無線経由のプロファイル配信と構成

2010-08-03

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
U.S.A.

アップルジャパン株式会社
〒163-1450 東京都新宿区西新宿
3 丁目20 番2 号
東京オペラシティタワー
http://www.apple.com/jp/

目次

図、リスト

はじめに

構成プロファイルは、構成情報をiOSベースのデバイスに配信するためのXMLファイルです。多数のデバイスを構成する必要がある場合や、独自の電子メール設定、ネットワーク設定、または証明書を多数のデバイスに提供する必要がある場合は、構成プロファイルを利用すると簡単にそれを実行できます。

iOSの構成プロファイルには、以下に示すような、指定可能な多くの設定が含まれています。

パスコードポリシー
デバイス機能の制限（カメラを無効にするなど）
Wi-Fi設定
VPN設定
電子メールサーバ設定
Exchange設定
LDAPディレクトリサービス設定
CalDAVカレンダーサービス設定
Webクリップ
認証情報と鍵
高度な携帯電話ネットワーク設定

注： 構成プロファイルはプロパティリスト形式になっており、Base64エンコーディングされたデータ値が格納されています。`.plist`形式は、任意のXMLライブラリで読み書き可能です。

これらのプロファイルの内容の詳細については、『iOS-Based Device Configuration Overview』または『*iPhone Configuration Profile Reference*』を参照してください。

構成プロファイルを配備するには、以下の4つの方法があります。

- デバイスを物理的に接続する
- 電子メールメッセージで送信する
- Webページ上に置く
- この文書で説明するようにOTA（over-the-air：無線）による構成を使用する

iOSは、暗号化されたプロファイルと暗号化なしのプロファイルの両方をサポートします。暗号化されたプロファイルは、データの完全性を保証し、機密のポリシー情報を盗聴から保護します。暗号化された構成プロファイルは、デバイスのID証明書に関連付けられた公開鍵で署名されています。この公開鍵は、次のいずれかの方法によって取得できます。1つは、iPhone構成ユーティリティ(iPCU)が実行されているコンピュータにUSB接続する方法、もう1つは、OTAによる登録を使用する方法です。

配備の前に各デバイスを1つのコンピュータに接続するほうが現実的な場合は、iPhone構成ユーティリティ(iPCU)を使用して、各デバイスに固有のプロファイルを暗号化できます。後から、電子メールやWebページを介して、更新されたプロファイルを安全に配布できます（暗号化されたプロファイルを必要としない場合は、デバイスを接続せずにiPCUを使用できます）。

手動による登録のほうがよい場合は、『*iPhone Configuration Utility*』、および『*Enterprise Deployment*』サブカテゴリ内のその他の文書をお読みください。

こうした方法は、大規模に配備を行う企業内利用のためにデバイスを構成する簡単な手段となりますが、配備プロセスを自動化したい場合もあります。

iOSのOTAによる登録と構成は、企業内で安全にデバイスを設定するための自動的な手段を提供します。このプロセスを利用すると、信頼できるユーザのみが会社のサービスにアクセスすること、またそうしたユーザのデバイスが規定のポリシーに従って適切に構成されることが保証されます。構成プロファイルは暗号化とロックの両方が可能なため、他人がその設定を削除したり、変更したり、あるいは共有したりできません。

地理的に分散している企業にとってもっと重要なことは、OTAプロファイルサービスを利用すると、iOSベースのデバイスをiPhone構成ユーティリティのホストに物理的に接続しなくてもデバイスを登録できる点です。

この文書で説明するプロファイルサービスが、その場で構成を作成し、デバイスはその構成をダウンロードします。デバイスは登録URLを記憶しています。そうすることで、登録以降に構成の有効期限が切れた場合や、VPN接続に失敗した場合でも、サーバから構成を更新できるようになります。

この文書では、OTAによる登録プロセスについて説明します。このプロセスでは、管理者は、電子メールやSMS通知を通じてURLを知らせることによって、登録プロセスの開始方法をユーザに指示できます。ユーザがプロファイルのインストールに同意した場合は、1回のセッションでそのユーザのデバイスが自動的に登録されて構成されます。

この書類の構成

この文書では、暗号化されたカスタムプロファイルをiOSベースのデバイスにOTAで配信できるようにサーバをセットアップするプロセスについても説明します。

- 「OTAによるプロファイル配信の概念」（11 ページ）では、OTAによる登録とプロファイル配信に関わる用語と基本的なセキュリティ概念について説明します。
- 「OTAによる登録と構成のためのプロファイルサーバの作成」（17 ページ）では、プロファイルサーバの参考実装について、デバイスの認証と登録からプロファイル配信まで、部分ごとに実行順序に沿って説明します。
- 「構成プロファイルの例」（29 ページ）では、サンプルプロファイルと、プロファイルを生成するためのコードを紹介します。

この文書は、Rubyプログラミング、XML、プロパティリスト、iPhone構成ユーティリティ、およびOpenSSLの基礎知識を前提としています。

関連項目

詳細については、以下の各ページを参照してください。

Cisco：Digital Certificates PKI for IPSec VPNs (PDF)
ウィキペディア：公開鍵基盤
IETF SCEP protocol specification

