BitVisorにおけるゲストOS・保護ドメイン・USB ドングル間の遠隔手続き呼び出し

新城靖、松下 正吾

筑波大学

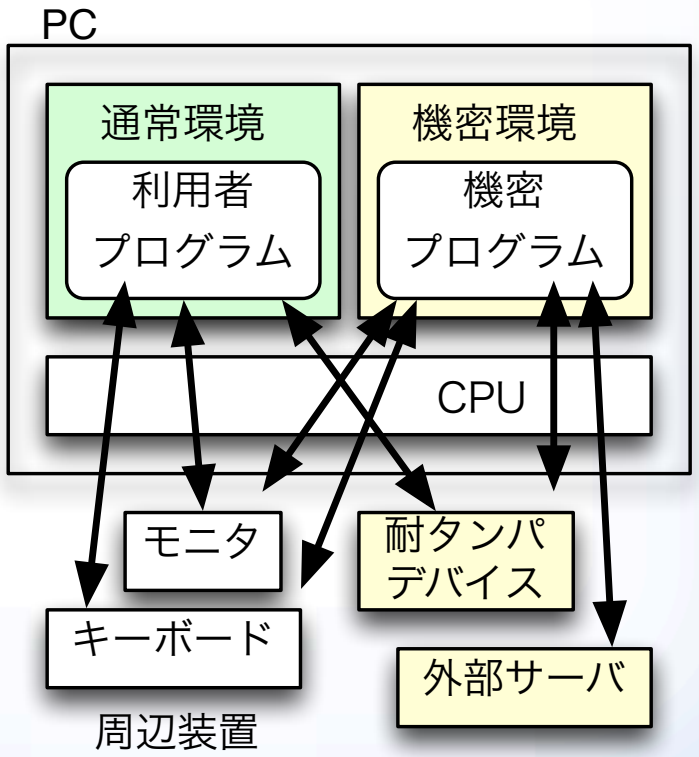
BitVisor Summit 2 (2013年12月5日)

背景: 耐タンパデバイス

- 耐タンパ性があるデバイス
  - 外部から内部のデータやコードが解析しにくいデバイス
  - 例:
    - IC カード
    - USBドングル: カードリーダ不要(USBポートが必要)
      - 飛天 Rockey6 (C51言語でのプログラミング)

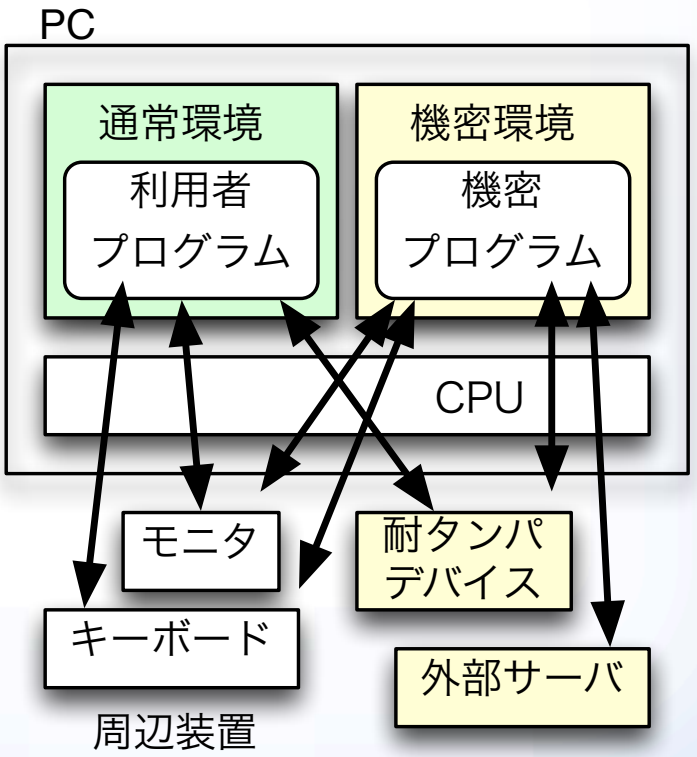
機密環境

- 通常の実行環境（通常環境）と同じコンピュータ内にあるが、データとコードを隠すことができるプログラムの実行環境。
- 一定の耐タンパ性が求められる。



機密環境

- 通常環境と機密環境は資源を共有する。
  - ハードウェア: メモリ、CPU時間、ネットワーク
  - ソフトウェア: TCPスタック、GUI、ファイルシステム



共有する所に意義（問題）がある。

機密環境の応用

- ディジタル署名
- 著作権保護、コード隠し、超流通
- 電子商取引トランザクションクライアント
- VPNゲートウェー

[1] Lenin Singaravelu, Calton Pu, Hermann Härtig, and Christian Helmuth. "Reducing TCB complexity for security-sensitive applications: three case studies.", ACM European Conference on Computer Systems (EuroSys), pp.161-174, 2006.

