推薦論文

耐タンパハードウェアを用いたモバイルエージェント保護手法

春木洋美[†,☆] 河口信夫[††,†††] 稲垣康善[†,☆☆]

近年，計算機の高性能化とネットワークの発達により，電子情報の手軽な交換・共有が可能になった．しかし，従来の技術では，一度データを配布すると，所有者がデータを管理できなくなるという問題が存在する．データ配布後にも所有者の意図を反映したデータ管理を行うために，モバイルエージェント技術を利用する．しかし，従来のモバイルエージェント技術では，移動先ホストからモバイルエージェントを保護できない．本論文では，各ホストに耐タンパハードウェアの存在を仮定したモバイルエージェント保護手法の提案をする．本手法では，耐タンパハードウェアと暗号化技術の利用により，ホストから秘密のデータを隠匿する．本手法により，モバイルエージェントを用いた柔軟なデータ管理が可能になる．

Protecting Mobile Agents against Malicious Hosts Using Tamper Resistant Hardware

Hiroyoshi Haruki,[†,☆] Nobuo Kawaguchi[††,†††] and Yasuyoshi Inagaki[†,☆☆]

Recently, it is getting possible to make simple data sharing and exchange among personal computers by advances in computers and wireless communication technology. However, in the existing technology, once a data has been sent to other hosts, it is impossible to control the redistribution of it. In this paper, we propose a data protecting and management method using mobile agents. However, mobile agent can be tampered by malicious hosts. We propose a mobile agent protection method using tamper resistant hardware for public key encryption. The method enables a flexible data management using mobile agents.

1. はじめに

近年，計算機の小型・高性能化とネットワークの急速な発達にともない，大量の電子情報を簡単に扱い，かつ，手軽に交換・共有することが可能になった．このような環境では，個人的な情報を含め文書・画像・音声・映像といった情報コンテンツを交換・共有する機会が増えて，利便性が著しく向上する．しかし一方で，一般に電子情報は自由に修正や再配布ができるため，情報のセキュリティの確保は難しくなる．

電子透かしや電子署名では，改竄の検知は可能であるが，盗聴や改竄自体を防ぐことはできない．電子情報を一度配布してしまうとその情報の利用を制限することは難しい．電子情報を配布した後にも，その読み書きや再配布の許可，回数や時間制限といった所有者の意図に沿った管理ができれば，電子情報の交換・共有を手軽に安心して行うことができる．ここで，所有者とは，個人情報の場合ならば対象となっている個人，あるいは，音楽・画像データならばその著作権者等，問題にしている情報に関する権利を有する者をいう．

そのような所有者の意図に沿った管理を行う情報配布手法の1つに，Adobe社が提案したPDF（Portable Document Format）がある[1)]．PDFでは，文書に対する制御情報として，フラグを付加することにより文書の保護を実現している．しかし，文書を扱うためのアプリケーションソフトを事前に使用装置にインストールしておく必要があり，また，PDFで決められ

† 名古屋大学大学院工学研究科計算理工学専攻
Graduate School of Engineering, Nagoya University
†† 名古屋大学情報連携基盤センター
Information Technology Center, Nagoya University
††† 名古屋大学統合音響情報研究拠点
Center for Integrated Acoustic Information Research (CIAIR), Nagoya University
☆ 現在，株式会社東芝研究開発センター
Presently with Corporate Research & Development Center, TOSHIBA Corporation
☆☆ 現在，愛知県立大学情報科学部
Presently with Faculty of Information Science & Technology, Aichi Prefectural University

本論文の内容は2001年6月のマルチメディア，分散，協調とモバイル（DICOMO2001）シンポジウムにて報告され，MBL研究会主査により情報処理学会論文誌への掲載が推薦された論文である．

た範囲内の文書保護しかできない．

これらの問題を解決するための手法として，配布後にも所有者の意図に沿った管理ができるコードと実行状態を持って移動するモバイルエージェントを用いることが考えられる．このような手法では，モバイルエージェント技術の利用により，PDF のようにデータを扱うためのアプリケーションを事前にインストールしておく必要がなく，しかも，所有者が自由にコードを作成・付加してデータを送ることができるため，従来の PDF 等で行われてきたデータ管理よりも柔軟な管理が実現できる．

しかし，モバイルエージェントには，移動先のホストコンピュータから攻撃を受けるというセキュリティ上の問題が存在する[2]．モバイルエージェントは，移動先のホストにコードと実行状態を持って移動する．すべてのコードと実行状態を渡す必要があるため，移動先のホストのモバイルエージェントの実行環境であるモバイルエージェントシステムが信頼できない場合，モバイルエージェントは実行状態の中に秘密にしたいデータ（以下，秘密データ）を持って移動することはできない．たとえ，秘密データを暗号化して移動したとしても，移動先のホストでモバイルエージェントを実行するためには，復号機能を移動先のホストが持つ必要がある．それがあれば，移動先のホストは暗号化された秘密データを盗聴・改竄できることになる．

一方，近年，出会ったその場で作られるネットワーク，アドホックネットワークの研究がさかんに行われている[3),4)]．本論文では，出会ったその場でデータの交換も想定するため，アドホックネットワーク上で動作するモバイルエージェントの保護も考慮する必要がある．しかし，アドホックネットワークでは，ホストの出入りが激しく，信頼できるホストの存在を仮定できない．

そこで，本論文では，各ホストに信頼できる箇所として，耐タンパハードウェアの存在を仮定したモバイルエージェント保護手法を提案する．ここで，耐タンパハードウェアとは，許可されていない変更を行うと正常に動作しないハードウェアである．

本論文の構成は，以下のとおりである．まず，2 章において，モバイルエージェントとセキュリティについて説明する．3 章において，耐タンパハードウェアを用いたモバイルエージェント保護手法と耐タンパハードウェアを用いる場合に生じる問題点をあげ，その解決策を提案する．4 章において，3 章で述べた手法を実現する手続きを述べる．5 章において，本手法を Java で実装する際に生じる問題点をあげ，その解決策を提案するとともに，本手法のプロトタイプの実装について報告する．6 章において，本手法の有効性について考察する．7 章において，関連研究との比較について述べる．

表 1 モバイルエージェントのセキュリティ
Table 1 Security of mobile agents.

