AMES for enhancing security in ubiquitous communication

石川, 直樹 / ISHIKAWA, Naoki

(発行年 / Year)
2007-03-24
(学位授与年月日 / Date of Granted)
2007-03-24
(学位名 / Degree Name)
修士(理学)
(学位授与機関 / Degree Grantor)
法政大学 (Hosei University)

AMES for Enhancing Security in Ubiquitous Communication

Naoki Ishikawa

Graduate School of Computer and Information Sciences,
Hosei University,
Koganei-shi, Tokyo 184-8584, Japan
naoki.ishikawa.af@gs-cis.hosei.ac.jp

Abstract

Most of current security strategies in communications use one or a few number of encryption methods. When the third party knows which encryption method used, the communication can not be secured and easy to be attacked. To enhance security in communications, this paper proposes an Automated Multi-Encryption System (AMES) to the problem. AMES is one of the security architectures for ubiquitous communication. This system sets up the encryption authority, named Encryption Station (ES), to process an encryption task. The ES has many encryption and decryption methods and dynamically assigns one of them to be used in communications. Thus, it makes the third party difficult to know what kind of encryption method used. The advantage of AMES is that the addition of an encryption method becomes easy and AMES is able to be applied for ubiquitous communication where ubiquitous devices are of futures of low power, light battery, and less storage. Consequently, communications among ubiquitous devices can be strengthened and a reasonable performance in terms of communication efficiency or security can be achieved.

Supervisor: Runhe Huang, Professor

1. 序論

1.1. ネットワークセキュリティ

昨今のコンピュータの進歩は目覚しく、安価になったコンピュータの存在は、私たちにとってとても身近なものになった。しかし、飛躍的にコンピュータの技術が伸びる一方、個々のコンピュータの性能は限界に近づいて来ていると言われ、その打開策として、グリッドコンピューティングなどの分散コンピューティングの技術に目が向けられるようになった。これによって、より多くの処理を並列に実行することが出来るようになり、ハードウェア、ソフトウェア両面で並列分散処理の研究が盛んになっている。

これら分散コンピューティングのようなこれからの情報社会を担う技術が身近になるにつれて、重要になってくるのが、コンピュータ間で組まれた「ネットワーク」である。コンピュータ単体で出来ることも限界に陥りつつある現状、ネットワークによって、情報の伝達・共有による新たな見解が見出されるようになっている。それによって、コンピュータ間の通信は、より複雑になり、その転送されるデータ量も膨大なものになった。かつて、アナログ回線使用時におけるワールドワイドなネットワークを構築するには、データ転送量がネックだったが、回線の速度も次第に速くなり、現在では一度に送ることができるデータ量が飛躍的に向上したことで容易となった。そこで、データの信頼性というものに着目したネットワークが注目を浴びるようになったのである。例えば、会社で使用される膨大なデータを扱ったデータベースやサーバークライアント型のシステムなど、コンピュータ間のネットワークは欠かせないもので、これからの Web2.0 [1] に代表されるような相互型のインターネットサービス社会では、特に重要な要素になって行くことと思われる。そして、ネットワークの普及に伴って、重要性が加速していく要素が「セキュリティ」である [2]。かつて、一台の PC 上でローカル作業される分にはさほど問題にならなかった第３者の悪意のある攻撃というものが、最近では非常に問題とされており、犯罪も後を絶たない。従って、情報の共有に付随する情報の一貫性、通信の安全性が問われるようなセキュリティが重要視される時代となってきたのは間違いない。そして、ネットワークにおけるデータを安全に送るセキュリティ技術が注目を浴びる中、私たちはそこに重点をおいたネットワークセキュリティの分野に注目した。

ネットワークが広範囲に普及するに連れて、それらのネットワークセキュリティの盲点をつく悪意のある第３者の攻撃が目立ち始めたのは言うまでもない。第３者の攻撃目的は、機密情報の盗聴、改ざん、技術の見せしめやなりすましなど様々である。現在、コンピュータは、数多くのコンピュータウイルス、スパイウェアやマルウェアなどの悪意のあるプログラムの被害にさらされ、深刻な問題になっている。それらは、日々、悪意のある者によって新しいものが作り出され、世界規模に広がったネットワークの脅威となっているのが現状である。考えられる攻撃方法は様々で、機密データを安全に通信するには、これら全てを想定した対策をしていく必要がある。

よって、ネットワーク技術が発展するにつれて、セキュリティの需要は増加し、サービスを展開する上で決して無視できない大きな要素となったことは間違いない。今や、どんな小さな規模のネットワークにおいても、決して大切なデータをそのままの形で通信してはいけない時代となったのだ。

また、ネットワークセキュリティを実現する上で、「Dependability」の確保が非常に大切である。「Dependability」は次の４つの要素から成り立っている [3]。「Safety」、「Reliability」、「Security」、「Availability」の４つ。４つの関係は、図１のような包含関係で構成される。

- 「Safety」は、如何に安全な通信ができるか。
- 「Reliability」は、認証を含め、如何に信頼できる通信が行えるか。
- 「Security」は、暗号化、アーキテクチャーを含めたセキュリティそのもの。
- 「Availability」は、障害の発生し難さや、障害発生時の修復速度など、如何に耐久性があるか。

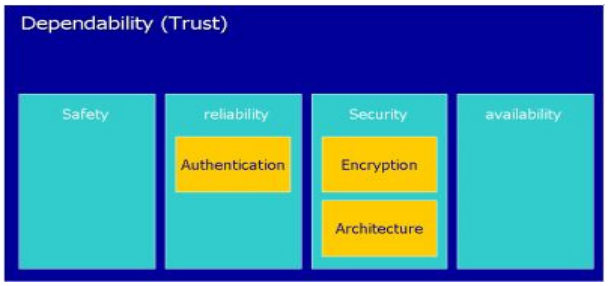


図 1. Dependability 関係図

これら４つの要素を、個別に十分なレベルで実装することで、ネットワークセキュリティに必要十分な「Dependability」を確保することができる。この十分な「Dependability」を実現することが、サービス提供者の安心と安全なサービスにつながるのである。私たちは、この「Dependability」の実現を目指し、この論文では、ネットワーク通信における、第３者からの悪意のある攻撃を防ぐための Encryption Station を提案する。

1.2. ユビキタスコンピューティング

IC 技術の発展に伴って、コンピュータの数はより増加し、その大きさもより小さくなった。それによって、分散システムにおけるネットワークデバイスは、従来のデスクトップ PC やノート PC からいわゆるユビキタスデバイスと呼ばれる PDA や携帯電話などの小型デバイスに移行することが可能となった。ユビキタスコンピューティングとは、こういったどこでも接続可能な小型のユビキタスデバイスを使い、「いつでも、どこでも、だれでも」気軽に使えるネットワー

クインフラの総称である [4]。これらユビキタスデバイスは、ネットワークを介して、有線もしくは無線でつながれる。

ユビキタス社会では、これまでの一台のコンピュータを複数の人が使うといった環境に対して、一人が複数のコンピュータを使う第 3 世代の環境を示したもので、１９８８年に Mark Weiser 氏が提唱した [5]。昨今のＩＰｖ６ [6]の広がりでどんなものにもＩＰアドレスを付加できる環境になったことで、それぞれのデバイスは途切れのない通信を行い、常時変化する情報を捉えることができるようになった。ネットワークを介して携帯電話だけでなく、センサーやカメラ、IC タグなど、あらゆるデバイスで構成されるユビキタスコンピューティングは、これからの情報社会の行先を担う重要な技術となっていくことが期待される。

また、ユビキタスネットワークを構成するデバイスは様々であるが、基本的に小型であり、いろいろな制限がある中で、ネットワークを組まなくてはいけない。インフラの構築の大変さや限定された CPU の処理能力、少ない電力供給、そして、限られた容量の記憶領域等、このような制限を考慮してネットワークを構築していく必要がある。このような、制限がある中で、第 3 者の悪意のある攻撃に対して、どのような方法で安全な通信を実現すればいいのだろうか。そして、災害や衝撃などのあらゆる局面においても、柔軟に対応できるような柔軟性も意識しなくてはならない。センサーを例に取ると、天候や温度によって、デバイスが壊れてしまった場合の対処やバッテリーが切れた場合の対処など、予期せぬ事態にも十分な対処ができるような柔軟性というものが求められる。

この論文では、ユビキタス環境を実現する際のネットワークセキュリティにおける「Dependability」を確保するべく、上記の制限を考慮したデータ通信時のセキュリティについて研究をしてきた。そして、これらの問題に対して、Automated Multi-Encryption System (AMES)を提案する。AMES はユビキタス通信のためのひとつのセキュリティアーキテクチャーであり、このシステムは、Encryption Station (ES)と名づけた暗号局を設けて、暗号メソッドを提供する。ES は数多くの暗号化もしくは復号化メソッドを持ち、通信に対して動的に割り当てることで、第３者に対して、どのような暗号メソッドが使われているか特定されにくくするといったメリットがある。AMES によって、誰もが安全にいつでもどこでもユビキタスデバイスを使って情報の共有や、リソース、サービスにアクセスするユビキタスコンピューティングの実現を可能にする。

