Vol.2011-OS-118 No.9
2011/7/27

中立的仮想計算機モニタによる耐タンパーデバイスのアクセラレータの実装

松下正吾[†1] 新城靖[†1] 榮樂英樹[†3]
松原克弥[†3] 東悠[†2]

耐タンパーデバイスはデジタル認証や著作権管理に広く用いられている。耐タンパーデバイスは外部からの攻撃に対して内部の情報を守ることができるが、計算能力が通常の PC と比べて格段に低いという問題がある。本研究では PC 上に 2 つの隔離された環境を構築することでこの問題を解決する。2 つの環境とは、通常の OS(Operating System) を実行する環境 (ユーザ環境) と耐タンパーデバイスを援助するプログラムを実行するための環境 (耐タンパーデバイス環境) である。これら 2 つの環境を実現するために、本研究では中立的な仮想計算機モニタを用いる。中立的な仮想計算機モニタとは、CPU などのハードウェアと同様に、期待された動作のみを行い、悪意がある行動をとらないような仮想計算機モニタである。

この論文では、中立的仮想計算機モニタを仮想計算機モニタ BitVisor を拡張することで実装する方法を示す。拡張された BitVisor は、環境間通信として耐タンパー環境 RPC (Remote Procedure Call) と呼ぶ仕組みを提供する。仮想計算機モニタが改竄されていないことを検証するために、BitVisor に TPM(Trusted Platform Module) を用いた Trusted Boot 機能を付加する。この論文では、耐タンパーデバイス環境を利用したアプリケーションとして、デジタル署名と著作権管理の実現方法を示す。

Implementing accelerators of tamper-resistant devices with a neutral virtual machine monitor

Shougo Matsushita,[†1] Yasushi Shinjo,[†1]
Hideki Eiraku,[†3] Katsuya Matsubara[†3]
and Yu Azuma [†2]

Tamper-resistant devices are widely used for electronic signature and digital rights management. While these devices can protect internal information against external attacks, they have much less computing power than regular personal computers (PCs). We overcome this problem by building two isolated environments in a PC: the user environment for running a normal operating system, and the tamper-resistant environment for running programs that help tamper-resistant devices. To realize these two environments, we use a neutral virtual machine monitor (VMM). A neutral VMM means a VMM that plays only its expected role as hardware devices including CPUs, and never performs malicious activities.

In this paper, we show an implementation of a neutral VMM by extending the BitVisor VMM. Our BitVisor provides an inter-environment communication facility called tamper-resistant environment RPC(Remote Procedure Call). We add the trusted boot feature using TPM (Trusted Platform Module) to BitVisor for verifying that the VMM is not compromised. In this paper, we show the implementations of electronic signature and digital rights management as applications that make use of the tamper-resistant environment.

1. はじめに

近年、Suica, Edy, B-CAS カード等の各種 IC カードがよく使われるようになっている。これらは**耐タンパー性** (Tamper-resistance) を持つデバイス（以下耐タンパーデバイス）であることが多い。耐タンパー性とはハードウェアやソフトウェアによる外部からの解析が困難な性質のことである。耐タンパーデバイスは外部からの攻撃を困難にしたり、攻撃された場合にそれを検出したりといったことを行う機能を備えている[5]。例えば実行コードを暗号化や難読化したり、コードの改竄を検出したり、異常な電圧やクロック周波数を検出する耐タンパーデバイスが存在する。

耐タンパーデバイスは利用者認証や著作権保護[8] に利用される。例えば、電子政府では住民基本台帳カードを個人認証に利用している。地上波デジタル放送では、放送内容を暗号化して電波に乗せ、B-CAS カードの鍵を利用して復号を行っている。

耐タンパーデバイスは、処理性能が PC(Personal Computer) と比較して格段に低い。もし耐タンパーデバイス上で動作しているプログラムを PC 上で動作させることができれば、プログラムの処理性能の向上が期待できる。しかし、元の耐タンパーデバイスが扱うのは機密情報であるため、そのようなプログラムを通常の OS が動作する環境で動作させることは

†1 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
Department of Computer Science of Systems and Information Engineering, University of Tsukuba
†2 筑波大学情報学群情報学類
College of Information Science, University of Tsukuba
†3 株式会社イーゲル
Igel Co., Ltd

