Ja-Net on Gridにおけるホスト利用の信頼性を向上させるためのモバイルエージェントのマイグレーション手法

沼田哲史[†1] 板生知子[†2] 小川剛史[†3]
塚本昌彦[†4] 西尾章治郎[†1]

Ja-Net on Gridは，動的に変化するグリッド環境における並列計算を容易にするためのモバイルエージェントシステムである．本システムではこれまでGlobus Toolkitのセキュリティ基盤GSIを用いて，信頼できるエージェントのみをホストが受け入れられるようにセキュリティを構築してきた．しかし，エージェント側のセキュリティは考慮していなかったため，悪意のあるホストにエージェントの内部変数を不正に書き換えられるなどエージェントがホストに攻撃される可能性がある．信頼できないホストへのエージェントの移動を禁止することで，エージェントのセキュリティを保つことが可能であるが，利用できるホストが減少してしまうため，グリッド環境で提供されている膨大な計算リソースを最大限に活用することができない．そこで本研究では，たとえ信頼できないホスト上であっても，モバイルエージェントが一定のセキュリティを確保し，その結果グリッド上のすべてのホストで処理を継続できる手法を提案する．本論文では，提案手法の詳細とプロトタイプの実装およびそのパフォーマンス評価について述べる．

A Migration Method of Mobile Agents on Ja-Net on Grid for Increasing Reliability of Host Utilization

Satoshi Numata,[†1] Tomoko Itao,[†2] Takefumi Ogawa,[†3]
Masahiko Tsukamoto[†1] and Shojiro Nishio[†1]

Ja-Net on Grid is a mobile agent system, which simplifies parallel calculations on dynamic changing grid environments. We have established its security by using a security infrastructure of Globus Toolkit, with which mobile agent hosts can accept only reliable agents. However, malicious hosts can rewrite instance variables of agents or extract important information from agents, since the security for agents has not been considered. We can prevent agents from migrating to unreliable hosts to keep the agents' security, though many of the resources connected on grid environment will be unavailable in the case. Thus, we propose a method that enables mobile agents to migrate securely onto unreliable hosts and continue their works even on unreliable hosts. In this paper, we describe the detail of the proposing method and its performance evaluation.

1. はじめに

我々の研究グループではこれまで，実空間に設置したカメラやセンサなどのデバイスを利用して大量のデータを収集し，それらのデータをネットワーク上の高度な計算機資源を活用して処理するための高度ユビキタス環境[10]の実現を目指して，グリッド上で自律的かつ適応的に動作するモバイルエージェントをベースにした，アプリケーション作成のためのフレームワークJa-Net on Grid[12]を提案し実装してきた．Ja-Net on Gridは，エージェントがグリッド環境におけるネットワークのパフォーマンス変化などに動的に対応して効率的に処理を行うためのモバイルエージェントシステムであり，グリッド上に分散配置されたデータベースへの効率的なアクセスも考慮している．

Ja-Net on GridではGlobus Toolkit[7]のGSI（Grid Security Infrastructure）を使用しており，そ

†1 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†2 NTT未来ねっと研究所
NTT Network Innovation Laboratories
†3 大阪大学サイバーメディアセンター
Cybermedia Center, Osaka University
†4 神戸大学工学部電気電子工学科
Department of Electrical and Electronics Engineering, Faculty of Engineering, Kobe University

れぞれのホストが実行を許可するエージェントを適切に識別してセキュリティを確保している．しかし，グリッド環境では膨大な計算リソースを利用できる一方，その中に悪意のあるホストがいる可能性もあり，エージェントの内部変数が不正に参照されたり，改竄されたりする可能性がある．信頼できないホストへの移動を禁止することでそれらの悪意のあるホストからエージェントを守ることが可能であるが，グリッド環境における膨大な計算リソースを最大限に利用することができない．グリッド環境では，SETI@home[5] のように任意のユーザが貸し出しているリソースを有効に利用することも目的としており[11]，信頼できないホストを介して移動を繰り返しても安全にタスクを処理できることは重要である．

本論文では，Ja-Net on Grid において，信頼できないホスト上であってもエージェントが安全にタスクを実行するための手法を提案する．具体的には，エージェントがホスト間を移動する際に，エージェントの持つ変数を暗号化して，移動先のホストでその変数を安全に復号化して使用することによって，ホストからの不正な参照や改竄を防止する．また本手法を利用する場合に必要な処理コストや実アプリケーションのパフォーマンスに対する本手法の影響を調べ，本手法の有効性を確認した．