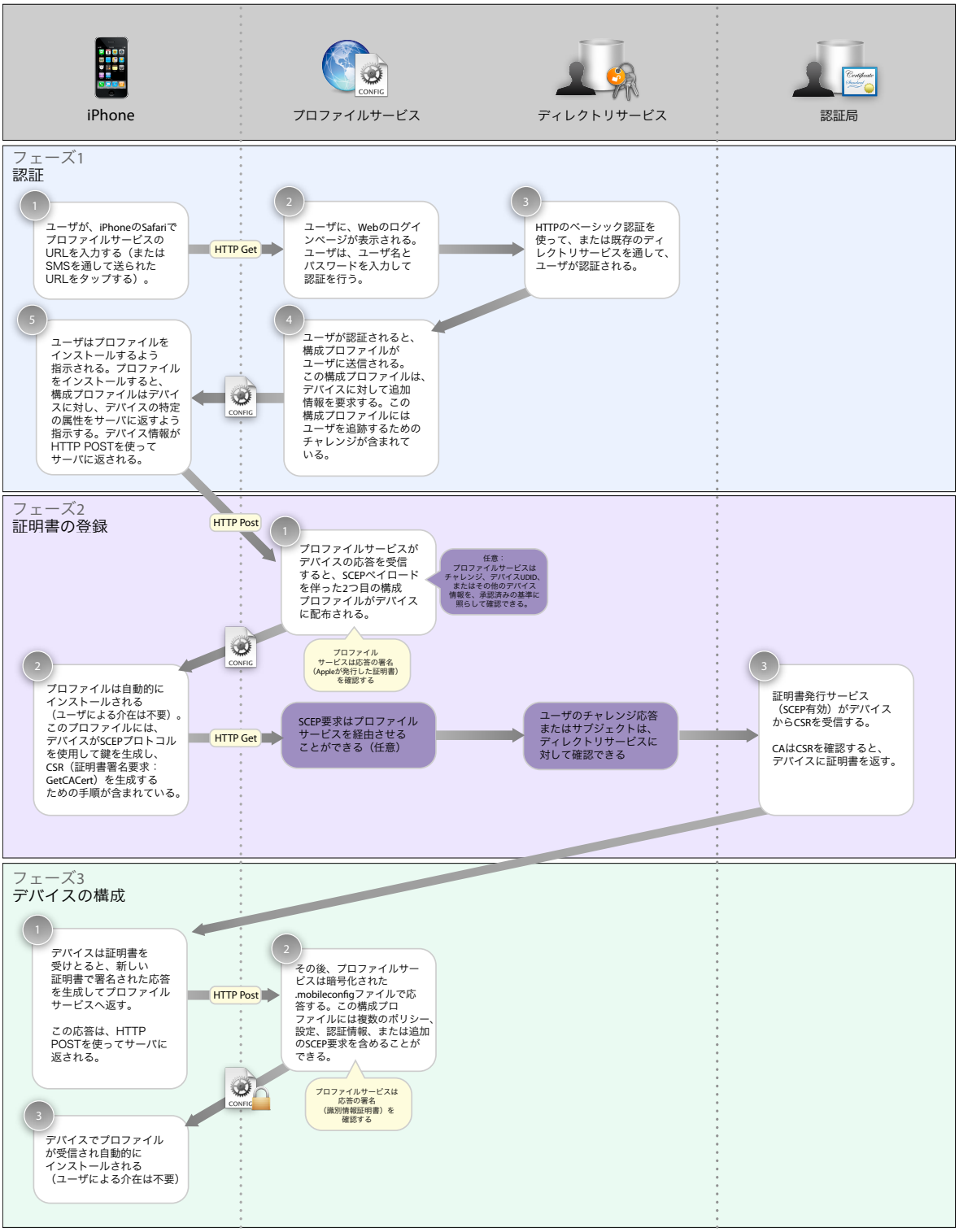
企業内環境でのiOSベースのデバイスに関する補足情報およびリソースは、http://www.apple.com/jp/iphone/business/から取得できます。ここには、『*iPhone Configuration Profile Reference*』も含まれています。この文書の付録には、独自のツールを作成しようと考えているデベロッパのために、`.mobileconfig`ファイルの形式が明記されています。

OTAによるプロファイル配信の概念

OTAによる登録と構成のプロセスは、認証、登録、デバイスの構成の3つのフェーズに分けられます。以降のセクションでは、これらのフェーズについて説明します。

このプロセスを図 1-1に示します。

図 1-1 デバイス登録プロセス



フェーズ1：認証

認証フェーズには2つの目的があります。1つは、受信した登録要求が、承認済みのユーザからのものであることを保証することです。もう1つは、証明書登録プロセスで使用するために、ユーザのデバイスに関する情報を取得することです。

デバイスを登録するときに、サーバは、ユーザまたはデバイス、あるいはその両方による認証を要求できます。

ユーザ認証は、ユーザが登録URLにアクセスしたときに要求できます。ユーザを認証するには、任意のWeb認証方式を使用できます。HTTPの一部（たとえば、ベーシック認証やNTML認証）でも、CGIスクリプトによって実装された独立の認証方式でもかまいません。ハイブリッド方式（ダイジェスト認証と、CGIによって処理する認証済みユーザリストの組み合わせなど）を使用することもできます。

必要であれば、デバイスを承認済みデバイスリストと照合することもできます。

認証フェーズの手順は以下のとおりです。

1. ユーザがルートURL (`/`)にアクセスすると、welcomeメッセージが返されます。このメッセージは、「welcomeページ(/) URLハンドラ」（19 ページ）で説明するハンドラによって提供されます。

2. ユーザは認証局のURL (`/CA`)にアクセスし、ルート証明書を取得します（必要な場合）。サーバ上では、このURLは、「ルート証明書(/CA) URLハンドラ」（20 ページ）で説明するコードによって処理されます。これは、自己署名ルート認証局証明書を使用する場合にのみ必要になります。

3. ユーザは、登録用URL (`/enroll`)にアクセスして、登録プロセスを開始します。このステップでは、ユーザは、HTTPのベーシック認証（この例の場合）または既存のディレクトリサービスを使用して、自身の認証を行うように要求されます。

4. サーバの登録ハンドラ（「登録(/enroll) URLハンドラ」（21 ページ）を参照）は、ユーザがデバイスの登録を許可されているかどうかを判断します。デバイスの登録が許可されている場合、サーバは、そのデバイスにプロファイルサービスペイロードを送信します（リスト 2-5（22 ページ））。

   このプロファイルサービスペイロード(`.mobileconfig`)には、デバイスが次のステップで提供しなければならないデバイス固有の追加情報についての要求も含まれています。

   このペイロードには、サーバが要求と要求元ユーザを関連付けられるように、`Challenge`トークンを含めることもできます。これを利用して、必要であれば、ユーザごとに構成プロセスをカスタマイズできます。この値は、検証可能でなければなりませんが、簡単に推測できるものであってはいけません。たとえば、ランダムな値をデータベースに保存して、それをユーザのログイン情報に関連付けることもできます。この機能の詳細は、サイトごとに固有です。

   このサービスが要求できるデバイス属性は、iOSのバージョン、Wi-Fiデバイス ID（MACアドレス）、製品のタイプ（たとえば、iPhone 3GSは`iPhone2,1`を返す）、携帯端末ID (IMEI)、およびSIMカードID (ICCID)です。

   プロファイルサービスペイロードの例については、『*Enterprise Deployment Guide*』の「Sample Phase 1 Server Response」を参照してください。

フェーズ2：証明書の登録（X.509 IDとSCEP）

2番目のフェーズ（登録）では、デバイスが認証局に問い合わせて、署名付きのX.509 ID証明書を取得します。この証明書は暗号化のために使われます。

IDを取得するために、デバイスは、まず非対称鍵ペアを生成し、それをキーチェーンに格納します。このキーチェーンの機密情報は、生成を行ったデバイスだけが読めます。

次に、デバイスはその公開鍵を認証局(CA)に送信します。認証局は署名付きのX.509証明書を送り返します。この証明書は、デバイス上の非公開鍵との組み合わせによってIDを構成します。

iOSは、このやり取りを可能にするSCEP (Simple Certificate Enrollment Protocol)をサポートしています。SCEPは、非公開の認証局へのネットワーク経由のフロントエンドを提供する通信プロトコルです。SCEPは、多くの認証局でサポートされています。また、SCEPをサポートする認証局の完全なオープンソースソフトウェアの実装もあります。

