社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－IOT－1 (14)
2008 / 5 / 9

認証デバイスを用いたOSの安全な起動制御

高田 真吾† 佐藤 聡† 新城 靖† 中井 央†† 板野 肯三†

† 筑波大学 システム情報工学研究科 〒305–8573 茨城県つくば市天王台 1-1-1
†† 筑波大学 図書館情報メディア研究科 〒305–8573 茨城県つくば市天王台 1-1-1

あらまし 企業や大学などでは，利用者に応じた独自のコンピュータ環境が用意できれば，利用者の作業効率は向上する．しかし，その管理コストは増加してしまう．また，既存の手法では，利用者や起動されるOSについて強固な認証と柔軟な起動制御を行うことはできない．そこで，本研究では，利用者の認証として認証デバイスの内部に格納された証明書を，ハードウェアの認証としてはそのハードウェアに固有な情報をそれぞれ用いることにより，利用者とハードウェアという組み合わせについての認証・起動制御を可能とするシステムを提案する．従来のBIOSに相当する位置に起動制御レイヤーを置き，認証や起動制御を行うことにより，OSの安全な起動制御を実現する．また，提案するシステムの有効性を確かめるために，仮想計算機モニタXenにおけるDomain0を現在のBIOSに相当するレイヤーのように扱い，起動制御処理を実装した．これにより，既存のハードウェアの構成に手を加えることなく，起動制御の機能を実現することができる．

キーワード 起動制御, 認証デバイス, USBトークン, 電子証明書, 仮想計算機

Secure OS Boot Control System using Authentication Device

Shingo TAKADA†, Akira SATO†, Yasushi SHINJO†, Hisashi NAKAI††, and Kozo ITANO†

† Graduate School of Systems and Information Engineering, University of Tsukuba Tennoudai 1–1–1, Tsukuba-shi, 305–8573 Japan
†† Graduate School of Library, Information and Media Studies, University of Tsukuba Tennoudai 1–1–1, Tsukuba-shi, 305–8573 Japan

Abstract In large enterprise or university campuses, to prepare customized computer environment for users will improve their work efficiency. But its administration cost will increase. And there's no existing method to authenticate users and operating systems and to control OS booting flexibly. So we propose a Secure Boot Control System which can authenticate by a couple of a user and a hardware using certificates stored in authentication devices and client systems. Secure OS boot contorolling will be realized by Boot Control Layer which is placed on the former BIOS's layer. It controls booting of OS and authenticates a user and a hardware. In order to examine the advantage of this system, we implement Boot Control Layer by using Virtual Machine Monitor Xen's Domain 0 treating as instead of current BIOS. Using this way, we don't have to modify our hardware architecture to realize the boot controlling.

Key words Boot Control, Authentication Device, USB Token, Electric Certification, Virtual Machine

1. はじめに

企業や大学等においては，多数のコンピュータを効率的に管理・運用することが求められる．コンピュータのハードウェアやOS環境などを統一することにより，管理効率を上げることができ，かつハードウェアの整備コストを下げることができる．しかし，そのシステムを利用する側からみれば，利用者一人一人に応じた使いやすいOS環境がなければ，作業の効率は下がってしまう．

利用者の要求にあわせて複数のOS環境をあらかじめ用意しておき，利用者にその選択を委ねる仕組みとしては，NetBootやPXE[4]がある．これらの仕組みでは，ネットワーク上のディスクサーバにOSイメージを格納しておくため，OS環境の集中管理が可能となる．しかし，これらの仕組みでは，OSの選択時に利用者の厳密な認証を行うことはできず，同時起動数を制限するといった柔軟な起動制御を行うこともできない．

また，特定の用途に特化したOS環境を，CDやDVDなどの外部メディアに格納しておき，そのメディアからOSを起動するというLiveCDのような方法も考えられる．しかし，外部メディアからのOSの起動を許可してしまうと，管理者が想定しないOS環境の起動も許可してしまうことになる．

