ードした.

本論文では，アプリケーション上で行われる暗号処理を暗号モジュールにオフロードするだけではなく，それと同時に CPU でも暗号処理を行う．これにより，暗号モジュール単体で暗号処理を行う場合に比べシステム全体での暗号処理速度の高速化を期待する．

3.2 提案システムの実装

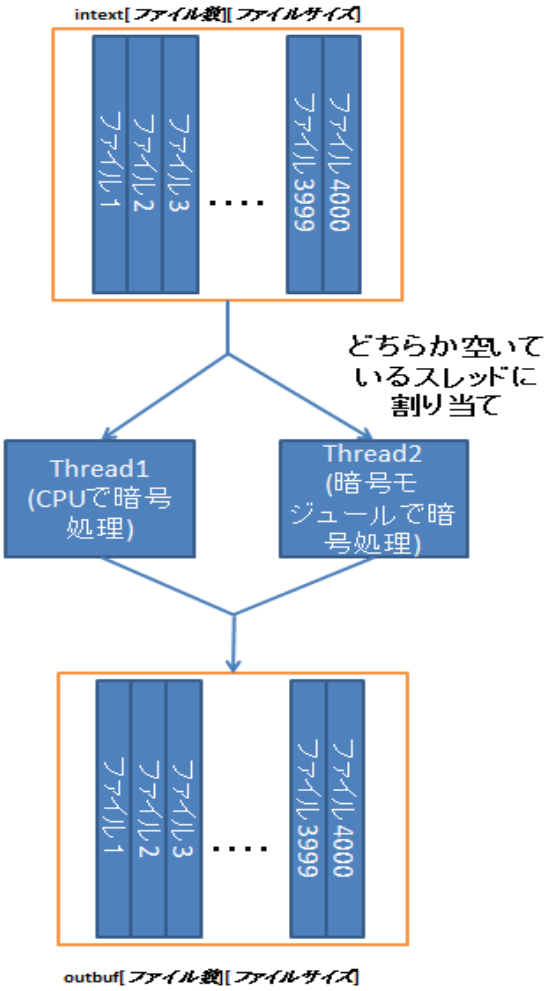

図 5:提案システムの概念図

提案システムでは，図 5 に示す通り，1 度に最大 4000 個の暗号化するファイルを 2 次元配列 intext に読み込む．読み込まれた最大 4000 個のファイルは，暗号モジュールにオフロードし暗号処理を行うスレッドの thread1 と，CPU を使い暗号処理をするスレッドの thread2 のいずれか空いている方に割り振り暗号処理を行う．暗号処理されたファイルは 2 次元配列 outbuf に格納された後，ファイルとして書き出される．

これらの処理は，主に，以下の機能により実現される．

1. 暗号モジュールにオフロードすることで暗号処理をするスレッド thread1 と，CPU で暗号処理をする thread2 の 2 つのスレッドの生成
2. 生成した 2 つのスレッドでそれぞれ暗号処理を行う関数．

スレッドの生成は次の図 6 の通りである．提案システムでは，pthread を用いた．

```
267     // スレッド1,2の作成と起動↓
268     pthread_create( &th1, NULL, thread1, (void *)NULL );
269     pthread_create( &th2, NULL, thread2, (void *)NULL );
270   ↓
271     // スレッド終了を待つ↓
272     pthread_join( th1, NULL );↓
273     pthread_join( th2, NULL );↓
```

図 6:スレッドの生成

268 行目と 269 行目の pthread_create 関数で、関数 thread1, thread2 をそれぞれ別のスレッドとして生成する．また，272 行目と 273 行目では，それぞれのスレッドが終了した時に待ち合わせを行っている．

次に，実際に CPU で暗号処理を行うスレッドと暗号モジュールを使い暗号処理を行うスレッドの実装の一部を図 7 及び図 8 に示す．

```
74 int do_crypt()↓
75 {↓
76 ENGINE *e;↓
77   ENGINE_load_cryptodev();↓
78   ↓
79   if(!(e = ENGINE_by_id("cryptodev"))){↓
80     fprintf(stderr,"error finding specified ENGINE¥n");
81     ENGINE_free(e);↓
82       exit(-1);↓
83     }↓
84   else if(!ENGINE_set_default(e,ENGINE_METHOD_ALL))↓
85       fprintf(stderr,"error using ENGINE¥n");↓
86   else{}↓
```