2. 関連技術

2.1. Kerberos

Kerberos [7]とは、マサチューセッツ工科大学 (Massachusetts Institute of Technology (MIT))によって、学生のいたずらを防止するために開発された認証セキュリティシステムである [8]。その構造は、認証をする側のサーバーと認証をされる側のクライアントのほかに認証局

（certificate authority (CA)）という機関を設け、認証を任せるといったものである。バージョン１から３は開発段階だったので公開されず、バージョン４になって初めて公開されることとなった。現在の最新バージョンは、Heimdal[9]と名づけられたバージョン５で、通信の暗号化を共通鍵暗号方式の DES（data encryption standard）だけで行っていたバージョン４に対し、DES だけでなく、さまざまな暗号メソッドの中から選択できるようになっている。バージョン４において、DES はアメリカ、カナダ以外の国への輸出が禁止されていたので DES をはずした e-Bones というものが諸外国によって開発された。また、e-Bones をもとにスウェーデンストックホルム王立技術研究所（KTH）が改良した KTH－KRB４ [10]が今でも幅広く使われている。やはり、この Kerberos を見ても、現在のセキュリティにおいて、数少ない暗号メソッドでは不安であるということが受け取れるのではないだろうか。Kerberos の使用例では、Microsoft が windows2000 より一部セキュリティ面で使用しているのを始め [11]、信頼あるメーラーである Eudora も採用していることが挙げられる。一部 Linux のバージョンでは標準的に採用されているなど、コンピュータの核となる OS に採用されるほどの信頼性をもつ Kerberos は、十分な信頼性を持った認証セキュリティアーキテクチャーと言えるだろう。私たちは、図２のようなサーバークライアントシステムの外側という立場を利用して成功している Kerberos のアーキテクチャーに習って、トライアングル構造を AMES に採用することにした。強固なセキュリティを実現するためには、CA のようなシステムの外側に置かれた機関が非常に重要な役割を果たす。なお、Kerberos のアーキテクチャーは、図２のように CA とクライアントは直接接続されておらず、完全なトライアングル構造ではない。これを拡張して、AMES では完全なトライアングル構造を提案する。

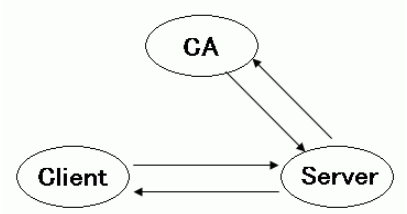


図 2. Kerberos のアーキテクチャー

2.2. センサーネットワーク

ユビキタスコンピューティングにおいて、もっとも小型となるデバイス「センサー」を使って構成されるネットワーク、それがセンサーネットワークである。センサーはその小ささゆえ、屋内、屋外問わず日常生活の様々な所で使えることが最大の利点と言うことができる。しかし、センサーは最も制約があるデバイスといっても過言ではない。屋外に置く完全に自動化されたネットワークにおいては、電力供給が一番の問題点となることだろう。CPU やセンシング機能、ストレージデバイスなどの過度な搭載は無駄に電力を消費する原因になりかねない。第一、小型化されたデバイスには、それほどの資源を搭載できる余裕はない。近年の集積回路の発達は、ある

程度の処理能力を持つプロセッサやその他のデバイスを比較的安価で手に入れることを可能にした。そのおかげで、比較的広範囲なセンサーネットワークの構築が容易になり、自然から多くの情報をより正確に取得できるとともに、その情報を使った技術の発達につながる結果となった。しかし、十分なネットワークを組むにはまだまだ個々の能力は乏しく、サービスを展開するには、取得したデータを扱う十分な処理能力をもったサーバーの力を借りる必要があるのが現状である。それらセンサーとサーバー間には、やはり十分に安全で信頼ある通信が必要不可欠で、処理能力が乏しいセンサーネットワークにおいて、安全な通信をするにはどうしたらいいのかといった問題に焦点が絞られる。

また、センサーから発生される信号はウェーブフロント拡張効果という作用によって、距離に応じて弱くなり、なんの制約もない空間においては、電磁波の強度は距離の 2 乗に比例して弱くなっていくということが判っている [12]。従って、無線で信号を受け取るサーバーは、確実にセンサーからの信号を捕らえ、処理する能力が問われる。外部からの情報を確実に取得するために、一定区域内のデバイスの数を多くすれば、それだけ、ネットワークが複雑になり、処理が重くなるのでコストがかさんでしまう。よって、センサーの配置を考慮する必要もあり、システムがデバイスを確実に検出するためには、ネットワークのコストがあがっても分散させるべきである [12]。このコストは、ネットワーク間の通信の複雑さからくるコストであり、センサー各々がある程度情報を処理できる能力があれば、抑えることができる。それには、センサー各々の処理能力が問われ、このことから、できるだけ多くの資源を使えるに越したことがないことが言える。しかし、プロセッサの負荷はバッテリーの消費につながるので、過度な搭載には注意しなくてはならない[13]。

これらを踏まえて、ユビキタスコンピューティングを考慮したネットワークを想定した場合、消費電力の観点からも、プロセッサの負荷やストレージデバイスやメモリの節約を考える必要があると言える。プロセッサの負荷に関しては、通信データ暗号化時のビット長を調節したり、処理を分散させたりすることで、アプローチできるのではないだろうか。AMES は、これらの問題に対して、有効なシステムになることだろう。

2.3. RFID

また、現在、ユビキタスコンピューティングにおいて、センサーとは違った分野で大きな注目を浴びているのが RFID(Radio Frequency Identification)である [14]。薄いカード状のＩＣタグのデータを無線リーダーで読み取って、そのデータに応じた処理を行う。センサーなどに比べ、比較的安価なＩＣタグを使用することによって、サービスの幅は大幅にひろがる。しかし、センサーとは違い、RFID 自体には処理能力はなく、それを受け取るリーダーによって様々な処理が行われる。大掛かりなリーダーを用意することは、コスト面やスペースを考慮した上で適切ではない。リーダーの性能がものをいうこのシステムは、接触させなければ反応しないものと、触れなくても広域な電波で読み取ることができるものの 2 種類が普及しており、後者は電波障害を起

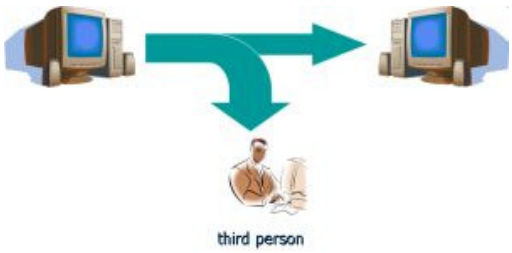
こしかねないとされ、問題となっているケースもあるようである。RFID を使ったシステムの重要な点は、サーバーに確実にデータを送ること、そして、確実にＩＣタグを個別特定できることである。普段身につけるさまざまなものにＩＣタグを付加することによって、街中の便利なサービスが受けられるような世の中になっていくことが期待されている。現在使用されている例としては、Suica [15]や Edy [16]が挙げられる。このような信頼がものをいう多数の人が使うようなサービスにおいては、決して誤認識やバグがあってはならない。よって、この RFID においても、無線通信やリーダーとサーバー間の通信といったところでセキュリティの需要は高まっていることは確かで、AMES の実力が発揮できる場となるのではないだろうか。

3. マイアプローチ

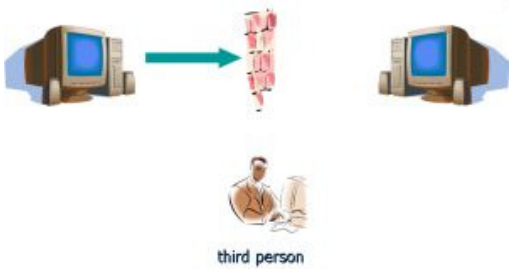
3.1. 攻撃に対する対処方法

最初に、ネットワークを構築するに当たって、考えられる第３者からの攻撃の種類を考えてみた。以下の５つが想定される攻撃の種類で、それぞれの攻撃を視野にいれて、対策をする必要がある。そして、ユビキタス環境を考慮した場合、従来のネットワークにおける方法とは違った対処方法を考える必要がある場合があり、それぞれに対して以下のような対処を考えた。

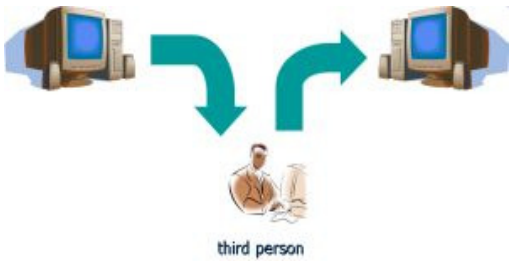
- 情報盗聴
  - 盗聴を防ぐために、解読されないような暗号化を施す。
  - 流用されないようなデータの形式にして送る。(ファイル形式ではなく、バイトコードなど)
  - 簡単に復号できないように、強固な暗号化を施す。
  - 安全なネットワークを構築する（VPN など）。



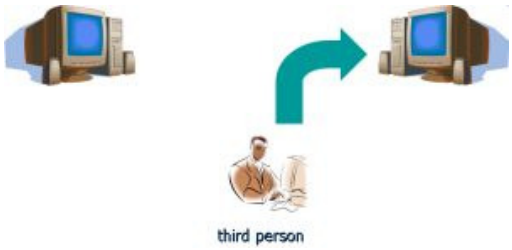
- 通信妨害
  - 違うルートを使って送受の通知を送る。
  - (相手からの受信通知を待つ)
  - データを複数回に分けて送る。(受信側は送られた回数によって判断する)