利用者が要求するOS環境をあらかじめ管理者が確認でき，またそれらをまとめて管理可能であり，そのOS環境を管理者が管理するハードウェア群のいずれのハードウェアにおいても柔軟な起動制御を行えるシステムがあれば，この問題は解決できると考えられる．利用者と利用するハードウェア，起動されるOSイメージについて認証を行うことにより，管理者は起動するOSの選択を利用者に移譲できることになる．

本研究では，これらの目標を実現するためのシステムの提案を行う．利用者とハードウェアの認証については公開鍵認証を用いる．また利用者についてはその秘密鍵を認証デバイスに格納し，システムの利用にはそのデバイスが挿入されていることを必須とする．これによりシステムの悪用を防ぐことができる．また，その通信経路はSSLを用いて暗号化することにより，盗聴や改竄といった脅威から保護することを可能とし，安全な起動制御を実現する．また，認証デバイスが挿入された状態でしか起動を認めない仕組みとすることにより，OSが同時に起動される数を制限することを可能とする．

このシステムを実現するには，従来のBIOSに相当するレイヤーに起動制御レイヤーと呼ばれる制御レイヤーを挿入し，そのレイヤーで起動制御や認証デバイスの制御，暗号化処理などを行う．この仕組みの有効性を確かめるために，本研究では起動制御を実現するために仮想計算機モニタXenを用い，また利用者の認証を行うためのデバイスとしてUSBトークンを用いて実装を行う．

2. 関連研究

2.1 起動処理におけるコンポーネント検証による認証

電源投入直後からOSの起動までの間に処理が行われるコンポーネントの状態値を検証することにより，不正なコンポーネントの挿入や改竄といった脅威からシステムを保護し，OSの起動を安全に行う試みがある．

AEGIS [1]は，拡張ボードに状態値を格納しておき，起動時にコンポーネントの状態値を比較することでシステムの安全性を検証する．sAEGIS [2]はこのAEGISをさらに拡張したもので，ブートローダとしてgrubをサポートしたことにより起動できるOSの種類が増え，また状態値の保存にスマートカードを採用したことにより，より高い安全性を実現している．

Intel TXT [6]は，こういった仕組みをCPUやTPM [5]を用いて実現する実装である．

これらの研究は，OS起動までの処理を検証することで安全性を保証しようという仕組みである．それに対し，本研究のシステムは，OS起動前よりもOS起動後の安全性に主眼を置いており，システムが起動したあとも安全性の保証を行う処理が継続して動作する．

2.2 ハードウェアの固有情報による実行制御

システムのハードウェアが持っている固有情報を用いて認証を行うことにより，特定のハードウェアでのみ動作するアプリケーションという仕組みを実現することができる．

XOM [3]はこれをCPUで実現するためのアーキテクチャであり，CPUが固有の鍵(公開鍵-秘密鍵対)を持っている．このCPU上で動作するアプリケーションをあらかじめCPUの公開鍵で暗号化しておくことで，アプリケーションがそのCPU上でのみ動作するということが実現可能となる．また，メモリアクセスにも公開鍵暗号を用いることでより安全にアプリケーションを実行可能としている．

これに対し，本研究のハードウェア固有情報による認証では，起動制御レイヤーにあらかじめ固有な証明書を格納しておき，その情報を用いて認証を行う．そのため，特別なハードウェアは必要なく，現行システムとの親和性が高い．

3. 提案方式

本研究では，認可された利用者とハードウェアの組み合わせでのみ，その人用に特化されたOSを利用できるようなシステムを提案する．このシステムは，次のような目標を満たす．