困難である。その理由は、OS 上で動作しているプログラムからの攻撃を完全に防ぐことができないからである。

本研究の目的は PC 上に通常の OS の環境 (ユーザ環境) と隔離された、耐タンパーデバイスのプログラムを動作させるための環境 (耐タンパー環境) を構築することである。これにより、耐タンパーデバイスの動作を実質的に高速化する。この目的のため、本研究では仮想計算機モニタを用いる。仮想計算機モニタはユーザ環境と耐タンパー環境に対して中立なものにする。これによりユーザ環境および耐タンパー環境の双方が安心してそれぞれの機密情報を扱えるようにする。本研究では仮想マシンモニタの中立性を保証するために、TPM(Trusted Platform Module) を用いた Trusted Boot を行い、仮想マシンモニタが改竄されていないことを両環境から検証可能にする。

本研究が実現されると、例えば IC カードによる認証処理を大幅に高速化することができる。またデジタル著作権管理では、従来は IC カードの中で行っていた復号処理を PC に行わせることが可能になる。

本論文の構成を以下に述べる。2 章では関連研究について述べる。3 章では、中立的仮想計算機モニタの要件を明確にする。4 章では BitVisor を拡張して実装した中立的仮想計算機モニタについて述べる。5 章では耐タンパー環境を利用したアプリケーションについて述べる。7 章でまとめを行う。

2. 関 連 研 究

Sony 社の PlayStation3 では Linux などのサードパーティ製の OS を実行を許すため、仮想計算機モニタを用いている[14]。この仮想計算機モニタはゲスト OS からハードウェアが解析されることを阻止し、DVD 等の著作物や Sony 社の権利を守ることは実現されている。しかしながら、その仮想計算機モニタがユーザの個人情報を盗むなどの悪意ある行動を取らないという保証はない。このため、ユーザが安心して自分の所有している PC を利用することはできない。本研究ではこの問題を解決し、ユーザが安心して自分の所有している PC を利用できるようにする。

文献 11) では、OS のシステムコールの一部を独自の仮想計算機モニタ上で実行し、そのコードを秘匿化することを提案している。本研究とは仮想計算機モニタ上で処理の一部を実行する部分が共通している。この研究と比較して本研究の特徴は、耐タンパーデバイスの利用を想定しているところにある。

文献 4) では、ソフトウェアによる耐タンパー性を持ったアプリケーションの実装を試みている。この方法では保護したいプロセスを Strata VM[12] 上で動作させ、その VM が暗号化、改竄の検出、復号されたキャッシュの削除といった処理を行うことで、プロセスを普通に動作させるよりも安全に実行できる。この研究では暗号化を行ってはいるが、他のプロセスからのデータの盗難を完全に防止することはできない。本研究では完全に隔離された実行環境を実現し、データの盗難を完全に防ぐ。

委託計算 (Server-aided computing)[3][7] とは、計算力の低いデバイスが重い処理を他の外部デバイスやサーバに処理させる手法のことである。委託計算では必ずしも信頼のおけない環境に計算を委託するため、渡したデータを解析されないように工夫する必要がある。文献 15) では、携帯電話デバイス上で行っている署名計算の一部を外部サーバに委託している。本研究は委託計算におけるサーバ側の処理を安全に PC 上で実行できる環境を提供する。

超流通[8] とは、デジタル情報の権利者が正当な対価を得られることを保証し、さらにユーザの利便性も同時に得られるようにすることを物理的に保証するソフトウェアの流通システムである。超流通では、アプリケーションの重要な処理を耐タンパーデバイス上で動作させる。本研究が実現されれば、超流通において保護する部分を増やすことで、著作権保護を強化できる。

Xen は TPM の仮想化[1] や Trusted Boot に対応しており、仮想計算機上から TPM を扱える。しかし Xen は Domain 0 と呼ばれるホスト OS(Linux Kernel) を必要としており、TCB(Trusted Computing Base) が大きいという問題がある。

3. 中立的仮想計算機モニタによる耐タンパー環境の実現

本研究では図 1 のように、耐タンパーデバイスともユーザとも中立な仮想計算機モニタを実装することで、耐タンパーデバイスのプログラムの実行を高速化する。**ユーザ環境**とは PC の所有者が使用する OS 及び、その上で動作するアプリケーションを動作させる環境である。この環境は耐タンパーデバイス以外の通常のデバイスを直接アクセス可能である。**耐タンパー環境**とは、耐タンパーデバイスにのみ直接アクセスできる環境である。ここでユーザ環境から隠したいプログラムを実行することで、耐タンパーデバイスの処理を実質的に高速化させる。

中立的仮想計算機モニタは次の要件を満たす必要がある。

(1) ユーザ環境と耐タンパー環境を提供する。各環境のメモリを隔離する。耐タンパー環境からのみ、耐タンパーデバイスへアクセス可能にする。ユーザ環境からのみ、その他の一般的なデバイスにアクセス可能にする。