機密環境の実装方法

- プロセス。ptrace() で破れる。
- OSカーネル。カーネル・モジュールで破れる。
- マイクロカーネル [1]
- VMM: **BitVisor保護ドメイン**
- CPU: Intel SGX (Software Guard Extensions)[2]

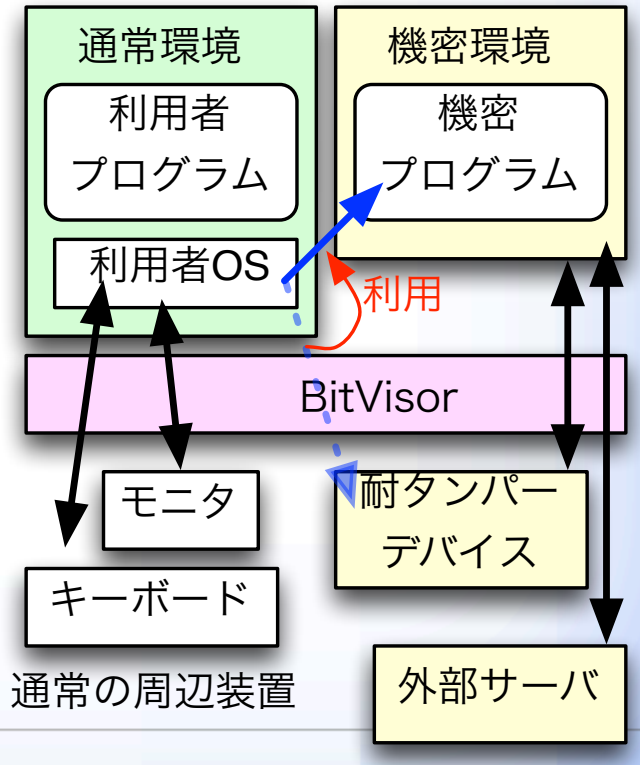
[2] Workshop on Hardware and Architectural Support for Security and Privacy, https://sites.google.com/site/haspworkshop2013/

BitVisor 保護ドメインによる機密環境の実装

- 耐タンパデバイスの高速化[3]
- 動画像著作権保護[4]

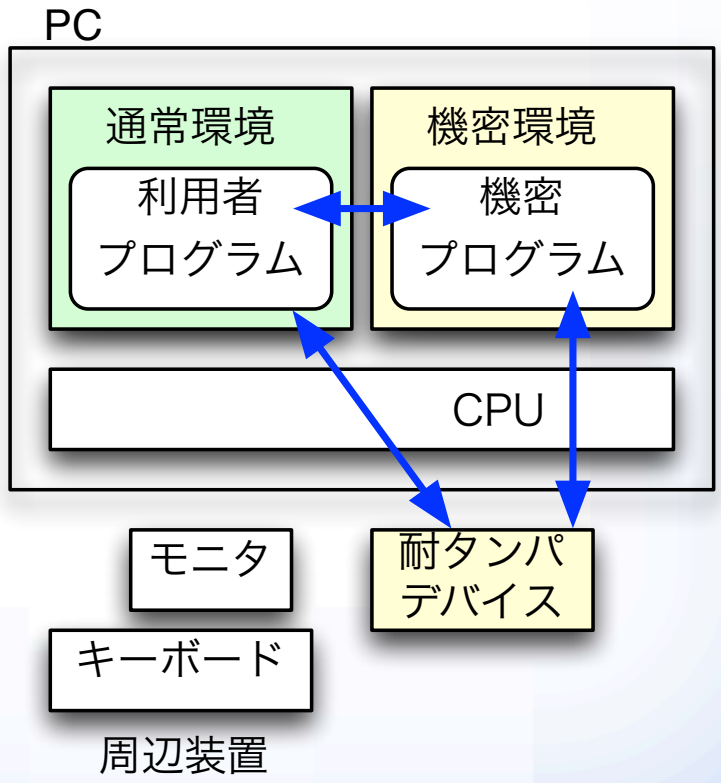
[3] 松下 正吾, 新城 靖, 榮樂 英樹, 松原 克弥, 東 悠: "中立的仮想計算機モニタによる耐タンパーデバイスのアクセラレータの実装", 情報処理学システムソフトウェアとオペレーティング・システム研究会,2011-OS-118(9) (2011).

