UltraSPARC T2における暗号モジュールの利用と評価

羅 鏡栄† 大釜 正裕† 杉浦 寛† 齋藤 孝道‡

†明治大学 大学院　‡明治大学

1 はじめに

インターネットでSSL/TLSやIPSecを利用した通信には，暗号化・復号などの高負荷な演算が伴うため，それらを高速に実行する暗号処理を専ら行うモジュール(以下，暗号モジュールと呼ぶ)が利用されている．

このような背景の中，Sun MicroSystems社のUltraSPARC T2が登場した．UltraSPARC T2は，1つのプロセッサに8つのコアを搭載したマルチコアプロセッサであり，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を搭載している．

UltraSPARC T2の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，Solaris10からPKCS#11（後述）が提供されている．しかし，PKCS#11を利用する場合は，プログラマは直接暗号モジュールの制御ができないため，暗号モジュールの利用効率は，PKCS#11に依存することとなる．

本論文では，UltraSPARC T2の暗号モジュールの制御を直接行うため，OCF(OpenBSD/FreeBSD Cryptographic Framework)[1]をSolarisに移植し，OCFをSolaris上で動かすためのカーネルモジュールを実装した．さらに，暗号処理を高速化するために，OCF内の改変を行った，本論文での実装システムとPKCS#11で暗号処理のパフォーマンスを計測し，それらの評価をそれぞれ行った．

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．

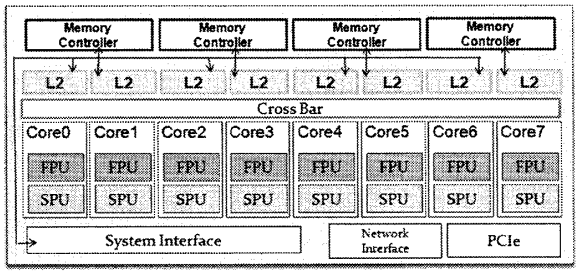

図1: UltraSPARC T2プロセッサ

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，各コアごとに暗号処理専用ユニットであるSPU (Stream Processing Unit) と浮動小数演算ユニットであるFPU (Floating point/Graphics Unit) を搭載している．これらはメインメモリを共有しており，各コアとSPU，FPUは並列に動作できる．

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は公開鍵暗号方式にサポートしており，FPUを利用して，RSA及び楕円曲線暗号を処理できる．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，MD5を利用できる．

3 PKCS#11

PKCS#11は，Solaris 10により提供されている暗号モジュールを利用するためのフレームワークである．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を用意しており，これを用いて暗号処理の実行を指示することで，暗号モジュールへ処理がオフロードされる．また，暗号モジュールを利用するための仕組みとして，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．

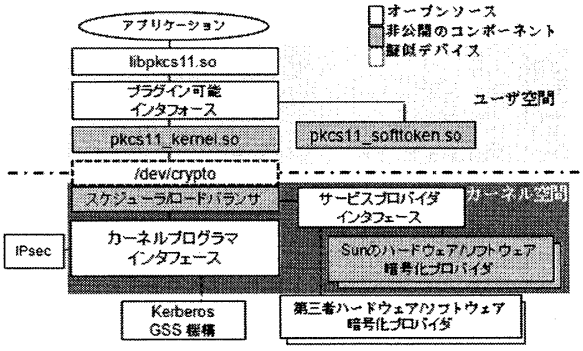

図2: PKCS#11暗号フレームワーク

4 OCF

OCFとは，OpenSSLなどを利用したアプリケーションから，様々なハードウェアアクセス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．オリジナルのOCFの構成は図3の鎖線部分に示している．ただし，OCFはLinux系のカーネルミドルウェアとして開発されているため，他の系統のOSで利用するには，その環境に合わせた改変が必要となる．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こ

Utilization and Evaluation of Hardware Cryptographic Module on UltraSPARC T2

†Keang-Weng Lo, Masahiro Ohgama, Kan Sugiura
‡Takamichi Saito
Graduate School of Meiji University(†), Meiji University(‡)
1-1-1, Higashimita, Tama-ku, Kawasaki-shi, Kanagawa 214-8571, Japan(†)(‡)
{oscar_weng, ohgama, kan_s, saito}@cs.meiji.ac.jp

とで，OCF からの利用が可能となる．

OCF では，暗号処理を実行する場合に，システムコールを使用して，鍵などの暗号情報の転送，暗号処理の実行，終了時のメモリ開放という 3 つの命令単位を一つのセッションとして扱い，処理を行っている．

5 実装

本論文では，OCF を Solaris に移植し，その OCF から SPU(以降，HW 暗号モジュールと呼ぶ) 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）の実装と，暗号処理の高速化のためにバッファの追加を行った．