攻撃名	詳細
M-H 攻撃	モバイルエージェント（M）からホスト（H）への攻撃
R-M 攻撃	中継ホスト（R）からモバイルエージェント（M）への攻撃
H-M 攻撃	移動先ホスト（H）からモバイルエージェント（M）への攻撃

2. モバイルエージェントとセキュリティ

モバイルエージェントは，コードと実行状態を持って自律的に移動するプログラムである．モバイルエージェントの実行には，モバイルエージェントシステムと呼ばれるランタイムシステムが各ホストに必要であるが，モバイルエージェントを用いることにより，プログラムの遠隔非同期実行やネットワーク負荷の軽減をはかることができる．また，プログラムの自動インストールや電子商取引等，モバイルエージェントを用いた数々の応用が考えられる．

モバイルエージェントを用いるうえで重要な問題の 1 つにセキュリティがあげられる．問題になる場面は大きく分けて，**表 1** に示すように，M-H 攻撃，R-M 攻撃，H-M 攻撃の 3 つがあげられる．ここで，中継ホストは，モバイルエージェントの送受信を行うが，実行しないホストであり，移動先ホストは，モバイルエージェントの送受信を行い，かつ，実行する可能性があるホストである．

M-H 攻撃については，署名技術を用いることによりモバイルエージェントの出所を調べ，出所によるアクセス制御を行うことにより，悪意のあるモバイルエージェントからの攻撃を防ぐことができる．R-M 攻撃に対しては，暗号化技術を用いることにより，中継ホストからの攻撃を防ぐことができる[5),6)]．H-M 攻撃に関しては，近年，さかんに研究が行われている[7)~10)]が，それらの手法は，信頼できるサーバを前提としていたり，モバイルエージェントが所持するデータの変更に対応していなかったりするため，所有者の意図に沿ったデータ管理を行うモバイルエージェントを保護するために必要となるセキュリティシステムとしては不十分である．この問題を解決するために，次章に耐タンパハードウェアを用いたモバイルエージェントシステムを提案し，それが持つべき機能について詳細を

検討する．

3. 耐タンパハードウェアを用いたモバイルエージェント保護手法

3.1 耐タンパハードウェアを用いたモバイルエージェントシステム

一般に情報セキュリティにおいては，攻撃に対処するためにかけられるコストは保護対象の重要度と比例する．本論文では，保護するデータとしてプライバシーに関するデータや音楽・映像等の電子データを対象としている．コストの問題を考慮し，ハードウェア，OS，ドライバやデバイス等の改竄による攻撃はないが，モバイルエージェントシステムの改竄による攻撃の可能性があると仮定する．

モバイルエージェントシステムが改竄されている場合，モバイルエージェントが保持するコードと実行状態は，移動先ホストによって盗聴，改竄される可能性がある．モバイルエージェントはコードと実行状態を持って移動し，モバイルエージェントシステムがモバイルエージェントのコードのロードとモバイルエージェントの登録を行う．そのため，移動先ホストは，モバイルエージェントの保持するコードと実行状態が獲得できる．そのとき，ホストは獲得したコードに実行状態の中にある秘密データを出力するコードを挿入することにより，そのデータを獲得できる．したがって，モバイルエージェントは悪意のある移動先ホストではデータを盗聴，改竄される可能性がある．

秘密データを持つモバイルエージェントを保護するためには，ホストがモバイルエージェントの保持するコードと実行状態を獲得し，コードを不正に書き換えたとしても，モバイルエージェントが持つ秘密データを盗聴・改竄されない方法が必要となる．そこで，耐タンパハードウェア（TRH：Tamper Resistant Hardware）を用いたモバイルエージェント保護手法を提案する．本論文では耐タンパハードウェアが持つべき機能やデータをできる限り小さくすることにより，実用的な手法を目指す．耐タンパハードウェアの一例として，メモリスティック☆やSDメモリカード☆☆があげられる．これらのハードウェアは，データの記憶領域だけでなく，暗号・復号機能，認証機能を持っている[11]．この認証機能により，耐タンパハードウェアが持つ秘匿データは特定のアプリケーションからしかアクセスできない．

☆ http://www.memorystick.org/
☆☆ http://www.panasonic.co.jp/avc/home/sd/

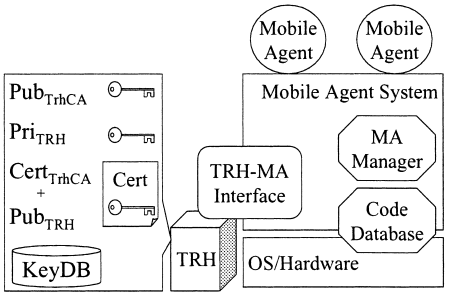

図 1 本論文で提案するモバイルエージェントシステム
Fig. 1 The structure of mobile agent system we propose in this paper.

図 1 に，本論文で提案するモバイルエージェントシステムの構成を示す．ここで，TRH-MA Interface（TMI）は耐タンパハードウェアとモバイルエージェントの仲介プログラムインタフェースであり，MA Managerはモバイルエージェントを管理するコンポーネントであり，Code Databaseはモバイルエージェントのコードを保管するコンポーネントである．

本システムでは，耐タンパハードウェアと公開鍵暗号方式を用いたデータの配布・交換を行う．例として，コード $Code1$ と秘密データ $Data1$ を含む実行状態 $State1$ を保持するモバイルエージェント MA がホスト A からホスト B に移動する場合を想定する．MA がホスト B に移動する前に，ホスト A の TRH を用いて $Data1$ の暗号化を行う．MA は，$Code1$ と暗号化された $Data1$ を含む実行状態 $State2$ を持って移動する．移動した MA は，ホスト B の TRH を用いて暗号化された $Data1$ の復号化を行う．ただし，TRH が保持する固有の公開鍵，秘密鍵は，製造メーカによって製造時に ROM に書き込まれるため，ユーザによって書き換えられることはない．

この方式で，

(1) 耐タンパハードウェア専用の CA（Certified Authority）の仮定による耐タンパハードウェア公開鍵偽装の防止，
(2) 復号要求元のモバイルエージェントの認証による耐タンパハードウェアへの不正アクセスの防止，
(3) データごとのシーケンスナンバ保持によるリプレイ攻撃の防止，

をすることによって，耐タンパハードウェアを用いたデータの暗号化・復号化による秘密のデータの漏洩防止を実現できる．以下に，それぞれについて詳述する．

3.2 耐タンパハードウェアの公開鍵の信頼性

本手法では，公開鍵暗号方式を用いるため，あらか

じめ互いの耐タンパハードウェアの公開鍵を交換しておく必要がある．しかし，悪意のあるホスト B が自分の公開鍵 Pub_B を耐タンパハードウェアの公開鍵と偽って，攻撃対象のホスト A と公開鍵を交換する可能性が考えられる．このとき，ホスト A が騙されて，ホスト B の公開鍵 Pub_B で秘密データを暗号化すれば，悪意のあるホスト B は秘密データを獲得できる．

ここでの問題は耐タンパハードウェアの公開鍵の信頼性にある．他ホストから獲得した公開鍵が『本当に』耐タンパハードウェアの公開鍵であるかを知る手段が必要になる．そこで，耐タンパハードウェアの公開鍵に対してのみ証明書を与える CA（以下，TrhCA）の存在を仮定する．TrhCA の運用ポリシとして，契約を結んだ TRH 製造メーカからの証明書発行要求に対してのみ応じることとする．また，耐タンパハードウェアは，端末固有の ID（NodeID）と TRH 固有の公開鍵付き証明書 $Cert_{TrhCA}(NodeID, Pub_{TRH})$ を持つ．公開鍵付き証明書，ノード ID は，公開鍵，秘密鍵と同様に，製造時に ROM に書き込まれるため，ユーザによって書き換えられることはない．また，TRH は，受け取った TRH 公開鍵，公開鍵証明書，および，TrhCA の公開鍵により，受け取った公開鍵が耐タンパハードウェアの公開鍵であることを確認する．これにより，悪意のあるホストは自分の公開鍵に対する証明書を獲得できないため，自分の公開鍵を耐タンパハードウェアの公開鍵として偽装できなくなる．