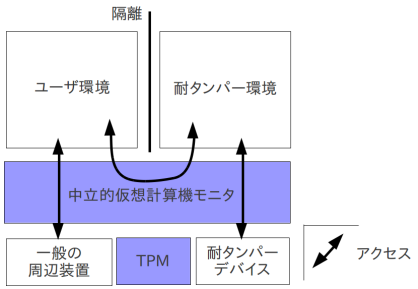

図 1 中立的仮想計算機モニタによる耐タンパー環境の構築

(2) 環境間で通信を行う機能を提供する。
(3) 各環境から仮想計算機モニタが改竄されていないことを検証可能にする。TCB が小さいことが望ましい。

本研究における仮想計算機モニタは、2 章で述べた PlayStation3 のもの[14] とは異なり、ユーザ環境からも耐タンパー環境からも中立性を持たせる。ここでいう中立性とは、CPU などのハードウェアと同様に仮想計算機モニタが期待された動作のみを行い、悪意のある行動をとらないことを意味する。仮想計算機モニタの中立性はソースコードを公開し、自由にチェックできるようにすることで担保する。

中立性がソースコードレベルで担保されたとしても、仮想計算機モニタのバイナリが改竄されれば中立性は失なわれてしまう。そこで本研究では仮想計算機モニタのバイナリが改竄されていないことを検証するために、TPM を用いた Trusted Boot を使用する。Trusted Boot とは、PC の起動時に **Chain of Trust**(信頼性の鎖) を構築することである。その起点となるのが PC のファームウェア部分を表す CRTM(Core Root of Trust for Measurement) である。Trusted Boot では次のプログラムに制御を渡す前に、そのプログラムや設定ファイルのハッシュ値を計算することで信頼性の鎖を構築する。Trusted Boot を行う際には計測 (measurement) に対応したファームウェアとブートローダが必要になる。

4. BitVisor を用いた中立的仮想計算機モニタの実現

3 章で述べた要件を満たす仮想計算機モニタを実現するために、本研究では仮想計算機モニタ BitVisor[13] を用いる。BitVisor とは筑波大学などを中心として開発されたハイパバイザ型の仮想計算機モニタである。BitVisor はハードディスクやネットワークの暗号化などセキュリティを保つために必要なデバイス以外はゲスト OS にそのままアクセスさせる。よって、デバイスドライバとデバイスのエミュレータを仮想計算機モニタ内に持つ必要がない。このため Xen などの他の仮想計算機モニタと比較するとソースコードが小規模で、TCB が小さく検証しやすい。

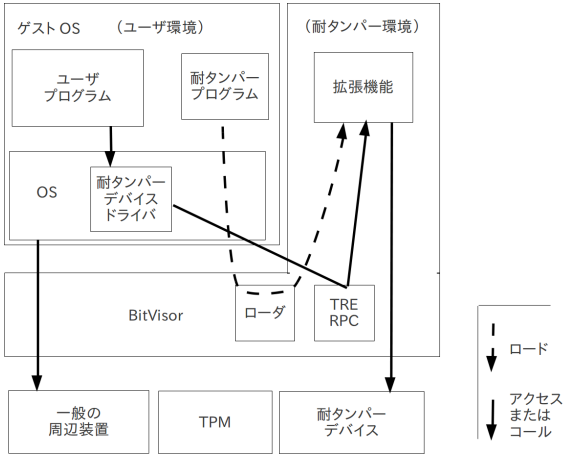

図 2 BitVisor による中立的仮想計算機モニタの実現

4.1 BitVisor の拡張機能による耐タンパー環境の実現

3 章で述べた中立的仮想計算機モニタの要件 (1)、すなわちユーザ環境と耐タンパー環境の構築は仮想計算機モニタにより 2 つの仮想計算機を生成し、それらを隔離すれば容易に実現可能である。しかし本研究で用いる BitVisor は、1 つのゲスト OS にしか対応していない。そこで本研究では、ユーザ環境を BitVisor 上のゲスト OS を実行するための仮想計算機として実装し、耐タンパー環境は BitVisor の拡張機能[2] を用いて実装する。この拡張機能では、独立した ELF(Executable and Linkable Format) 形式のプログラムを保護されたメモリ内で実行することができる。そのプログラムはゲスト OS のメモリをアクセスできないように設定できる。また、BitVisor ではゲスト OS は仮想計算機モニタのメモリ領域にアクセスすることはできない。こうすることでユーザ環境と耐タンパー環境の間でメモリを完全に隔離することができる。