このフロントエンドサービスは、チャレンジを利用してアクセス制御を行うようにセットアップできます。チャレンジは、実際には証明書の発行を自動化するための認証トークン（ワンタイムパスワード、またはユーザ／デバイス情報を含む署名付きの暗号化されたBLOB）です。

登録フェーズの各ステップは以下のとおりです。

1. ユーザが、フェーズ1で受信したプロファイルのインストールに同意します。
2. デバイスは要求された属性を検索し、チャレンジ応答を追加します（提供されている場合）。次に、デバイスの内蔵ID（Appleが発行した証明書）を使用して、この応答に署名し、HTTP POSTを使用してプロファイル配布サービスにその応答を送り返します。

   注： デバイスが以前登録されたことがあり、単に新しい構成を要求しているだけの場合は、以前フェーズ3で認証局から提供された証明書を使用して、この要求に署名します。

   この例では、デバイスはこの応答を`/profile` URLに送信します。

   このフェーズでのプロファイルの例については、『*Enterprise Deployment Guide*』の「Sample Phase 2 Device Response」を参照してください。
3. サーバのプロファイル要求ハンドラ（「プロファイル要求(/profile) URLハンドラ」（23 ページ）を参照）は、「フェーズ2：証明書の登録（X.509 IDとSCEP）」（14 ページ）で説明したように、SCEPを使用して登録するようデバイスに指示する構成プロファイルを送り返します。サーバはこのプロファイルに署名しなければなりません。

   構成プロファイルの例については、『*Enterprise Deployment Guide*』の「Sample Phase 3 Server Response With SCEP Specifications」を参照してください。
4. SCEPを使用してデバイスが登録されると、有効なID証明書がデバイスにインストールされます。

フェーズ3：デバイスの構成と暗号化されたプロファイル

OTAによるプロファイル配布と構成の3番目のフェーズは、実際のプロファイル配布そのものです。このフェーズでは、サーバは特定のデバイス用にカスタマイズされたプロファイルを送信します。

環境によっては、会社の設定とポリシーを確実に盗聴から保護することが重要になります。この保護を提供するために、iOSでは、1つのデバイスからしか読むことができないように、プロファイルを暗号化できます。

暗号化されたプロファイルは、構成プロファイルペイロードが、デバイスのX.509 IDに関連付けられた公開鍵で暗号化されていること以外は、通常の構成プロファイルとまったく同じです。

悪意のある者によって内容が変更されないようにするために、暗号化された構成プロファイルはこのサービスによって署名されます。暗号化と署名のために、iOSではCMS (Cryptographic Message Syntax)を使用します。これは、S/MIMEでも使われている標準です。ペイロードは、PKCS#7のエンベロープデータを使用して暗号化されます。プロファイルは、PKCS#7の署名データを使用して署名されます。

デバイス構成フェーズの各ステップは以下のとおりです。

1. デバイスは、最終的なプロファイルを要求するために、`/profile`ハンドラに署名付きの要求を再度送信します（この要求は、前のステップで取得したID証明書を使用して署名されます）。

   このフェーズでのプロファイルの例については、『*Enterprise Deployment Guide*』の「Sample Phase 4 Device Response」を参照してください。

2. サーバのプロファイル要求ハンドラ（「プロファイル要求(/profile)URLハンドラ」（23 ページ）を参照）は、最終的な暗号化されたプロファイルをデバイスに送信します。

登録後について

最終的な暗号化されたプロファイルを受信すると、デバイスはそれをインストールします。プロファイルの有効期限が切れたり、VPN接続に失敗した場合は、自動的に再構成が行われます。

構成プロファイルによって行われた設定は、デバイス上で変更することはできません。これらの設定を変更するには、更新されたプロファイルをインストールしなければなりません。

構成プロファイルの更新は、自動的にはユーザにプッシュされません。プロファイルの有効期限が切れる前に、ほかのプロファイルの更新を行う必要がある場合は、それらを手動で配布するか、ユーザに再登録を要求しなければなりません（デバイスは、VPN認証に失敗したときにも新規プロファイルを要求します）。

更新されたプロファイルは、電子メールの添付ファイルやWebページを介してユーザに配布できます。プロファイルを更新するには、以下の条件が満たされていなければなりません。

- プロファイルIDが一致していなければなりません。

  IDの詳細については、『*Enterprise Deployment Guide*』の「General Settings」を参照してください。

- プロファイルが署名されていた場合は、置き換え用のプロファイルも同じ発行者によって署名されていなければなりません。

プロファイルの作成時に指定した一般設定ペイロードによっては、ユーザがそのプロファイルを削除できる場合があります。

- プロファイルを削除するためにパスワードが必要な場合、ユーザが「削除」をタップするとパスワードの入力を求められます。
- プロファイルをユーザが削除できない設定になっている場合、「削除」ボタンは表示されません。

重要： 構成プロファイルを削除すると、そのプロファイルに関連付けられているすべての情報が削除されます。これには、ポリシー、デバイス上に保存されているExchangeアカウントデータ、VPN設定、証明書、メールメッセージ、その他の情報が含まれます。

プロファイルのセキュリティ設定の詳細については、『*Enterprise Deployment Guide*』の「General Settings」を参照してください。

OTAによる登録と構成のためのプロファイルサーバの作成

プロファイルサーバを作成する場合は、以下の手順を実行する必要があります。

1. インフラストラクチャを構成します。これについては、「インフラストラクチャの構成」（17 ページ）で説明します。
2. サーバ用のSSL証明書を取得します。これについては、「SSL証明書の取得」（18 ページ）で説明します。
3. テンプレートとなる構成プロファイルを作成します。これについては、「テンプレートとなる構成プロファイルの作成」（18 ページ）で説明します。
4. サーバのコードを作成します。サーバの部品については、「サーバの起動」（19 ページ）と「プロファイルサービスのハンドラ」（19 ページ）で説明します。
5. 環境に固有の適切な認証方式を追加します。
6. サービスをテストします。

以降の各セクションでは、プロファイル配布サービスのソースコードのさまざまな部分を紹介します。

インフラストラクチャの構成

OTAによる登録と構成を実現するには、認証サービス、ディレクトリサービス、および証明書サービスを統合する必要があります。このプロセスは、標準のWebサービスを使用して配備できますが、事前にいくつかの重要なシステムをセットアップしておく必要があります。

ディレクトリサービス

ユーザ認証には、HTTPのベーシック認証、または既存のディレクトリサービスとの統合認証を使用できます。使用するサービスにかかわらず、登録を要求してきたユーザに対してWebベースの認証方法を提供する必要があります。