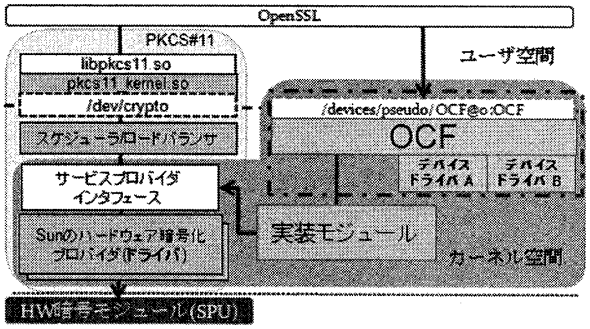

図 3: OCF と実装モジュールの概要

5.1 実装モジュールの詳細

実装モジュールは，HW 暗号モジュールのデバイスドライバとして動作する，OCF の暗号処理を HW 暗号モジュールへオフロードするためのモジュールである．その実装モジュールを介して，OCF から HW 暗号モジュールを直接制御することが可能となる．実装モジュールの位置付けを図 3 に示す．

実装モジュールの主な役割は，以下に示す 3 点がある．(1)PKCS#11 のスケジューラにリンクし，PKCS#11 の HW 暗号モジュールにアクセスするための内部関数を呼び出しす．(2)OCF から暗号鍵，平文などの暗号処理に必要なパラメータを内部ドライバ関数で利用できる形式に変換する，(3)PKCS#11 スケジューラと同一のセマフォを使用し，HW 暗号モジュールへのアクセス競合を回避する．

5.2 OCF へのバッファ追加

OCF で暗号処理を実行する場合には，平文を一時保存するため，毎回カーネルメモリの確保と開放を行っている．本実装では，これらの CPU オーバーヘッドを抑えるため，同一セッション内で継続して利用するバッファを追加し，暗号処理実行時には，セッション開始からセッションの終了まで同一のバッファを使用する．

5.3 実装モジュールの動作例

ここでは，OpenSSL の暗号処理を，OCF と実装モジュールを介して，HW 暗号モジュールへオフロードする場合の動作例を図 4 中の番号と対応させて説明する．

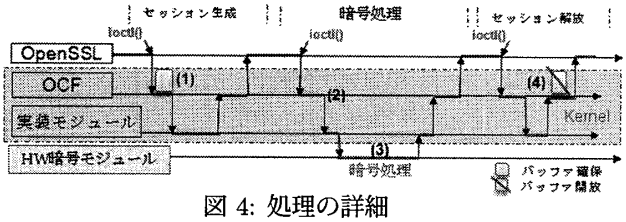

図 4: 処理の詳細

(1) OpenSSL が OCF とのセッション確立を行う．ここで，暗号鍵と暗号化方式を OCF へ転送し，OCF が平文のためのバッファを確保する．また，セッション ID を生成して，OpenSSL へ転送する
(2) OpenSSL が，OCF にセッション ID と平文，IV を受け渡す．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 を内部ドライバ関数で使用できる形式に変換する．その後，ドライバ関数を呼び出して暗号処理を行う
(3) HW 暗号モジュールが暗号処理を行う．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 OpenSSL へ返す
(4) OpenSSL がセッションを開放し，メモリの開放を行う

6 評価

実装モジュールの評価のために，表 1 に示す環境で OCF を使用した場合と PKCS#11 を使用した場合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[3]	その他	OCF-20041201
OS	Solaris Express		OpenSSL-0.9.8a

6.1 評価項目

実装モジュールの性能評価として，OpenSSL の speed コマンドと EVP API を使用した計測用コードそれぞれを用いた．ここで，OpenSSL の speed コマンドでは，単一のプロセスによる処理時間しか計測できないため，計測用コードを用意している．計測用コードは，fork() により複数のプロセスを生成し，それぞれのプロセスが 10Mbyte のデータを，暗号処理している．計測方法は，gettimeofday() 関数を使用し，プロセスの生成から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．また，計測に用いる暗号方式は，いずれの計測でも AES の CBC モードである．

OpenSSL speed コマンドによる計測結果を表 2 に示す．但し，表中のデータサイズは，一度暗号処理を行うデータサイズを表している．ここで，PKCS#11 はオリジナル，OCF は本論文での実装をそれぞれ示す．

表2：OpenSSLのspeedコマンド結果

データサイズ	64bytes	254bytes	1024bytes	8192bytes
PKCS#11(kbyte/s)	1110.68k	4213.59k	14292.31k	20733.95k
OCF(kbyte/s)	791.93k	3130.00k	11866.03k	75474.64k

次に，計測用コードによる計測結果を図 5 に示す．

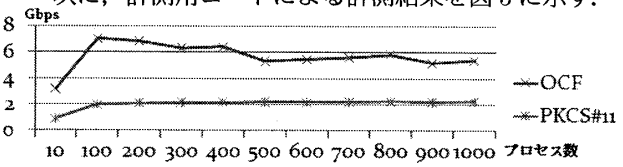

図 5: 暗号処理スループット

いずれのブロックサイズでも本論文での実装が優れていることが分かる．

7 まとめと今後の課題

本論文では、OCF を利用して OpenSSL からの暗号処理を，UltraSPARC T2 の HW 暗号モジュールへオフロードする独自方式を実装した．その評価を行った結果，従来方式より高速化することができた．

今後の課題として，プロセス数の増加により，スループットの低下を抑えるため，OCF に暗号プロセスを管理するスケジューラを組込むことが挙げられる．

参考文献

[1] http://ocf-linux.sourceforge.net/
[2] Solaris kernel build 117 ソース，
http://dlc.sun.com/osol/on/downloads/b117/