セキュア携帯機のためのアプリケーション機密分離および監視方式

太田 賢[†] 吉川 貴[†]
中川智尋[†] 稲村 浩[††]

携帯電話機の信頼性と機能性の両立のため，障害や攻撃からの端末システムおよびアプリケーション内の機密リソースの保護と，新たなサービスやハードウェアに対応するためのソフトウェア導入の柔軟性が要求される．従来のアプリケーションや OS 単位のドメイン分離技術によって信頼環境に機密リソースを隔離できるが，安定性のため，信頼環境には新たなソフトウェア導入が制限される問題がある．本研究は，1. 最小限の機密リソースを信頼環境に閉じ込める，細粒度の機密分離アルゴリズムと，2. 機密リソースにアクセスする API の誤用による障害やポリシ侵害を防止する機密監視方式を提案する．機密情報がアプリケーション実行中に内部で拡散する問題に対して，本アルゴリズムは，関数や引数に付与された機密の伝播解析により，機密データを扱う関数を分離し，信頼環境と通常環境で分散実行可能とする．一方，機密リソースごとに異なる監視ポリシに対応するため，機密監視方式は，API 呼び出しの引数や呼び出し順序，アクセス制御サーバへの照会やログ等多様で組み合わせ可能な監視規則をサポートする．機密分離ツールを実装してテストプログラムに適用するとともに，分散実行した際の関数/アプリケーションレベルのオーバヘッドを測定した．

Confidentiality Separtion and Monitoring for Mobile Applications

Ken Ohta,[†] Takashi Yoshikawa,[†] Tomohiro Nakagawa[†]
and Hiroshi Inamura[††]

Reliablity and security are mandatory requirements for mobile handsets. Critical functions and confidential data should be protected against failure and malware attacks. This paper addresses partitioning and monitoring for confidential resource in a mixed execution environments consiting of trusted and untrusted environments. We propose 1. application confidential separation algorithm to lock confidential resources in trusted domain, and 2. monitoring method to prevent misuse of API for confidential resource access. The confidential separation algorithm extracts functions handling confidential data from an application source code by confidential propagation analysis based on application and system confidential policies. The monitoring method supports various, incorporable verification rules. We applied an application confidential separation tool we developed to two test applications. We also measured funtion-level and application-level overhead of separated applications in trusted and untrusted VMs and PCs.

1. はじめに

生活やビジネスに密着した携帯電話機（端末と呼ぶ）には，様々な個人情報やビジネス情報の機密データ，電話や認証，課金等の機密機能が含まれ，高い安全性と信頼性が要求される．しかし，端末 OS やアプリケーションには，新たなサービスやコンテンツ，ハードウェアへの対応のため，高機能性と進化の速さが求められており，安全性と信頼性との両立が求められている．今後，携帯電話を対象としたウィルスの増加が予想されるのに対し，ソフトウェアの大規模化や多様化，無線 LAN や Bluetooth 等の新たなチャネルも加わり，端末の脆弱性は増加する傾向にある．

従来，障害や不正ソフトの攻撃から機密リソースを保護するため，ドメイン分離技術が用いられている．多くの端末に導入されている Java 仮想マシン（JVM）により，Java アプリケーションのバグや攻撃から，端末の機密を保護できる．また，Xen[1] や VMware を含む仮想マシンモニタ（VMM: Virtual Machine Monitor）[2] により，機能性に富む非信頼環境と，機密リソースを保持するクローズドな信頼環境を VM レベルで隔離して共存できる．Terra[4] は，バンキング等の各セキュアアプリケーションに対して 1 つの信頼 VM を割り当て，他の VM 内から隔離する．しかし，信頼

† NTT ドコモマルチメディア研究所
Multimedia Laboratories, NTT DoCoMo, Inc.
†† ドコモコミュニケーション研究所 USA
DoCoMo Communications Laboratories USA

VM には新しいライブラリやミドルウェアを自由に導入することは許されないため，機能性が犠牲になる．

本研究は，アプリケーションの機密リソースのみを信頼環境に閉じ込める細粒度の機密分離アプローチをとり，リッチな非信頼環境でアプリケーションを動作可能とする．非信頼環境と信頼環境が共存した実行環境として，端末だけでなく，PC や情報家電，サーバ等の外部デバイスと端末が連携した分散環境も想定する．入出力や処理能力に制約のある端末は外部デバイスとの連携により，画面サイズや操作性等のユーザ体験の向上，処理の負荷分散をはかることができる．しかし，外部デバイスの信頼度はその種別や利用形態に依存する．PC はソフトウェアの自由な入れ替えが可能であり，公共（キオスクやネットカフェ）で共有される外部デバイスには，不正ソフトの導入や攻撃の脅威があり，信頼できない．