証明書サービス

登録のプロセスには、標準のx.509 ID証明書をiOSユーザに配備する必要があります。それには、SCEP (Simple Certificate Enrollment Protocol)を使用してデバイスの証明書を発行するCA（認証局）が必要になります。

Cisco IOSとMicrosoft Server 2003（証明書サービス用のアドオンを含む）は、どちらもSCEPをサポートします。また、Verisign、Entrust、RSAなど、SCEPをサポートするPKIサービス提供業者もたくさんあります。PKI、SCEP、およびそれに関連するトピックへのリンクは、「はじめに」（7 ページ）の「関連項目」（9 ページ）を参照してください。

プロファイルサービス

このプロセスを実現するには、プロファイルサービスを開発する必要があります。これは、プロセス全体にわたってiOSベースのデバイスの接続を管理したり、ユーザ用の構成プロファイルを生成したり、その過程でユーザ認証情報を検証したりするHTTPベースのデーモンです。

プロファイルサービスは、以下の重要な機能を提供する必要があります。

- ユーザがアクセス可能なWebサイトをホストして、HTTPSセッションをサポートします。
- Webベースの認証方式（ベーシック認証、またはディレクトリサービスとの統合認証）を使用して、受信したユーザ要求を認証します。
- プロセスのフェーズに応じて、必要な構成プロファイル（XML形式）を生成します。
- 公開鍵暗号を使用して、構成プロファイルを暗号化して署名します。
- このプロセスの手順全体にわたってユーザを追跡します（タイムスタンプとログを利用）。
- 認証局やディレクトリサービスへの接続を管理します。

SSL証明書の取得

プロファイルサービスをセットアップするための最初のステップは、Webサーバ用のSSL証明書を取得または生成することです。プロファイルサーバをホストしている場合は、各iOSベースデバイスは、このサーバにセキュアな接続ができなければなりません。そのための最も簡単な方法は、iOSによってすでに信用されている公認の認証局からSSL証明書を取得することです。信用できる認証局の網羅的なリストについては、「iOS 3.x：信用できるルート証明書の一覧」を参照してください。

あるいは、独自のルート証明書を生成して、それに自己署名することもできます。ただし、その場合、その証明書を信用するかどうかユーザが判断を求められます。

テンプレートとなる構成プロファイルの作成

プロファイルサービスは、テンプレートとなる構成プロファイルをベースとして使用し、特定のデバイス用にそのプロファイルを変更します。事前にこのテンプレートを作成して、ディスク上のファイルに保存しておく必要があります。iPhone構成ユーティリティは、このようなベースプロファイルを簡単に作成する手段を備えています。

一般設定のほかに、この構成プロファイルには、遵守させたいエンタープライズポリシーも定義しておくべきです。会社が所有する機器の場合は、ユーザがプロファイルをデバイスから削除できないようにするために、ロックされたプロファイルにする必要があります。

これらのプロファイルの詳細については、『iOS-Based Device Configuration Overview』、または『*Enterprise Deployment Guide*』の「Configuration Profile Format」を参照してください。

サーバの起動

SSL証明書を入手したら、プロファイルサービスと、証明書を発行するSCEP対応の認証局をホストするWebサーバを構成する必要があります。

初期化関数`init`は、HTTPサーバの証明書とSSL秘密鍵をロードします。これらの鍵と証明書は、再利用できるように、最後に発行された証明書のシリアル番号と一緒にディスクに保存されます。この関数をリスト 2-1に示します。

リスト 2-1　Webサーバの起動

```
world = WEBrick::HTTPServer.new(
  :Port            => 8443,
  :DocumentRoot    => Dir::pwd + "/htdocs",
  :SSLEnable       => true,
  :SSLVerifyClient => OpenSSL::SSL::VERIFY_NONE,
  :SSLCertificate  => @@ssl_cert,
  :SSLPrivateKey   => @@ssl_key
)
```

この例では、rootとして実行する必要がないように、ポート`8443`でサーバを起動します。`:DocumentRoot`の値には、プロファイルサービスディレクトリ内の空ディレクトリのパスを含めます。

SSLを有効にし、「SSL証明書の取得」（18 ページ）で取得した実際のSSL証明書と鍵を指すように、`SSLCertificate`と`SSLPrivateKey`の値を設定します。

また、クライアントデバイスはまだ検証可能なIDを持っていないため、クライアント証明書の認証は無効にします。

プロファイルサービスのハンドラ

基本のWebサーバが準備できたら、登録と配布プロセスで使用するさまざまなページ用のハンドラを記述する必要があります。

フェーズ1：認証

welcomeページ(/) URLハンドラ

welcomeページは、新規ユーザがサイトのルートレベル(/)にアクセスしたときに表示される最初のページです。このページ用のハンドラをリスト 2-2に示します。

リスト 2-2 / URL用のハンドラ

```
world.mount_proc("/") { |req, res|
    res['Content-Type'] = "text/html"
    res.body = <<WELCOME_MESSAGE

<style>
body { margin:40px 40px;font-family:Helvetica;}
h1 { font-size:80px; }
p { font-size:60px; }
a { text-decoration:none; }
</style>

<h1 >ACME Inc. Profile Service</h1>

<p>If you had to accept the certificate accessing this page, you should
download the <a href="/CA">root certificate</a> and install it so it becomes
trusted.

<p>We are using a self-signed
certificate here, for production it should be issued by a known CA.

<p>After that, go ahead and <a href="/enroll">enroll</a>

WELCOME_MESSAGE

}
```

上のように自己署名した証明書を使用した場合は、ユーザがこのページにアクセスすると、このサーバのSSL証明書を信用するかどうかをSafariが尋ねます。これに同意すると、このページが表示されます。ただし、登録するには、これで十分ではありません。

サイト証明書が自己署名したものかどうかにかかわらず、SCEPサービスを利用した登録プロセスでは、デバイスがカスタム認証局のルート証明書を信用する必要があります。つまり、この認証局のルート証明書をデバイスのトラストアンカーリストに追加する必要があります。それには、証明書に適切なMIMEタイプを提供するURLハンドラを作成しなければなりません。

ルート証明書(/CA) URLハンドラ

welcomeページ内の/CAへのリンクは、カスタム認証局のルート証明書をデバイスのトラストアンカーリストに追加する手段をユーザに提供します。これは、登録プロセスのSCEPの段階で必要になります。

iOS上のSafariは、そのURLからルート証明書をロードすると、この新しいルート証明書をデバイスのトラストアンカーリストに追加するための許可をユーザに求めます（このページには、セキュアな接続を介してのみアクセスするべきです）。