以下，2 章では Ja-Net on Grid の概略とその問題点について述べる．3 章では関連研究について概要をまとめ，本論文における問題点への適用の可能性について述べる．4 章ではエージェントを保護する手法を提案し，その詳細について述べる．5 章では本手法の実現にかかるコストを分析してその有効性について考察した後，本手法を実アプリケーションに適用した場合のパフォーマンスについて述べる．6 章では本論文のまとめと今後の課題について述べる．

2. Ja-Net on Grid

本章では，Ja-Net on Grid のグリッド環境における動作の仕組みと現在のセキュリティについて述べ，その問題点について述べる．

Ja-Net on Grid では，サイバーエンティティ（Cyber Entity，CE）と呼ぶ自律的なサービスコンポーネント（モバイルエージェント）が，ランタイムシステムである ACERE（Abstract Cyber Entity Runtime Entity）の上で他のいくつかの CE と協調してタスクを実行することで，ユーザにサービスを提供する．CE は ACERE から提供されるリソースが減少すると，リソースが潤沢な他の ACERE に移動して実行を続けることができる．サービスを構成する複数の CE 間で行った通信の内容に応じて，その CE 間にリレーションシップ（関係性）が生成される．サービスを利用したユーザがそのサービスを良いと評価した場合，そのサービスを構成する CE 間のリレーションシップは強められ，同様のサービスがより多く提供される．Ja-Net on Grid では，リレーションシップが強く協調動作を行う CE を同時に移動させることで，より効率的なタスク処理を可能としている．

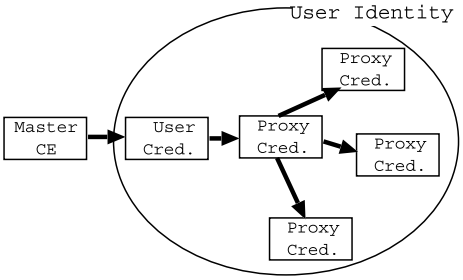

図 1 Ja-Net on Grid の認証動作
Fig. 1 Authentication of Ja-Net on Grid.

図 1 に示すように，CE はその CE を生成したユーザの証明書から派生したプロキシ証明書を持ち運ぶ．各 ACERE は CE のプロキシ証明書を参照することで，実行を許可しているユーザが生成した CE だけを動作させることが可能となる．

現在は信頼できる ACERE に接続されている ACERE はすべて信頼できるものとして移動先を決定しているが，これでは 1 章で述べたような，任意に貸し出されている膨大な計算リソースの安全な利用ができないため，任意の ACERE において CE が自分のセキュリティを確保できるような手段が必要となる．

3. 関 連 研 究

悪意のあるホストからエージェントを保護する手法に関する研究が行われている[3]．

3.1 部分結果のカプセル化

モバイルエージェントを用いて構築したオークションシステムでは，エージェントが各ホストを巡回して入札情報を収集し，落札者を決定する．このとき，あるホストが他のホストの入札情報を改竄してしまうと，正常な入札が行えない．そこで，文献 1) では，各ホストに独自の証明書を保持させ，その証明書を用いて各ホストが処理した変数に署名するようにしている．このようにすることで，他のホストが処理した情報の不正な参照や改竄が不可能となり，エージェントのタスクが正常に行われる．この手法を利用することで，

エージェントを保護し，処理結果の信頼性を高めることが可能であるが，事前に信頼できるホストに証明書を配布しなければならず，信頼できないホストのリソースを活用したタスクの実行は実現できない．

3.2 実行の追跡

文献 2) では，部分結果のカプセル化と同様に，各ホストに独自の証明書を保持させ，それぞれのホストにおいて移動時の状態を署名付きで記録しておくことにより，どのホストにおいて改竄があったのかを検出している．この手法により，信頼できないホスト間を移動しても改竄があった場合にはそれを検出できるようになる．しかしこの手法では悪意のあるホストからのエージェントに対する不正なアクセスを防ぐことはできないため，本論文における目的は達成できない．