（1） 認証デバイスを用いることにより，起動を行おうとしている利用者を特定できること

（2） ハードウェアの管理者がそのハードウェア上でOSを起動できる人とそのOSを限定できること

（3） 認証デバイスがハードウェアから抜かれた場合，利用者用のOSが停止すること

3.1 起動制御

提案方式では，OSの起動を制御するために，ハードウェアとOSの間に起動制御レイヤーを配置する．そして，そのレイヤーがUSBトークンやICカードなどの物理的な認証デバイスの抜き取り監視や認証処理を行う(図1)．これらを行う利用者側のハードウェア・ソフトウェアをクライアントシステムと呼ぶ．

提案方式ではクライアントシステムの電源を投入しただけではOSは起動しない．起動制御レイヤーは認証デバイスが挿入されたことを検知すると，利用者とハードウェア情報に応じて適切なOSのイメージを取得し，そのOSを起動する．また，物理的認証デバイスが抜かれたことを検知すると，起動したOSを停止させる．ハードウェアと利用者，利用するOSとの関連およびOSイメージを管理するために認証サーバを用いる．

従来のコンピュータにおいて，ハードウェア上に搭載されているBIOSがOSの起動を行っている．したがって，提案方式では起動制御レイヤーがBIOSに替わって起動制御を行うものであり，従来のコンピュータシステムとの親和性が高い．また，ハードウェアやOSに対する変更点が少ないという利点がある．

3.2 認証処理を含むOSイメージの転送方法

ネットワークにおいて，安全に認証やデータ転送を行うプロトコルとして，HTTPSがある．HTTPSでは，SSLサーバ認証とクライアント認証の両方を同時に用いることにより，サーバはクライアントを，クライアントはサーバを互いに識別する

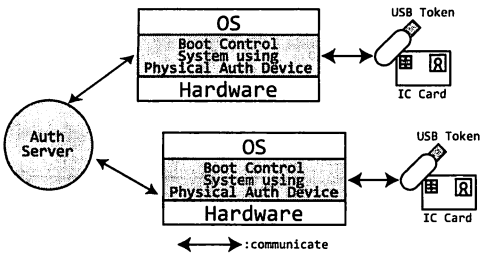

図 1 提案システム概要

Fig. 1 Overview of proposal system

ことができる．このとき，提案方式ではサーバ側の証明書としては通常の HTTPS 通信と同じく認証サーバを識別できる証明書を用いるが，クライアント側の証明書としてはクライアントシステムが実行されているハードウェアに関連づけられたハードウェア用証明書を用いる．これにより，通信経路が暗号化されるほか，サーバ側でクライアントシステムが実行されているハードウェアについての認証を行うことができる．

同様に，利用者用証明書を用いて SSL クライアント認証を行えば，利用者についても認証を行うことができるが，ハードウェアについて認証を行ったセッション上でそれを行うことはできない．そのため，SSL クライアント認証を行うにはもう 1 つ別のセッションが必要になってしまう．そうすると 2 つのセッションの間で利用者・ハードウェアについて結び付けを行う必要がある．その部分がセキュリティリスクになる可能性があるため，本実装ではハードウェアについて認証を行ったセッション上で，HTTP を用いた独自プロトコルにより利用者の認証を行うこととした．

提案するプロトコルにおける認証の流れを次に示す．

（1） クライアントは USB トークンからユーザ名 (利用者用証明書の CN) を取得する．

（2） クライアントは，サーバに対しハードウェア用証明書を用いた HTTPS SSL クライアント認証セッションを確立する．このセッションを用いて，以下の処理を行う．

（a） クライアントはユーザ名を名乗り，認証要求を出す．

（b） サーバはこのセッションを確立したハードウェア名 (ハードウェア用証明書の CN) を取得する．

（c） サーバはユーザ名とハードウェア名からアクセス可否を決定する．

（d） アクセスが許可される組み合わせの場合，ランダムに作成したデータをチャレンジデータとしてサーバに保存する．そのチャレンジデータを HTTP 応答としてクライアントに送信する．許可されない場合，HTTP エラー応答を送信する．