リスト 2-3に示すハンドラは、ルート証明書を送信します。

リスト 2-3 /CA URL用のハンドラ

```
world.mount_proc("/CA") { |req, res|
    res['Content-Type'] = "application/x-x509-ca-cert"
    res.body = @@root_cert.to_der
}
```

ユーザが信用できるWebサーバからHTTPSを介してルート証明書をダウンロードすると、ユーザは、クリックして登録プロセスを継続できます。

登録(/enroll) URLハンドラ

リスト 2-4は、welcomeページ上の`/enroll`リンク用のハンドラです。

リスト 2-4 `/enroll` URL用のハンドラ

```
world.mount_proc("/enroll") { |req, res|
    HTTPAuth.basic_auth(req, res, "realm") {|user, password|
        user == 'apple' && password == 'apple'
    }

    res['Content-Type'] = "application/x-apple-aspen-config"
    configuration = profile_service_payload(req, "signed-auth-token")
    signed_profile = OpenSSL::PKCS7.sign(@@ssl_cert, @@ssl_key,
            configuration, [], OpenSSL::PKCS7::BINARY)
    res.body = signed_profile.to_der

}
```

上のハンドラは、非常に限定的な認証を実行してユーザを識別します。ユーザは、HTTPのベーシック認証によって認証された接続を介して、ユーザ名とパスワードとして`apple`という単語を送信することでログインします。実運用サーバ環境では、代わりに、このコードをディレクトリサービスやその他のアカウントシステムと結び付けるべきです。Rubyアプリケーションをディレクトリサービスに結び付ける方法の詳細については、「Using Open Directory from PHP and Ruby to Manage Mailing Lists for Leopard Server」を参照してください。

このハンドラは、応答のMIMEタイプを`application/x-apple-aspen-config`に設定します。これによって、iOS上のSafariは、この応答を構成プロファイルとして扱います。

`profile_service_payload`関数（「プロファイルサービスペイロード」（21 ページ））は、携帯電話にプロファイルサービスへの登録を指示する特殊な構成を作成します。`"signed-auth-token"`というリテラル文字列は、ユーザの認証情報を検証した認証サービスからの認証トークンに置き換えてください。

最後に、この関数は、`OpenSSL::PKCS7.sign`を呼び出してプロファイルに署名し、署名されたプロファイルをデバイスに送信します。

セキュリティに関する注意： このやり取りは、ユーザ名、パスワード、および署名された認可トークンを保護するために、HTTPSを介して行われます。したがって、SSL証明書による署名を行っても、それ以上のセキュリティは提供されません。ただし、プロファイルサービスが別のSSL証明書を使用し、別のHTTPSサーバ上に存在する場合は、意味があります。

プロファイルサービスペイロード

（登録が許可された後に）デバイスに最初に送信されるペイロードは、プロファイルサービスペイロードです。このペイロードは`/enroll`ハンドラ（リスト 2-4）から`profile_service_payload(req, "signed-auth-token")`を呼び出すことによって送信されます。

プロファイルサービスペイロードの例については、『*Enterprise Deployment Guide*』の「Sample Phase 1 Server Response」を参照してください。

リスト 2-5　profile_service_payload関数

```
def profile_service_payload(request, challenge)
    payload = general_payload()

    payload['PayloadType'] = "Profile Service" # 変更不可
    payload['PayloadIdentifier'] = "com.acme.mobileconfig.profile-service"

    # UIに表示される文字列（カスタマイズ可能）
    payload['PayloadDisplayName'] = "ACME Profile Service"
    payload['PayloadDescription'] = "Install this profile to enroll for secure
 access to ACME Inc."

    payload_content = Hash.new
    payload_content['URL'] = "https://" + service_address(request) + "/profile"
    payload_content['DeviceAttributes'] = [
        "UDID",
        "VERSION"
=begin
        "PRODUCT",              # 例：iPhone1,1、iPod2,1
        "MAC_ADDRESS_EN0",      # WiFiのMACアドレス
        "DEVICE_NAME",          # 与えられたデバイス名"iPhone"

        # 以下の項目はiPhoneでのみ利用可能
        "IMEI",
        "ICCID"
=end
        ];
    if (challenge && !challenge.empty?)
        payload_content['Challenge'] = challenge
    end

    payload['PayloadContent'] = payload_content
    Plist::Emit.dump(payload)
end
```

この関数は、general_payloadを呼び出すところから処理を開始します。これは、バージョンと組織（これらの値は、サーバ上で変化しない）を設定し、プロファイル用のUUIDを提供するテンプレートペイロードを返します。

このペイロードの内容には、（HTTP POSTを使用して）デバイスのIDを送信する対象となるURLと、サーバがデバイスに要求する属性（ソフトウェアのバージョン、IMEIなど）のリストが含まれています。

認可トークン（ユーザ認証を表す）が呼び出し元から渡された場合は（リスト 2-4 （21 ページ）を参照）、そのトークンがChallenge属性として追加されます。

要求に応えて、デバイスは要求された属性のリストおよび対応する属性値を送り返します。サーバが要求の中にChallenge値を入れて送信した場合、デバイスは、要求されたデバイス属性と一緒にこの値も含めます。最後に、iOSベースのデバイスであることを証明するために、デバイスは、自身のデバイス証明書でこのIDに署名をします。この応答は/profile URL用のハンドラに送信されます。

値に関する注意： ペイロードタイプを変更してはいけません。携帯電話は"Profile Service"というリテラル文字列を期待しています。

ペイロード識別子は、逆DNSスタイルの適切な識別子に変更する必要があります。この識別子は、個々のプロファイルサービスに対して固定でなければなりません。

ペイロードの表示名と説明の値は、これから実行しようとする処理をユーザに説明するために、ユーザインターフェイスに表示されます。

フェーズ2：証明書の登録

プロファイル要求(/profile) URLハンドラ

/profileURL用のハンドラは2回呼び出されます。1回目は、SCEPを使用した登録が許可されていないときに、デバイスの認証要求を送信するため。2回目は、SCEPのステップの後に、最終的なプロファイルをデバイスに配信するためです。

このハンドラ内で、プロファイルサーバはPKCS#7で署名されたデータペイロードをデバイスから受信します。そして、それを展開して検証します。このプロファイルの例については、『*Enterprise Deployment Guide*』の「Sample Phase 2 Device Response」を参照してください。

理解しやすくするために、/profileハンドラをいくつかの部分に分けて説明します。このハンドラの最初の部分をリスト 2-6に示します。

リスト 2-6 /profile URL用のハンドラ 1/7

```
world.mount_proc("/profile") { |req, res|

    # CMS BLOBを検証する。ただし、署名者の証明書はチェックしない
    p7sign = OpenSSL::PKCS7::PKCS7.new(req.body)
    store = OpenSSL::X509::Store.new
    p7sign.verify(nil, store, nil, OpenSSL::PKCS7::NOVERIFY)
    signers = p7sign.signers
```