3 章で述べたように中立的仮想計算機モニタでは、耐タンパー環境からのみ、耐タンパーデバイスへアクセス可能にする必要がある。BitVisor はデバイスをゲスト OS の環境から

```
int vmmcall_tre_rpc(int desc, int proc_number,
        struct iovec args, int arg_count,
        struct iovec results, int result_count)
```

図 3 ハイパバイザーコール vmmcall_tre_rpc のインタフェース

隠す機能がある。この機能を利用して耐タンパーデバイスをゲスト OS から隠す。また、BitVisor は USB デバイスやシリアルポート等の一部のデバイスを仮想計算機モニタ本体から利用できる機能がある。この機能を利用して、耐タンパー環境から耐タンパーデバイスへアクセスさせる。また、その他のデバイスについては、ユーザ環境からのみアクセスさせる。

このように、BitVisor の拡張機能を用いることで、3 章で述べた要件 (1) を満たす。

4.2 耐タンパー環境との RPC

3 章で述べた要件 (2) を満たすために、本研究ではユーザ環境から耐タンパー環境に対する **RPC**(Remote Procedure Call) を提供する。この RPC を TRE RPC(Tamper-Resistant Environment) と呼ぶことにする。

本研究では TRE RPC を BitVisor に vmmcall_tre_rpc というハイパバイザーコールを追加して実装する。このハイパバイザーコールのインタフェースを図 3 に示す。desc は拡張機能のプログラムを識別する番号である。proc_number は呼び出す手続きの番号である。RPC の要求は args で与え、RPC の応答を受けとるための領域を results で指定する。arg_count と result_count には引数や戻り値の数を与える。args と results は struct iovec 型となっており、ゲスト OS 空間のメモリのアドレスと長さを含む。struct iovec はバッファを指すポインタ iov_base とバッファの長さ iov_len からなる構造体で、readv や writev といったシステムコールで標準的に使用される型である。

vmmcall_tre_rpc が呼び出されると、仮想計算機モニタは次のような処理を行う。

(1) args で指定された大きさのメモリを確保し、そのメモリの内容をゲスト OS のアドレス空間から仮想計算機モニタのアドレス空間にコピーする。
(2) result で指定された大きさのメモリを仮想計算機内のアドレス空間に確保する。
(3) (1) と (2) で確保したメモリ領域を引数として、desc で指定された拡張機能のプログラムを呼び出す。
(4) 拡張機能のプログラムが (1) で確保されたメモリから引数を取り出して手続きを実行し、(2) のメモリに処理の結果を格納する。
(5) 仮想計算機モニタのアドレス空間にある (2) のメモリを、引数 results で指定されたゲスト OS のアドレス空間にあるメモリに対してコピーする。

```
 1 int add(int a, int b) {
 2         struct iovec args[2]; struct iovec result[1];
 3         int c;
 4         args[0].iov_base = &a; args[0].iov_len = sizeof(a);
 5         args[1].iov_base = &b; args[1].iov_len = sizeof(b);
 6         result[0].iov_base = &c; result[0].iov_len = sizeof(c);
 7         r = vmmcall_tre_rpc(r, TRE_ADD, args, 2, result, 1);
 8         if (r < 0) {
 9                 fprintf(stderr, "vmmcall_tre_rpc:␣%d\n", r);
10                 return -1;
11         }
12         return c;
13 }
```

図 4 C 言語による耐タンパー環境 RPC のクライアント側のコード例 (加算)

ユーザ環境における、vmmcall_tre_rpc を用いた RPC のクライアント側のコードの例を図 4 に示す。これは耐タンパー環境への RPC を用いて足し算を行うプログラムである。4〜5 行目では vmmcall_tre_rpc のための引数のアドレスと長さをセットしている。この例では TRE_ADD は 2 つの引数で、2 つのメモリアドレスを args に代入する。6 行目では戻り値を受けとる変数のアドレスと長さをセットしている。RPC の関数 TRE_ADD の戻り値は 1 つのみなので、iovec を 1 つだけ作成する。7 行目で実際に vmmcall_tre_rpc() を呼び出し、結果を関数の戻り値とする。