（3） クライアントは受け取ったチャレンジデータを，USB トークンの署名機能を用いて署名し，レスポンスデータを作成する．

（4） クライアントは新たにサーバに対しセッションを確立する．

（a） クライアントはユーザ名とともに，レスポンスデータをサーバに送る．

（b） サーバは受け取ったレスポンスデータを，あらかじめ持っているそのユーザの公開鍵を用いて復号化する．

（c） 復号化したデータと，認証要求時に保存しておいたデータを比較し，一致すれば HTTP 応答としてそのユーザ用の OS イメージを送信する．一致しなければ HTTP エラー応答を送信する．

このような認証プロトコルとすることで，次のように安全性が保証される．

まず，通信には SSL を用いるため，鍵の機密情報が漏洩しない限り通信の安全性が保証される．通信内容が盗聴され，その情報を元に悪用・攻撃される可能性はきわめて低い．また，起動制御レイヤーが持つ認証局証明書を適切に保護することで，中間者攻撃を防ぐことができる．

次に，公開鍵暗号の理論から，認証サーバにおいてチャレンジデータとレスポンスデータの比較に成功した場合，クライアント側には間違いなくその利用者の認証デバイスが挿入されていることが保証される．利用者が持つ USB トークンで署名を行ったデータを正しく復元できるのは，そのトークンに格納された証明書の公開鍵だけである．そのため，仮にチャレンジデータの盗聴に成功したとしても，正しく受理されるレスポンスデータを捏造することは事実上不可能である．また，一度クライアント側からレスポンスデータが送られ，比較に失敗すると，その利用者のチャレンジデータはサーバ上から削除される．そのため，仮にレスポンスデータを盗聴できたとしてもそのレスポンスデータは既に使用できないものになっている．

また，認証デバイスには PIN コードと呼ばれるパスフレーズが設定できる．認証デバイスを用いるにはその認証デバイスと PIN コードが必要となるため，仮に認証デバイスが盗難に遭っても，PIN コードがわからなければ認証を行えない．

3.3 認証サーバ

利用者やハードウェアの情報，利用者用 OS のディスクイメージなどを一括管理するために，提案方式では認証サーバを設置する．認証サーバでは，次のような情報を格納，管理する．

（1） ユーザ名，及び対応する公開鍵情報

（2） OS イメージ名と OS イメージファイルの対応リスト

（3） ハードウェア名，ユーザ名，OS イメージ名の対応リスト

ハードウェアについては，SSL クライアント認証により認証を行う．その正当性の検証は「クライアント証明書に署名を行った CA が信頼できる CA であるかどうか」について行われるため，対応する公開鍵情報は不要である．(1) では，利用を許可するユーザのリストとともに，そのユーザが持つ USB トークンに格納された証明書に対応する公開鍵情報を保持する．これは，利用者の認証の際にクライアント側から送られてくる署名データ (レスポンスデータ) を復号化する際に必要となるためである．(2) は，クライアントに送信するディスクイメージの名前と実際のファイルとの対応を保持する．これにより，ディスクイメージファイルの管理が容易になる．最後に，(3) では，利用を許可するハードウェア名，ユーザ名，OS イメージ名の

組み合わせを保持する．

4. 仮想計算機モニタ Xen による実装

提案方式の起動制御レイヤーは，現在の BIOS を置き換えるものである．よって，その実現には BIOS と同じレイヤーでの開発が必要となる．BIOS と同じレイヤーであり，かつプログラマブルな環境としては EFI [7] があるが，HTTP や SSL などのライブラリが十分ではなく，認証デバイスへのアクセス手段も標準で提供されないため，提案方式を実装するための難易度が非常に高い．

1 台のハードウェア上に複数の仮想計算機 (Virtual Machine:VM) を構築する仮想計算機技術の中で，Type-I VM と呼ばれるものは，BIOS と OS の間で仮想計算機モニタ (Virtual Machine Monitor:VMM) が動作する．VMM は実計算機資源の VM への割り当てや，VM の管理などを行うため，VM 上で動作するゲスト OS にとっては BIOS に相当するレイヤーとみなすことができる．