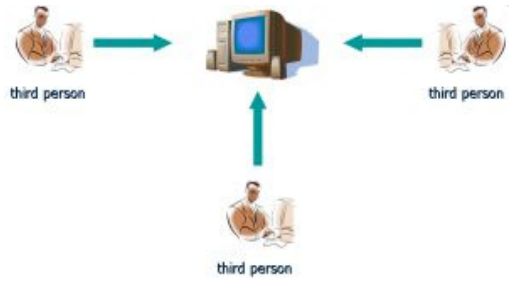
- データ改ざん
  - 情報盗聴と同じ対処方法。
  - 改ざんにかかる時間を想定して、タイムスタンプから判断する。
  - シーケンシャルなデータ通信を行う。



- なりすまし
  - 互いに理解できる複雑な通信規約を設定しておく。
  - 受信側は、送信側に確認を取る。
  - 特定のポート以外の通信を遮断する。(HTTP や FTP でよく使われる８０や２０は使用しないようにする)
  - データが送られてくるべき時間以外は通信を遮断する。



- DoS (denial of service)攻撃
  - 一箇所から何度もデータを送ってくる相手を遮断する。
  - 一度、データを受信したら、特定の場所からの通信を一定の時間遮断する。



- デバイス（センサーなど）の破壊
  - 頑丈な作りにしておく。
  - 人目につかない壁などに埋め込む
  - 衝撃を受けたら、アラームが鳴る。

現実的ではない対処方法となるかもしれませんが、これらの対処方法を効率も兼ねて実装する必要があり、特に、第３者からの攻撃を防ぐだけではなく、受けた場合の対処方法ということも考える必要がある。これらのほかにも、セキュリティーホールなどの脆弱性をついた攻撃も想定される。

また、強固なセキュリティの実現と言った場合、その強さというものを証明するのは大変難しい [17]。そのシステムにおいて、考えられる脆弱性を熟知し、それらに対して十分な対処ができてこそセキュリティの強さというものを表せるのではないだろうか。セキュリティの強さというものは、明示的に、外部からの攻撃を受けた際のかかる時間と費用によって数値化し比較するといった方法もある。脆弱性を認識していないシステムでは、悪意のある第３者の介入によって、どれだけの被害を受けるのか、想像の範囲ではない。特に、データの漏洩が起こってはいけない

企業などの通信などにおいては、通信にかかる時間を考慮するよりも先に安全性を確保する必要があることは言うまでもない。

3.2. AMES

私たちは、これまで述べてきたネットワークセキュリティの重要性に着目し、特に現在の情報社会で最も注目を浴びているユビキタスコンピューティングを想定した AMES (Auto mated Multi-Encryption System)を提案する。AMES とは、データ送信時の暗号化において、自動で暗号メソッドが変化し、第３者からの特定を阻止するためのシステムである。自動で選択される暗号メソッドは、４章で述べる特定の選択法を使用する。このシステムの狙いは、資源の限り有る環境下であるユビキタス環境での安全な通信と、少ない資源の有効活用である。従来のサーバークライアントシステムを拡張し、外部に Encryption Station(ES)を設置する図３のようなトライアングル構造で実現する。

3.2.1. Architecture

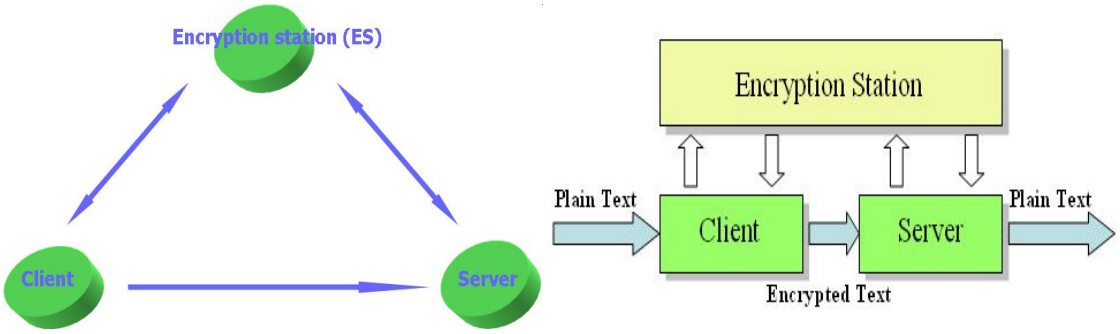


図 3. AMES のトライアングル構造

ＡＭＥＳの暗号化の仕組みと流れを以下に示す。

① ユビキタスデバイスであるクライアントがデータを取得する（RFID のリーダーや、センサーでのデータ取得）。

② クライアントは、接続可能な ES を探し、接続する。接続できる ES のアドレスは、あらかじめリスト化しておく必要がある。そのリストの中から現在通信可能（ダウンしていない負荷の少ないところ）な ES を割り出して接続を行う。接続可能かどうかは、ping コマンドを使用した。

③ 接続した ES に暗号メソッドの要求メッセージとデータのタイプ(文字列か数字列か)、長さを送信。

④ ES は特定の方法（４章にて後述）で、送るべき暗号メソッドを判別し、そのメソッドに応じた鍵を生成する。このとき、共通鍵暗号なら、共通鍵をひとつ生成し、公開鍵暗号においては、公開鍵と秘密鍵を対で生成する。公開鍵はクライアントに送るのに対し

て、秘密鍵はクライアントの ID を付加して、サーバーからの鍵要求に備えて ES に保持しておく。また、ES は予め十分な数の暗号メソッドをコンパイルしクラスファイルの状態で保持しておく。

⑤ クライアントに、ES 独自で決定した暗号メソッドと公開鍵、それと適当なメソッドで暗号化したタイムスタンプを送信する。送信される暗号メソッドは、バイトコードとして送信することが可能で、セキュリティを考慮したシーケンシャルな通信を行う。シーケンシャルな通信は、第３者に読み取られても、データを解読されない手段として有効。他には、https や ftps などの安全なプロトコルを使用する手段も考えた。

⑥ クライアントは、送られてきたメソッドと鍵を使用して暗号化を行う。このとき、送るべき暗号データに接続した ES のアドレスと自分自身の情報（クライアント ID）を付加する。

⑦ クライアントは暗号化されたデータをサーバーに送信する。ここでも、シーケンシャルな通信を行う。

⑧ サーバーはクライアントからの暗号文を受け取ったあと、付加されている ES のアドレスをたどって ES に鍵の要求をする。鍵の要求メッセージには、クライアントの ID を付加して送信。

⑨ ES はサーバーから送られてきたクライアントの ID から判別して、復号メソッドと秘密鍵を特定。サーバーに復号メソッドと秘密鍵を送信する。このとき、タイムスタンプの復号メソッドをリストから探し、復号化する。もし、かかった時間などから第３者の改ざんが特定された場合、通信は遮断される。

⑩ サーバーは、送られてきた復号メソッドと鍵を使って復号化を試み、成功したら平文を取得できる。

⑪ サーバーは無事複合化できたメッセージをクライアントに送信する。

⑫ 今回、クライアントはメッセージを受け取り、使用した暗号メソッドを消去する。今回メソッドの消去の仕方は、java のガーベッジコレクションにまかせた。

3.2.2. メリットとデメリット

AMES を使う最大のメリットとして、通信の安全性と資源の有効活用が挙げられる。少ない種類の暗号メソッドを持つクライアントで、暗号化したデータをサーバーに送り続けていると、いずれ解読されてしまうといった危険性がある。つまり、暗号強度は、時間の経過に比例して、弱くなっていくのである。２，３種類の暗号メソッドを使ったシステムであれば、第３者が通信にどんなメソッドで暗号化しているのか特定されやすく、特定された時点から、そのメソッドだけに対する解読が始まることになり、解読されるのも時間の問題となってしまう。それに対し、AMES を使用すると、複数の暗号化メソッドの中から自動で選択されるメソッドによって暗号化されるので、どのような暗号が施されているのか特定されにくくなるといったメリットが得ら

れる。その暗号メソッドの数は、多ければ多いほど良く、メソッドの種類の多さに従って、第３者からの特定は困難になる。しかし、現在安全とされる暗号長を実装するのは、資源の少ないユビキタスコンピューティングを想定したこのシステムでは難しい。例えば、RSA において、これまで５１２ビットまで解読されてしまったという事実があり、現行では、１０２４ビットの実装が様々なシステムに実装されているが、いずれ解読されてしまう危険性があることを考えると、それよりも長いビット長で暗号化する必要性が近い将来生じてくるだろう。そうなると、暗号化するために使用するパワー（CPU や電力）は大きなものになってしまう。今現在、汎用の CPU では処理がおいつかず、IC カードなどの小型デバイスでは、専用のコプロセッサを回路に組み込んで処理しているのが現状で、搭載メソッドを違うものに変えようと思っても物理的な回路の制約から簡単には行えない。そこで、私たちは、このような問題に対処するために AMES を提案する。数多くの暗号メソッドを用意すれば、ひとつの暗号化メソッドにおける有効暗号ビットを下げられるのではないかという考えに至った。例えば、１つの暗号メソッドで１０２４ビットを実装するのに対し、２つの暗号メソッドを実装すれば、ひとつあたりのメソッドの暗号長を下げても同じかそれ以上の強度を実現できるのではないかと考えた。それによって、現行の有効的なビット数（危険性の薄い最低のビット長）を下げて暗号化が可能になり、暗号化１回１回に要するプロセッサのパワーを節約できるとともに、電力消費を抑えることができるのではないだろうか。暗号ビットが長ければ、それだけプロセッサのパワーと消費電力はかさむ［13］。さらに、物理的な制限のために困難であった暗号メソッドの追加も容易になる。よって、少ない資源のもとで行うユビキタスコンピューティングに適した暗号化が行えるのではないだろうか。