耐タンパー環境で動作する、RPC のサーバ側のコードの例を図 5 に示す。vmmcall_tre_rpc() が呼び出されると、BitVisor により _start 関数が呼ばれる。_start 関数の引数はどのような呼び出し方で呼び出されたかを表す msgcode、手続きの番号を表す proc_number、RPC の引数と戻り値が格納される buf、buf の数を表す bufcnt からなる。struct msgbuf とは BitVisor 内で標準的に使用される struct iovec と類似の構造体である。メモリ領域へのポインタ base とその長さ len をメンバに持つため iovec に似ているが、メモリ領域が書き込めるかを表す rw フラグが存在しているところが異なっている。_start 関数では 1 行目で呼び出し方法をチェックする。msgcode が MSG_BUF の場合は、拡張機能のプログラムへの RPC を意味する。RPC によって呼び出された場合は、proc_number を見て処理を振り分ける。このコードでは TRE_ADD のみ処理するようにしている。proc_number が

```
1  int _start(int msgcode, int proc_number, struct msgbuf *buf, int bufcnt) {
2          if (msgcode == MSG_BUF) {
3                  switch (proc_number) {
4                  case TRE_ADD:
5                          return tre_add_svc(buf, bufcnt);
6                          break;
7                  default:
8                          return -1;
9                  }
10         }
11         return -1;
12 }
13 int tre_add_svc(struct msgbuf *buf, int bufcnt) {
14         int a, b, c;
15         int *result;
16         if (bufcnt < 2 /* Args check. */
17                 || buf[0].len < sizeof (a) || buf[0].rw
18                 || buf[1].len < sizeof (b) || buf[1].rw
19                 || buf[2].len < sizeof (result) || !buf[2].rw) { return -1; }
20         a = *((int *)buf[0].base);
21         b = *((int *)buf[1].base);
22         c = a + b;
23         result = (int *)buf[2].base;
24         *result = c;
25         return 0;
26 }
```

図 5 C 言語による耐タンパー環境 RPC のサーバ側のコード例 (加算)

TRE_ADD であった場合は、tre_add_svc 関数を呼び出している。tre_add_svc 関数は 16 行目から 19 行目で引数のチェックを行う。引数の rw メンバをチェックすることで、引数のバッファは読み取り専用になっているか、戻り値のバッファは書き込めるかをチェックする。不正な引数でなかった場合、20〜21 行目で引数を取り出す。24 行目で結果を戻り値のバッファに代入する。

4.3 中立的仮想計算機モニタの検証

3 章で述べた要件 (3) を満たすために、本研究では BitVisor に対して TPM による Trusted Boot 機能を追加した。起動時、BitVisor はブートローダ GRUB-IMA 等により計測 (measurement) され、制御が渡される。BitVisor は次のコード (ゲスト OS のブートローダやゲスト OS 自身) を計測し、その結果を TPM の PCR(Platform Control Register) とログに保存して制御を渡す。

システムが起動した後、ユーザ環境側と耐タンパー環境側の双方で、計測された結果が期待された通りであることを検証する必要がある。ユーザ環境側では、OpenPTS[9] を用いて検証を行なう。OpenPTS は OS の Trusted Boot が行われていること、および OS 上で動作するプログラムが改竄されていないことを外部のサーバを用いて検証するソフトウェアである。OpenPTS ではあらかじめ、改竄されていない状態の PCR の値とログをサーバに記録しておく。検証したい時、OpenPTS のクライアントは、PCR の値とログを外部のサーバに送信する。PCR の値とログが異なっていた場合、サーバはエラーを返す。

BitVisor のハッシュ値はブートローダ等により TPM の PCR4 に格納されている。したがって、両方の環境で PCR0〜PCR4 までの PCR 値が期待するものと一致すれば改竄されていないことになる。PCR の値は TPM の秘密鍵によって署名されているため、TPM の公開鍵を用いれば値の改竄を検出できる。ログの改竄はハッシュ値を計算しなおすことで検出する。

一方、耐タンパー環境側では OpenPTS が使用できない。そこで耐タンパーデバイスが OpenPTS における遠隔のサーバと同じ役割を果たすことにする。ユーザ環境は TPM によって署名された PCR の値とログを耐タンパー環境のプログラムに送信する。耐タンパー環境のプログラムは、耐タンパーデバイスに PCR の値とログを送信する。耐タンパーデバイスは送られてきた PCR の値とログが期待されるものと一致するか判定する。

4.4 仮想計算機モニタへの署名付きプログラムの動的なロード

従来の BitVisor では、拡張機能のプログラムを仮想計算機モニタに静的に組み込む必要があった。このため、新しい耐タンパーデバイスをサポートするたびに仮想計算機モニタごと再コンパイルが必要である。仮想計算機モニタを再コンパイルすると、Trusted Boot 用のデータを作り直さなければならない。この作り直しの手間は小さくない。

この問題を解決するために、本研究では拡張機能のプログラムを動的に BitVisor へロードするための機能を実装する。この機能をハイパバイザーコール vmmcall_tre_exec により利用可能にする。安全性を保つため、ロード可能なプログラムは、あらかじめ仮想計算機モニタの管理者に署名されたものに限定する。