3.3 関数の暗号化

文献 4) では，実行対象となるメソッドを移動前に変換し，移動先のホストに意味のある値や演算を参照できないようにすることでエージェントの動作の保護を試みている．ホストに戻った後，そのエージェントは計算結果に対して逆の変換を行い，意味のある値を結果として取り出す．この手法では関数の動作を暗号化することでエージェントの保護を試みているが，暗号化できる対象はメソッド内に含まれる多項式と有理関数に限られており，悪意を持ったホストが繰り返しローカルに復号を試みることで値を取り出せる危険性がある．

3.4 Locker パターン

Locker パターン[9] では，ホスト上にロッカーと呼ぶオブジェクトの保管庫を用意し，エージェントが他のホストに移動する際に，セキュリティ上問題があると思われるすべての情報をそのロッカーに預けてから移動する．これにより，エージェントがたとえ悪意のあるホストに移動しても，重要な情報に不正にアクセスされることはない．しかし，ロッカーに入れてしまった情報に関するタスクは，他のホストに移動すると継続できない．

4. CE の保護

本章では，移動先の ACERE による変数の覗き見や改竄が不可能な CE の保護手法を提案する．

4.1 提 案 手 法

Ja-Net on Grid のこれまでの実装では，名前を動的に指定して変数にアクセスできる Java のリフレクション API を用いることで，ACERE が CE の内部変数を覗き見たり改竄したりすることが可能である．本論文ではこの問題に対処する方法について述べる．なお我々の想定する ACERE の利用形態は，ユーザが自分の所有するリソースの上で ACERE を実行しており，ユーザがその ACERE に投入したタスクがサブタスクに分割され，グリッドネットワーク上に存在する ACERE に分配されるというものである．そのため，CE の移動元の ACERE はつねに信頼できるものとしている．また，バイトコード自体や直列化されたバイト列が改竄されることはないものと仮定している．

まず考えられる簡単な対処方法は，CE の変数を private なメンバとして宣言しておくことである．これにより，デフォルトの環境ではリフレクション API を使用しても外部からこの変数にアクセスすることができなくなる．しかし，セキュリティマネージャがない場合や，ReflectPermission（“suppressAccessChecks”）に対応した checkPermission() メソッドが SecurityException 例外をスローしないセキュリティマネージャを用意している場合には，その変数の Field クラスの setAccessible() メソッドを true を引数にしてコールすることでアクセスできるようになる[6]．そのため，CE の変数を private なメンバとして変数を宣言しても，ACERE から簡単に覗き見たり改竄したりすることが可能であり，完全に CE を悪意のある ACERE から保護することはできない．

信頼できる ACERE のリストを事前に CE に与えておき，そのリストから移動先を選択するようにすれば CE の安全性を保持できるが，利用する ACERE を限定してしまい，グリッド環境に接続された多くのリソースを利用できなくなってしまう．本論文では，一定のセキュリティを保ったうえで，ネットワーク上のすべての ACERE を利用して計算できるようにすることが目的である．

提案手法では，CE の持つすべての変数を暗号化し，タスク実行時にのみ復号化してデータを参照・更新できるようにする．具体的な処理手順を以下に示す．

(1) **変数の暗号化** CE が移動する際に，移動元の ACERE が鍵を生成する．CE は，その鍵を使用して CE の持つすべての変数を暗号化する．
(2) **鍵の取得** CE は移動先で移動元の ACERE と通信して鍵を受け取る．
(3) **変数の参照と更新** 鍵は処理が終了するまで各メソッドに引数として渡され，CE は変数を参照するときと変数を書き換える場合にこの鍵を使用して復号化・暗号化を行う．

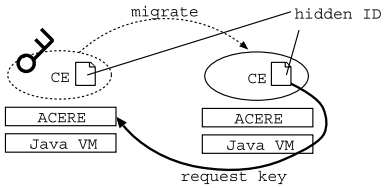

図 2 ACERE を介さない鍵の取得
Fig. 2 Key acquisition not through ACERE.