一方、デメリットとして、やはり、外部から暗号メソッド取得するということで、通信に時間がかかってしまうといったことが挙げられる。これについては、上記のように、メソッド一つあたりの暗号長を下げることで対応し、出来る限りのスピードアップを図ることでアプローチした。しかし、ここで注意しなければならないのが、ビット長の下げすぎである。もしも、使用メソッドの特定と同時に暗号文を盗聴されてしまう危険性を考えると、ある程度の強度を保つ必要がある。また、数多くの暗号メソッドを保持している ES に負荷が集中してしまうといったデメリットも挙げられる。ES は、クライアントとサーバーに対してひとつずつのエージェントを配置するので非常に負荷がかかり、これによる、ES ダウンを回避するための策として、１つのクライアントが接続できる ES を複数設置する対策を考えた。これは、ひとつの ES がダウンしてしまっても違う ES に接続できるといった対処が出来、また一つあたりの ES の負荷を分散できるといった利点がうまれる。また、悪意のあるウイルスが入って、接続するための IP アドレスを変えられるといった問題に同時に対処でき、さらに、通信をする前に、接続可能かどうかを最初にチェックするので、悪意のある第３者のなりすましによるＥＳに接続する心配がなくなる。そのためにも、ES に接続する最初の段階から、平文を ES に送って暗号化してもらうといった危険なアプローチはするべきではない。

3.2.3. タイムスタンプ

さらに、AMES のセキュリティを高めるために、タイムスタンプを組み込むことにした。日本では、２００５年、４月１日から e-文書法という法案が施行され [18]、今まで紙ベースでの保存が義務づけられていた文書の電子保存を認められることとなった。この法案が施行されてから、電子保存する時間の信頼性に目が向けられ、タイムスタンプの重要性が増した。タイムスタンプを付加することによって、間違いなく、その時間にデジタルデータが作成されたことを証明することができ、データの信頼性を向上させることができる。また、この証明は、第３者からの改ざんに対する対処にもなり、有効なアプローチといえるのではないだろうか。改ざんには時間がかかるので、もし、盗聴したデータの意味を解読することができて、改ざんすることができたとしても、そのデータを転送するのにかかった時間から割り出し、そのデータの受信を阻止することができるのである。しかし、そのタイムスタンプを付加する場所が信頼できる機関でなくては意味がない。例えば、デジタル文書を作成するときに付加したタイムスタンプを、つけた本人が故意に変更できたら意味がなくなってしまう。もし変更できたとしたら、なんら信頼性を持たなくなってしまう。

これらを踏まえ、セキュリティ向上に向けて AMES にタイムスタンプを実装することにした。では、この AMES において、どこでいつタイムスタンプを付加したらいいのかを考えたところ、一番信用でき、かつ効果的な場所は、ES であるという結論に至った。それは、クライアントからもサーバーからも影響を受けない外側にいるからで、ES がタイムスタンプを発行すれば、信頼ある証明になる。タイムスタンプを付加するタイミングは、クライアントに暗号メソッドと鍵を送るときに同時に送るのがベストだろう。ただ単に付加するだけでは、第３者に読み取られた場合に、タイムスタンプだということも特定され、タイムスタンプ自体も改ざんされてしまうといった危険性が出てしまうので、これに対処するために次のような方法でアプローチした。

$$S\_Time = Encrypt(C\_Time + \alpha)$$

S_Time はクライアントに送るべきタイムスタンプ、C_Time は現在のタイムスタンプ、α はランダムな整数を指す。現在のタイムスタンプは現在の時刻がミリ秒で表される。例 1165127418974。ランダムな整数は、＋－問わずランダムで生成された整数を表す。例 1234567890。この式で得た数値を暗号化して S_Time としてクライアントに送信する。この暗号化するためのメソッドは、それほど強度を強くする必要はない。なぜなら、せいぜい１３桁程度の数字であるのと、もし解読されてしまっても、ランダムな整数が足されているといった理由で何のデータかはっきりしないだろう。暗号化計算を簡略化し、無駄な時間をかけないためにも、簡単なメソッドで暗号化するべきである。ここで、ランダムな整数と暗号化メソッドは ES のスタティックなリストにクライアント ID と一緒に登録しておく。ES のタイムスタンプを含んだクライアントからの暗号データがサーバーに送られ、サーバーから鍵と復号メソッドを問い合わせられたとき、

クライアント ID から復号メソッドを割り出し、タイムスタンプを復号化する。リストに登録されているランダムな整数を減算することで得られたタイムスタンプによって信頼ある有効な通信かどうかを特定できるのではないだろうか。あまりにも時間がたっていたり、タイムスタンプを含まない無効なメッセージを受信したりした場合、通信は中断され、失敗したことがメッセージとしてサーバーに通達される。このように、タイムスタンプを導入することによって、セキュリティ強度の向上が期待できるのではないだろうか。

3.3. 実装

今回の実装では、クライアントをユビキタスデバイスと想定しているので、まず、モバイル向け Java SDK の J2ME で実装を試みた。その中でも組み込み機器向けの Java 仮想マシン KVM(K virtual machine)における、携帯電話や PDA などの小型デバイスを対象とした CLDC (Connected Limited Device Configuration) 環境での実装を目指した。しかし、強度な暗号化を実現するためには、膨大な桁の計算をする必要があり、それに必要なクラス BigInteger が J2ME には存在しない。BigInteger クラスとは、Java で実装されている多倍長整数クラスで、有効桁数は無限とされる。データを送信するときには、String 型に変換して送ることができるが、計算をする段階で String なのは都合が悪い。同じく有効な長さが無限とされる String に対して、long 型でも有効桁数が６４ビットなのを考えると、暗号計算における多倍長クラスの重要性が理解できるだろう。J2SE の BigInteger クラスのソースを参考に J2ME への実装を試みたが、Math クラスの多くのメソッドが使えない状態で、非常に困難であったため、今回の実装では、カーナビやセットトップボックスなどの中・大型デバイスを対象にした CDC (Connected Device Configuration) 環境を対象にした J2SE を使って行うことにした。CLDC 環境における暗号化は、J2SE から、部分的に javax.crypto クラスなどをインポートして実現しているようである。また、J2ME でのセキュリティを実現するための API 集 JSR１７７というプロファイルが存在する。これは、Java カードと呼ばれる特殊なＩＣカードに特化したセキュリティプロファイルであり、PKI を使って暗号通信を実現しているようであるが、今回の実装にあたっては、十分でない。

この AMES は端末への出力なしに、バックグラウンドですべて処理されるシステムであるが、途中経過や流れが分かりやすいように簡単な GUI を設けて実装にあたった。

3.3.1. BCEL

BCEL（Byte Code Engineering Library）とは、Jakarta Project の API のひとつである。この API は、Java クラスをコンパイルして生成される class ファイル（バイナリ形式のクラスファイル）を容易に解析、生成、操作可能とする事を目的としていて、ファイルから、もしくはソケットからの通信で得られるバイトコードストリームからもクラスを操作できる。一度、コンパイルされたあとのバイナリデータであるクラスを読み込んで、プログラムから操作して変更したあと、再

びクラスファイルにダンプすることも可能である。今回 AMES の実装にあたり、ES から取得した暗号、復号メソッドをクライアントもしくは、サーバーで実行可能なクラスに変換するときにこの API を使用した。ES に蓄えられた暗号メソッドの class ファイルをバイトコードで送信し、それを受け取った側が BCEL でクラスにダンプする。このとき、ローカルのファイルには落とさず、メモリ上に展開したクラスのオブジェクトを生成し、encrypt メソッドと decrypt メソッドを実行する。これによって、ストレージデバイスの節約につながるとともに、ファイル IO という時間がかかる操作を省くことができます。