本研究は，機密の分離と監視の 2 つを要求条件とし，1. 最小限の機密リソースを信頼環境に閉じ込める，細粒度の機密分離アルゴリズムと，2. 機密リソースにアクセスする API（機密 API と呼ぶ）の誤用による障害やポリシ侵害を防止する機密監視方式を提案する．第 1 に，アプリケーション独自の機密データや，機密リソースにアクセスするための API を通じて信頼環境から非信頼環境に持ち出された機密データは，アプリ実行中に内部で伝播する問題がある．本アルゴリズムは，関数や引数に付与された機密の伝播解析により，機密データを扱う関数を分離し，信頼環境と通常環境で分散実行可能とする．第 2 に，機密リソースの種別は多様であり，その利用手順やパラメータ，重要性は異なるため，単純なアクセス可否やログの規則だけではポリシを反映できない．機密監視方式は，API 呼び出しの引数や呼び出し順序，アクセス制御サーバへの照会や API のロック，入出力のログ等多様で組み合わせ可能な監視規則を記述可能とする．

実用性評価のため，機密分離アルゴリズムを組み込んだ機密分離ツールを実装し，2 つのテストプログラムに適用した．分離プログラムは RPC により分散実行されるため，端末内の VM 隔離構成に加えて，端末と外部デバイスによるデバイス隔離構成でも動作可能である．両構成において，関数およびアプリケーションレベルのオーバヘッドを測定した．

以下，2 章で関連研究について述べ，3 章で端末アーキテクチャを説明した後，機密分離アルゴリズムと機密監視方式を提案する．4 章で動作検証と性能評価を行い，5 章でまとめとする．

2. 関連研究

2.1 機密分離

アプリケーション内の機密リソースを保護するため，従来，難読化，暗号化，多様化，アンチデバッグ等をソースコードやバイナリに施す耐タンパソフトウェア技術[3] が開発されている．本研究は，アプリケーションの機密を分離する方式に位置づけられる．

アプリケーション分離技術の Privtrans[5] は，権限昇格防止のため，管理者特権や機密保護を要求する関数セットをライブラリに分離する．ソースコード中の変数や関数に対して開発者が付与する機密属性のアノテーションを伝播解析し，機密の関数の呼び出しを RPC に置き換える．しかし，ソースコードに機密保護要求を組み込むため，アプリケーション開発者や端末の管理者等による複数のポリシの反映や，別の環境における異なる機密保護要求に対応するのは難しい．本論文は，ソースコードと独立したポリシファイルを反映可能な機密分離アルゴリズムを提案する．

また，携帯端末用 Java プロファイルである DoJa☆は，XString と呼ばれる端末外に持ち出せないシステムの機密文字列データを表すクラスを提供している．たとえば電話帳を扱う Java アプリは，端末内の電話番号を保持する XString インスタンスを利用して，電話番号の画面表示や電話の発信ができるが，文字列の取り出しや比較，外部出力を禁止して，機密漏洩を防止している．ただし，アプリケーション独自の機密リソースを保護する仕組みを持たない．

2.2 機密監視

サンドボックス技術の Software Pot[6] は，アプリケーションごとにシステムの機密リソースに対するアクセス制御の可否をポリシとして与えることができる．本研究は，アクセス制御の可否のみでだけではなく，アクセス制御の管理サーバへの照会やログ等を組み合わせてポリシに記述可能としている．

Semantic Remote Attestation[7] は，Java VM 等の VM ランタイムに，クラス階層やプログラムの実行状態，入出力等アプリの動作を監視するモジュールを組み込む，VM ベースの動作監視技術である．VM ランタイムが信頼できることを前提としているため，本研究の想定する非信頼環境の動作は監視できない．

☆ http://www.nttdocomo.co.jp/english/p_s/i/make/java/index.html

3. 機密分離と監視

3.1 端末アーキテクチャ

本論文が対象とする 2 種の実行環境を**図 1** に示す．(a) は VM，(b) はデバイスレベルの隔離構成をとり，端末（MH: Mobile Host）上のシステム VM は信頼環境，ユーザ VM と外部デバイス（XH: External Host）は非信頼環境であるものとする．非信頼環境はユーザが任意のソフトウェアを導入可能であるのに対し，信頼環境はデジタル署名の検証等の手段により，信頼ソフトウェアのみ導入可能であるとする．システム VM 内には，システムミドルウェア，機密ライブラリ，モニタの 3 つのビルディングブロックが含まれる．