(4) 変数の復号化 タスクの処理を完了した後，CE は移動元の ACERE に戻って変数を復号化する．

手順 (2) におけるソケット通信を Java API を使用して行うことで，ACERE は通信に介入できなくなる（**図 2**）．移動する CE のクラスには，その CE に対応した外部に公開されない ID がコンスタント・プールに埋め込まれる．コンスタント・プールは，バイトコード中に現れる定数をまとめて管理するための場所である．そして移動後に CE が鍵を取得する際に，自分の ID を移動元の ACERE に送る．移動元の ACERE は鍵の取得要求が CE からのものであることを確認したうえで鍵を送信する．復号化された変数の値はつねにスタック上にのみ格納され，メンバ変数の値の更新はその値を暗号化してから元の値と入れ替えるため，変数はつねに暗号化された状態となって，ACERE からはアクセスすることができなくなる．以上のように，本手法では CE のメンバ変数を暗号化し，他の ACERE に移動した後も，ACERE がアクセスできない領域に必要なデータを保持することによって，CE の安全性を確保している．

本手法は，従来の Ja-Net on Grid システム用に構築されたアプリケーションのソースコードを書き換えることなく適用することが可能であり，本手法を用いることで，各 CE は自分の状態を盗み見られたり改竄されたりすることなく任意の ACERE 上でタスクを処理できるようになる．

4.2 実　　装

本節では，Ja-Net on Grid における提案手法の実装について述べる．本手法は，CE のバイトコードを変換することで実現できる．まず保護すべき変数を参照しているすべてのコードを，鍵を使用して値を複合化するメソッドに変換し，変数に値を代入するすべてのコードを，鍵を使用して値を暗号化するメソッドの呼び出しに変換する．元のオブジェクト（int 型などのプリミティブ値の場合にはそれに対応する Integer クラスなどのオブジェクト）を直列化してバイト列に変換した後，そのバイト列を暗号化する．暗号化されたバイト列は，バイトコードの変換によって追加した Map オブジェクトに，変数名をキーとして格納する．Ja-Net on Grid では，変数を参照するためのメソッド getVariable() と，変数を更新するためのメソッド setVariable() を提供している．具体的には，getVariable() が Map オブジェクト内の暗号化されたバイト列を読み出して復号化し，setVariable() は新しい値を暗号化したバイト列を Map オブジェクトに格納し直す．CE が自身の変数を参照したり書き換えたりする場合には，その CE のバイトコードが ACERE に読み込まれたときに，getVariable() メソッドと setVariable() メソッドが埋め込まれ，参照と書き換えはこれらのメソッドを使って行うように変更される．例として，**図 3** に示すソースコードから生成されるバイトコードは，**図 4** に示すソースコードと同じ働きをするように変換される．

```
class MyCE extends AbstractCyberEntity {
  public MyObj foo;
  public void work() {
    MyObj foo2 = /* foo を用いた操作 */;
    foo = foo2;
  }
}
```

図 3 変換元の CE
Fig. 3 Source class.

鍵の取得のための ID をコンスタント・プールに埋め込んでいるのと同様に，コンスタント・プールに直接鍵を埋め込むことも可能であるが，バイトコードの解析と比較してコンスタント・プールの参照は容易であり，直接埋め込むようにはしていない．バイトコードにおける変数の実際の値はそのコードを 1 度実行するまでは確定できず，単純には参照できないが，コンスタント・プール内の定数は簡単なフォーマットのバイト列となっており，パーサを用意するだけでその定数の値が分かるためである．移動元の ACERE から移動先の ACERE に対してソケット通信を通じて鍵を送るのは 1 度だけとしており，ダミーの ID をコンスタント・プールに混在させることでセキュリティを高めることができる．またその ID をさらに何らかの手法（バイト列の並びを変える，定数を加算するなどの操作をランダムに組み合わせる，など）で変換しておいて ID を利用する際に復元するメソッドを介するようにすれば，バイトコードの参照なしに鍵を取得することができなくなり，試行を繰り返して鍵を復元することはできなくなるため，さらにセキュリティを高めることができる．ACERE はコンスタント・プール

```
class MyCE extends AbstractCyberEntity {
  public MyObj foo;
  public Map variableMap;
  private Object getVariable(String name,
      Key key) {
    byte[] encryptedBytes =
  (byte[]) variableMap.get(name);
    byte[] decryptedBytes =
      /* 暗号化したバイト列を key で複合化 */;
    Object obj = /* 復号化したバイト列から
      オブジェクトを復元 */;
    return obj;
  }
  private Object setVariable(String name,
      Key key, Object obj) {
    byte[] bytes =
      /* obj を直列化したバイト列 */
    byte[] encryptedBytes =
      /* バイト列を key を用いて暗号化 */
    variableMap.put(name, encryptedBytes);
  }
  public void work() {
    Key key = /* ソケット通信で鍵を取得 */;
    work_(key);
  }
  public void work_(Key key) {
    MyObj foo_=(MyObj)getVariable("foo",key);
    /* foo_ を用いた操作 */
    setVariable("foo", key, );
  }
}
```

図 4 変換後の CE
Fig. 4 Modified CE.