3.3.2. クライアント

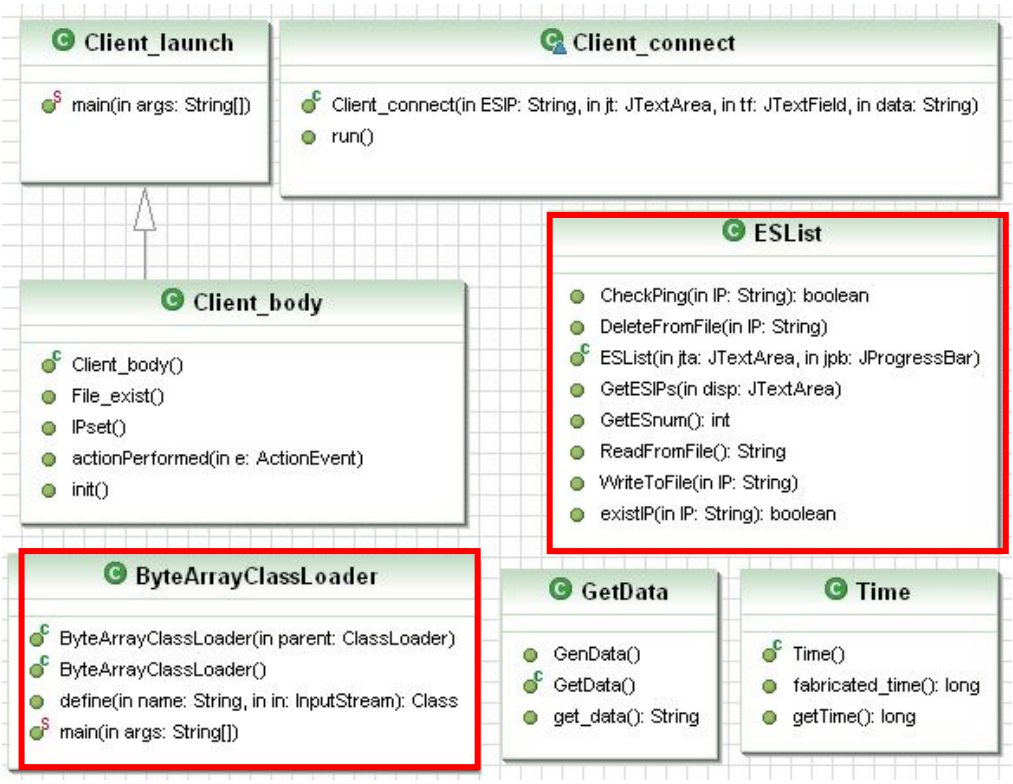


図 4. クライアントのクラス図

Client_connect クラスは ES、サーバーとそれぞれ通信するためのクラスで、一つのクラスで ES とサーバー、それぞれに通信するために、マルチスレッドで動作させた。GetData クラスでは、暗号化するためのデータを外部から取得する。CLDC 環境を想定すると、外部からのデータとは、センサーや ID タグなどを使って読み取ったデータ列のことである。ES_list クラスは、あらかじめ登録されている ES の IP アドレス群の中から、接続可能な IP アドレスを割り出すクラスで、ping コマンドを OS に送り、その結果に応じて、接続可能かそうでないかの情報を判断する。Java におけるコマンドを OS に送る方法は次の通り（例：ping コマンド）。

```
String command = "ping " + IP;
Process process = Runtime.getRuntime().exec(command);
InputStream is = process.getInputStream();
BufferedReader br = new BufferedReader(new InputStreamReader(is));

while ((line3 = br.readLine()) != null) {  結果に対する処理  }
```

これによって、コマンドを OS に送った際のレスポンスに応じて、動作を行うことが可能である。ここでは、ping コマンドが通るかどうかによって、その ping を送った ES が生きているかどうかを判断している。また、この ES_list クラスでは、接続するべき ES の IP アドレスリストを編集することもできるようにし、登録の場合には、一度 ping コマンドを実行してから、接続可能なことを確かめてから登録できるようにしている。重複して登録できないような仕組みも用意した。ByteArrayClassLoader クラスは、ES より流れてくる暗号メソッドのバイトストリームをクラス化する BCEL を使ったクラスで、クラス化された暗号メソッドを使って暗号するべき平文を Ubiquitous Mobile Encode Agent (UMEA)で暗号化する。このとき、クライアントと ES の間の通信において、暗号メソッドを送るバイトストリームと鍵の送信を一緒にするべきではない。なぜなら、一度に両方とも第３者に読み取られた場合の危険性が増してしまうからで、これに対して、バイトストリーム用のソケットとメッセージ交換、鍵を送信するソケットを用意することで、別々に送信する対処でアプローチした。なお、バイトストリーム用のソケットでは、シーケンシャルな通信を行う。このシーケンシャルな通信をすることで得られる利点として、第３者が盗聴したとしても、どの時点のデータを盗聴したか特定することができないことが挙げられる。一連の送信全部を盗聴できて初めて一つのデータとしての意味を持つことから、データを改ざんされる心配を減らしてくれることになる。これについては、IO のスピードを上げるために行われる Buffered 機能を敢えて使わず、順次送信という形を取った。

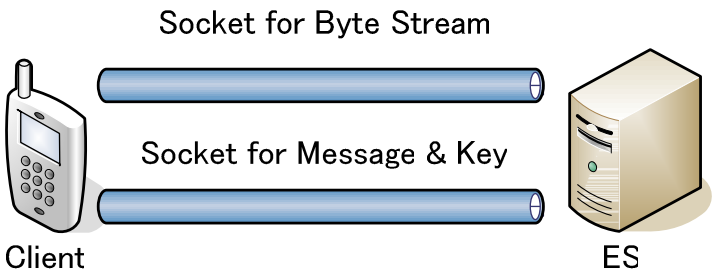


図 5.2 本のソケット通信路

暗号終了後、クライアントが、サーバーからの復号成功通知を受けた段階で、暗号メソッドは不要になる。この不要なメソッドのインスタンスは、Java のガベージコレクションに任せることになるが、その作業を促すために、System.gc()メソッドを実行することにした。このメソッドは、ガベージコレクションを促し、無駄なメモリの領域を確保することで、意図しているメモリの節約につながるのだが、リンクをたどって参照されていないものを検出するので、非常に処理が重く、復号化が行われるまでの途中で実行すべきではない。よって、復号成功の通知を受けてから実行することにした。そして、このサーバーに暗号データを送信する際にも、シーケンシャルな通信を行う。

3.3.3. サーバー

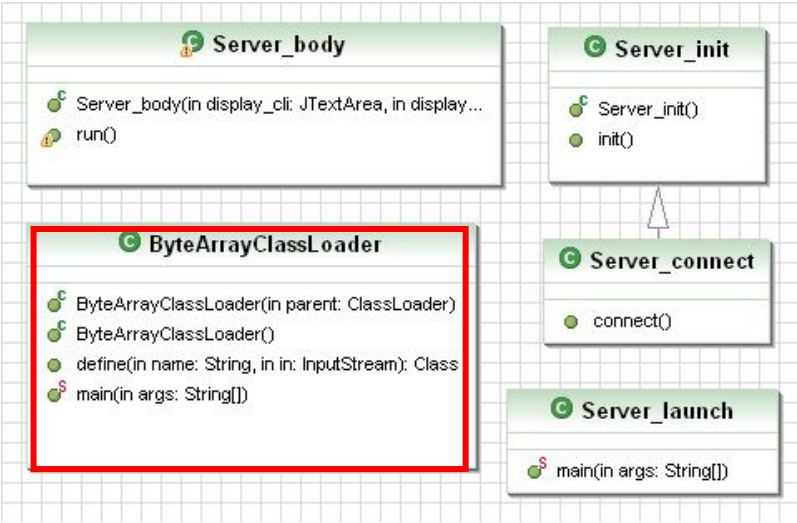


図 6. サーバーのクラス図

サーバーは、クライアントと ES との通信を行う。クライアントから送られてきた暗号データを復号化するにあたって、クライアントと同じ、ByteArrayClassLoader クラスを用意し、ES から取得するバイトストリームから復号メソッドを復元して実行する。Server_body クラスの中では、バイトストリームや鍵の取得を行っている。サーバーと ES 間の通信においても、セキュリティ上、上記のクライアントと同様に 2 本のソケットを用意して、バイトストリームと鍵を別に送ることにした。

そして、サーバーに予めいくつかの復号メソッドを用意しておくことでスピードアップを期待できるが、ES からその度にメソッドを取得することによって、あらかじめ持ち合わせるメソッド以外の暗号化に対応できる汎用性を考えると、望ましくない。完全にサーバーも復号メソッドを ES より取得することによって、AMES を使うことを想定していないサーバークライアントシステムにおいても、その可能性を広げることになり汎用性が生まれる。

さらに、サーバーにおいては、ユビキタスデバイスであることを想定していないので、資源の限りあるデバイスと仮定していない。よって、クライアントのように、メモリ確保のためにガベージコレクションを促す必要はないと言える。また、複数のクライアントの接続を可能にするために、マルチスレッドでサーバーを実現した。