仮想計算機モニタ Xen [8] では，この VMM 部分 (ホスト OS) が Linux として動作するため，VMM のレイヤーで既存の HTTP や SSL などのライブラリやソフトウェア，デバイスドライバなどが利用できる．Xen ではゲスト OS を DomainU と呼ばれる領域で起動し，DomainU は Domain0 と呼ばれる VMM 部分により管理される．利用者用 OS を DomainU として起動することにより，Domain0 で起動制御を行うことができるようになる．これにより，提案方式の有効性を迅速かつ容易に確認することができる．

本実装では認証デバイスとして飛天ジャパン社の USB トークン ePass2000 [9] を用いる．Domain0 の OS としては Fedora7 を，VMM としては Xen 3.1 を用いる．

認証サーバについては，HTTPS 上の独自プロトコルを実装する必要があるため，Web サーバ上の CGI(Common Gateway Interface) プログラムとして実現する．認証サーバの OS としては Linux 2.6.18(Fedora7) を用い，HTTP デーモンとしては Apache 2.2.6 を，HTTP 上の認証プロトコル処理の実装には PHP 5.2.4 を用いる．

4.1 利用者用 OS の制御の流れ

提案システムにおける利用者用 OS の起動の流れを図 2 に示す．電源が投入されると，本システムは，利用者に対してまず USB トークンの挿入を要求する．USB トークンが挿入されたことを検知すると，本システムは USB トークンから取得したユーザ名と，あらかじめ Domain0 内に保存されているハードウェア用証明書を用いて，認証を伴う利用者用 OS イメージの取得処理を行う．

認証サーバからの利用者用 OS イメージの取得が完了すると，そのイメージを仮想計算機上で起動し，利用者がその OS を利用できるようにする．また，同時にホスト OS 上で USB トークンの抜き取りを監視するデーモンを実行する．

利用者用 OS の稼働中に USB トークンが抜かれた場合，監視デーモンがそれを検知し VMM のシャットダウン命令を利用して利用者用 OS を停止させる (図 3)．これにより，USB トークンが挿さっていない場合には利用者用 OS は実行できないため，利用者用 OS が複数の環境で同時に実行されることはなく，ユニーク性が保証される．

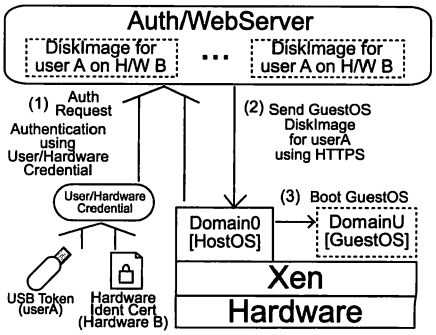

図 2 システム起動の流れ

Fig. 2 System booting flow

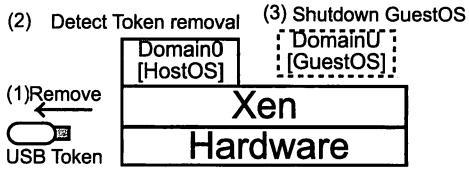

図 3 USB トークンが抜かれた場合

Fig. 3 Process flow on USB token has been removed

4.2 クライアントシステムの実装

本システムでは，クライアントシステムのハードウェアを識別するためにそれぞれのハードウェアに対してユニークな証明書をあらかじめ発行しておく．このハードウェア用証明書は，クライアントシステムに配置する起動制御レイヤー内に同梱する．その形式としては，SSL クライアント認証で使用することができる X.509 証明書を用いる．また，証明書・秘密鍵ともに PEM 形式のファイルとして利用する．ハードウェア用証明書の格納については，将来 TPM を用いることにより，より安全に実現可能となる．