3.3 復号要求元の認証

悪意のあるホストは，モバイルエージェント $MA1$ が保持するコード $Code1$ と暗号化データ $ER(Pub_{TRH}, Data1)$ を含む実行状態 $State1$ を獲得できる．ここで，$ER(Key, Data)$ は公開鍵 Key で $Data$ を暗号化したデータを示す．このとき，悪意のあるホストは TRH への不正な復号化要求により，復号化されたデータを獲得できる．たとえば，**図 2** のように，悪意のあるホストは獲得したコードに TRH で復号化したデータを出力するコードを挿入したコード $Code2$ を作成する．そして，$Code2$ と $State1$ を持つ $MA2$ を作成し，$MA2$ から TRH へ暗号化データの復号化要求を行う．このとき，TRH が，復号化した秘密データを $MA2$ に返すことにより，結果として悪意のあるホストにデータを漏らすことになる．

ここでの問題は，TRH が秘密データに対してデータの持ち主以外のモバイルエージェントにも復号化したデータを渡してしまうことにある．そこで，復号要求元の認証が必要になる．

ここで，モバイルエージェントのコードは移動しても変わらない．また，悪意のあるホストは，モバイルエージェントのコードを書き換えてモバイルエージェントが保持する秘密データを獲得しようとする．したがって，復号要求元の認証方法として，コードを用いた共通鍵暗号方式を実施する．具体的には，モバイルエージェントが移動する前にコードから，たとえばハッシュ関数等を利用して，共通鍵 $sKey1$ を作成し，この $sKey1$ で秘密データを暗号化する．モバイルエージェントが移動した後，復号要求元のコードから共通鍵 $sKey2$ を作成し，この $sKey2$ で暗号化データを復号化する．ここで，復号要求元のモバイルエージェントが正当な場合のみ，$sKey1 = sKey2$ となるため，不正なモバイルエージェントからの復号化要求は受け入れられない．つまり，コードの完全性の検証により，データの持ち主であるモバイルエージェントに対してのみ復号化データを渡すことになる．

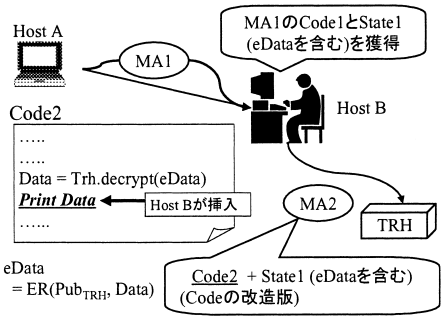

図 2 TRH に対する不正な復号化要求
Fig. 2 Wrong decryption request for TRH.

3.4 データの新しさの検証

悪意のあるホストは，モバイルエージェントのコードと実行状態を獲得できるため，リプレイ攻撃を防ぐことができない．たとえば，実行回数をカウントするモバイルエージェント $MA1$ を考える．そのとき，$MA1$ を受け取ったホスト A は，$MA1$ のコード $Code1$ と実行状態 $State1$ を獲得する．次に，A が $MA1$ を実行すると，$MA1$ は実行回数のカウントを 1 つ増やし，実行状態は $State2$ になる．しかし，ホスト A は，$State1$ を保持しているため，再度 $Code1$ と $State1$ を保持するモバイルエージェントを実行する．つまり，実行回数をごまかすことができる．

ここでの問題は，モバイルエージェントの実行状態に対する新しさに関する情報の欠如にある．先ほどの例では，$MA1$ を実行した時点で，$State1$ は古い実行状態で $State2$ が新しい実行状態である．この後に，$MA1$ を実行するには $State2$ を用いるべきであるにもかかわらず $State1$ を用いている．

表 2 本手法の TRH が保持するデータ
Table 2 TRH data.

変数名	内容
$Cert_{TrhCA}(NodeID, Pub_{TRH})$	NodeID と TRH 公開鍵を含む TrhCA による証明書
Pub_{TrhCA}	TrhCA の公開鍵
Pri_{TRH}	TRH 秘密鍵
$KeyDB_{TRH}$	他ホストの TRH 公開鍵付き証明書保管用データベース
$List(DataID, SeqNum)$	データ ID ごとのシーケンスナンバのリスト

表 3 本手法のモバイルエージェントが保持するデータ
Table 3 Mobile agent data.

変数名	内容
$DataID$	データごとのユニークな ID
$SecretData$	秘密データ
$List(NodeID, SeqNum)$	ノード ID ごとのシーケンスナンバのリスト

そこで，データに対して ID を付加し，その ID ごとのシーケンスナンバ（SeqNum：Sequence Number）のリスト $(DataID, SeqNum)$ を TRH が保持する．また，各モバイルエージェントはデータごとに $(NodeID, SeqNum)$ のリストを保持する．SeqNum はモバイルエージェント作成時に初期化され，データが変更されたとき，モバイルエージェントが SeqNum の加算を行い，耐タンパハードウェアに対して SeqNum の登録を行う．**表 2** に，本手法で想定する耐タンパハードウェアが保持すべきデータとその内容を示す．耐タンパハードウェアに必要なメモリ容量については 6.2 節で述べる．**表 3** に，モバイルエージェントが保持すべきデータとその内容を示す．これらのデータは，耐タンパハードウェアで暗号化の対象になるデータである．

各モバイルエージェントは，まず，暗号化されたデータを復号化する．次に，復号化されたデータに含まれるデータ ID とシーケンスナンバを用いて，そのデータが最新のデータであるかどうかを TRH に問い合わせることによりデータの新しさを確かめる．耐タンパハードウェアは入力されたシーケンスナンバが自分が保持するシーケンスナンバ以上であれば，最新のデータであることをモバイルエージェントに知らせ，リストに最新のシーケンスナンバを登録する．モバイルエージェントは，最新のデータであれば，復号化されたデータを受け入れ，最新のデータでない場合は，復号化されたデータを破棄する．

3.5 セッション ID の導入

3.2 節〜3.4 節で述べたことに加え，さらに配慮すべき事項がある．3.5 節〜3.7 節に，それらについて述べる．

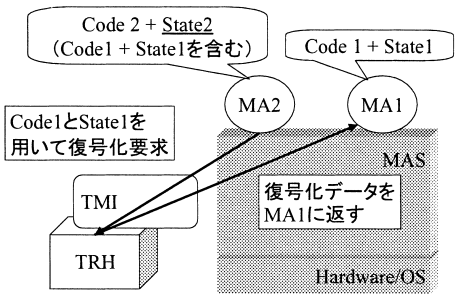

図 3 TRH に対する復号化要求の不正実行
Fig. 3 Illegal execution of decryption request for TRH.