図 7:暗号モジュール利用の手続き

CPU で暗号処理を行うスレッドと暗号モジュールを使うスレッドの暗号処理を行う部分は，do_crypt 関数と do_crypt2 関数であり，それぞれスレッドとして起動された関数 thread1，thread2 から呼び出される.

do_crypt 関数は，暗号モジュールにオフロードを行い，暗号処理を行う関数である．そのため，図 7 に示す通り，図 3 と同様の OpenSSL から OCF を利用するための ENGINE オブジェクトを定義する手続きが必要になる．対して，do_crypt2 関数は CPU を使い暗号処理を行う関数であるので，図 7 で示した ENGINE オブジェクトの定義は行わず，図 8 の OpenSSL の処理を行う.

OpenSSL で暗号処理を行う部分は図 8 のようになっている.

```
 EVP_CIPHER_CTX ctx;↓
 EVP_EncryptInit(&ctx, EVP_aes_256_cbc(), (unsigned char *)key, (unsigned char *)iv);↓
↓
 for( ; loop<nof; ){//ここではthread2が既に計測を始めているかもしれないのでloopを初期化しない
   if(!EVP_EncryptUpdate(&ctx, outbuf[loop],&outlen,intext[loop],len[loop]))↓
     return 0;↓
//  dumpData( outbuf, len);↓
↓
↓
 if(!EVP_EncryptFinal(&ctx, outbuf[loop] + outlen, &tmplen))↓
     return 0;↓
loop++;↓
}↓
 EVP_CIPHER_CTX_cleanup(&ctx);↓
↓
 return 1;↓
}↓
```

図 8:OpenSSL での暗号処理

暗号処理自体の流れは，2.3.2 で説明したとおりである．do_crypt 関数と do_crypt2 関数はこの部分では同一であるが，暗号モジュールを利用するための ENGINE オブジェクトの定義の有無により，暗号処理に暗号モジュールか CPU の使用するかを決定する.

4 評価

4.1 評価環境

提案システムの評価を行うために，以下に示す環境を用意した．表 1 に示す環境上で，多数のファイルの暗号化のみの処理を行い，処理時間を計測した．今回は，「CPU のみでの暗号化」した場合（「CPU」と表記），「暗号モジュールへのオフロードのみでの暗号化」した場合（「暗号モジュール」と表記），及び，提案システムである「オフロードすると同時に CPU も使って暗号化」した場合（「暗号モジュールと CPU の同時利用」と表記）の 3 つの場合を比較した.

表 1:評価環境

CPU	AM3358 プロセッサ
メモリ	512MB DDR2
Linux カーネル	Linux3.2
OpenSSL	OpenSSL 1.0.0e
OCF	ocf-linux-20120127
暗号方式	aes-256-cbc

4.2 計測方法

ファイルサイズが，5 バイトから 7K バイトの間となるような 9 種類を用意した．ファイルの内容は乱数列とし，ファイルサイズごとにダミーファイルを 4000 個用意した．評価において，ファイルサイズごとにそれぞれ計測した．また，ファイルの入出力処理は時間に含めず，暗号処理のための thread1 及び thread2 を生成してから両方のスレッドが終了するまでの時間を計測した.

計測値は，それぞれのケースで 5 回試行し，それらの平均値とした.

4.3 計測結果

3.2 で説明した提案システムによる暗号処理時間の計測結果について表 2 及び図 9 に示す.