- システムミドルウェア：端末の機密機能（認証や電話機能等）や機密データ（電話帳等）の管理機能を実装し，そのアクセスのための機密 API（認証 API，電話帳 API 等）を提供する．
- モニタ：アプリケーションからの機密 API 呼び出しを受け付け，アクセス制御と監視を行う．
- 機密ライブラリ：アプリケーション本体から分離された機密関数のセット．本体と機密ライブラリは別々か，一緒に配布され，本体は非信頼環境，機密ライブラリは信頼環境に導入されるものとする．

本論文は，デバイス隔離構成に対応するため，機密 API 呼び出しに RPC（Remote Procedure Call）を利用する．ただし，提案の機密分離・監視方式は API 呼び出し手段とは独立であり，他のドメイン間通信手段に適用できる．RPC の利用において，API 呼び出しの引数や戻り値のシリアライズと通信処理のため，アプリケーション側にはクライアント stub，機密ライブラリやシステムミドルウェア側にはサーバ stub が必要となる．システムミドルウェアとアプリケーションの開発者は，rpcgen 等の stub 作成ツールを利用して，システムミドルウェア用の stub，機密ライブラリ用の stub を用意するものとする．以下，アプリケーションは C 言語で記述されているものと仮定する．

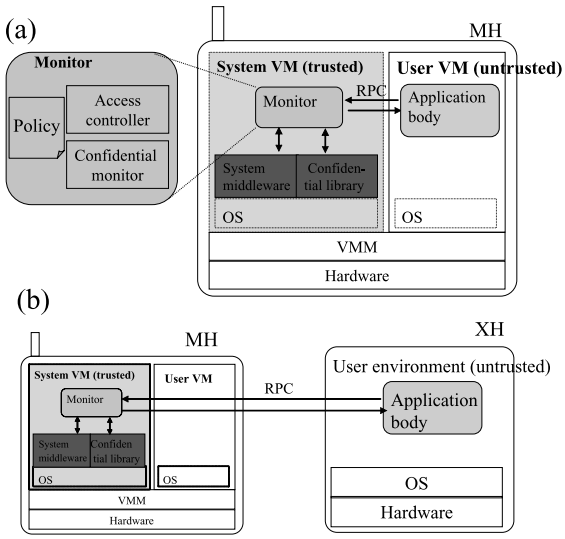

図 1 信頼/非信頼実行環境：(a) VM 隔離構成，(b) デバイス隔離構成

Fig. 1 Trusted and untrusted execution environments: (a) VM-based domain separation, (b) Device-based domain separation.

3.2 機 密 分 離

機密分離の手順を**図 2** に示す．開発者は，アプリケーションを作成して動作確認した後，機密分離ツール（CST）にソースコードと機密システム/アプリケーションポリシを入力する．CST は，機密分離アルゴリズムに基づき，関数呼び出しやデータ操作による機密伝播を解析し，機密データを扱う関数を機密ライブラリとして分離する．最後に，RPC 用の stub と合わせてコンパイル，リンクされ，アプリケーション本体の executable と機密ライブラリが生成される．

図 3，**図 4** にサンプルのコードとポリシを示す．システムミドルウェアとして，ライセンスの再生回数を制御する *create_license_handle*()，*get_count*()，*set_count*() という API が提供され，アプリ開発者は，

