# UltraSPARC Tx における暗号処理のオフロード方式のスケジュール機能の改良と評価

村上 智祐† 笠原 竜大† 天野 桂輔† 渥美 裕太† 齋藤 孝道‡

†明治大学大学院 ‡明治大学

**あらまし** 複数の暗号モジュールを搭載したプロセッサにおいて，暗号処理を行う際，その処理を行うプロセスが増えると，不要なコンテキストスイッチが頻繁に発生してしまい，暗号モジュールへのオフローディングの効率が悪くなる場合がある．そこで，その解決策の一つとして，本論文では，UltraSPARC Tx 上で，暗号プロセスを管理するスケジュール機能を組込み，プロセスの生成や切り替えの制御を行う方式を提案し，その実装と評価を行う．

# An Process Scheduler for Off-loading of Cryptographic Operation on UltraSPARC Tx

Tomosuke Murakami† Ryuta Kasahara† Keisuke Amano† Yuta Atsumi†
Takamichi Saito‡

†Graduate School of Meiji University ‡Meiji University

**Abstract** In case of cryptographic processing on cryptographic modules, a context switch causes to be increased when executing many processes. Then, it could decrease efficiency of offloading cryptographic operation into hardware cryptographic module. In the paper, we introduced a scheduler for cryptographic processes. We show that the scheduler reduces the number of context switches on UltraSPARC Tx.

## 1 はじめに

インターネットを利用する通信システムにおいて，個人情報など秘匿性の高い情報を盗聴や改ざんから保護するためにSSL(Secure Socket Layer) / TLS(Transport Layer Security)や IPSec [1]などのセキュリティプロトコルは欠かせないものとなっている．SSL / TLS や IPSecを利用した通信では，暗号化・復号処理を利用する必要がある．しかし，これら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，数多くの暗号化・復号処理を同時に行う Web サーバなどでは，大きな負担となり，通信のボトルネックとなってしまう場合がある．

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エアモジュール（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 [2][3]．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増えてきた [4][5]．

本論文では，複数の暗号モジュールを搭載したプロセッサとして，Oracle 社の UltraSPARC T2 を利用して，暗号化・復号処理のオフロード技術について検討する．UltraSPARC T2 は 8 個のコアを搭載した CPU であり，それぞれのコアに，共通鍵暗号方式や公開鍵暗号方式，ハッシュ処理

などをサポートする暗号モジュール（後述）を搭載している．また，UltraSPARC Tx 用の OS である Solaris では，暗号モジュールを利用するためのフレームワークとして，PKCS#11 ライブラリ（後述）が提供されている．しかし，PKCS#11 を利用する場合，プログラマは直接暗号処理の制御ができないため，暗号モジュールの利用は PKCS#11 に依存することになる．

そこで，本論文では，UltraSPARC Tx の暗号処理の制御を直接行うため，Solaris に単純に移植した OCF（OpenBSD / FreeBSDCryptographicFramework）[6]及び OCF から暗号モジュールを利用するためのカーネルモジュール（以降，実装モジュールと呼ぶ）[7]を利用し，暗号処理を効率化する．しかしながら，単純に移植した OCF では，プロセス数の増加に伴い，条件によっては，スループットが低下してしまうことが分かった．そこで，本論文では，プロセスの増加によるスループットの低下を抑えるため，OCF にプロセスを管理するスケジュール機能を実装し，その評価を行った．