同時に起動されている OS の数を管理するために，起動制御レイヤーでは USB トークンの監視処理を行う．この処理は次のような流れで動作する．まず，異なるユーザの USB トークンが挿された場合にそのことを検出できるように，利用者用 OS を起動する前にユーザ名をクライアントシステム上に記録しておく．その後，ホスト OS 上で定期的に USB トークンにアクセスし，ユーザ名が記録されているものと一致するかを確認する (Busy Wait 方式)．USB トークンにアクセスできない場合や，取得したユーザ名が記録されているものと異なる場合，利用者用 OS を立ち上げたユーザの USB トークンが抜かれたとして判定する．

USB トークンが抜かれたことを検出した場合，まず DomainU をシャットダウンさせるコマンドを実行する．実際には，次のコマンドを実行している．ここで，`domain` は DomainU の名

前である．

```
# xm shutdown domain
```

コマンドを実行すると DomainU にシャットダウン命令が送られ，このコマンドを受け取った DomainU はシャットダウンを開始する．

上記のコマンドを実行した後も定期的な確認を行い，DomainU のシャットダウンが確認できない場合には DomainU の強制終了コマンドを実行する．このコマンドは DomainU の状態に関係なく DomainU を強制終了させる．これは実際の PC における電源断に限りなく近い．実際には，次のコマンドを実行している．同様に，`domain` は DomainU の名前である．

```
# xm destroy domain
```

以上のような方法で利用者用 OS を停止させるため，一度 USB トークンが抜かれたことが検出された場合，USB トークンを挿しなおしても，クライアントシステムはシャットダウンする．

4.3 認証サーバの実装

認証サーバは，認証リクエスト処理とレスポンス処理からなる．認証リクエスト処理は，リクエストを受け，アクセス制御のチェックとチャレンジデータの作成，保管を行う処理である．レスポンス処理は，送られてきたレスポンスデータをもとに，検証とディスクイメージの送信を行う処理である．いずれの処理の実装にも PHP を用いた．

認証リクエスト処理を行うスクリプトは，パラメータとしてユーザ名を受け取り，環境変数を経由してハードウェア用証明書の CN をハードウェア名として取得する．ユーザ名，ハードウェア名の組を用いてアクセス制御のチェックを行う．起動が許可される組み合わせであれば，48 バイトのランダムなデータを作成し，チャレンジデータとする．ここで，48 バイトというのは使用する USB トークンで署名を行える最大のデータ長である．

起動が許可されない組み合わせの場合，HTTP 応答として，ステータスコード 404 Not Found を返す．この場合，チャレンジデータはサーバ上に保存されない．

クライアントから送られてきたレスポンスデータに対する処理を行うスクリプトは，パラメータとしてユーザ名とレスポンスデータを受け取る．

レスポンス処理では，まず対応するチャレンジデータが保存されているかを確認する．認証が正しい手順で行われており，かつ起動が許可される組み合わせの場合にはサーバ上にチャレンジデータが保存されているはずである．もし存在しない場合には，クライアントに対し HTTP 応答として，ステータスコード 404 Not Found を返す．

受け取ったレスポンスデータは，あらかじめサーバ上に保存されているユーザの公開鍵を用いて復号化される．復号化して得られたデータは，サーバ上に保存されたチャレンジデータと比較される．比較の結果，チャレンジデータと一致した場合には認証成功とし，そのユーザに結び付けられたディスクイメージを HTTP 応答として送信する．一致しなかった場合には認証失敗とし，404 Not Found を返す．どちらの場合でも，ユーザのチャレンジデータはサーバ上から削除される．なお，ユーザとディスクイメージのマッピングについては，特定のディレクトリにあるユーザ名.img を使うといったようにして行っている．万が一，ユーザに対応したディスクイメージが存在しない場合には，認証失敗として扱う．