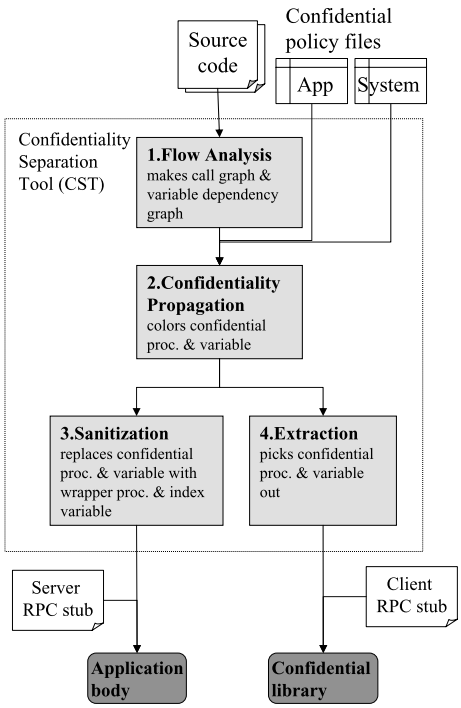

図 2 機密分離手順

Fig. 2 Confidentiality separation procedure.

```
bool f2(license_info license){
 bool rv;
 int count;
 rv = get_count(license, &count);
 count = count-1;
 if(count < 0) return FALSE;
 rv = set_count(license, count);
 return rv;
}
void f3(){
 // some critical processing
}
bool f1(license_info license){
 f3();
 return f2(license);
}
int main(int argc, char **argv){
 license_info license;
 bool rv;
 rv = create_license_handle("license1",&license);
 if(rv == FALSE) return FALSE;
 f1(license);
}
```

図 3 サンプルコード
Fig. 3 Sample code.

```
// 機密アプリケーションポリシー
<function name="f3" sensitive=true>
</function>
// 機密システムポリシー
<function name="create_license_handle" >
 <return sensitive="false" />
 <arg sensitive="false" name="filename" />
 <arg sensitive="true" name="license" />
</function>
<function name="get_count">
 <return sensitive="false" />
 <arg sensitive="true" name="license" />
 <arg sensitive="true" name="count" />
</function>
<function name="set_count">
 <return sensitive="false" />
 <arg sensitive="true" name="license" />
 <arg sensitive="true" name="count" />
</function>
```

図 4 サンプルポリシ
Fig. 4 Sample policy.

ライセンスにある再生回数を 1 減らす関数 f2 と f2 を呼び出す f1 を作成している．機密システム/アプリケーションポリシは，それぞれシステムミドルウェアとアプリケーションの開発者によって提供される．ポリシには，各関数の保護が必要な引数（arg）と戻り値（return）に機密属性（sensitive=true）が指定される．機密システムポリシとして関数 *create_license_handle*() については，引数 license に機密属性が与えられている．また，機密アプリケーションポリシとして関数 f3 自身に機密属性が与えられている．

3.2.1 機密分離アルゴリズム

アルゴリズムは，ソースコードからコールグラフと変数依存グラフを作成した後（図 2 のステップ 1），機密アプリケーション/システムポリシを適用して機密伝播解析を行う（ステップ 2）．両ポリシの関数内で機密属性が与えられている変数に sensitive フラグを立てる．次に，以下の手順を新たな sensitive フラグがセットされなくなるまで繰り返す．

(1) sensitive フラグが立った変数（機密変数と呼ぶ）の使用箇所に注目．関数の引数に指定されている場合，ポリシ内の当該関数の引数と戻り値の機密属性を参照し，ソースコード中の変数に sensitive フラグをセット．
(2) 同様に，機密変数の使用箇所に注目し，代入された変数に sensitive フラグをセット．

そして，機密変数を直接，操作している関数を機密関数としてマークする．たとえば，関数内で機密変数を加算や乗算等の演算子で操作したり，strcpy や puts 等の標準ライブラリの関数の引数として利用したりする場合である．ステップ 2 の最後に，アプリケーションと機密ライブラリの境界となる関数を見つけ，境界機密関数としてマークする．コールグラフを main から順にたどり，現在注目している関数の呼び出し先を順に探索する．呼び出し先が機密関数の場合，それが境界機密関数となる．境界機密関数の呼び出し先は探索の必要がないため，枝狩りしながら，コールグラフの全探索を行う．

ステップ 3 は，機密情報の浄化のため，ソースコードの書き換えを行う．具体的には，境界機密関数の呼び出しをクライアント stub の呼び出しに置き換え（sensitive_接頭辞を関数前に付与），グローバルおよびローカル変数の機密変数の型を，システム VM 内の実体を指し示すインデックス型（sensitive_t）に書き換える．最後にステップ 4 で，機密ライブラリのソースコードとして，境界機密関数と機密関数の実体を抽出する．

図 3 の場合，機密システムポリシによって，変数 license と変数 count の機密属性が f1，f2 に伝播する．**図 5** に対応するコールグラフと変数依存グラフを示す．次に機密関数の抽出において，f2 は演算子を利用して count を操作しており，f3 はポリシによって機密属性が与えられているため，機密関数としてマークされる．最後に，境界機密関数として f2 と f3 が選ばれ，f1 は以下のように書き換えられる．