3.3.4. Encryption Station

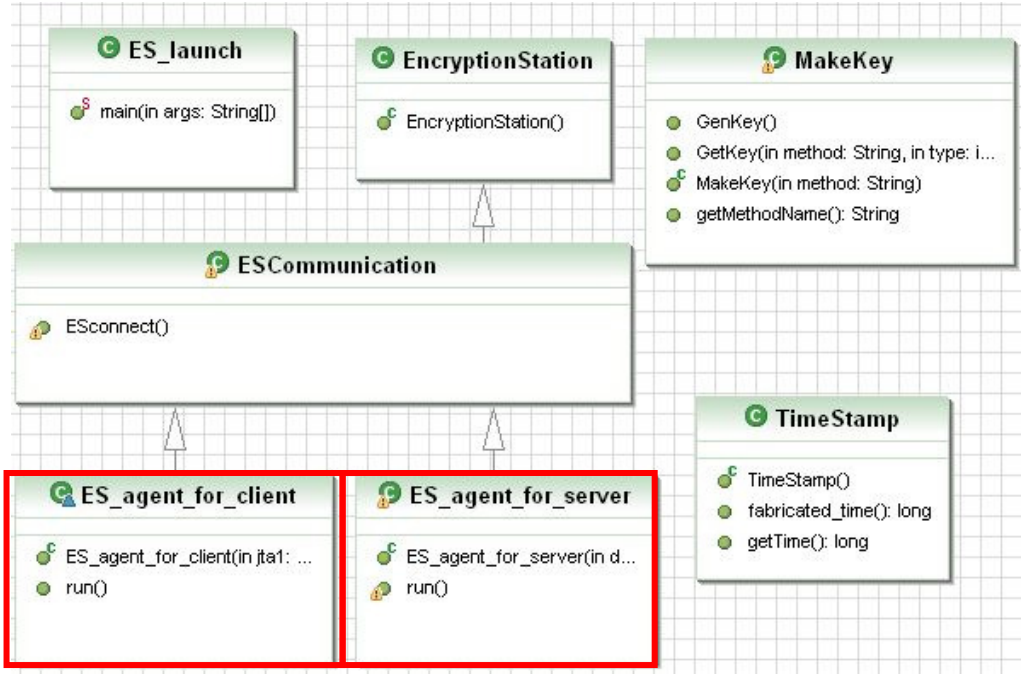


図 7. Encryption Station のクラス図

Encryption Station は、このシステムの一番重要な核となる存在である。クライアント用対応エージェント Communication Agent for Ubiquitous Client(CAUC)と、サーバー用対応エージェント Communication Agent for Ubiquitous Server(CAUS)を配置し、複数のクライアント、サーバーに対応できるようにマルチスレッドで実現する。ES は、クライアントから受けた要求に対し、暗号メソッドを決定し、それと同時に、MakeKey クラスで決定したメソッドに応じた鍵を生成する。MakeKey クラスのコンストラクタでは、引数に暗号メソッドの名前を必要とし、その名前がインスタンスフィールドにセットされる。そのインスタンスに対して、GenKey メソッドを実行することで、その暗号メソッドにあった鍵の生成がされ、GetKey メソッドを実行すると、メソッドに応じた鍵を取得できるようにした。TimeStamp クラスでは、付加する暗号化されたタイムスタンプを取得する。CAUC を介して、クライアントに送信するデータ列は図８の通り。

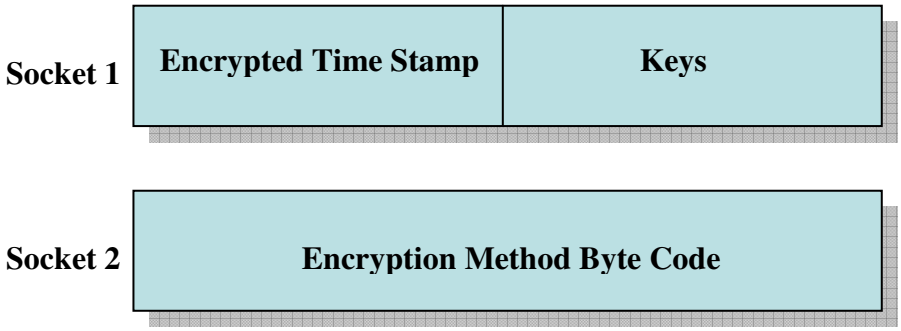


図 8. クライアントに送信されるデータ

また、クライアントの ID とタイムスタンプの復号用メソッド（メソッド名）はタイムスタンプを暗号化した段階で、リスト化して保持される。これは、全てのエージェントがアクセスできるスタティックな ArrayList で実現した。

次にサーバーから要求を受けたときの対処としては、サーバーごとに、CAUS を配置し、送られてくるクライアント ID から判断して、復号メソッドと鍵を送信する。そして、サーバーから送られてくる暗号化されたタイムスタンプに対して復号化を行い、TimeStamp クラスは、かかった時間を割り出し、ここで、かかった時間によって改ざんが特定された場合、メッセージを破棄し、サーバーに通知される。最終的に使われなくなった各々のエージェントは時間に応じて破棄される。

4. 実験と結果

4.1. 暗号化メソッドの選択方法

ES において、クライアントから送られてくる暗号メソッド要求に対して、どの暗号化メソッドを選択するのかという方法を考えた。目的は、効率の良い暗号メソッドの選択により、暗号化スピードを向上させようというものである。メソッドの決定要素は次の５つ。

- 暗号化、復号化時間
- 鍵作成時間
- 通信路の帯域幅
- 暗号メソッド、復号メソッドのファイルサイズ
- 平文の長さによる暗号長

この５つのうち、暗号化、復号化時間と鍵作成時間をまとめた「総暗号時間」と通信路の帯域幅と暗号メソッド、復号メソッドのファイルサイズ、平文の長さによる暗号長をまとめた「総通信時間」を計算に取り込むことにする。本実験では、Java での Cipher クラスを使った暗号化メソッド５種（DES、AES、TripleDES、blowfish、RC2）と、BigInteger クラスを使用してアルゴリズムを実装した暗号メソッド１種（RSA）を実装し、上記の総暗号時間と総通信時間を計測してみた。

「総通信時間」とは、ES からバイトストリームでクライアントとサーバーにそれぞれ、暗号メソッドと復号メソッドを送信する時間なので、通信路の大域幅とメソッドのファイルサイズが関わってくることになる。実装した暗号メソッドの種類とクラスファイル（.class）のサイズは、表１の通り。なお、使用したライブラリは含まない。今回、平文の長さに応じた暗号長の違いは、メソッドによって大差が生じないので計算に取り入れないことにした。

表 1. それぞれのメソッドのファイルサイズ（ライブラリ除く）

暗号名	暗号メソッドサイズ(byte)	復号メソッドサイズ(byte)
RSA	859	900
DES	1430	1093
AES	1663	1326
TripleDES	1690	1347
Blowfish	1688	1346
RC2	1663	1326

表１から判断すると、現状の回線通信速度では、考慮するほどではない微小な値であることが言える。アナログ回線５６Kbps(bit per second)の回線においても、実質の通信速度の約5.5Kbps(byte per second)で計算すると、一番大きなTripleDESのクラスファイルでさえ、1690 Bytes ÷5.5 Kbps× 1024 bytes = 0.3 秒程度に収まることになる。これを考えると現在の赤外線通信 IrDA での速度において、最低基準である 115Kbps では、通信時間が気にならない程度になるので、暗号メソッドを選択する式の中には含めないことにした。メソッドと同時に、計算に必要なライブラリも送ることを考えると、これらを考慮する必要性がでてくるが、クライアント側に使用するライブラリを必要最小限用意しておくことで対処できる。

次に、まったく暗号選択方法を設定せず、ランダムに暗号メソッドを選択した場合を考える。なにも選択式を与えないで、６つの暗号をランダムで 10000 回選択したときの時間を測定した。計測する時間は、鍵作成、暗号化、復号化のあわせた総暗号化時間である。ここでは、作成され

る鍵の長さに関係する平文の長さを変化させて以下の４台のコンピュータで計測した。なお、RSA を除いたそれぞれの暗号メソッドにおける鍵長は、平文の長さに比例するので、一定となる。RSA においては、512 ビットの暗号長で実験した。図８は、平文に数字列 100~1000 桁と半角英文字 100～1000 文字を設定したグラフである。なお、この半角英文字においては、後述する理由で RSA を除外した。

- PC1　CPU : Intel Pentium 3 700MHz, 320MB Memory
- PC2　CPU : Intel Pentium 4 2.8GHz, 1GB Memory
- PC3　CPU : Intel Pentium M 1.6GHz, 1.5GB Memory
- PC4　CPU : AMD Athlon XP 2800+, 1GB Memory

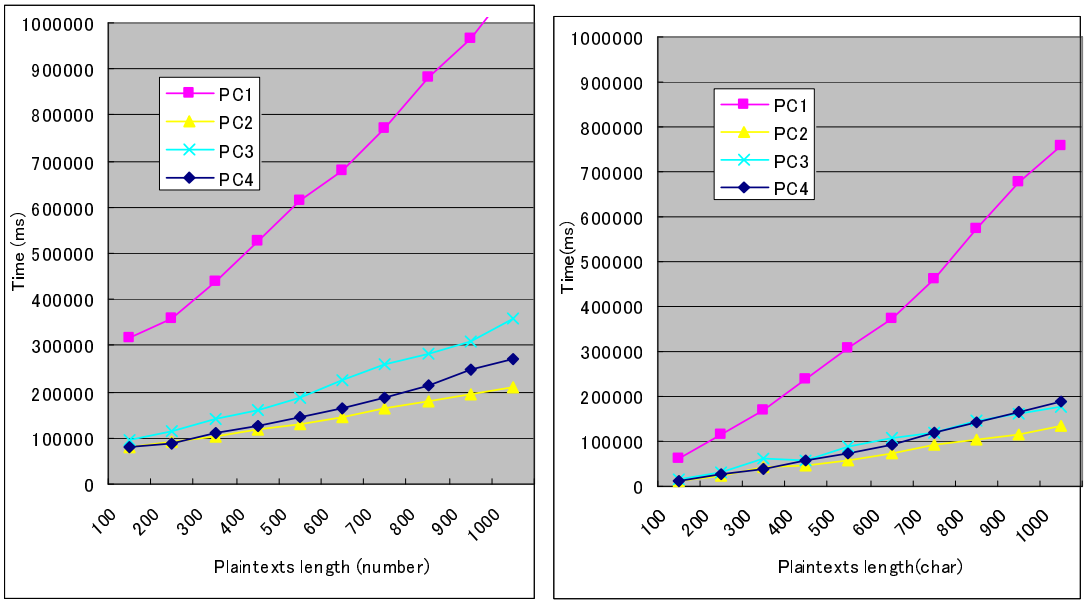


図 9. 数字列と文字列（ＲＳＡを除く）を平文にしたときの暗号時間

次にメソッド選択法を適用して実験を試みる。メソッドの選択法は次のような基準で暗号化メソッドに対してウエイトをつけ、そのウエイトにしたがって、選択する。なお、メソッドに使用される鍵長は、平文の長さから生成されるので、平文が一定であれば、同じ長さとなるが、鍵長が同じだからといって､暗号メソッドのアルゴリズムによって暗号強度は変化することに注意しなければならない。