5. 評　　価

本研究で実装した提案方式の評価を行うため，起動所要時間の計測を行った．

5.1 ハードウェア構成

実験には，次のようなハードウェア構成の認証サーバとクライアントを用いた．それぞれのハードウェアは，1000BASE-T の LAN で接続されているとする．(図 4)

- 認証サーバ
  - CPU: Intel Core2Quad Q6600 2.4GHz
  - Memory: DDR2 SDRAM 4096MB
  - HDD: SerialATA 250GB
- クライアント
  - CPU: Intel PentiumD 820 2.8GHz
  - Memory: DDR2 SDRAM 1024MB
  - HDD: SerialATA 160GB

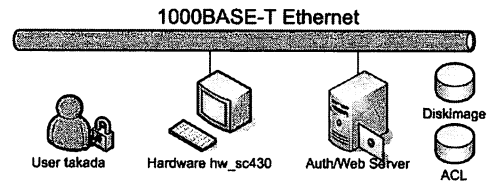

図 4　実験環境概略図
Fig. 4　Experimentation Environment

5.2 実験方法

本システムの起動処理にかかる時間を計測する．クライアントシステムの電源を投入してから，利用者が利用者用 OS にアクセスできるようになるまでの時間がユーザにとっての起動処理所要時間である．しかし，利用者用 OS の種類や構成によりその時間は様々である．そこで，本システムの評価として，認証にかかる時間と転送にかかる時間を計測する．前者は起動制御レイヤーに処理が移ってから，ディスクイメージの受信が始まる前までの時間で，これを認証フェーズ所要時間と呼ぶ．後者はディスクイメージの受信開始から受信完了までの時間で，これを転送フェーズ所要時間と呼ぶ．

転送するディスクイメージのサイズとして，540MB, 1080MB, 1540MB の 3 種類を用意し，それぞれについて 3 回計測を行った．

5.3 実験結果

ディスクイメージのサイズを 540MB, 1080MB, 1540MB としたときの認証フェーズ，転送フェーズの所要時間を，それぞれ 3 回計測した．その実験結果を表にまとめたものを表 1 に，それをグラフに表したものを図 5 に示す．

結果より，すべてのイメージサイズで認証フェーズの所要時

間は約 11 秒と一定であるが，転送フェーズの所要時間はディスクイメージサイズにほぼ比例することが読み取れる．

表 1　所要時間計測結果
Table 1　Average process time

Image Size[MB]	Avg. Auth[s]	Avg. Trans[s]	Avg. Total[s]
540	11	18	29
1080	11	36	47
1540	12	52	64

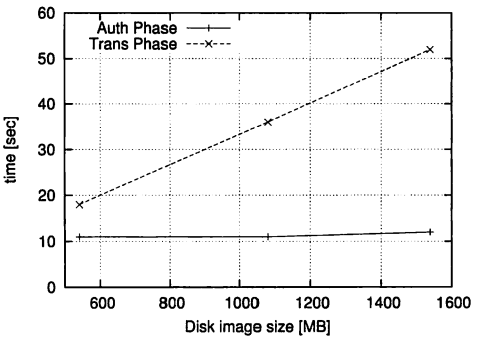

図 5　平均所要時間計測結果
Fig. 5　Average process time graph

5.4 考　　察

5.4.1 認証フェーズ所要時間

実験結果では，認証フェーズにおける平均所要時間は約 11 秒であった．今回使用した USB トークンでは，ユーザ名の取得に約 4 秒かかり，チャレンジデータの署名に約 7 秒かかった．両者を合わせると約 11 秒となり，認証フェーズにおける平均所要時間の大半が USB トークンアクセスにかかる時間であり，アクセス制御処理の時間は無視できる．

今回の実装では，ACL はプレーンテキストに直接記述し，また参照時には先頭から順次走査を行うものであった．この方法では，ユーザやハードウェアの数が増えると記述量が爆発的に増加し，それに伴って参照にかかる時間も増加してしまう．