からランダムに抜き出した値を使用して鍵の要求を繰り返すことで一定数の CE に対して不正なアクセスを行うことが可能であるが，正しい ID をともなわない不正な鍵の要求を繰り返す ACERE は想定している以上に危険な ACERE であると判断して，その利用を中断することが可能である．

なお，上記と同様に，コンスタントプールに直接鍵を埋め込み，それとともにダミーの鍵を埋め込む手法も考えられるが，ネットワークを介して鍵を取得する場合とは異なり，解錠試行可能な鍵がコンスタントプールに埋め込まれているため，悪意を持った ACERE はコンスタントプール内のすべての鍵を使用して解錠を試みることが可能である．この場合コンスタントプールの解析を行うだけで CE の情報が覗き見られることになり，移動元の ACERE はそれを検知することができない．それに対して，コンスタントプールに ID とダミーの ID をともに埋め込んだ場合には移動元の ACERE からいったん鍵を取得する必要があるため，ダミーの ID で鍵を取得しようとした時点で移動先の ACERE が不正なアクセスを試みようとしていることを検知できる．

4.3 ACERE への影響の考慮

本手法では，ACERE を介さずに直接 Java API にアクセスすることでスタック上に鍵を取得し，ACERE からその鍵を参照されることを防いでいる．これは ACERE にとってはセキュリティ上好ましくないが，CE のソケット通信は特定のポートでのみ行うものとしておけば，CE が作成するソケット通信用の Socket オブジェクトが，移動元のホスト名および鍵取得用のポート番号の定数でのみ初期化されることをバイトコードレベルで確認することが可能であるため，そのようなライブラリを ACERE に提供しておくことで，ACERE は不用意にポートを空けられ，DOS 攻撃などに利用される心配はなくなる．

5. 性能評価

本手法を適用した場合に発生するコストは，暗号化・復号化のコストと鍵を送るためのコストである．本章では，提案手法を実現する際のそれぞれのコストを分析し，この手法を実アプリケーションに適用した場合のパフォーマンスについて述べる．

5.1 実験環境

性能評価のための実験には，次のような構成の 2 台のマシンを使用した．

Linux マシン
- CPU：Intel Pentium III 1.4 GHz
- HDD：120 GB
- メモリ：512 MB
- OS：RedHat Linux 7.3
- JDK：JDK 1.4.1_03-b02

PowerBook G4
- CPU：PowerPC G4 800 MHz
- HDD：60 GB
- メモリ：1 GB
- OS：Mac OS X 10.3.5
- JDK：JDK 1.4.2_05-141

暗号化の鍵には，共通鍵暗号アルゴリズムである DES（Data Encryption Standard）を用いている．

5.2 暗号化・復号化のコスト

6 文字のアルファベットからなる Java の文字列オブジェクト，すべての CE が持つ WorkingRange オブジェクト，そして int 型のプリミティブ値を表すための Java の Integer オブジェクトの暗号化と復号化にかかるコストを計測した．暗号化の対象はオブジェクトを直列化して得られたバイト列であり，それぞれのバ

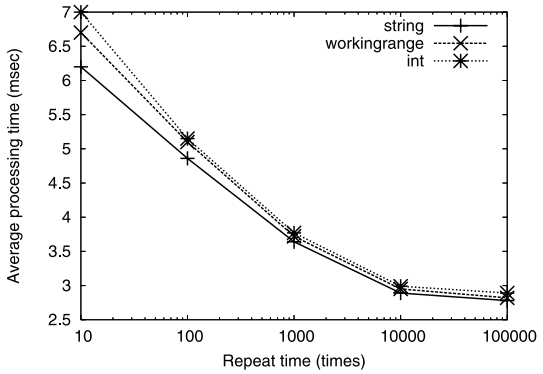

図 5 Linux マシンにおける暗号化のコスト
Fig. 5 Cost of encryption on Linux machine.