次の２点を考慮した実験を行った。

- RSA では、基本的に数式計算を行って暗号化を行う。よって、全角文字、もしくは半角英文字の平文の場合、１文字ずつを取得し、それぞれを数値コードで表す必要があるため、非常に時間がかかることになる。よって、RSA の選択基準は、平文が数字のみの時とし、半角英字、全角が混ざっていたら、選択肢からはずす。

- それぞれの暗号時間を計測し、同レベルのセキュリティを実現できるメソッド同士で暗号時間が短いものから、それに応じたウエイトを付加していく。

各々のメソッドにかかる総暗号時間は、表２の通り。表２は、上記ＰＣ３（CPU : Intel Pentium M 1.6GHz, 1.5GB Memory）において、1000 文字を 5000 回暗号化したときの時間をまとめたものである。

表 2. それぞれのメソッドの暗号化時間

暗号名	数字列の平文	文字列の平文
RSA	215320	×
DES	106373	94466
AES	98241	94066
TripleDES	134713	106303
Blowfish	108336	104761
RC2	95277	96499

結果より、RSA の遅さが顕著に見られるが、これは、共通鍵暗号である AES、DES などにくらべ、公開鍵、秘密鍵の両方を作る必要があり、また、膨大な桁数の複雑な計算を行っているためである。よって、RSA のウエイトは低くする必要があるが、今まで述べてきたように、AMES は暗号の種類によってセキュリティの強さを高めるシステムであるので、暗号時間が遅いからといって完全に除外すべきではない。

処理時間から付加するウエイトは、一番遅いメソッド(今回は数字列の平文で RSA、文字列の平文で TripleDES)を１としたとき、2 番目以降のメソッドのウエイトは次の式で決定する。

$$TargetMethodWeight = (\mathrm{int})\left(\frac{SlowestMethodTime - TargetMethodTime}{SecondFastestMethodTime - FastestMethodTime} + 0.5 \right) + 1$$

この式の０．５は、括弧内部を四捨五入するためで、１は最も遅いメソッドを１とするための正規化である。表２の処理時間をこの式に適用して得たそれぞれの暗号メソッドのウエイトは、表３の通り。この表のウエイト値の大きさは、優先度を示す。

表 3. メソッドへのウエイト付け

メソッド名	数字列の平文	文字列の平文
RSA	1	×
DES	38	31
AES	41	33
TripleDES	29	1
Blowfish	38	5
RC2	42	26

ただ単に、処理時間の短いものだけを選択するのでは、従来の暗号化システムと変わらず、セキュリティの向上は見込めない。よって、速いメソッドに対する相対的な値でウエイトを決定する。しかし、セキュリティの向上を考慮すると、ある程度の選択のばらつきをもたせる必要があり、極端な偏りは避けるべきである。

上記のウエイトを加味した上で、メソッドを選択する式を適用するとトータル時間は次のようになる。

$$TotalEncryptionTime = \sum^{TheNumberOfMethod} \frac{EachMethodWeight}{TotalWeight} \times EachEncryptionTime$$

この式を適用して時間を計り、図１０のような結果を得た。

4.2. 実験結果

この式を使って実行した結果、図１０のような結果を得ることができた。

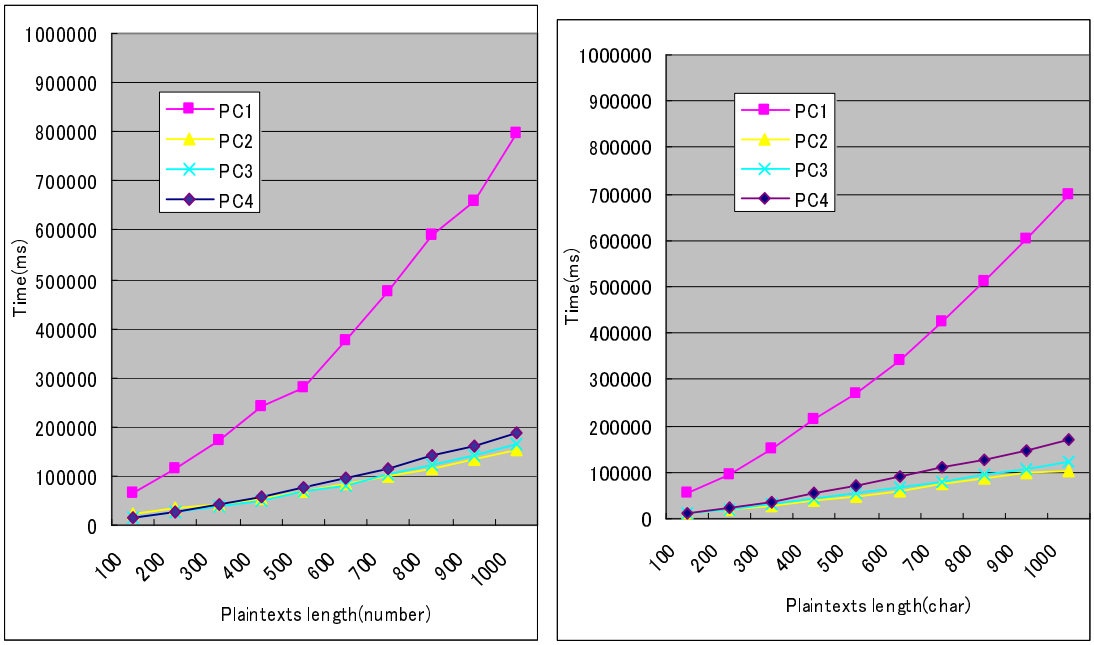


図 10. メソッド選択法を使用して
数字列の平文と文字列の平文（ＲＳＡを除く）を暗号化した時間

グラフから読み取ると、ランダムに選択した方法より、メソッド選択法を使用した方法のほうが確かに早くなっていることがわかる。今回、６つの暗号メソッドを実装したが、より多くのメソッドを実装することを考えると、今回のような的確なチューニングによって、セキュリティを保ったままのスピードアップを図ることが出来ると言える。これを実現するためには、暗号アルゴリズムや暗号長によって、どれだけ暗号時間がかかるのか、また暗号の強度を得られるのかを熟知しなくてはならない。

5. 結論

この論文では、従来のサーバークライアントシステムの外部に Encryption Station を設けるこの AMES を実装することで、セキュリティの向上を実現できた。特に、ユビキタスコンピューティングを実現するような環境においては、数少ない資源の節約を期待できる。選択する暗号メソッドの暗号時間によってウエイトをかける今回の実験は、ランダムにメソッドを選択する方法よりも確かに AMES の効率をあげることができた。しかし、暗号メソッドの選択方法を誤ると、無駄な負荷がかかり、非常に重いシステムとなってしまうので、適切なチューニングが重要になるだろう。今回の実装では、６つだけの暗号メソッドの実装を行ったが、これから、ES におけ

る暗号メソッドの数を増やす必要性が出てきたとしても、ES に置かれている暗号メソッドだけを変更すればいいので、簡単にメソッドの追加を行うことができる。もちろん、暗号メソッドの数が多くなるにつれて、この暗号システムは複雑になり、チューニングの作業も難しくなるが、それにしたがってセキュリティの強度は強くなるので、できるだけ多くの暗号メソッドを実装することが望ましい。

また、AMES は VPN [19]のような安全な環境において、最も効果を発揮するだろう。なぜなら、トライアングル構造の通信間において、第３者の妨害を最小限に抑えることができるからである。VPN を使うことで、AMES における問題点でもある IP アドレスを変化させるようなウイルスの進入は最小限に抑えることができるだろう。

さらに、データの送受信を監視するスニファーのような第３者の盗聴は、送達チェックや到達順序保障をしてくれるＴＣＰ通信などのシーケンシャルな通信を各々の機関間に確立することで防ぐことができるのではないだろうか。シーケンシャルな通信が保障されている環境においては、もし盗聴されたとしても、連続的なデータ送信なために、改ざんされにくくなることが言える。AMES は、シーケンシャルな処理がされている Edy、Suica のようなシステムに応用できるのではないだろうか。