[4] 小清水 滋, 新城 靖, 板野 肯三, 榮樂 英樹, 松原 克弥: "中立的VMMによる動画像を対象とした著作権保護", 情報処理学システムソフトウェアと オペレーティング・システム研究会, 2012-OS-123(8), (2012).OS-118(9) (2011).



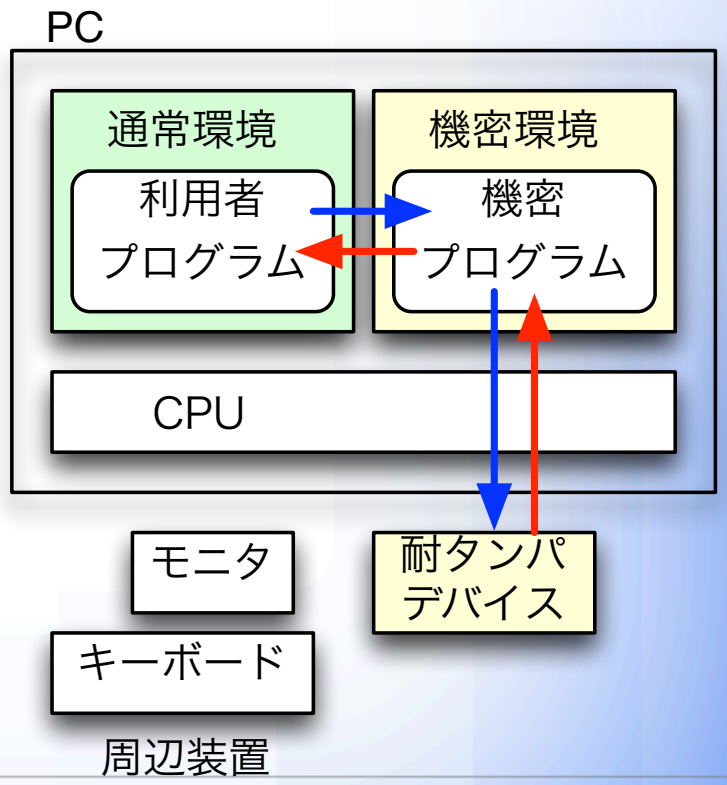
問題: 環境間通信はどうあるべきか

- 通常環境と機密環境の間
- 通常環境と耐タンパ・デバイスの間
- 機密環境と機密環境の間



提案: RPC+逆RPC

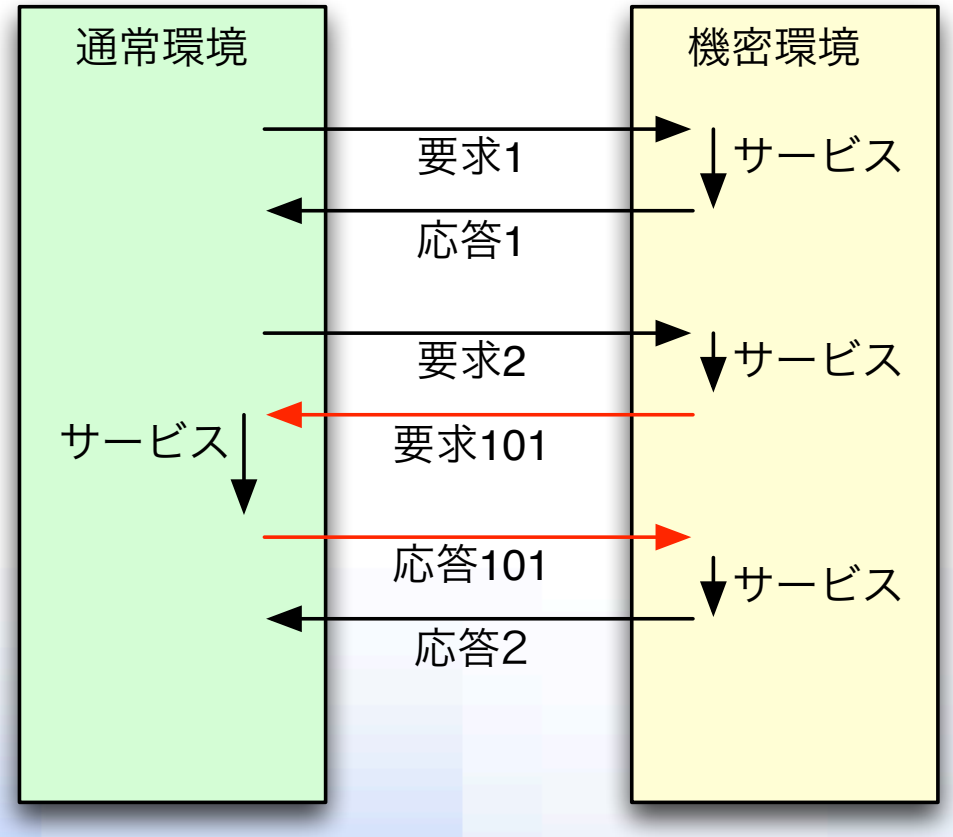
		サーバ		
		通常	機密	デバイス
クライアント	通常		RPC	
	機密	逆RPC		RPC
	デバイス		逆RPC	