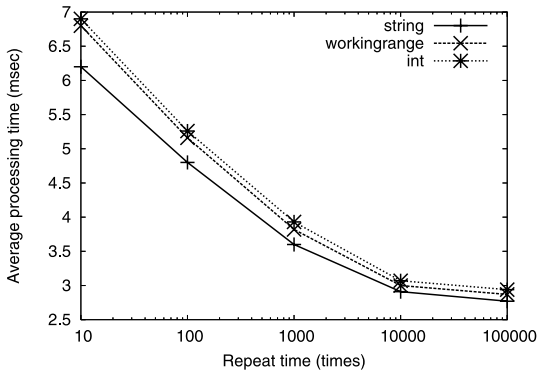

図 6 Linux マシンにおける復号化のコスト
Fig. 6 Cost of decryption on Linux machine.

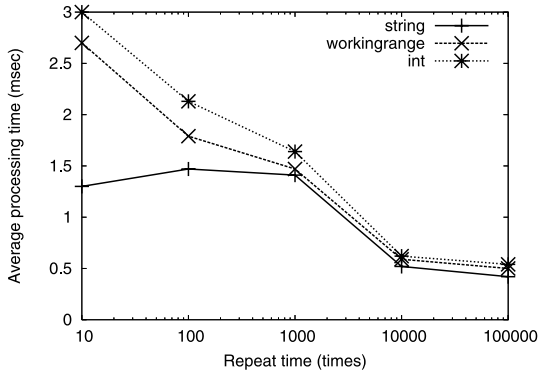

図 7 PowerBook G4 における暗号化のコスト
Fig. 7 Cost of encryption on PowerBook G4.

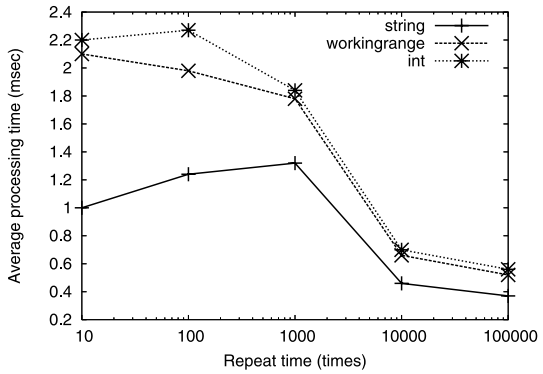

図 8 PowerBook G4 における復号化のコスト
Fig. 8 Cost of decryption on PowerBook G4.

イト列のサイズは，文字列オブジェクトが 12 バイト，WorkingRange オブジェクトが 61 バイト，Integer オブジェクトが 81 バイトである．

図 5 と**図 6** に Linux マシンにおける暗号化および復号化のコストを，**図 7** と**図 8** に PowerBook G4 における暗号化および復号化のコストを示す．横軸は連続して暗号化あるいは復号化を行った回数であり，縦軸は総所要時間を繰返し回数で割って算出した，オブジェクト 1 個あたりの暗号化あるいは復号化にかかった時間である．

この結果より，Linux マシンにおいても PowerBook G4 においても，暗号化と復号化のコストはほぼ同じである．PowerBook G4 の方がプロセッサ速度が 2 倍近い Linux マシンよりも 2〜3 倍速く暗号化と復号化の処理を終えているが，Linux と Mac OS X における Java VM の最適化処理，暗号化・復号化処理の実装の違いによるものである．

実際にユーザが実装する CE の持つ変数の数は 10〜100 個程度であると考えられるため，暗号化および復号化にかかるコストは，Linux マシンの場合でおよそ 5〜7 msec，PowerBook G4 の場合でおよそ 1.5〜3 msec であると考えられる．これより，800 MHz 以上のプロセッサ速度を持つマシンを使用した場合には，10 個の変数を持つ CE の場合におよそ 10.5〜70 msec，100 個の変数を持つ CE の場合におよそ 105〜700 msec のコストが生じることとなる．分散処理させた場合には，移動距離が 1 ホップ伸びるごとにこのコストがホップ数分かかってくるため，何ホップの移動を許可するかによってリニアにコストが加算される．1 ホップの場合には，700 msec が最悪のコストである．実際のアプリケーションにおいてはこのコストと，次に示す鍵送信のコストが加算されたものが本手法を適用する際に全体としてかかるコストになるものと考えられる．

5.3 鍵送信のコスト

鍵を送信するためのコストを計測するため，802.11b の無線 LAN（最大 11 Mbps）と 1000BASE-T のギガビットイーサネットワーク（最大 1Gbps）の 2 種類のネットワークを用意し，それぞれのネットワークにおいて鍵の送付にかかる時間を計測した．この計測は，ACERE から CE への鍵データの送信を開始し

表 1 実アプリケーションにおける暗号化のコスト
Table 1 Encryption cost in a real application.