セキュリティに関する注意： 参考用の実装では、ここで署名者の検証はしません。署名者の検証は、デバイス認証局からルート認証局までの中間証明書で構成されたトラストストアや、プロファイルサービスIDを発行するために使用する階層と照合することによって行う必要があります。

デバイスが、プロファイルサービスIDを発行する階層に属する証明書を使用して要求に署名した場合（つまり、このデバイスが以前に登録されたことがある場合）、1回目のパスに従って実行されます（リスト 2-7を参照）。このパスでは、更新済みの暗号化された構成を発行するか、ここで示す実装のように、デバイスをリダイレクトして再登録します。テストのために、以前プロファイルを取得したことのあるデバイスは再登録しなければなりません。

リスト 2-7 /profile URL用のハンドラ 2/7

```
    # ここでは、署名者が、発行済みの証明書かどうかをチェックする
    #
    if (signers[0].issuer.to_s == @@root_cert.subject.to_s)
        print "Request from cert with serial #{signers[0].serial}"
            " seen previously:
#{@@issued_first_profile.include?(signers[0].serial.to_s)}"
```

```
            " (profiles issued to #{@@issued_first_profile.to_a}) \n"
        if (@@issued_first_profile.include?(signers[0].serial.to_s))
          res.set_redirect(WEBrick::HTTPStatus::MovedPermanently, "/enroll")
            print res
```

この時点までに、完全に登録されたことのあるすべてのクライアントは、再登録のために登録ページにリダイレクトされます。

コードはこのステップを通過した時点で、プロパティリストか、最終的なプロファイルを求める新たな要求のいずれかを受信しています。

リスト 2-8では、暗号化されたプロファイルを生成しています。この部分は、フェーズ3（デバイスの構成）に含まれるため、ここではこれ以上説明はしません。詳細は、「/profileハンドラの詳細」（25 ページ）で説明します。

リスト 2-8 `/profile` URL用のハンドラ 3/7

```
        else
            @@issued_first_profile.add(signers[0].serial.to_s)
            payload = client_cert_configuration_payload(req)
                        # vpn_configuration_payload(req)

            #File.open("payload", "w") { |f| f.write payload }
            encrypted_profile = OpenSSL::PKCS7.encrypt(p7sign.certificates,
                payload, OpenSSL::Cipher::Cipher::new("des-ede3-cbc"),
                OpenSSL::PKCS7::BINARY)
           configuration = configuration_payload(req, encrypted_profile.to_der)
        end
```

リスト 2-9のコードでは、デバイスがデバイスIDを送信するケースを扱います。この部分では、理想的には有効なデバイス証明書によって応答が署名されていることを検証して、その属性を解析します。

リスト 2-9 `/profile` URL用のハンドラ 4/7

```
    else
        #File.open("signeddata", "w") { |f| f.write p7sign.data }
        device_attributes = Plist::parse_xml(p7sign.data)
        #print device_attributes
```

次のコードリスト 2-10は、`=begin`と`=end`によってコメントアウトされています。このコードは、プロファイルの発行を（一意のデバイスID、つまりUDIDによって）1つのデバイスに限定して、その`Challenge`が、以前発行された`Challenge`値と同じであることを検証する方法を示しています。

実運用環境では、通常、この部分は、認可トークンを検証するためにディレクトリサービスに問い合わせたり、組織が所有するデバイスの認可済みのUDID値をデータベースに問い合わせたりするサイト固有のコードに置き換えられます。

リスト 2-10 `/profile` URL用のハンドラ 5/7

```
=begin
        # プロファイルの発行を1つのデバイスに限定し、チャレンジ値を検証する
       if device_attributes['UDID'] == "213cee5cd11778bee2cd1cea624bcc0ab813d235"
 &&
            device_attributes['CHALLENGE'] == "signed-auth-token"
        end
```

```
=end
```

次に、リスト 2-11のコードでは、登録プロセスの完了方法を指示するペイロードを取得してデバイスに送信します。この構成の詳細については、encryption_cert_payloadを論じているところで説明します。

リスト 2-11　/profile URL用のハンドラ 6/7

```
        configuration = encryption_cert_payload(req, "")
    end
```

最後に、この関数で送信するものが何もない場合は、例外が発生しHTTP要求は失敗します。それ以外の場合、この関数は、送信するプロファイルに署名して、それを返します。これらのコードをリスト 2-12 （25 ページ）に示します。

リスト 2-12　/profile URL用のハンドラ 7/7

```
    if !configuration || configuration.empty?
        raise "you lose"
    else
    # プロファイルサービス証明書を登録するための構成を送信する、
    # または、このデバイスに固有のプロファイルを送信する
    res['Content-Type'] = "application/x-apple-aspen-config"

        signed_profile = OpenSSL::PKCS7.sign(@@ssl_cert, @@ssl_key,
            configuration, [], OpenSSL::PKCS7::BINARY)
        res.body = signed_profile.to_der
        File.open("profile.der", "w") { |f| f.write signed_profile.to_der }
    end
}
```

この関数が、デバイスに登録方法を指示する構成を送信すると、デバイスはSCEPを使用してデバイスIDを登録します。次に、最終的なプロファイルを取得するために、このハンドラに関連付けられている/profile URLに2回目の要求を送信します。

実際のペイロードについては、「構成プロファイルペイロード」 （26 ページ）と「暗号化証明書ペイロード」 （26ページ）で説明します。構成プロファイルの例については、『*Enterprise Deployment Guide*』の「Sample Phase 3 Server Response With SCEP Specifications」を参照してください。

フェーズ3：デバイスの構成

/profileハンドラの詳細

前述のリスト 2-8 （24 ページ）では、暗号化されたプロファイルの生成プロセスを示しました。そこで示したコードは、実際にはフェーズ3まで実行されません。このため、詳しい説明はしませんでした。ここでは、/profileハンドラのその部分に戻って詳しく説明します。

暗号化されたプロファイルは、以下のような手順で生成されます。

- 1組の構成ペイロードによって、1つの構成が生成されます（これらのペイロードの内容の詳細については、『*Enterprise Deployment Guide*』の「Configuration Profile Format」を参照してください）。

この参考用の実装では、どのデバイスも同じプロファイルを得ます。ただし、必要であれば、Challenge情報を使用して、プロファイルを要求しているユーザを識別し、そのユーザ固有のプロファイルを生成することもできます。

同様に、提供されたデバイス情報を使用して、そのデバイスに固有のプロファイルを生成したり、特定のタイプのデバイスに固有のプロファイルを生成することもできます（たとえば、iOSベースデバイスのモデルごとに、異なるプロファイルを提供するなど）。