計測結果より，個々のファイルのサイズが 1000 バイトより小さい時には「CPU のみ」を使

って暗号化した場合がもっとも高速に暗号処理することができたことがわかる．これは，ファイルサイズが小さい分，暗号処理の高速化の影響が少ないので，オフロードによって生じるオーバヘッドの割合が暗号処理時間において増えることが理由である．しかし，ファイルサイズが 1000 バイト超えると，「暗号モジュールと CPU の両方を利用した場合」が最も高速に暗号処理をすることができ，ファイルサイズが大きいほど，提案システムが高速に処理できることがわかる．今回計測したファイルサイズで最も大きい 7 キロバイトでは，「暗号モジュールと CPU を同時利用した場合」は，「CPU のみ」で暗号処理を行った場合に比べ，2 倍以上高速であった．また，計測を行った 9 種類のファイルサイズ全てにおいて，「暗号モジュールと CPU を使った場合」が単純に暗号モジュールのみを利用した場合に比べ高速に処理できると言える．

表 2：計測結果

ファイルサイズ	CPU(ms)	暗号モジュール(ms)	暗号モジュールと CPU の同時利用(ms)
5B	85.14	357.85	203.25
20B	110.66	465.09	193.51
50B	100.71	568.48	197.14
100B	126.16	569.49	186.22
500B	278.6	660.58	415.5
1KB	490.63	736.82	517.55
3KB	1307.22	1047.09	938.72
5KB	2253.51	1406.22	1263.03
7KB	3111.15	1707.46	1441.1

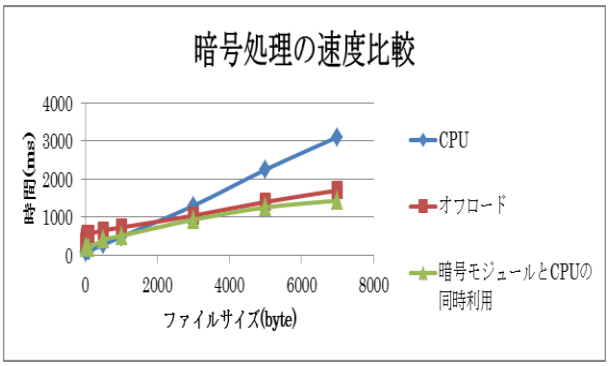

図 9：計測結果

5 まとめ

本論文では，モバイル端末上で行われる暗号処理を暗号モジュールにオフロードする．また，暗号モジュールにオフロードを行うことに加え，CPU も同時に利用することで，モバイル端末における暗号処理速度の高速化を図り，その評価を行った．その結果，暗号モジュールと CPU の両方で暗号処理を行うと，暗号モジュールのみで暗号化する場合に比べ，高速であることが言える．このことから，CPU リソースが空いていれば暗号モジュールと CPU の両方で暗号処理を行うとより高速化できることがわかった．

また，ファイルサイズが 1K バイト以下では CPU のみで暗号化した時が最も高速に暗号化するが，1K バイト以上であれば，提案システムが有効であることがわかった．

よって，CPU リソースに空きがあり，暗号処理をするデータサイズがある一定以上大きければ，提案手法が最も高速に暗号処理ができると言える．

謝辞

本論文の作成にあたって，柿祐輔氏には有益なコメントを頂きました．記して感謝します．

参考文献

[1] Intel® IXP425 Network Processor, http://download.intel.com/design/network/ProdBrf/27905105.pdf

[2] UltraSPARC Tx Processor http://www.opensparc.net/pubs/preszo/09/hotChips_spracklen-final.pdf

[3] Hughes ,J., Morton, G., Pechanec, J., Schuba, C., Spracklen, L. and Yenduri, B.: Transparent Multi-core Cryptographic Support on Niagara CMT Processors, Sun Microsystems, Inc. 10 Network Circle Menlo Park, CA 95025 - USA

[4] 齋藤，大釜，羅，杉浦，IXP425 における暗号処理の効率的なオフロード方式の実装と評価，情報処理学会論文誌，Vol.51 No.9（2010），pp1530-1541

[5] AM3358 Processor, http://www.ti.com/product/am3358

[6] OCF, http://ocf-linux.sourceforge.net/

[7] John Viega・Matt Messier・Pravir Chandra 共著，齋藤孝道 監訳，OpenSSL - 暗号・PKI・SSL/TLS, http://www.openssl.org/

[8] OpenSSH, http://www.openssh.org/