この機能の概観を図 7 に示す。中立的仮想計算機モニタの管理者は、公開鍵と秘密鍵の組を生成し、公開鍵をあらかじめ中立的仮想計算機モニタのバイナリに埋め込んでおく。耐タンパー環境のプログラム (耐タンパープログラム) の開発者は、開発した拡張機能のプログラムを中立的仮想計算機モニタの管理者へ送る。管理者はその内容を確認し、自身の秘密鍵で署名する。耐タンパープログラムの開発者は、ユーザにその署名された拡張機能のプログラムを送る。ユーザは、vmmcall_tre_exec ハイパバイザーコールでその署名された拡張機

```
int vmmcall_tre_exec(ulong binary_addr, ulong binary_size,
        ulong sign_addr, ulong sign_size)
```

図 6 ハイパバイザコール vmmcall_tre_exec のインタフェース

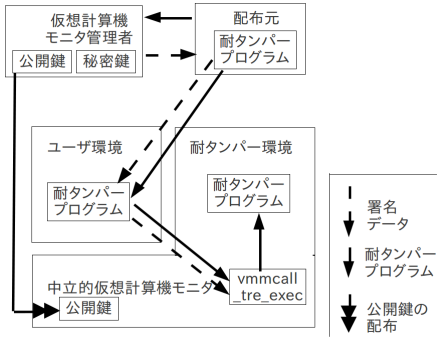

図 7 署名付き耐タンパープログラムの仮想計算機モニタへのロード

能のプログラムを中立的仮想計算機モニタにロードすることを要求する。中立的仮想計算機モニタは、その署名された拡張機能のプログラムの署名を自身が持つ管理者の公開鍵で検証し、それが正しければロードする。

ハイパバイザーコール vmmcall_tre_exec のインタフェースを図 6 に示す。binary_addr はロードするバイナリのアドレス、binary_size はロードするバイナリのサイズである。sign_addr は中立的仮想計算機モニタの管理者による署名のアドレス、sign_size は署名のサイズである。

5. アプリケーション

提案方式を評価するために、次のようなアプリケーションを実装する。

- デジタル署名
- 著作権管理

5.1 デジタル署名

著作権管理アプリケーションとして次の 2 つを実装することを検討している。第 1 のデジタル署名を行うアプリケーションでは、文献 15) と同様の処理を本研究の耐タンパー環境を用いて実装する。文献 15) では計算力の低い携帯端末から HTTP を用いてサーバにデータを送り委託計算を行っていた。本研究においてアプリケーションは、図 8 のような耐タンパー環境側の署名サーバプログラムと、耐タンパーデバイス側の署名計算プログラムからなる。耐タンパーデバイスが耐タンパー環境に署名処理の一部を委託する。例えばショート署名に用いられる楕円曲線暗号においてのスカラ倍計算や、RSA 暗号における累乗と剰余の計算を耐タンパー環境に委託することが考えられる。従来の委託計算とは異なり、本研究の耐タンパーデバイスを用いた委託計算はネットワークに接続しなくても動作する利点がある。

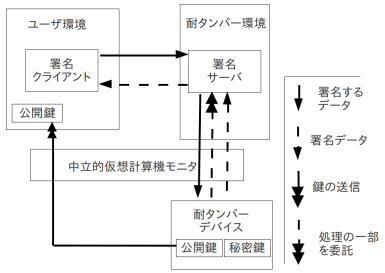

図 8 耐タンパー環境内での委託計算のサーバの実行

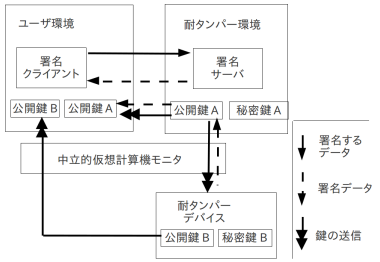

図 9 耐タンパー環境内での動的に生成した鍵による署名

第 2 のデジタル署名を行うアプリケーションでは、図 9 で示されているように、ユーザ環境は耐タンパーデバイスの公開鍵 B をあらかじめ受けとっておく。ユーザ環境の署名クライアントが耐タンパー環境に対してデータの署名を要求した場合、耐タンパー環境内の署名サーバは秘密鍵 A と公開鍵 A を動的に乱数で生成する。次に、署名サーバはその公開鍵 A を耐タンパーデバイスの秘密鍵 B で署名する。次に、署名サーバはデータを秘密鍵 A で署名する。最後に、署名サーバは署名と公開鍵 A をユーザ環境に渡す。署名の検証には、まず公開鍵 A が正しいことを公開鍵 B によって検証する。次に、公開鍵 A で署名を検証する。