- この構成は、元々の要求に署名したデバイスの公開鍵によって暗号化されます。
- 暗号化されたデータBLOBは、構成プロファイルでラップされます。

この暗号化されたBLOBの詳細については、client_cert_configuration_payload（リストA-1（29ページ））とconfiguration_payload（「構成プロファイルペイロード」（26ページ））のところで説明します。

リスト 2-13　/profile URL用のハンドラ 3/7（再掲）

```
else
    @@issued_first_profile.add(signers[0].serial.to_s)
    payload = client_cert_configuration_payload(req)
                # vpn_configuration_payload(req)

    #File.open("payload", "w") { |f| f.write payload }
    encrypted_profile = OpenSSL::PKCS7.encrypt(p7sign.certificates,
        payload, OpenSSL::Cipher::Cipher::new("des-ede3-cbc"),
        OpenSSL::PKCS7::BINARY)
  configuration = configuration_payload(req, encrypted_profile.to_der)
end
```

構成プロファイルペイロード

構成プロファイルペイロード（configuration_payloadによって提供される）は、「プロファイルサービスペイロード」（21 ページ）で説明したプロファイルサービスペイロードに似ています。唯一の違いは、ペイロードの内容です。

このフェーズでのプロファイルの例については、『*Enterprise Deployment Guide*』の「Sample Phase 4 Device Response」を参照してください。

暗号化証明書ペイロード

リスト 2-14では、暗号化証明書ペイロードについて説明しています。このペイロードは、クライアントに登録プロセスを完了する方法を指示します。

リスト 2-14　encryption_cert_payload関数

```
def encryption_cert_payload(request, challenge)
    payload = general_payload()

    payload['PayloadIdentifier'] = "com.acme.encrypted-profile-service"
    payload['PayloadType'] = "Configuration" # 変更不可

    # UIに表示される文字列（カスタマイズ可能）
```

```
    payload['PayloadDisplayName'] = "Profile Service Enroll"
    payload['PayloadDescription'] = "Enrolls identity for the encrypted profile
 service"

    payload['PayloadContent'] = [scep_cert_payload(request, "Profile Service",
 challenge)];
    Plist::Emit.dump(payload)
end
```

scep_cert_payload関数については、「SCEP証明書ペイロード」（27 ページ）で説明します。

SCEP証明書ペイロード

scep_cert_payload関数という名前のとおり、リスト 2-15に示す関数は、SCEPペイロードを作成します。このペイロードは、証明書を登録するために必要な情報をデバイスに提供します。

リスト 2-15　scep_cert_payload関数

```
def scep_cert_payload(request, purpose, challenge)
    payload = general_payload()

    payload['PayloadIdentifier'] = "com.acme.encryption-cert-request"
    payload['PayloadType'] = "com.apple.security.scep" # 変更不可
```

com.apple.security.scepというペイロードタイプは、SCEPペイロードを表し、そのコンテンツはパラメータを指定します。

```
    # UIに表示される文字列（カスタマイズ可能）
    payload['PayloadDisplayName'] = purpose
    payload['PayloadDescription'] = "Provides device encryption identity"

    payload_content = Hash.new
    payload_content['URL'] = "https://" + service_address(request) + "/scep"
```

真っ先に、SCEPサービスのベースURLがきます。便宜上、このベースURLもサンプルのサービスで処理します。この部分は、IOS (http://scep-server/cgi-bin/pkiclient.exe)の場合と、Windows SCEPサーバ(http://scep-server/certsrv/mscep/mscep.dll)の場合とでは若干の違いがあります。

```
=begin
    # SCEPインスタンス。注：MS SCEPサーバでは必須
    payload_content['Name'] = ""
=end
```

このサービスは、最終的なURLの一部になるName値でパラメータ化されたさまざまな証明書発行サービスを提供できます。Windowsの場合は、この値を設定する必要があります。どんな値でもかまいません。

```
    payload_content['Subject'] = [ [ [ "O", "ACME Inc." ] ],
        [ [ "CN", purpose + " (" + UUIDTools::UUID.random_create().to_s + ")"
] ] ];
    if (!challenge.empty?)
        payload_content['Challenge'] = challenge
    end
```

サブジェクトを利用することで、クライアントは要求されたサブジェクトを指定できます。この例では、サブジェクトはプロファイルサービスによって値が設定されます。クライアントによるサブジェクトの指定を許可したくない場合は、Challengeを使用して、要求者のIDをエンコードすることもできます。

X.509のサブジェクトは複雑な構造になっているため、ここでは、それを完全に指定するために、配列の配列として表しています。キーと値のペアは、それぞれ1つの配列として定義されます。キーが最初の要素で、ID（たとえば、"0.9.2342.19200300.100.1.25"はDC）、または認識可能な略称（CN、C、ST、L、O、OU）のいずれかの値を持つ文字列です。上の例は、"/O=ACME Inc./CN={purpose} ({random UUID})"と表示されるサブジェクトを表します。

```
    payload_content['Keysize'] = 1024
```

次に、いくつかの単純なパラメータがきます。これらには、いくつかの留意点があります。鍵サイズ(Keysize)は、特定のサイズの鍵ペアを生成するようにデバイスに要求します。1024ビットと2048ビットの鍵サイズのみを使用しなければなりません。2048ビットを超える鍵はサポートされていません。一般に、2048ビットの鍵の生成はオーバーヘッドを伴うため、1024ビットの鍵が推奨されています。

```
    payload_content['Key Type'] = "RSA"
```

鍵タイプ(Key Type)は、必ずRSAでなければなりません。この参考用の実装では（実際のSCEPでも）RSA鍵だけがサポートされます。

```
    payload_content['Key Usage'] = 5 # デジタル署名(1) | 鍵による暗号化(4)
```

鍵の使用法(Key Usage)は、鍵の使用目的を指定します。これはビットマスクです。ビット0（値1）はデジタル署名を表し、ビット2は鍵による暗号化を表します。MS SCEPサーバは、署名の発行と暗号化のいずれか一方を行い、両方は行いません。

```
=begin
    payload_content['CAFingerprint'] =
StringIO.new(OpenSSL::Digest::SHA1.new(@@root_cert.to_der).digest)
=end
```

認証局証明書が帯域外で検証されている限り、HTTPの上でSCEPを動作させることができます。この機能は、（上に示すように）現在は無効になっています。それは、iOSが現在この機能をサポートしていないからです。この関数では、登録プロセス中に携帯電話がHTTPSを介してダウンロードするSCEPペイロードに、フィンガープリントを追加することによって、このような操作をサポートします。以下にそれを示します。

```
    payload['PayloadContent'] = payload_content;
    payload
end

            payload = client_cert_configuration_payload(req)
                      # vpn_configuration_payload(req)
```