ホストは，モバイルエージェント $MA1$ のコード $Code1$ と実行状態 $State1$ を獲得できる．本システムでは，入力されたモバイルエージェントのインスタンスに対する関数呼び出しによって暗号化データ，あるいは，復号化データを渡すため，**図 3** のように，復号化要求の不正実行が行われる可能性がある．たとえば，悪意のあるホストが，実行状態の中に $Code1$ と $State1$ を持つモバイルエージェント $MA2$ を作成する．$MA2$ が $Code1$ と $State1$ を用いて TRH へ復号化要求を行うと，$MA1$ に復号化データが与えられる．このとき，$MA1$ が暗号化データの復号化回数を数えている場合，$MA2$ により $MA1$ の復号化要求を超える復号化回数がカウントされる可能性がある．

この問題の解決策として，モバイルエージェントが復号化要求をするとき，復号化要求用の ID として，セッション ID を作成し，自分で覚えておく．セッション ID には，SeqNum を利用してもよい．また，その ID を TRH に復号化要求するときに入力する．復号化されたデータは，ID とともに返される．モバイルエージェントは復号化されたデータに含まれる ID と自分が持つ復号化要求 ID が一致することを確認できれば，そのデータを受理する．これにより，不正な復号化要求による結果を受け入れることはない．

3.6 インタフェースの認証

耐タンパハードウェアとモバイルエージェントを仲介するインタフェース TMI は，暗号化データ，復号化データを扱う．そのため，このインタフェースの正当性を調べる必要がある．

正当性を調べるための手法として，乱数を用いたチャレンジレスポンス認証がある．TMI と TRH は同じハッシュ関数，TRH は乱数発生器を保持し，以下の手順で認証を行う．

(1) TRH は，TMI からアクセスされると乱数発生

器を用いて乱数を生成し，生成した乱数を TMI に渡す．

(2) TMI は，ハッシュ関数と受け取った乱数を用いてハッシュ値を生成し，TRH に渡す．

(3) TRH は，ハッシュ関数と生成した乱数からハッシュ値を生成し，受け取ったハッシュ値と比較する．同じであれば認証成功とし，そうでなければ認証失敗とする．

この手法は，SD メモリカードやメモリスティックにおける認証でも用いられている．

3.7 安全なモバイルエージェント コード

モバイルエージェントは，復号化データを扱うため，復号化データをともなった移動，永続化，他からの不正アクセスに注意しなければならない．したがって，モバイルエージェントの作成者が注意すべき問題として，復号化データの移動，永続化，他からのアクセスが行えないコードを作成することが求められる．

4. 本手法の手続き

本手法では，(1) 耐タンパハードウェアの公開鍵交換，(2) 移動元でのデータの暗号化，(3) 暗号化モバイルエージェントの移動，(4) 移動先でのデータの復号化，(5) 復号化データのデータの新しさの確認の 5 つの手続きからなる．ただし，$ED(Key, Data)$ は，共通鍵 Key で $Data$ を暗号化したデータを示し，$ER(Key, Data)$ は，公開鍵 Key で $Data$ を暗号化したデータを示す．また，**表 4**，**表 5** は，TMI，および，TRH が保持している機能を示している．TMI の機能は MA が利用し，TRH の機能は TMI が利用する．ここで，NodeID は移動先ホストの ID を，sID はセッション ID を示す．さらに，本手法を利用するモバイルエージェントは正当なモバイルエージェントに結果を返すため，かつ，暗号化/復号化データの対応関係を調べるために，setEncData(sID, eData, hash(data), NodeID) と setDecData(sID, Data, hash(eData)) というインタフェースを持つ．モバイルエージェントを実行するために必要となるコードは，モバイルエージェント初期実行時にコード配送エージェント，あるいは，モバイルエージェント自身があらかじめ Code Database に登録しておく必要がある．Code Database は，モバイルエージェントシステムが実行前，あるいは，実行中のモバイルエージェントのコードを獲得するためのデータベースである．認証可能であれば，ファイルシステムによる実現でもよい．この構成方法や認証方法は言語やシステム依存になるため，ここでは記述せず，Java 言語を用いたシステムにおける Code Database の構成方法，ならびに，認証方法について 5.1 節で述べる．**図 4** は，4.2 節〜4.4 節の手続きを，**図 5** は，4.1 と 4.5 の手続きを UML シーケンス図を用いて表現した図である．

表 4 本手法の TMI が保持する機能
Table 4 TMI function.

関数名	内容
enc(sID, MA, data, NodeID)	MA に対応するコードを獲得し，TRH の enc 関数を実行
dec(sID, MA, eData)	MA に対応するコードを獲得し，TRH の dec 関数を実行
setCert(Certificate)	TRH の setCert 関数を実行
getCert()	TRH の getCert 関数を実行
regData(DataID, SeqNum)	TRH の regData 関数を実行

表 5 本手法の TRH が保持する機能
Table 5 TRH function.

関数名	内容
enc(sID, code, data, NodeID)	NodeID の TRH 公開鍵と code から生成した鍵で data を暗号化
dec(sID, code, eData)	TRH 自身の秘密鍵と code から生成した鍵で data を復号化
setCert(Certificate)	耐タンパハードウェアの公開鍵付き証明書を登録
getCert()	TRH 自身の証明書を渡す
regData(DataID, SeqNum)	入力 SeqNum が最新であるかをチェック

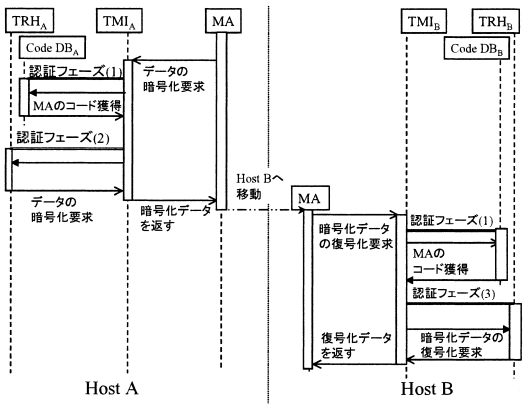

図 4 本手法の流れの UML シーケンス図 (1)
Fig. 4 UML sequence diagram No.1 for this technique.