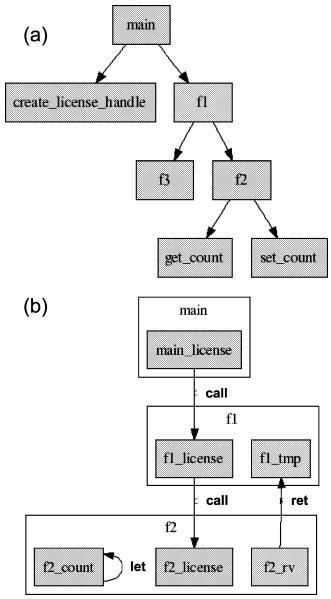

図 5 機密伝播解析
Fig. 5 Confidential propagation analysis.

```
bool f1(sensitive_t license){//index var(handle)
  bool tmp ;  {
  sensitive_f3();              // client RPC
  tmp = sensitive_f2(license); // client RPC
  return (tmp);}
}
```

3.2.2 機密変数管理機構

機密分離アルゴリズムにより，機密変数はシステム VM 内の実体を指すインデックス型変数（ハンドル）に書き換えられる．アプリケーションは機密 API の引数にハンドルを指定して実体を操作可能となる．ハンドルと実体の対応付けは，機密ライブラリに組み込まれる機密変数管理機構がテーブルで管理する．機密 API の引数にハンドルが指定されていた場合，機密変数管理機構は，テーブルから実体の値を取得して置換する．一方，API 呼び出しの戻り値としてハンドルを返す際，テーブル中にそのハンドルに対するレコードがあるかを変数名で検索する．存在しない場合は，乱数生成器によって生成したハンドルの値と，変数名，実体の値（文字列や整数，ポインタ等）の 3 つのフィールドを持つ新規レコードをテーブルに追加する．

なお，機密変数の管理の単純化と障害に対する頑健性のため，機密ライブラリとアプリケーションのインスタンスは，セッション ID を介して 1 対 1 に対応させる設計をとった．アプリケーションが開始時に通知するシグナルをトリガとして，モニタは機密ライブラリの新しいインスタンスを起動し，機密ライブラリはセッション ID を作成してアプリケーションに通知する．アプリケーションが異常終了した場合に，セッションを正常に終了できない問題に対し，本実装では，アプリケーションに一定周期のキープアライブ API の呼び出しを義務付けることで，タイムアウトの際に機密ライブラリを終了してリソースを開放可能とした．

3.3 機密監視

モニタ内の機密監視機能は，システムミドルウェアやアプリケーション提供者が与えた機密監視規則を利用して検証を行う．機密監視規則は，アプリケーションの悪意やバグを検知するために API 実行順序や引数を検査する API 呼び出し検証規則と，API 呼び出しの前後に実行すべき処理を規定する API 付帯規則を含み，以下の例のように組合わせて記述できるものとする．

- 当該 API の実行前に特定の API が，正常な戻り値で実行済みであることを検証．
- 引数に特定の文字列が現れた場合，管理サーバか，ユーザに問い合わせる．
- API の戻り値が特定の値である場合，指定の API のロックと，ログの記録を行う．

本論文では，監視規則に上記のような条件分岐や変数等の複雑な規則の記述を許すため，ruby スクリプトを利用することとした．機密監視規則は，*TDP* :: *PolicyValidator* クラスを継承したクラスに，各機密 API 名に pre_ と post_ を付与した 2 つのメソッドを追加する形で記述する．モニタは，ある API 呼び出しのアクセス制御の判定前に pre_ メソッドを，戻り値を返す前に post_ メソッドを起動する．

例として，4 章で述べるミュージックプレーヤ用の機密監視規則を**図 6** に示す．ライセンス関連の API の呼び出しが，*license_operation*，*license_operation*，*license_info_decrypt* の順で，同一のライセンス情報を引数として実行されたかどうかを，*license_info_load* API で読み出されたライセンス情報のポインタを追跡することで検証する．異常が検知された際，warn コマンドにより，ユーザに警告ダイアログを表示する．ほかにも以下のコマンドを記述可能である．