5.2 著作権管理

著作権管理アプリケーションとして 2 種類実装を行う。第 1 の著作権管理アプリケーションは超流通[8] の仕組みを用いる。このアプリケーションは図 10 のように、ユーザ環境で動作する保護対象アプリケーションと耐タンパー環境で動作する保護したいコードからなる。保護したいコンテンツ (プログラム) の内部には耐タンパー環境との RPC を行う処理を埋め込んでおく。保護したいコードはあらかじめ暗号化し、拡張機能のバイナリに埋め込んでおく。ロードされた暗号化コードを耐タンパーデバイスの鍵を用いて復号する。この方式では保護したい処理のほとんどを、耐タンパー環境内で処理することで高速化する。低い頻度

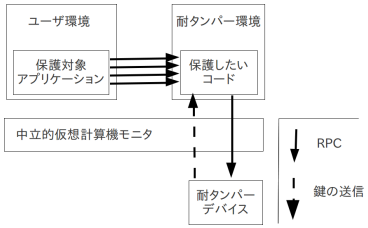

図 10 超流通による著作権管理アプリケーション

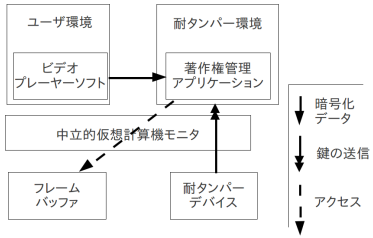

図 11 直接描画による著作権管理アプリケーション

で重要な処理を耐タンパーデバイスで行うこともできる。

第 2 の著作権管理アプリケーションは図 11 のように、直接ユーザ環境の出力デバイス (この例ではフレームバッファ) にアクセスする。このアプリケーションはユーザ環境のビデオプレーヤーソフトから、暗号化された映像のデータを受けとる。そしてこのアプリケーションは耐タンパーデバイスから受けとった鍵によって映像を復号し、仮想計算機モニタによるフレームバッファへのアクセス機能[10] を使い直接描画する。この時、ユーザ環境からのフレームバッファへのアクセスを一時的に禁止する。

5.3 実験環境

耐タンパーデバイスを独自に開発するのはコストがかかる。そのため本研究ではユーザの PC と別の PC をシリアルケーブルでつなぎ、別の PC を耐タンパーデバイスとみなしてアプリケーションを実行する。

6. 耐タンパー環境が提供する耐タンパー性

耐タンパー環境がどれだけ耐タンパー性を持っているかの評価を行う。現在の耐タンパー環境ではユーザ環境と完全に隔離されているため、ユーザ環境側からデータを盗み見られることはない。中立的仮想計算機モニタに脆弱性があったり改竄されてしまった場合は盗み見られる可能性がある。この問題に対しては Trusted Boot によりユーザ環境・耐タンパー環境双方で改竄を検知できるようにすることで対処する。OS 上で動作するプログラムの場合、デバッガ上で実行することで解析されることがあるため、耐タンパー性を持つプログラムを実装するためにはデバッガの検知機能を実装する必要がある。本研究では、耐タンパー環境のプログラムは決してデバッガ上で実行されることはないので、デバッガの検知機能を実装する必要はない。耐タンパー環境の誤った出力から内部の処理を類推するフォールト・ベース攻撃や処理時間の違いを利用するタイミング攻撃は耐タンパー環境のプログラム内で対処すれば防ぐことができる[5]。例えばフォールト・ベース攻撃の場合は出力を返す前に不正な値でないかチェックしたり処理に乱数を利用して戻り値の推測を防ぐという対処方法が知られている。タイミング攻撃の場合はランダムな処理の遅延を発生させたり、入力メッセージに乱数を影響させるメッセージブラインディングの手法[6] が知られている。

耐タンパー環境は特別なハードウェアを使用していないので、ハードウェアに対する攻撃には対処できない。ハードウェアに対する攻撃としては、メモリを抜かれることによるデータの盗難、処理による消費電力の違いを利用した内部の解析などが考えられる。

7. おわりに

耐タンパーデバイスには処理速度が遅いという問題がある。この論文では、PC 上に通常の OS(Operating System) の環境 (ユーザ環境) と隔離された、耐タンパーデバイスのプログラムを動作させるための環境 (耐タンパー環境) を構築することでこの問題を解決する方法を提案した。これらの 2 つの環境を隔離するため、本研究では中立的な仮想計算機モニタを用いる。中立とは、CPU などのハードウェアと同様に、仮想計算機モニタが期待された動作のみを行い、悪意がある行動をとらないことを意味する。中立性は、ソースコードの開示と TPM(Trusted Platform Module) を用いた Trusted Boot により担保される。