4.1 耐タンパハードウェアの公開鍵交換

データの暗号化を行うために，事前にホスト間で TRH 公開鍵を交換する必要がある．そこで，TRH の証明書を獲得し，他ホストへ移動し，獲得した証明書を登録するモバイルエージェント（TRHCertTransAgent）を用いて TRH 公開鍵を交換する．TRHCertTransAgent は自ホストの通信相手のホストの TRH 公開鍵を保持していない場合に，システムによって相手ホストに移動される．相手ノード

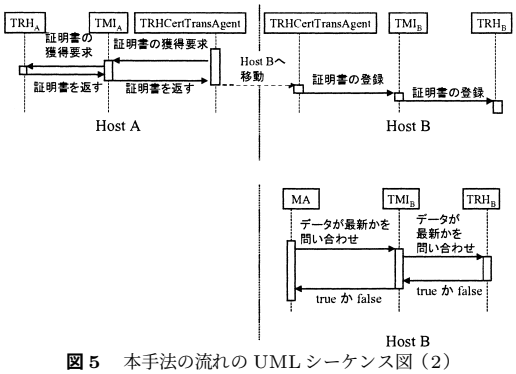

図 5 本手法の流れの UML シーケンス図（2）
Fig. 5 UML sequence diagram No.2 for this technique.

では，逆に自ノードに対して TRHCertTransAgent を送り直す．これにより，公開鍵の交換を終了する．以下は，ホスト A からホスト B へ TRH 公開鍵を移動させる場合について示す．

(1) TRHCertTransAgent は，TMI_A の getCert() を実行し，TRH_A から $Cert_{TrhCA}(NodeID, Pub_{TRH_A})$ を獲得．(3.2 節)

(2) TRHCertTransAgent は，ホスト A からホスト B へ証明書 $Cert_{TrhCA}(NodeID, Pub_{TRH_A})$ を持って移動．

(3) TRHCertTransAgent は，TMI_B の setCert ($Cert_{TrhCA}(NodeID, Pub_{TRH_A})$) を実行して，TRH の鍵データベースに TRH の公開鍵 Pub_{TRH_A} を登録．

(a) TRH_B は，入力された公開鍵と公開鍵証明書を用いて，公開鍵が TrhCA によって信頼されているかを Pub_{CA} を用いて確認することにより入力された公開鍵が TRH 公開鍵であるかどうかを確認．

(b) TRH_B は，認証成功時に，$KeyDB$ (TRH_B) に同一の NodeID が含まれていないか，あるいは，同一の Pub (TRH_A) が含まれていなければ，$KeyDB_{TRH_B}$ に公開鍵 Pub_{TRH_A} を登録．

4.2 移動元でのデータの暗号化

モバイルエージェントの機密性を満たすために，移動前に TMI を利用して秘密にしたいデータの暗号化を行う．

(1) モバイルエージェント $MA1$ は，暗号化前に $id1$（セッション ID）を作成．

(2) $MA1$ は，$id1$ を記憶しておき，TMI_A の enc($id1$, $MA1$, $Data$, Host B) を実行し，データを暗号化する．

(a) TMI_A は，入力されたデータから MA のコード $code1$ を獲得．

(i) TMI_A による Code Database の認証．(認証フェーズ (1))

(ii) TMI_A は Code Database から MA のコード $code1$ を獲得．

(b) TMI_A は，TRH_A と相互認証．(3.6 節，認証フェーズ (2))

(c) TMI_A は，TRH_A の enc($id1$, $code1$, $Data$, Host B) を実行し，データを暗号化．

(i) TRH_A は，入力された $code1$ から共通鍵 $sKey$ を作成．

(ii) TRH_A は，共通鍵 $sKey$ で ED ($sKey, Data$) を作成．

(iii) TRH_A は，移動先の TRH 公開鍵 Pub_{TRH_B} で $ER(Pub_{TRH_B}, ED$ ($sKey, Data$))(= $eData$) を作成．

(iv) TRH_A は，TMI_A に暗号化データ $eData$ を返す．

(d) TMI_A は，data ハッシュ値 hash(data) を計算し，$MA1$ の setEncData($id1$, $eData$, hash(data), Host B) を実行．

(3) $MA1$ は，setEncData が実行されると，入力された $id1$ が保持するセッション ID と同じであるか，自身が保持する data のハッシュ値と返されたハッシュ値が等しいかどうかを確認する．同じであれば，入力された暗号化データ $eData$ を受け取る．

4.3 暗号化モバイルエージェントの移動

中継ホストによる攻撃を防ぐため，モバイルエージェントのコードと実行状態を暗号化する配送エージェント（CryptAgent）を用いてモバイルエージェントを移動させる．ただし，セキュアな通信路上では，CryptAgent を用いず，MA1 自身の移動能力を用いて移動する．

(1) ホスト A は，$MA1$ が保持する $code1$ と $eData$ を含む $state1$ を移動先ホストの公開鍵 Pub_B で暗号化し，暗号化 Code と暗号化 State を作成．

(2) ホスト A は，暗号化 State に対して，ホスト A の秘密鍵を用いて署名を作成．

(3) 配送エージェントは，暗号化 Code と暗号化 State，署名を持って移動．

(4) ホスト B は，ホスト A の公開鍵を用いて署名を確認し，確認できれば暗号化 Code と暗号化

State を自ホストの秘密鍵 Pri_B で復号化し，$code1$ と $state1$ を作成.

(5) ホスト B は，$code1$ と $state1$ を用いてモバイルエージェント $MA1$ の作成，実行を行う.

4.4 移動先でのデータの復号化

モバイルエージェントが移動した後に，あらかじめ暗号化したデータを利用したいときに，TMI を利用してデータの復号化を行う.

(1) モバイルエージェント $MA1$ は，$Data$ を獲得するために，復号化前に $id2$（セッション ID）を作成.

(2) $MA1$ は，$id2$ を記憶しておき，TMI_B の dec($id2$, $MA1$, $eData$) を実行.

(a) TMI_B は，MA のコード $code1$ を獲得.（4.2 節-(2)(a) と同じ）

(b) TMI_B は，TRH_B と相互認証.（3.6 節，認証フェーズ (3)）

(c) TMI_B は，TRH_B の dec($id2$, $code1$, $eData$) を実行し，データを復号化.

(i) TRH_B は，自分の TRH 秘密鍵 Pri_{TRH_B} で $ED(sKey, Data)$) を復元.

(ii) TRH_B は，$code1$ から共通鍵 $sKey$ を作成.

(iii) TRH_B は，共通鍵 $sKey$ でデータ $Data$ を復元.

(iv) TRH_B は，TMI_B に復号化データ $Data$ を返す.

(d) TMI_B は，eData のハッシュ値 hash(eData) を計算し，$MA1$ の setDecData($id2$, $Data$, hash(eData)) を実行.

(3) $MA1$ は，setDecData が実行されると，入力された $id2$ が保持するセッション ID と同じであるか，自身が保持する eData のハッシュ値と返されたハッシュ値が等しいかどうかを確認する．同じであれば，入力された復号化データ $Data$ を受け取る.

4.5 データの新しさの確認

モバイルエージェントは，リプレイ攻撃を防ぐために，復号化されたデータが最新であるかどうかを確かめる必要がある．そこで，TMI を利用して現在のデータが最新であるかどうかの確認を行う.