- query(server/user)：管理サーバやユーザに当該 API の実行可否を照会する．
- history(api_name=nil)：実行履歴 DB から，呼び出し順に機密 API の引数と戻り値を取り出す．省略可能な引数 api_name の指定により，特定 API の履歴のみを取得できる．
- log(info)：管理サーバに通知するログに記録する．
- delete(file)：指定のファイルを消去する．

```
class MPlayerPolicyValidator < ::TDP::PolicyValidator
 ...
 def post_license_info_load(req, res)
  if handle = res["license"]
   @license_info[handle] = false
  end
  return true
 end
 def post_license_operation(req, res)
  if res["rv"] == 0
   handle = req["license"]
   if @license_info.has_key?(handle)
    @license_info[handle] = true
   end
  end
  return true
 end
 def pre_license_info_decrypt(req)
  if handle = req["license"]
   unless @license_info[handle]
    warn("Misuse of API is detected")
    return false
   end
  end
  return true
...
```

図 6 機密監視規則例
Fig. 6 Monitoring policy example.

4. 実装と評価

4.1 機密分離ツール CST

提案の機密分離アルゴリズムを，C プログラムの解析・変換ツールである CIL[8] に追加して，CST を実装した．CIL によるコールグラフと変数依存グラフの解析結果を利用して，機密伝播とソースコードの書き換えを行っている．RPC は，xml-rpc（http://www.xmlrpc.com/）を利用した．ONC RPC（RFC 1831）と比較して，XML のパース処理による性能低下の懸念があるが，デバイス隔離構成におけるセキュリティ確保のための SSL や署名処理等との親和性から xml-rpc を選択した．テストプログラムとして，ミュージックプレーヤ（プレーヤと呼ぶ）と在庫管理ソフトの 2 つを作成し，プレーヤ用にライセンス操作，復号化，音声出力のためのシステムミドルウェアとその stub も実装した．

表 1 にテストプログラムに CST を適用した結果を示す．プレーヤの場合，システムミドルウェアの提供者は 32 の関数にポリシを与えたのに対して，アプリケーション開発者はポリシを与えていない．機密分離の結果，60 個の全関数のうち，25 個が機密関数，さらにそのうちの 22 個が境界機密関数とマークされた．21 個の境界機密関数は，システムミドルウェアの呼び出しであり，stub がすでに用意されていたが，残りの 1 つはアプリ独自の関数であり，アプリ開発者は新たにその stub を作るように CST に指示された．この関数は，システムミドルウェアが返したライセンス情報を直接操作するものであり，信頼環境で実行する必要があった．このように，アプリケーション開発者がポリシを与えなくても，端末システムから伝播した機密リソースを保護できることを確認できた．なお，機密分離によるバイナリのコードサイズの増加は，77%であり，主に RPC のサポートと，CIL によるコードの正規化（テンポラリ変数の追加等）による．

表 1 テストプログラムの機密分離
Table 1 Test results.

テストプログラム種別	プレーヤ	在庫管理
機密システムポリシ内関数（個）	32	1
機密アプリポリシ内関数（個）	0	1
関数の総数（個）	60	40
機密関数（個）	25	8
境界機密関数（個）	22	8
スタブ記述が必要な関数（個）	1	8
オリジナルコード（byte）	101,464	74,436
アプリ本体（byte）	101,120	73,780
機密ライブラリ（byte）	78,788	51,756
コード増加割合（%）	77	69

一方，在庫管理ソフトの場合，機密システム/アプリポリシとしてそれぞれ 1 つの関数に機密属性を与えた．機密伝播解析の結果，認証や在庫データを扱う 8 つの関数が境界機密関数と判定され，アプリ開発者は新たに 8 個の stub を作るように CST に指示された．複数のポリシファイルを反映できること，アプリ独自の関数も機密分離できることを確認した．

4.2 機密監視方式

機密監視の動作確認のため，ミュージックプレーヤにおけるライセンス検査関連の API 呼び出しに誤用がないことを，図 6 の機密監視規則に基づき，検証した．機密監視規則では，1．ライセンス情報の読み出し（license_info load）2．音楽再生回数のカウント（license_operation），3．ライセンス情報に含まれる鍵による音楽データの復号化（license_info_decrypt）が正常であると規定している．2 をスキップするようにアプリケーションを改造した結果，warn コマンドによる警告ダイアログが表示され，再生をブロックできることを確認した．また，API 呼び出しの引数についても，機密監視規則を記述し，不正範囲外の引数の値を検出できることを確認した．

4.3 性能評価

図 1 のとおり，MH と XH，ミュージックプレーヤ用のコンテンツサーバの 3 ホストから構成されるプロトタイプを構築した．MH には VMM として Xen を利用した．各ホストは，Pentium4 1 GHz，メモリ

表 2 関数呼び出し時間（ms）
Table 2 Function call time (ms).