## 2 UltraSPARC T2 のアーキテクチャ

UltraSPARC T2 プロセッサのアーキテクチャを図 1 に示す．

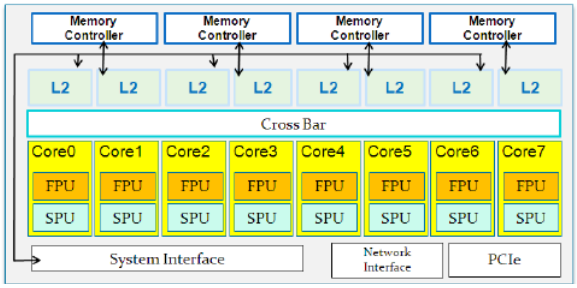


図1: UltraSPARC T2 プロセッサ

UltraSPARC T2 プロセッサは Oracle 社のチップ・マルチスレッディング・テクノロジー（CMT：Chip Multithreading Technology）に基づくマルチコアプロセッサであり，1 つのプロセッサに 8 つのコアが搭載されている．それぞれのコアは，論理的に 8 つの CPU として動作するため，最大 64 個のスレッドを同時に実行可能である．また，各コアごとに浮動小数演算ユニットである FPU（Floating point / Graphics Unit）と暗号処理ユニットである SPU（Stream ProcessingUnit）を搭載している．これらはメインメモリを共有しており，各コア，SPU 及び FPU は並列に動作できる．

UltraSPARC T2 の Memory Controller はオンチップ化により，メインメモリへのアクセス・レイテンシを低減させている．合計 4MB の L2 キャッシュが 8 つのコアに均等に分割され，Cross Bar 経由で，それぞれのコアとのデータ転送を行っている．

暗号処理ユニット SPU は主に MAU（Modular Arithmetic Unit）と暗号 / ハッシュ・ユニットから構成される．MAU は FPU を利用し，公開鍵暗号方式 RSA，及び楕円曲線暗号を処理する．暗号 / ハッシュ・ユニットは共通鍵暗号方式やハッシュ関数に対応しており，DES，3DES，AES，RC4，SHA1，SHA256 と MD5 を利用できる．

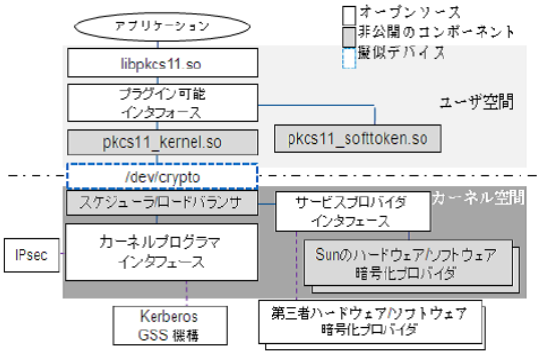


図 2: PKCS#11 暗号フレームワーク

## 3 PKCS#11

PKCS#11 では，アプリケーションからのインターフェースとなるライブラリとして libpkcs11.so [8]を用意しており，これを用いて，暗号モジュールへ処理をオフロードすることができる．また，暗号モジュールの利用状況を監視し，暗号処理の実行対象を決定するスケジューラ / ロードバランサと，暗号モジュールを制御するためのデバイスドライバである暗号化プロバイダを備えている（図 2）．

# 4 OCF

ここでは，OCF の概要とOpenSSL [9]からOCFを利用する方法を示す

## 4.1 OCF の概要

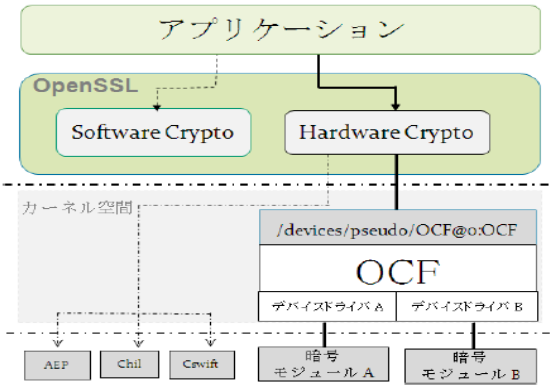


図 3: 暗号処理専用モジュールの利用

OCF(図 3)とは，OpenSSL，OpenSSH 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．

## 4.2 OCF の利用方法

ここでは，OCF の利用方法について説明する．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，Linux の標準的な 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．

## 4.3 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/devices/pseudo/OCF@0:OCF に対して，OCF に発行する．その OCF への制御リクエストには，CIOCGSESSION，CIOCCRYPT，CIOCFSESSION の 3 種類があり，以下に示すように使い分ける．

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．ここでセッションとは，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．この制御リクエストを発行すると，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．

# 5 OpenSSL

OpenSSL(図 3)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.