(1) モバイルエージェント $MA1$ は，獲得したデータが最新であるかどうかをチェックするため，TMI の regData($dataId1$, $seqNum1$) を実行.

(a) TMI は，TRH の regData($dataId1$, $seqNum1$) を実行.

(i) TRH は，$dataId1$ に対応するシーケンスナンバ $seqNum$($dataId1$) をリストから獲得.

(ii) TRH は，$seqNum1$ が $seqNum$($dataId1$) 以上であれば，$seqNum1$ を $dataId1$ のシーケンスナンバとしてリストに登録し，true を返す．そうでなければ，登録せずに false を返す.

(b) TMI は，TRH の実行結果を $MA1$ に伝える.

(2) $MA1$ は，実行結果が ture であれば最新版のデータであることを知り，false であれば古いデータであることを知ることができる.

5. 実　　装

今回，システムの実装において，Java 言語（JDK1.3, JCSI☆）を用いた．ここで，本システムでは，ハードウェア，OS，JavaVM の改竄による攻撃はないが，モバイルエージェントシステムの改竄によって攻撃される可能性があると仮定した．また，耐タンパハードウェア専用の CA の存在を仮定した.

5.1 Java バイトコード獲得問題

本手法では，復号要求元の認証に，要求元のモバイルエージェントのコードから生成された鍵を用いた共通鍵暗号方式を利用している．しかし，Java で実装する場合，要求元のモバイルエージェントのバイトコードをどのように獲得するかという問題がある．Java には，実行中のオブジェクトからバイトコードを獲得するためのメソッドが存在しない．そこで，オブジェクト名に基づいて Code Database からバイトコードを獲得する必要がある.

しかし，**図 6** のように，Code Database にあるバイトコードは実行中のモバイルエージェントのバイトコードに等しいとは限らない．以下に例を示す．まず，ホストは，モバイルエージェント $MA1$ からコード $Code1$ と暗号化データを含む実行状態 $State1$ を獲得できる．次に，ホストは $Code1$ に TRH で復号化したデータを出力するコードを挿入した $Code2$ を作成する．そして，$Code2$ と $State1$ を持つモバイルエージェントを登録，実行する．この後，悪意のあるホストは，Code Database にある $(MA2, Code2)$ というデータの対を $(MA2, Code1)$ に書き換える．このと

☆ http://www.wedgetail.com/jcsi/index.html

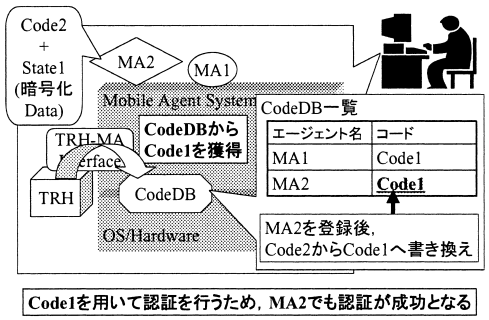

図 6 Java バイトコード獲得問題
Fig. 6 Java byte code acquisition problem.

き，耐タンパハードウェアは Code Database にあるバイトコードに基づいて認証を行ってしまうと MA2 でも認証が成功してしまう．

この問題の解決策として，ClassLoader が（$Class$, $Code$ の MD 値）をデータの対とするテーブル（ClTable）を，MA Manager が（$Object$, $Code$ の MD 値）をデータの対とするテーブル（MmTable）を持つ．ここで，Code の MD 値とは，Code から作られる Message Digest である．また，ClTable と MmTable は，モバイルエージェントシステムが終了するとともに消滅してしまうものとする．以下にテーブルの作成手順を示す．

(1) ClassLoader がクラス名 $Name1$ とバイトコード $Code1$ からクラス $Class1$ を作成．
- ClTable に $Class1$ と $Code1$ の MD 値 $MD(Code1)$ を登録．

(2) MA Manager がオブジェクト $Object1$ をモバイルエージェントとして登録．
- $Object1$ からクラス $Class1$ を求め，クラスに対応するコード $Code1$ を獲得．
  - (a) ClassLoader は，入力されたクラス $Class1$ からクラス名 $Name1$ を獲得．
  - (b) Code Database からクラス名に対応するコード $Code1$ を獲得．
  - (c) 獲得した $Code1$ の MD 値を求め，ClTable にある MD 値と一致すれば，MA Manager に $Code1$ を渡す．
- MmTable に $Object1$ と $Code1$ の MD 値 $MD(Code1)$ を登録．

(3) TMI は，復号要求元のモバイルエージェントのコードを MA Manager から獲得．
- MA Manager は，オブジェクトからクラス名 $Name1$ を獲得．
- Code Database からクラス名に対応するコード $Code1$ を獲得．
- 獲得した $Code1$ の MD 値を求め，MmTable の MD 値と一致すれば，TMI に $Code1$ を渡す．

最後に，テーブルの正当性を保つために，TMI が ClassLoader と MA Manager を認証する必要がある．TMI は MA Manager に対してコードを要求するときに，MA Manager，ClassLoader の認証を開始する．TMI は，MA Manager をロードした ClassLoader のインスタンスを獲得☆1する．この ClassLoader は TRH が信頼できる団体☆2,☆3による署名付き JAR ファイルからロードされていると仮定する．そして，オブジェクトに対する署名を偽装されないために，その JAR ファイルを Java Native Loader 上で実行する☆4．TMI は ClassLoader が Java Native Loader で実行されており☆5，かつ，信頼できる団体による署名☆6がなされているかを確認することにより，ClassLoader を認証する．次に，MA Manager も同様に，TMI は，MyClassLoader から獲得した MA Manager から獲得できる署名者が信頼できる団体であるかどうかを確認することによって MA Manager の認証を行う．

5.2 プロトタイプの作成

本手法のプロトタイプをモバイルエージェントシステム MAGNET（Mobile AGent NETwork）[12] に実装した．**図 7** に，プロトタイプの概略図を示す．

プロトタイプでは，仮想の耐タンパハードウェアを想定して Java で実現している．また，下線は認証を必要とするコンポーネントを示し，図にかかれた矢印では，コード要求と要求元の認証が行われている．

6. 考　　察

6.1 本手法における防御可能範囲

表 6 に，様々な攻撃に対する本手法を利用した場合の防御の可否を示す．ただし，表中の○は防御できる

☆1 MAManager.getClass().getClassLoader()
☆2 信頼できる団体は，ClassLoader，MA Manager の動作検証を行い安全性が確認されたときに，署名を発行する．
☆3 TrhCA が行う場合は，TRH の公開鍵証明書を発行するだけでなく，信頼できる ClassLoader，MA Manager に対する署名を与えると仮定．
☆4 Java Native Loader では署名付き Jar ファイルをロードすると，Jar ファイル内にあったクラスに対して，署名の対応付けを行う機構がある．
☆5 MyClassLoader.getClass().getClassLoader() == Java Native Loader
☆6 MyClassLoader.getClass().getProtectionDomain().getCertificates() から獲得．

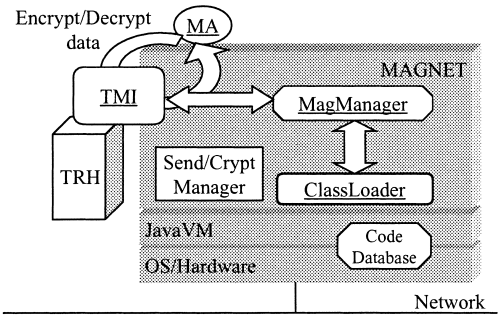

図 7 プロトタイプシステムの概略図
Fig. 7 Prototype system.