参照にかかる時間を軽減する方法としては，ACL の管理に RDBMS を使うという方法がある．これにより，大規模な ACL に対しても高速にアクセスすることができる．また，データベースサーバを別に用意することでスケーラビリティを確保することもできる．

5.4.2 転送フェーズ所要時間

起動所要時間について，転送フェーズの所要時間がディスクイメージサイズにほぼ比例することが分かった．転送速度はどの場合においてもほぼ 30MBps で，scp を使って同じディスクイメージを同じ経路で転送した場合の転送速度とほぼ等しい．

6. ま と め

本論文では，より柔軟で安全な OS の起動制御を実現するための仕組みである，起動制御レイヤーについて述べた．この起動制御レイヤーは，利用者とハードウェアについての認証を行い，また認証サーバとの通信，利用者用 OS の起動制御を行う．

起動制御レイヤーの有効性を確認するため，仮想計算機モニタ Xen と USB トークンを用いてシステムを構築し，その起動にかかる所要時間を計測した．その結果，PXE のような従来からある方式に比べ大きなオーバーヘッドもなく，安全かつ柔軟な OS の起動制御を行えることがわかった．

今後の課題としては，利用者への GUI の提供がある．現在の実装では，利用者用 OS のインターフェースとして CUI のみ提供されている．利用者に対し GUI を提供するためには，どこかで X サーバを立ち上げる必要がある．利用者用 OS で X サーバを立ち上げようとしても，仮想計算機に割り当てられる端末の関係で実現できず，起動制御レイヤー (Domain0) 側で X サーバを立ち上げた場合，いかにして Domain0 の安全性を確保するかという問題がある．

Intel VT や AMD-V のような，ハードウェアベースの仮想計算機支援技術を用いることで，Windows のような，OS のソースコードに手を加えることができない OS も Xen 上で起動することができる．利用者に対し GUI を安全に提供できるようになれば，本システムで Windows などを稼働させることが可能となり，より幅広い使い道が期待できる．

文　　献

[1] W.A. Arbaugh, D.J. Farber, and J.M. Smith, "A secure and reliable bootstrap architecture," In *Proceedings 1997 IEEE Symposium on Security and Privacy*, pp. 65-71, May 1997.

[2] N. Itoi, W.A. Arbaugh, J. McHugh, and W.L. Fithen, "Personal secure booting," In *Proceedings of the Sixth Australian Conference on Information Security and Privacy*, pp. 130-144, July 2001.

[3] D. Lie, C.A. Thekkath, and M. Horowitz, "Implementing an untrusted operating system on trusted hardware," In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 178-192, December 2003.

[4] "Preboot Execution Environment(PXE) Specification Version 2.1," pp. 1-103, September 1999. http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf (参照 2008-04-08)

[5] Sundeep Bajikar, "Trusted Platform Module(TPM) based Security on Notebook PCs-White Paper," pp. 1-20, 2002. http://www.intel.com/design/mobile/platform/downloads/Trusted_Platform_Module_White_Paper.pdf (参照 2008-01-10)

[6] "Intel Trusted Execution Technology Preliminary Architecture Specification," pp. 1-102, 2007. http://download.intel.com/technology/security/downloads/31516804.pdf (参照 2008-01-29)

[7] United EFI Inc, "Unified Extensible Firmware Interface Specification Version 2.1," pp. 1-1706, 2007. http://www.uefi.org/specs/ (参照 2008-01-22)

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," In *ACM Symposium on Operating Systems Principles*, pp. 164-177, 2003.

[9] 飛天ジャパン株式会社 - USB トークン・ドングル, http://www.ftsafe.co.jp/products/epass2000.htm (参照 2007-12-26)

[10] 高田真吾, 佐藤聡, 新城靖, 中井央, 板野肯三, "USB トークン認証を用いた OS の安全な起動制御," 情報処理学会 第 70 回全国大会, 分冊 3, no.2Y-7, pp.99-100, March 2008.