OpenSSLが提供するライブラリはlibsslとlibcryptoがあり,前者はSSL / TLS通信を,後者は暗号技術を提供するライブラリである.共通鍵暗号化方式としてDES, 3DESやAESなど,公開鍵暗号化方式としてRSAやDH(Deffie-Hellman)など,ハッシュ関数としてSHA-1やMD5など,主要なアルゴリズムが利用可能である.特に,共通鍵暗号化方式とハッシュ関数については,共通のインタフェースでの利用を可能とするEVP APIを提供している.

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.

### 5.1 OpenSSL からの 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4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4: OpenSSLからのオフロードの手続き

図4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.

### 5.2 OCF への暗号処理の受け渡し

図5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 6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とCIOCFSESSION制御リクエストに対応している.

```
7.  EVP_CIPHER_CTX ctx;
8.  EVP_EncryptInit( &ctx, EVP_des_cbc(), key,  iv );
9.
10. EVP_EncryptUpdate( &ctx, output, &outlen, input, len);
11. EVP_EncryptFinal( &ctx, outbuf+outlen, &tmplen);
12.
13. EVP_CIPHER_CTX_cleanup( &ctx );
```

図5: 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 とのセッションが生成される.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に暗号処理が受け渡される.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.

## 6 提案システム

提案システムでは,暗号モジュールへ処理をオフロードする際,プロセスの増加によるスループットの低下を抑えるため,暗号処理を行うプロセス(以

降，暗号プロセスと呼ぶ）の実行・停止をスケジューリングする．

### 6.1　提案システムの概要

実装においては，後述の一方向リスト（以降，制御リストと呼ぶ）を導入し，暗号プロセスを FIFO でのスケジューリングを行い，実行できる暗号プロセス数を制限する．提案システムにおいて，制御リストは 64 個存在し，暗号プロセスは OCF とのセッション生成時にプロセス ID に対応した制御リストに登録される．同じ制御リストに暗号プロセス A，B がこの順で登録された場合，先に登録された暗号プロセス A が暗号モジュールを利用することができ，後に登録された暗号プロセス B は A の暗号モジュールの利用が終わるまで，スケジューラによって強制的に sleep させる．A による暗号モジュールの利用が終了後，スケジューラにより B が起こされ，B が暗号モジュールを利用することができる．制御リストに他の暗号プロセスが登録されていない場合は制御リストに登録され，速やかに処理が実行される．

### 6.2　制御リストの概要

制御リストは暗号プロセスの情報を格納するため，下記に示すように，3 つのメンバーを持つ構造体（以降，制御構造体と呼ぶ）を用意する．

```
struct sched_entry{
        list_node_t list
        pid_t pid
        kcondvar_t se_waitq
};
```

各メンバーの役割は以下の通りである：

- list — 制御リストを連結するためのリスト構造体
- pid — 暗号プロセスのプロセス ID を記憶するための変数
- se_waiq — sleep 状態に遷移するためのウェイトキュー

提案システムでは 64 個の制御リストを設け，実行できる暗号プロセス数を UltraSPARC T2 で同時実行可能な最大数である 64 個に制限し，制御リストへの登録作業を並列化させることを可能にした．

### 6.3　提案システムの動作例

ここでは，ユーザ空間で実行される OpenSSL を用いた暗号プロセスの処理を，OCF と実装モジュールを介して，暗号モジュールへオフロードする場合の動作例を図 6 中の番号と対応させて説明する．

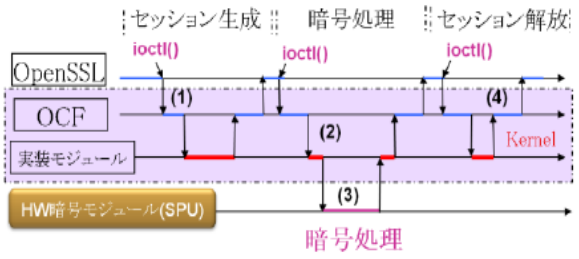


図6:処理の詳細

(1) OpenSSL と OCF の間でセッションを確立する．セッションの確立時に暗号プロセスを制御リストに登録する．この際，制御リストが空ならば，暗号プロセスを制御リストに繋ぎ，次の(2)の処理に移る．制御リストが空でなければ，暗号プロセスを制御リストにつないだ後，wait キューにつなぎ，sleep 状態に遷移させる