表 6 本手法における防御可能範囲
Table 6 Protection with this protection system for some attacks.

攻撃者	攻撃内容	用いる手法	安全性
モバイル	システムの盗聴	署名技術による	○
エージェント	システムの改竄	アクセス制御	○
	DoS 攻撃		△
中継ホスト	コードの盗聴，改竄	暗号化技術	○
	データの盗聴，改竄		○
移動先	コードの盗聴		×
ホスト	コードの改竄	TRH を用いた	△
	データの盗聴	モバイル	○
	データの改竄	エージェント	○
	TRH のなりすまし	保護手法	○
	リプレイ攻撃		○

ことを示し，△は一部防御できることを，×は防御できないことを示す．

本手法では，コードの盗聴を防ぐことはできない．コードの盗聴に関しては，プログラムの難読化技術[13]によってある程度防御できると考えられる．DoS 攻撃については，攻撃を防ぐことはできないが，署名による認証により，攻撃者を簡単に特定できるので，安易には攻撃はできないと考えられる．

6.2 耐タンパハードウェアに必要なメモリ量

本手法では，耐タンパハードウェアを用いてモバイルエージェントを保護し，保護されたモバイルエージェントを用いてコンテンツ管理を行う手法を提案した．本システムでは，データ管理上，多くのリストをかかえるため，耐タンパハードウェアに必要なメモリ量についての考察を行った．

耐タンパハードウェアの構成は表 2 である．これらそれぞれについて，必要なメモリ量を**表 7** に載せた．公開鍵証明書はファイル格納分として 4 KByte，TrhCA 公開鍵，ならびに，TRH 専用の秘密鍵を 256 Byte，KeyTable については，他の 1024 ホスト分の公開鍵を格納するため，256 KByte とした．DataList については，DataID，SeqNum のビット数，および，人間が扱うコンテンツ量を仮定する必要がある．DataID は，TRH 製造メーカの IP Address（128 bit）にコンテンツ ID（128 bit）を付加して 256 bit とする．SeqNum については，50 年の間に 1s に 1 度書き換える（16 億回）とすると 64 bit あれば十分である．人間は 50 年の間に 1 時間に 1 つの割合で，新規コンテンツを扱うと仮定すると，50 万コンテンツとなる．つまり，DataList に必要な容量は，20 MByte となり，耐タンパハードウェアに必要なメモリ量は，21 MByte のメモリがあれば十分である．この数字は，現在のフラッシュメモリで安価に実現が可能である．

表 7 耐タンパハードウェアに必要なメモリ量
Table 7 Memory capacity required for tamper resistant hardware.

TRH データ	メモリ内訳	メモリ量
公開鍵付き証明書	ファイル格納分	4 KByte
TrhCA 公開鍵	RSA2,048 bit	256 Byte
TRH 秘密鍵	RSA2,048 bit	256 Byte
KeyDB	1,024 ホスト分	256 KByte
DataList	DataID: 256 bit,	
	SeqNum: 64 bit,	
	コンテンツ量：50 万	20 MByte

6.3 本手法を用いた安全で柔軟なデータ管理手法

1 章で述べたように，柔軟なデータ管理を行う手法としてモバイルエージェントを用いた柔軟なデータ管理手法を提案した．モバイルエージェントを用いることにより，事前にアプリケーションをインストールしておく必要がなく，また，データ所有者は自由にデータの管理法を作成できる．

本手法により，これまで問題となっていたホストによるモバイルエージェントへの攻撃を防ぐことができる．したがって，モバイルエージェントを用いた安全で柔軟なデータ管理が可能になる．柔軟なデータ管理の例として，ユーザ ID ごとの登録，削除，複製の制御や実行回数，永続化回数，閲覧時間設定があげられる．

また，エージェント単体ではなく，複数のエージェントの組合せにより，コンテンツによるビンゴゲームのような新規サービスが実現できる．

6.4 グループ間のモバイルエージェント保護

本手法では，1 対 1 通信におけるモバイルエージェント保護手法を提案した．しかし，グループ通信と呼ばれる 1 対多通信では本手法をそのまま用いることはできない．我々は，グループ ID を保持するモバイルエージェント（グループエージェント）を分散管理し，同じグループ ID を持つグループエージェントの有無によるグループ管理手法を提案した[14]．

そこで，グループエージェントがグループ間の共通

鍵（グループ鍵）を持ち，グループ間では TRH 公開鍵ではなく，グループ鍵を用いた暗号化・復号化を行うことにより，グループ間のモバイルエージェント保護手法を行う．グループ間で利用されるモバイルエージェントのデータはグループ鍵によって保護され，グループ鍵を持つグループエージェントは本手法によって保護されているため，データの保護が可能になる．

7. 関連研究

セキュリティを考慮したモバイルエージェントシステムとして，Aglets[7]，D'Agent[8]がある．

Aglets は，IBM により開発された Java モバイルエージェントシステムである．Aglets ではセキュリティドメインと呼ばれる同質のセキュリティが期待される領域内でのみ安全な通信を認めている．各セキュリティドメインには 1 つのマネージャが存在し，ドメインマネージャがセキュリティドメインに属するサーバに対して，ドメインの証である共通鍵を配布する．しかし，この手法では，安全な通信がドメイン外では保証されないことや，ドメインを形成する際にはマネージャが存在する必要があるため，アドホックネットワーク等のように出会ったその場で通信を行う場合には適さない．

D'Agent は，Tcl，Java，Scheme といった複数言語で書くことができるモバイルエージェントシステムである．D'Agent では，モバイルエージェントをコンポーネント化し，秘密データを持つコンポーネントは通常移動せず，秘密データが必要なときに移動先から他のコンポーネントが秘密データを持つコンポーネントを連れていく方式をとっている．しかし，この手法では，信頼できるサーバ，あるいは，作成者自身のホストが実行時に通信可能なネットワーク上に必要になる．

耐タンパハードウェアを用いたモバイルエージェント保護手法として，Wilhelm の手法[9]，山崎らの手法[10]がある．

Wilhelm は，耐タンパハードウェア内にモバイルエージェントシステム全体が保存され，モバイルエージェントを耐タンパハードウェア内で実行する手法を提案した．しかし，システム全体の挿入は，高性能で大容量のメモリを持つ耐タンパハードウェアが必要になるため，実用的とはいえない．