	暗号化なし	暗号化あり
無線 LAN	141.4 msec	265.0 msec
GbE	37.2 msec	114.4 msec

た時点から，鍵の受信が完了したことを知らせるために CE から ACERE へ送られる 1 バイトのデータを ACERE が受け取るまでの時間を計測している．その結果，無線 LAN 上では平均 7.33 msec，ギガビットイーサネットワーク上では平均 2.67 msec で鍵の送付が完了することが分かった．これより，本手法を適用した場合のパフォーマンスの低下は，1 ホップの場合で，4〜710 msec となる．なお，鍵の生成は，いずれのマシンでも 1 個の鍵の生成あたり 1 msec 以下の処理時間となっているため，ほとんど影響はない．

5.4 実アプリケーションでの評価

本手法を実際のアプリケーションに適用して，計測を行った．実測に用いたプログラムは，円周率の計算をモンテカルロ法によって 10 万回の繰返し回数で行うものである．使用したマシンは，暗号化のコスト分析のために使用した PowerBook G4 が 1 台と，同じくコスト分析に使用した同じ構成の Linux マシンが 2 台である．無線 LAN 環境における計測は PowerBook G4 と Linux マシンの間で行い，ギガビットイサーネットワークにおける計測は Linux マシン 2 台の間で行った．いずれの場合も 2 台の ACERE を用意しており，最初の ACERE にタスクを投入すると CE が生成され，もう片方の ACERE に移動して計算を行い，計算が完了した時点で元の ACERE に戻る．**表 1** に，CE の生成時から計算を終了して元の ACERE に CE が戻る時点までの時間を，暗号化なしと暗号化ありのそれぞれの場合について 5 回計測した平均を示す．これより，無線 LAN における暗号化のコストは 123.6 msec，ギガビットイーサネットワークにおける暗号化のコストは 77.2 msec となり，本手法の適用によるパフォーマンスの低下は予測した範囲内に収まっていることが分かった．

6. おわりに

本論文では，グリッド環境で適応的な動作を行うモバイルエージェントシステム，Ja-Net on Grid におけるモバイルエージェントのセキュリティに注目し，CE がタスクを実行する ACERE からの攻撃に対処する移動手法を提案し，その詳細について述べた．また，本手法を適用する際のコストを評価し，実アプリケーションに適用した際に，パフォーマンスが一定以上低下しないことを確認した．本手法により，悪意のある ACERE が Grid ネットワークに存在した場合でも，その ACERE を避けることなく，CE は安全にタスクを遂行できる．

本論文ではバイトコードが改竄されないことを前提にエージェントのセキュリティを確立したが，今後はバイトコードの一部が改竄された場合についても，その改竄をエージェントが検知し動的に対応できるように，本手法の拡張を検討し進めていく予定である．また Ja-Net on Grid では，あるタスクから派生したサブタスクにおいて最終的な結果にどれだけ寄与するかといった重要性が異なる場合に，それぞれのサブタスクの重要性を考慮しつつ，計算リソースを効率的に使用することを目的としている[12]．今後は，これらのタスクの重要性を ACERE の信頼度やパフォーマンスの違いとともに考慮し，より効率的にタスク処理が行えるように Ja-Net on Grid を拡張していく予定である．

謝辞 本論文を執筆するにあたってご協力をいただいた西尾研究室の諸氏に感謝する．なお，本研究の一部は，文部科学省 21 世紀 COE プログラム「ネットワーク共生環境を築く情報技術の創出」，文部科学省特定研究領域（C）「Grid 技術を適応した新しい研究手法とデータ管理技術の研究」（プロジェクト番号：13224059），ならびに，科学研究費補助金（基盤研究（B）（2））「大規模な仮想空間システムを構築する放送型サイバースペースに関する研究」（プロジェクト番号：15300033）によっている．ここに記して謝意を表す．