この論文では、中立的仮想計算機モニタを BitVisor を拡張することで実装する方法を示した。BitVisor はゲスト OS を 1 つしか実行できないという問題があるが、拡張機能があり、独立したバイナリを保護された領域で実行することができる。この機能を用いて、耐タンパー環境を実現する。ユーザ環境と耐タンパー環境の間の通信としては、RPC(Remote Procedure Call) を提供する。BitVisor が改竄されていないことを検証するために、BitVisor に Trusted Boot 機能を実装した。Trusted Boot による検証のコストを下げるために、BitVisor に対して動的に拡張機能プログラムをロードする仕組みを付加した。この論文では、耐タンパー環境を利用したアプリケーションとして、デジタル署名と著作権管理の実現方法を示した。デジタル署名では、委託計算を用いる方法、および、動的に公開鍵を生成する方法を示した。著作権管理では、超流通と同じ方法、および、仮想計算機から直接出力デバイスをアクセスする方法を示した。

現在までに、BivVisor の拡張機能のプログラムを動的にロードする機能を実装した。また、ユーザ環境と耐タンパー環境との間の RPC も動作している。今後は、アプリケーションを実装し、提案方式による高速化が可能であることを示す。また、耐タンパー環境からも TPM を用いて改竄されていないことを検証する機能を実装する。

謝辞 本研究で利用した BitVisor の拡張機能は、筑波大学システム情報工学研究科コンピュータサイエンス専攻 (現在、東京大学情報基盤センター) の品川高廣氏の設計による。この拡張機能がなければ、耐タンパーデバイス環境を実装することは困難であった。ここに深く感謝の意を表する。

本研究の一部は、日本学術振興会科学研究費補助金 (挑戦的萌芽研究)、および、総務省戦略的情報通信研究開発制度 (SCOPE) の支援を受けて行われた。

参 考 文 献

1) Stefan Berger, RSailer, and LVan Doorn. vTPM Virtualizing the Trusted Platform Module. *Proceedings of the 15th USENIX Security Symposium*, pp. 305–320, 2006.
2) BitVisor Project. Bitvisor1.1.1. `http://www.bitvisor.org/`, 2010.
3) John Burns and Chris J. Mitchell. Parameter Selection for Server-Aided RSA Computation Schemes. *IEEE Transactions on Computers*, Vol. 43, pp. 163–174, February 1994.
4) Sudeep Ghosh, JasonD. Hiser, and JackW. Davidson. A Secure and Robust Approach to Software Tamper Resistance. *Proceedings of the 12th International Conference on Information Hiding*, pp. 33–47, 2010.
5) 情報技術標準化センター. 耐タンパー性研究委員会報告書. 財団法人　日本規格協会, 2003.
6) Paul Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. *CRYPTO '96*, pp. 104–113, 1996.
7) ChaeHoon Lim and PilJoong Lee. Security and Performance of Server-Aided RSA Computation Protocols. *CRYPTO' 95*, pp. 70–83, 1995.
8) 森亮一. ディジタル革命とその未来を支える基盤技術. 情報処理学会論文誌, Vol.41, No.11, pp. 2950–2957, 2000.
9) OpenPTS. Open Platform Trust Services. `http://sourceforge.jp/projects/openpts/`, 2011.
10) 大山恵弘. ハイパバイザによる広告表示. 情報科学技術フォーラム講演論文集, pp. 341–342, 2010.
11) 佐久間充, 大山恵弘. 仮想マシンモニタを用いた OS コードの秘匿化. 情報処理学会研究報告. CSEC, [コンピュータセキュリティ], Vol. 2010, No.35, pp. 1–6, 2010.
12) Kevin Scott and Jack Davidson. Strata: A Software Dynamic Translation Infrastructure. *IEEE Workshop on Binary Translation*, 2001.
13) Takahiro Shinagawa and etal. BitVisor: a Thin Hypervisor for Enforcing I/O Device Security. *2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments(VEE '09)*, pp. 121–130, 2009.
14) Sony Computer Entertainment Inc. Documents of PS3 Linux Distributor's Starter Kit, 2008.
15) 田村洸太, 金山直樹, 金岡晃, 伊藤忠彦, 満保雅浩, 岡本栄司. 検証機能を有する楕円曲線上のスカラ倍算の委託計算. 電子情報通信学会技術研究報告. ISEC (情報セキュリティ), Vol. 109, No. 207, pp. 73–78, 2009.