	Original program	VM isolation	In same VM	Device isolation
is_sound_play	0.04	26.71	24.27	40.61
get_license	0.11	27.26	26.01	54.66
musicsrv_list	55.66	90.49	90.31	133.8
on_but_play	527.78	984.24	886.04	1,496.05

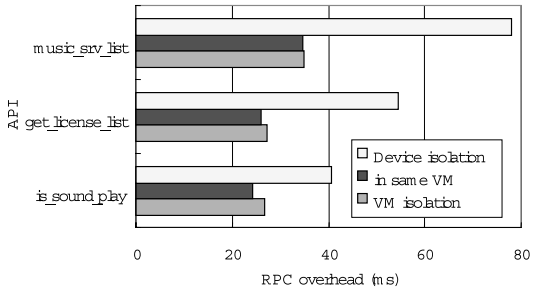

図 7 RPC オーバヘッド
Fig. 7 RPC overhead.

表 3 ONC RPC オーバヘッド（ms）
Table 3 ONC RPC overhead.

Parameter	VM Separation	Device Separation
integer	0.28	3.54
string	0.30	4.15
struct	0.37	4.39
buf（1 KB）	0.31	4.15
buf（4 KB）	0.62	9.22
buf（16 KB）	1.39	19.37
buf（64 KB）	5.60	64.21

1 GB の PC を利用した．そして，Xen による VM 隔離とデバイス隔離構成における API 呼び出しのオーバヘッドを調べるため，関数レベルとアプリケーションレベルで性能測定を行った．比較のため，機密分離を行う前の元プログラム（Original と呼ぶ）と，同一の VM 内で分散実行する場合も測定した．

表 2 に，ミュージックプレーヤの 4 つの API の呼び出し時間（1,000 回の試行の平均値）を示す．最初の 3 つは単一の関数であり，RPC のシリアライズの影響を調べるため，戻り値の XML データサイズが異なる API を選択した（それぞれ，384，937，3,089 バイト）．**図 7** は各 API の Original との呼び出し時間との差を示しており，これが機密分離のための XML-RPC オーバヘッドとなる．is_sound_play は処理量と応答データ量がともに少ない API であり，相対的に約 25 ms の RPC コストが大きくなる．一方，musicsrv_list はコンテンツサーバへのアクセスを含む比較的重い処理であり，そのオーバヘッドは 1.63 倍に抑えられる．また，VM 隔離と同一 VM の差に比べて，デバイス隔離のオーバヘッドが大きいことを確認した．

一方，表の 4 つ目の on_but_play は，アプリケーションレベルのオーバヘッド評価のために選択した関数であり，ユーザの再生要求に対する応答時間を示す．この関数は，再生ボタンを押した際に呼ばれ，内部でライセンス検査や復号化等の機密関数を呼び出して再生を開始する．VM 隔離の場合，応答時間は Original の 1.86 倍となり，ユーザビリティに大きな影響はなかったが，デバイス隔離の場合は応答時間が 1 秒を超え，応答性が悪化した．

アプリケーションレベルのオーバヘッドは，API 呼び出し回数 N と各 API によるデータ転送量 S に依存する．今後，N の削減のため，機密分離を最適化して複数の API 呼び出しを 1 つに集約させたり，オーバラップした呼び出しを行ったりする手法を導入することが考えられる．一方，S の削減には，XML-RPC の代替として，Xen ドメイン間通信や，ONC RPC 等の軽量の通信手段を利用できる．**表 3** に，ONC RPC による 7 つのテスト関数呼び出しの応答時間（5,000 回の試行の平均値）を示す．それぞれ，1 つの整数（integer），32 バイトの文字列（string），整数と 256 バイトの文字列を含む構造体（structure），1 KB から 64 KB のバルクデータを引数とし，引数の一部の表示処理を行い，整数を戻り値として返す機密関数を実装した．結果，XML-RPC よりもオーバヘッドが小さいこと，引数の違いによるシリアライズ処理時間の差は少なく，データサイズの影響が大きいことが確認できた．さらに，ローカル呼び出しに近づけるため，VM 隔離構成では Xen のイベントチャネルや共有メモリの仕組みが利用できる．