RPC, 逆RPCの要件

- 機密環境／耐タンパ・デバイスは、基本的には RPC のサーバとしてサービスを提供する。
- 機密環境／耐タンパ・デバイスは、 時々、外部のサービスをクライアントとして利用する。
  - 暗号の委託計算(server-aided computation)
  - ネットワーク通信、ストレージ
  - ユーザ・インタフェース
- 機密環境／耐タンパ・デバイスは、 スリープできない。

逆RPCを実装するメッセージ



メッセージの構造

- ヘッダ: 固定長
  - 種類: 要求か応答か
  - ID: 要求と応答のマッチングに使う数。クライアントは、要求を発信する度にユニークなIDを生成。
  - RPCの手続き番号
  - 状態(エラーコード)
  - メッセージの長さ(ヘッダ+本体)
- 本体: 可変長の opaque

BitVisor 保護ドメインRPC

- ハイパバイザコールによる実装

  int vmmcall_rpc(int desc, struct tre_rpc_request *request,

  struct tre_rpc_result *result)

```
enum tre_rpc_direction {
        TRE_RPC_DIRECTION_REQUEST,
        TRE_RPC_DIRECTION_RESULT
};
struct tre_rpc_request {
        enum tre_rpc_direction direction;
        ulong id;
        ulong  proc_number;
        ulong  len;
        char  args[0];
};
```

```
struct tre_rpc_result {
        enum tre_rpc_direction direction;
        ulong id;
        ulong  status;
        ulong  len;
        char  results[0];
};
```

BitVisor 保護ドメインRPC のクライアント(ゲストOS)

```
int add(int desc, int a, int b)
{
    request = malloc(sizeof(struct tre_rpc_request)+ sizeof(*arg));
    request->direction = TRE_RPC_DIRECTION_REQUEST;
    request->id = request_count ++;
    request->proc_number = TRE_PROC_ADD;
    request->len = sizeof(struct tre_rpc_request)+ sizeof(*arg);
    arg = &request->args[0];
    arg->a = a;
    arg->b = b;
    result = malloc(sizeof(struct tre_rpc_result)+ sizeof(*res));
    memset( result, 0, sizeof(struct tre_rpc_result)+ sizeof(*res) );
    result->len = sizeof(sizeof(struct tre_rpc_result)+ sizeof(*res));
    r = vmmcall_rpc(desc, request, resut);
    res = (tre_rpc_add_result *)&result->results[0];
    c = res->c;
    free(result);
    return c;
}
```

BitVisor 保護ドメインRPC のサーバ (1/2)

```
int _start( int msgcode, int proc_number, struct msgbuf *buf, int bufcnt)
{
    if (msgcode == MSG_BUF) {
        return tre_svc(buf, bufcnt);
    }
}
int tre_svc(struct msgbuf *buf, int bufcnt)
{
    request = (struct tre_rpc_request *)buf[0].base;
    result  = (struct tre_rpc_result *)buf[1].base;
    switch( request->proc_number ) {
    case TRE_PROC_ADD:
        ret = tre_svc_add(request, result);
    default:
        ret = -1;
    }
    return ret;
}
```

BitVisor 保護ドメインRPC のサーバ (2/2)

```
int tre_svc_add(struct tre_rpc_request *request,
                struct tre_rpc_result *result) {
    arg = (struct tre_rpc_add_arg *)(request->args);
    res = (struct tre_rpc_add_result *)(result->results);
    res->c = arg->a + arg->b;
    result->len = sizeof(*result)+sizeof(*res);
    result->direction = TRE_RPC_DIRECTION_RESULT;
    result->id  = request->id;
    result->status = 0;
    return;
}
```