(2) OpenSSLはOCFにセッションIDと平文，IV を渡す．さらに，実装モジュールがセッション IDに対応した暗号鍵，OpenSSLから受け取った平文及びIVをHW暗号モジュールで使用できる形式に変換する

(3) HW暗号モジュールは暗号処理を行う．ここで，暗号処理が終わると，ハードウェア割り込みが発生し，結果をOCF経由でOpenSSLへ返す

(4) セッションの解放を行う．自暗号プロセスが登録されていた制御リストから自らを削除し，その制御リストの先頭の暗号プロセスを実行可能状態に遷移させる

## 7　評価

### 7.1　計測環境

提案システムの評価のために，表 1 に示す環境

を用意した．この環境で，単純に移植した OCF（以降，単純移植版 OCF と呼ぶ），PKCS#11 ライブラリを利用したものと本論文で実装した OCF（以降，OCF-sched と呼ぶ）の処理時間を計測し，比較した．

表1：評価環境

| | | | |
|---|---|---|---|
| CPU | 1.2GHz UltraSPARC T2 (コア) | メモリ | 16GB |
| カーネル | Solaris kernel build 117 [5] | その他 | OCF-20041201<br>OpenSSL-0.9.8a |
| OS | Solaris Express | | |

### 7.2 計測項目

性能評価として，EVP API を使用し，AES の CBC モードを行うオリジナルの計測用コードを用いた．計測用コードは，fork()関数により暗号プロセスを複数生成し，それぞれの暗号プロセスで 10Kbyte のデータを 1024 回（合計 10Mbyte）暗号処理する．計測方法は，gettimeofday()関数を使用し，暗号プロセスの生成から暗号プロセスの終了までを処理時間として，単位時間当たりのスループットを求めた．計測はそれぞれ 10 回行い，その平均値を結果とした．図 7 に計測用コードによる計測結果を示す．

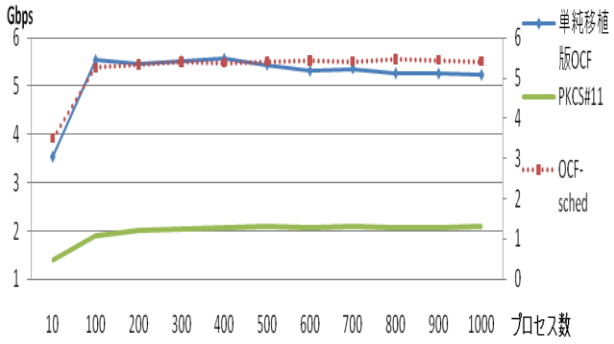


図7: 暗号処理スループット

図7より，プロセス数が500を超えた辺りから，単純移植版OCFよりもOCF-schedのスループットの方が高くなる．プロセス数がさらに増加した場合においても，単純移植版OCFのスループットは低下するが，OCF-schedのスループットは低下せずに，一定の値を保っており，狙い通りの効果を確認できた．

## 8 まとめ

本論文では，暗号処理を行うユーザプロセスを管理するスケジュール機能を実装し，その評価を行った．その結果，プロセスの切り替えの増加に伴うスループットの低下を抑えることができた．

## 参考文献


[1] IPSec http://www.ietf.org/rfc/rfc1825.txt

[2] Intel® Xeon® Processor 5600 Series, http://download.intel.com/jp/business/japan/pdf/323501-003JA.pdf

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

[4] 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

[5] 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.

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

[7] 羅，大釜，杉浦，齋藤，UltraSPARC T2 における暗号モジュールの利用と評価，暗号と情報セキュリティシンポジウム (2010)

[8] libpkcs11.so http://docs.sun.com/.

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

[10] Solaris kernel build 117 http://dlc.sun.com/osol/on/downloads/b117/