5. ま と め

携帯機の信頼性と機能性の両立のため，VMM によって信頼環境と非信頼環境を隔離した端末アーキテクチャにおける，細粒度の機密分離アルゴリズムと，機密監視方式を提案した．本アルゴリズムにより，アプリケーションから最小限の機密リソースを信頼環境に隔離し，機能やリソースが豊富な非信頼環境において，アプリケーションを動作させることができる．また，機密監視方式により，多様な機密リソースのポリシに応じた，API 呼び出し順序や引数の検査，管理サーバへの照会，データ消去等の多様な規則を，機密リソースの利用の際に強制可能とする．機密分離ツールを実装してテストプログラムに適用し，細粒度のアプリケーション分割が自動で実行できることを確認す

るとともに，ミュージックプレーヤを VM 隔離構成で分散実行するケースにおいて，アプリケーションレベルのオーバヘッドは許容範囲であることを確認した．

今後の課題は，VMM 上のドメイン間通信手段の安全化と高速化の両立，API の呼び出し頻度やデータ転送量等の正常・異常範囲を指定する機密監視規則の実現，アプリケーションやシステムミドルウェアの障害およびエラーに対するロバスト性確保のためのセッション/状態管理等である．

参 考 文 献

1) Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P. and Neugebauer., R.: Xen and the art of virtualization, *ACM Symposium on Operating Systems Principles* (Oct. 2003).
2) Rosenblum, M.: The reincarnation of virtual machines, *ACM Queue*, Vol.2, No.5 (July 2004).
3) Naumovich, G. and Memon, N.: Preventing piracy, reverse engineering, and tampering, *Computer*, Vol.36, No.7, pp.64–71 (2003).
4) Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M. and Boneh, D.: Terra: A virtual machine-based platform for trusted computing, *Symposium on Operating Systems Principles* (*SOSP*), (Oct. 2003).
5) Brumley, D. and Song, D.: Privtrans: Automatically partitioning programs for privilege separation, *13th USENIX Security Symposium* (Aug. 2004).
6) Oyama, Y. and Kato, K.: Softwarepot: An encapsulated transferable file system for secure software circulation, *Proc. Int. Symp. on Software Security*, pp.112–132, LNCS-2609, Springer (2003).
7) Haldar, V., Chandra, D. and Franz, M.: Semantic remote attestation: A virtual machine directed approach to trusted computing, *USENIX Virtual Machine Research and Technology Symposium* (May 2004).
8) Necula, G., McPeak, S., Rahul, S. and Weimer, W.: Cil: Intermediate language and tools for analysis and transformation of c programs, *Conference on Compiler Construction* (2002).

(平成 17 年 12 月 2 日受付)
(平成 18 年 3 月 2 日採録)

太田 賢（正会員）

平成 6 年静岡大学工学部情報知識工学科卒業．平成 8 年同大学大学院修士課程修了．平成 10 年同大学院博士課程修了．博士（工学）．平成 11 年 NTT 移動通信網（株）入社．現在，（株）NTT ドコモマルチメディア研究所勤務．平成 9 年度日本学術振興会特別研究会特別研究員．モバイルコンピューティング，端末セキュリティ，マルチメディア通信，分散システムに関する研究に従事．訳書『コンピュータネットワーク第 3 版』（プレンティスホール出版）等．電子情報通信学会，ACM 各会員．

吉川 貴（正会員）

平成 11 年慶應義塾大学総合政策学部卒業．平成 13 年同大学大学院修士課程修了．同年（株）NTT ドコモ入社．以来，モバイルコンピューティングに関する研究に従事．現在，同社マルチメディア研究所勤務．

中川 智尋（正会員）

平成 10 年東京大学工学部電子情報工学科卒業．平成 12 年同大学大学院修士課程修了．同年（株）NTT ドコモ入社．以来，アドホックネットワーク，P2P ネットワーク，端末セキュリティの研究に従事．現在，同社マルチメディア研究所勤務．電子情報通信学会会員．

稲村 浩（正会員）

平成 2 年慶應義塾大学大学院理工学研究科計測工学専攻修士課程修了．同年日本電信電話（株）入社．分散トランザクションシステム，分散ファイルシステムの研究開発に従事．平成 6〜7 年カーネギーメロン大学計算機科学科にて訪問研究員．平成 10 年より NTT ドコモ．モバイル環境におけるトランスポートプロトコルに関する研究開発に従事．電子情報通信学会，ACM 各会員．