参 考 文 献

1) Bennet, S.Y.: A sanctuary for mobile agents, *Secure Internet programming: Security issues for mobile and distributed objects*, Springer-Verlag (2001).
2) Giovanni, V.: Protecting Mobile Agents through Tracing, *Proc. 3rd ECOOP Workshop on Mobile Object Systems*, Jyvälskylä, Finland (1997).
3) Jansen, W.: Countermeasures for Mobile Agent Security, *Computer Communications, Special Issue on Advanced Security Techniques for Network Protection*, Elsevier Science BV (2000).
4) Sander, T. and Tschudin, C.F.: Protecting Mobile Agents Against Malicious Hosts, *Mobile Agents and Security*, Lecture Notes in Computer Science, Vol.1419, pp.44–60, Springer-Verlag (1998).
5) SETI@home: Search for Extraterrestrial Intel-

ligence at home.
http://setiathome.ssl.berkeley.edu/
6) Sun Microsystems: AccessibleObject.
http://java.sun.com/j2se/1.4.2/docs/api/java/lang/reflect/AccessibleObject.html
7) The Globus Project. http://www.globus.org/
8) Tim, L. and Frank, Y.（著），村上雅章（訳）：Java 仮想マシン仕様，ピアソンエデュケーション (2001).
9) Yariv, A. and Danny, B.L.: Agent Design Patterns: Elements of Agent Application Design, *Autonomous Agents 98* (1998).
10) 板生知子，塚本昌彦，山本 淳，田中 聡：高度ユビキタス環境のための Ja-Net アーキテクチャ，電子情報通信学会総合大会論文集 (2004).
11) 中田秀基：グリッドコンピューティング—Globus, Java CoG キットによるグリッドポータル構築，*JAVA PRESS*, Vol.27 (2002).
12) 沼田哲史，板生知子，小川剛史，塚本昌彦，西尾章治郎：動的なグリッド環境における効率的でセキュアなリソース利用のためのモバイルエージェントシステム Ja-Net on Grid，情報処理学会論文誌：データベース，Vol.45, No.SIG 14(TOD 24) (2004).

(平成 16 年 9 月 20 日受付)
(平成 16 年 12 月 29 日採録)

（担当編集委員 石田 和成）

沼田 哲史（学生会員）

1978 年生．平成 14 年大阪電気通信大学大学院工学研究科情報工学専攻博士前期課程修了．同年大阪大学大学院情報科学研究科マルチメディア工学専攻博士後期課程入学．モバイルエージェントを主としたプログラミング環境の研究開発に興味を持つ．

板生 知子（正会員）

1994 年東京工業大学電気電子工学科卒業．平成 8 年 Stanford 大学大学院 Computer Science 学科修士課程修了．同年日本電信電話（株）入社．同社未来ねっと研究所研究員．ネットワークサービスシステムの研究開発に従事．

小川 剛史（正会員）

1997 年大阪大学工学部情報システム工学科卒業．1999 年同大学院工学研究科博士前期課程修了．2000 年同研究科博士後期課程中退後，大阪大学サイバーメディアセンター情報メディア教育研究部門助手となり，現在に至る．博士（情報科学）．グループウェア，ヒューマンインタフェース，バーチャルリアリティ，オーグメンティッドリアリティに興味を持つ．ACM，電子情報通信学会，日本バーチャルリアリティ学会会員．

塚本 昌彦（正会員）

1987 年京都大学工学部数理工学科卒業．1989 年同大学院工学研究科修士課程修了．同年，シャープ（株）入社．1995 年大阪大学大学院工学研究科情報システム工学専攻講師，1996 年同専攻助教授となる．2002 年より同大学院情報科学研究科マルチメディア工学専攻助教授となり，現在に至る．工学博士．ウェラブルコンピューティングおよびユビキタスコンピューティングに興味を持つ．ACM，IEEE 等 8 学会の会員．

西尾章治郎（フェロー）

1975 年京都大学工学部数理工学科卒業．1980 年同大学院工学研究科博士後期課程修了．工学博士．京都大学工学部助手，大阪大学基礎工学部および情報処理教育センター助教授，大阪大学大学院工学研究科情報システム工学専攻教授を経て，2002 年より同大学院情報科学研究科マルチメディア工学専攻教授となり，現在に至る．2000 年より大阪大学サイバーメディアセンター長，2003 年より大阪大学大学院情報科学研究科長を併任．この間，カナダ・ウォータールー大学，ビクトリア大学客員．データベース，マルチメディアシステムの研究に従事．現在，Data & Knowledge Engineering，Data Mining and Knowledge Discovery 等の論文誌編集委員．情報処理学会フェローを含め，ACM，IEEE 等 9 学会の会員．