山崎らは，コードと秘密データのそれぞれに対し，また，全体に対して証明書をつけ，その証明書を確認するモジュールを耐タンパハードウェアが保持する手法を提案した．しかし，この手法では，秘密データを変更することができない．たとえば，1 章で述べたデータを管理するモバイルエージェントの場合，残り時間や残り実行回数等のデータがときどき刻々と変わるため，この手法を利用できない．

本論文で提案した手法は，アドホックネットワーク上でも扱えるモバイルエージェント保護手法であり，かつ，実用的な規模の耐タンパハードウェアで扱うことができる．さらに，モバイルエージェント上でのデータの変化に対しても対応できる．

8. おわりに

本論文では，所有者が電子情報の配布後にもその利用や再配布を管理でき，かつ，柔軟なデータ管理を実現する手法として，モバイルエージェントを用いたデータ管理手法を提案した．そして，そこで生じるセキュリティ問題を解決する手法として，耐タンパハードウェアを用いたモバイルエージェント保護手法を提案した．本手法では，各ホストに耐タンパハードウェアの存在を仮定し，耐タンパハードウェアを用いた公開鍵暗号方式を採用することにより，安全な暗号化・復号化を実現した．そして，耐タンパハードウェアを用いて復号化できるモバイルエージェントをデータごとに制限した．また，データごとに ID を設定し，TRH が ID とシーケンスナンバを保持することにより，リプレイ攻撃にも対応した．これによって，モバイルエージェントが保持するコードと実行状態を獲得されても実行状態の中にある秘密データの盗聴，改竄を防ぐことができた．これにより，モバイルエージェントを用いた安全で柔軟なデータ管理が実現できた．

今後の課題として，まず，モバイルエージェントが保持する秘密データをエージェント間で共有することがあげられる．次に，モバイルエージェントが保持するコードのバージョンアップ時に，モバイルエージェントのデータを安全にバージョンアップする方法が必要となる．本手法では，コードを用いた暗号化を行っているため，他のエージェント，あるいは，バージョンアップしたエージェントにデータを渡す際に，データの再暗号化が必要となる．また，データを渡すモバイルエージェントが信頼できるモバイルエージェントであるかどうかの認証も必要となる．

参考文献

1) Adobe Systems Incorporated: *Portable Document Format Reference Manual Version 1.3* (1999). http://partners.adobe.com/asn/developer/acrosdk/docs/pdfspec.pdf
2) Chess, D.M.: Security Issues in Mobile Code

Systems, Lecture Notes in Computer Science, Vol.1419, pp.1–14 (1998).
3) 片桐秀樹，河口信夫，外山勝彦，稲垣康善：赤外線通信を用いた頑健なモバイルアドホックネットワーク構築手法，情報処理学会研究報告（MBL），Vol.98, No.110, pp.63–70 (1998).
4) 戸崎貴資，河口信夫，外山勝彦，稲垣康善：漸増的に端末を認識するアドホックネットワーク構築手法の評価，情報処理学会研究報告（MBL），Vol.2000, No.87, pp.47–54 (2000).
5) DiCelie, J. and Kino, S.: Concordia and Its Security Features, *Proc. Japan Society of Software Technology Workshop*, pp.133–140 (1998).
6) 田原康之，長谷川哲夫，大須賀昭彦，本井田真一：知的ネットワークエージェント plangent のセキュリティ・セーフティ，インターネットテクノロジーワークショップ（WIT98）論文集 (1998).
7) Ono, K.: A Security Model for Mobile Agent Framework Aglets, コンピュータソフトウェア, Vol.17, No.4, pp.2–14 (2000).
8) Gray, R.S., et al.: D'Agents: Security in a Multiple-Language, Mobile Agent System, Lecture Notes in Computer Science, Vol.1419, pp.154–187 (1998).
9) Wilhelm, U.G.: Increasing privacy in mobile communication systems using cryptographically protected objects, *Verlasliche IT-Systeme*, pp.319–334 (1997).
10) 山崎重一郎，岩尾忠重，塩内正利，和田裕二，岡田　誠，荒木啓二郎：P2P 型エージェントプラットフォームにおける信用ドメインの構成について，マルチメディア，分散，協調とモバイル（DICOMO 2001）シンポジウム論文集，pp.681–686 (2001).
11) 上野正巳，庵　祥子，三宅延久，武井英明：不正コピー防止を考慮したコンテンツ販売システム，情報処理学会研究報告（IM），Vol.2000, No.36, pp.17–24 (2000).
12) Kawaguchi, N., Toyama, K. and Inagaki, Y.: MAGNET: Ad Hoc Network System based on Mobile Agents, *Computer Communication*, Vol.23, No.8, pp.761–768 (2000).
13) 門田暁人，高田義広，鳥居宏次：プログラムの難読化法の提案，第 51 回情報処理学会全国大会，pp.263–264 (1995).
14) Miyagoshi, Y., Kawaguchi, N. and Inagaki, Y.: Flexible Group Management on Ad Hoc Network using Mobile Agents, *International symposium on Wireless Personal Multimedia Communications*, pp.835–840 (2001).

(平成 14 年 3 月 28 日受付)
(平成 15 年 4 月 3 日採録)

推　薦　文

本論文は，耐タンパハードウェアを用いることにより，移動先のホストからモバイルエージェントを保護する手法を提案している．提案手法は，信頼できるサーバを必要とせず，アドホックネットワーク環境でも動作することから応用範囲が広く，必要とされる耐タンパハードウェアの容量も少なく，実用性が高い．プロトタイプの作成も行い，実装上の問題点を解決している点が論文として評価できる．

（MBL 研究会主査　高橋　修）

春木　洋美（学生会員）

平成 12 年名古屋大学工学部電気電子情報工学科卒業．平成 14 年同大学院工学研究科計算理工学専攻修了．現在，（株）東芝研究開発センター所属．

河口　信夫（正会員）

平成 2 年名古屋大学工学部電気工学科卒業．平成 7 年同大学院情報工学専攻博士課程修了．同大学助手，講師，助教授を経て，平成 14 年より同大学情報連携基盤センター助教授．工学博士．モバイルコミュニケーション，マルチモーダルインタフェースの研究に従事．電子情報通信学会，日本音響学会，日本ソフトウェア科学会，人工知能学会，IEEE，ACM 各会員．

稲垣　康善（フェロー）

昭和 37 年名古屋大学工学部電子工学科卒業．昭和 42 年同大学院博士課程修了．同大学助教授，三重大学教授を経て，昭和 56 年名古屋大学工学部教授，平成 15 年より名古屋大学名誉教授，愛知県立大学情報科学部教授，工学博士．オートマトン言語理論，ソフトウェア基礎論，代数的仕様記述法，人工知能，自然言語処理に関する研究に従事．電子情報通信学会，人工知能学会，日本ソフトウェア科学会，言語処理学会，IEEE，ACM，EATCS 各会員．