6. 今後の課題

今回、クライアントにおいて ES が生きていることを確認する際に行った PING コマンドは、レスポンスが来るまで時間を要する。特に、ping コマンドが通らないときに起こるタイムアウトは非常にレスポンスが遅く、windows の DOS コマンドで ping を通す際のパケット４つ分の待ち時間を待っていては、暗号とは別なところで時間をくってしまう。今回は、１つ目のパケットのタイムアウトで打ち切ることで対処しているが、これでもレスポンスには数秒かかるので、他の手段を見つける必要があるだろう。複数の登録した ES がすでに接続可能な状態であることが分かっていれば、ping が通るかどうかチェックする必要もなくなるので、無駄な時間を節約できるかもしれない。これは、あくまでも安全な環境で、IP アドレスを変更するようなウイルスが入ってこないことを仮定した場合なので、さらに、悪意のあるＥＳでないことが保障されているような環境では、この限りではない。

また、J2ME のいろいろなライブラリの制約のために、CLDC 環境での実装ができなかったので、小型デバイスに搭載できる J2ME 上で、暗号メソッドを模索し、数多くのメソッドを実装することが望ましい。そのためには、多倍長クラス然り、Cipher クラスなど暗号に必要なクラスを実装することが必要であるだろう。

そして、AMES に実装する暗号メソッドの数を増やすことが望まれる。この AMES においては、それほど複雑な暗号メソッドでなくてもシステムに適用できるといったことがあげられ、メソッドの数を多くすればするほど、そういった簡単なメソッドで暗号化することが可能となる。単に

データをシフトし、そのシフトした値を鍵にするような単純なメソッドも、メソッドの数に応じては使えるかもしれない。

参考文献

[1] T O'Reilly, *what Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software,* 2005.

[2] PW Dowd, JT McHenry, *Network security: it's time to take it seriously,* IEEE, 1998.

[3] Joseph S. Valacich, *Ubiquitous Trust: Evolving Trust into Ubiquitous Computing Environments,* Case Western Workshop on Ubiquitous Computing, 2003.

[4] 中村 康昭, 長谷 佳明, *ユビキタスコンピューティング*, May 2001.

[5] Weiser, M. The Computer for the 21st Century. ScientificAmerican, 1991, 265 (3), pp.94-104.

[6] D Johnson, C Perkins, *Mobility Support in IPv6*, Work in Progress, 1998

[7] Kerberos: The Network Authentication Protocol, Massachusetts Institute of Technology (MIT), http://web.mit.edu/Kerberos/

[8] Brian Tung, *KERBEROS A Network Authentication System*, piason education.

[9] Heimdal, Kungliga Tekniska H skolan (KTH), http://www.pdc.kth.se/heimdal/

[10] KTH-KRB, Kungliga Tekniska H skolan (KTH), http://www.pdc.kth.se/kth-krb/

[11] Microsoft, Windows 2000 Kerberos Authentication, http://www.microsoft.com/windows2000/techinfo/howitworks/security/kerberos.asp

[12] G.J. Pottie, W.J. Kaiser, *Wireless Integrated Network Sensors,* Communications of the ACM, 2000.

[13] Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, Niraj K. Jha, *A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols,* IEEE Transactions on Mobile Computing, VOL 5, NO.2, February 2006.

[14] 末永 俊一郎, *Two Surges for RFID Application*, UNISYS TECHNOLOGY REVIEW 第 78 号, AUG, 2003

[15] JR 東日本, Suica
http://www.jreast.co.jp/suica/

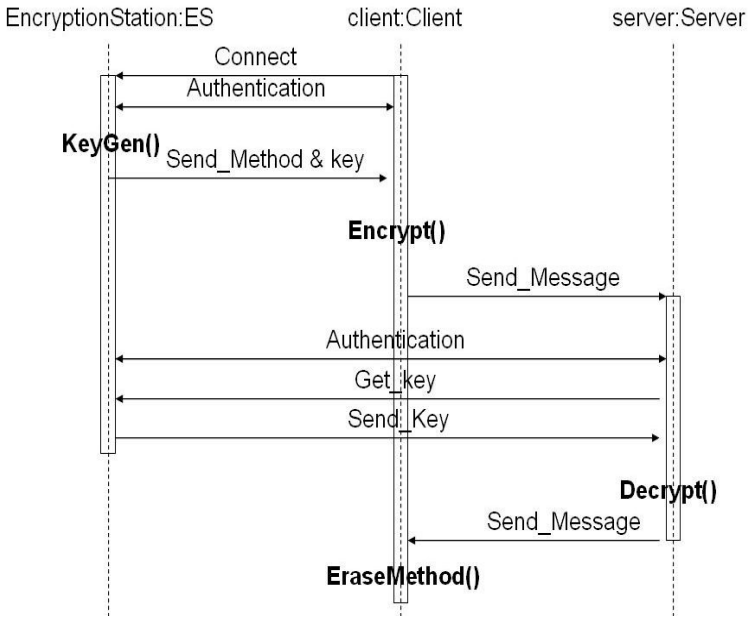
[16] ビットワレット株式会社, Edy
http://www.edy.jp/

[17] 中野秀男, *情報科学*, 大阪市立大学
http://www.media.osaka-cu.ac.jp/~nakano/

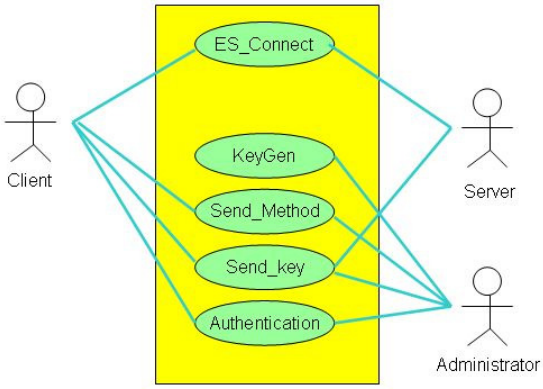
[18] 経済産業省, e-文書法
http://www.meti.go.jp/index.html

[19] Netgear, *Security & Savings with Virtual Private Networks.*

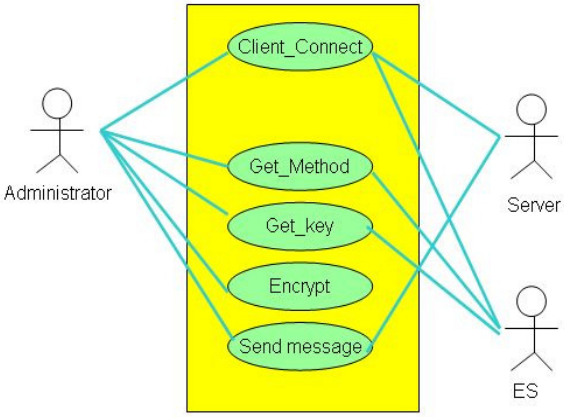
付録



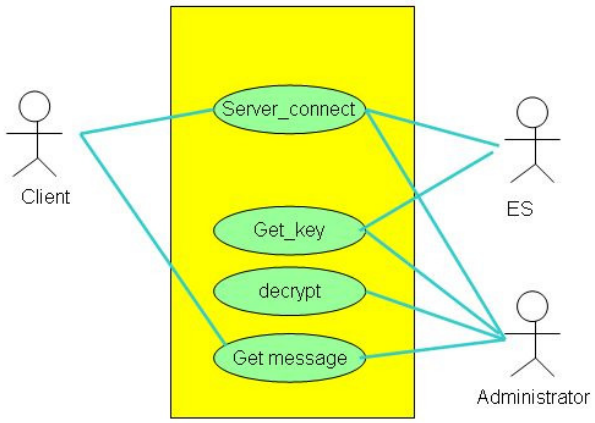
付録 1. AMES のシーケンス図



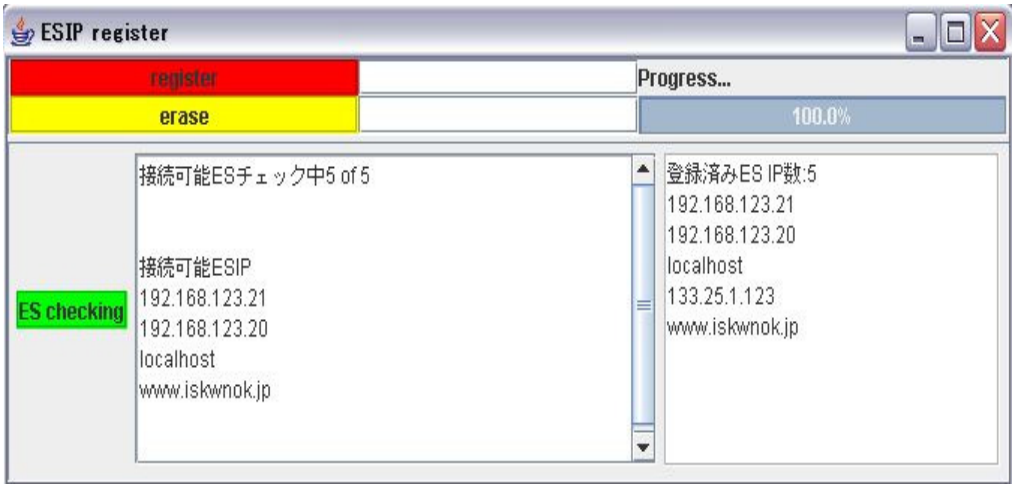
付録 2. ES のユースケース図



付録 3. Client のユースケース図



付録 4. Server のユースケース図



付録 5. ESIP register