BitVisor 保護ドメインRPC の実装

```
int vmmcall_rpc (ulong desc,struct tre_rpc_request *request,
                 struct tre_rpc_result *result)
{
    copy_from_guest(&request_header, request, sizeof(struct tre_rpc_request));
    copy_from_guest(&result_header, result, sizeof(struct tre_rpc_result));
    arguments_buffer = allocate_buffer(request_header.len);
    results_buffer   = allocate_buffer(result_header.len);
    setmsgbuf(&vmm_buffers[0], arguments_buffer, request_header.len, 0);
    setmsgbuf(&vmm_buffers[1], results_buffer, result_header.len, 1);
    copy_from_guest(vmm_buffers[0].base, (u8 *)request, request_header.len);
    memcpy(vmm_buffers[1].base, &result_header, sizeof(struct tre_rpc_result));
    ret = msgsendbuf(desc, VMMCALL_RPC_NUMBER, vmm_buffers,
        sizeof(vmm_buffers)/sizeof(struct msgbuf));
    copy_to_guest(result, vmm_buffers[1].base, vmm_buffers[1].len);
    free_page_phys( arguments_buffer );
    free_page_phys( results_buffer );
}
```

保護ドメインからの逆RPC

- 保護ドメイン: 応答メッセージを作成する代わりに、要求メッセージを作成する。
- ゲストOS: メッセージを受信する。
  - 応答メッセージなら通常通りの処理
  - 要求メッセージなら、それに従ってサービスを提供する。

Rockey6 RPC

- クライアント: 保護ドメインのコード
- サーバ: Rockey6 内のコード
  - C51 言語で記述

Rockey6 RPC のクライアント

```
int rockey6_add( int a, int b)
{
    request_len = sizeof(struct rockey6_rpc_request)
        + sizeof(struct rockey6_rpc_add_request);
    request = malloc( request_len );
    result_len = sizeof(struct rockey6_rpc_result)
        + sizeof(struct rockey6_rpc_add_result);
    result = malloc(result_len);
    make_add_request( request, a, b );
    rockey6_write(hic, g_add_cmddata, run_id, request);
    rockey6_read(hic, g_add_cmddata, result);
    c = get_add_result( request );
    free( request );
    free( result );
    return( c );
}
```

Rockey6 RPC のサーバ (1/2)

```
void main()
{
    get_input(&request.id, 0, 2, 1);
    get_input(&request.direction, 4, 2, 1);
    get_input(&request.proc_number, 8, 2, 1);
    get_input(&request.len, 12, 2, 1);
    switch (request.proc_number) {
    case ROCKEY6_RPC_PROC_ADD:
        rockey6_svc_add(&request);
        break;
    }
    exit();
}
```

Rockey6 RPC のサーバ (2/2)

```
void rockey6_svc_add(struct rockey6_rpc_request *request)
{
    struct rockey6_rpc_result result;
    dword arg_a, arg_b, res_c;
    byte xdata result_buffer[20];
    get_input(&arg_a, 16, 2, 1);
    get_input(&arg_b, 20, 2, 1);
    res_c = arg_a + arg_b;
    result.id = request.id;
    result.direction = ROCKEY6_RPC_DIRECTION_RESULT;
    result.status = 0;
    result.len = 20;
    set_result_response(result_buffer, (byte *)&result, sizeof(result), 0);
    set_result_response(result_buffer, (byte *)&res_c, 4, sizeof(result));
    set_response(20, &result_buffer);
}
```

性能

- ThinkPad X61, Core2Duo T8300 2.4GHz
- RPCの往復時間
  - 保護ドメインRPC: 0.029ms
  - Rockey6 RPC: 224ms

その他の方法、改善

- BitVisor msgsendbuf()
  - 保護ドメイン側でのバッファの確保すれば、allocate_buffer(), ページのマップ／アンマップのオーバヘッドが減らせる。
- 保護ドメインとゲストOSの間の共有メモリがあれば、コピーが減らせる。

まとめ

- 機密環境は、promising。
- 保護ドメインは、機密環境として適している。特に、耐タンパデバイスと組み合わせが良い。
- RPC+逆RPCは、わかりやすい。

今後の課題

- 保護ドメインRPCの通信の標準化と最適化
- 逆RPCによるゲストOSのサービス利用の標準化