構成プロファイルの例

この付録は2つのセクションから構成されています。1つ目の「構成プロファイルペイロードのコード例」（29 ページ）では、基本のプロファイルペイロードをプログラムで作成する方法を示します。2つ目の「応答の例」（30 ページ）では、典型的なSCEP登録セッション中にやり取りされるプロパティリストの例を示します。

構成プロファイルペイロードのコード例

リスト A-1に示した構成プロファイルペイロードの例には、ユーザをイントラネットサイトに誘導するためのWebクリップが含まれています。また、保護されている資産にアクセスするために必要なSSLクライアント証明書の登録を、携帯電話に許可するためのペイロードを提供します。

リスト A-1　client_cert_configuration_payload関数

```
def client_cert_configuration_payload(request)

    webclip_payload = general_payload()

    webclip_payload['PayloadIdentifier'] = "com.acme.webclip.intranet"
    webclip_payload['PayloadType'] = "com.apple.webClip.managed" # 変更不可

    # UIに表示される文字列（カスタマイズ可能）
    webclip_payload['PayloadDisplayName'] = "ACME Inc."
   webclip_payload['PayloadDescription'] = "Creates a link to the ACME intranet
 on the home screen"

   # ユーザにWebクリップの削除を許可する
    webclip_payload['IsRemovable'] = true

    # リンク
    webclip_payload['Label'] = "ACME Inc."
   webclip_payload['URL'] = "https://" + service_address(request).split(":")[0]
 # + ":4443/"
```

このWebクリップによって、指定のURLにユーザを誘導するアイコンを作成されます。この例では、ユーザがWebクリップを削除できます。

```
    client_cert_payload = scep_cert_payload(request, "Client Authentication",
"foo");
```

クライアント証明書は、暗号化されたペイロードの復号に使用する証明書と同様に、SCEPペイロードを作成することによって登録されます。現実の実装では、通常、鍵の使用法、ポリシー、およびサブジェクトの別名を指定する補足情報を追加します。これは、サーバ側で、登録済みのIDと、特定のユーザおよびそのユーザの機能を簡単に一致させることができるようにするためです。

```
    Plist::Emit.dump([webclip_payload, client_cert_payload])
```

```
end
```

この関数は、ペイロードの配列をそのままダンプして終了します。呼び出し元は、リスト 2-8 （24 ページ）に示したように、これらを構成プロファイルでラップして署名します。

利用可能なペイロードタイプの詳細については、http://manuals.info.apple.com/en_US/Enterprise_Deployment_Guide.pdfにある『*iOS Enterprise Deployment Guide*』を参照してください。

応答の例

このセクションには、OTAによる登録フェーズと構成フェーズを示すプロファイルの例が含まれています。これらは抜粋なので、読者の要求事項はこれらの例に示したものとは異なるでしょう。構文の詳細については、この付録の前半で示した情報を参照してください。各フェーズの説明については、『*Over-the-Air Profile Delivery and Configuration*』を参照してください。

フェーズ1-サーバ応答の例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>PayloadContent</key>
        <dict>
            <key>URL</key>
            <string>https://profileserver.example.com/iphone</string>
            <key>DeviceAttributes</key>
            <array>
                <string>UDID</string>
                <string>IMEI</string>
                <string>ICCID</string>
                <string>VERSION</string>
                <string>PRODUCT</string>
            </array>
            <key>Challenge</key>

            <string>optional challenge</string>
```

または

```
            <data>base64-encoded</data>

        </dict>
        <key>PayloadOrganization</key>
        <string>Example Inc.</string>
        <key>PayloadDisplayName</key>
        <string>Profile Service</string>
        <key>PayloadVersion</key>
        <integer>1</integer>
        <key>PayloadUUID</key>
        <string>fdb376e5-b5bb-4d8c-829e-e90865f990c9</string>
        <key>PayloadIdentifier</key>
```

```
        <string>com.example.mobileconfig.profile-service</string>
        <key>PayloadDescription</key>
        <string>Enter device into the Example Inc encrypted profile
service</string>
        <key>PayloadType</key>
        <string>Profile Service</string>
    </dict>
</plist>
```

フェーズ2-デバイス応答の例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>UDID</key>
        <string></string>
        <key>VERSION</key>
        <string>7A182</string>
        <key>MAC_ADDRESS_EN0</key>
        <string>00:00:00:00:00:00</string>
        <key>CHALLENGE</key>
```

次のいずれか :

```
        <string>String</string>
```

または

```
        <data>"base64 encoded data"</data>

    </dict>
</plist>
```

フェーズ3-SCEP仕様によるサーバ応答の例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Inc//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>PayloadVersion</key>
        <integer>1</integer>
        <key>PayloadUUID</key>
        <string>Ignored</string>
        <key>PayloadType</key>
        <string>Configuration</string>
        <key>PayloadIdentifier</key>
        <string>Ignored</string>
        <key>PayloadContent</key>
        <array>
            <dict>
                <key>PayloadContent</key>
```

```
                <dict>
                    <key>URL</key>
                    <string>https://scep.example.com/scep</string>
                    <key>Name</key>
                    <string>EnrollmentCAInstance</string>
                    <key>Subject</key>
                    <array>
                        <array>
                            <array>
                                <string>O</string>
                                <string>Example, Inc.</string>
                            </array>
                        </array>
                        <array>
                            <array>
                                <string>CN</string>
                                <string>User Device Cert</string>
                            </array>
                        </array>
                    </array>
                    <key>Challenge</key>
                    <string>...</string>
                    <key>Keysize</key>
                    <integer>1024</integer>
                    <key>Key Type</key>
                    <string>RSA</string>
                    <key>Key Usage</key>
                    <integer>5</integer>
                </dict>
                <key>PayloadDescription</key>
                <string>Provides device encryption identity</string>
                <key>PayloadUUID</key>
                <string>fd8a6b9e-0fed-406f-9571-8ec98722b713</string>
                <key>PayloadType</key>
                <string>com.apple.security.scep</string>
                <key>PayloadDisplayName</key>
                <string>Encryption Identity</string>
                <key>PayloadVersion</key>
                <integer>1</integer>
                <key>PayloadOrganization</key>
                <string>Example, Inc.</string>
                <key>PayloadIdentifier</key>
                <string>com.example.profileservice.scep</string>
            </dict>
        </array>
    </dict>
</plist>
```

フェーズ4-デバイス応答の例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>UDID</key>
```

```
        <string></string>
        <key>VERSION</key>
        <string>7A182</string>
        <key>MAC_ADDRESS_EN0</key>
        <string>00:00:00:00:00:00</string>
    </dict>
</plist>
```

構成プロファイルの例

書類の改訂履歴

この表は「*無線経由のプロファイル配信と構成*」の改訂履歴です。

日付	メモ
2010-08-03	『Enterprise Deployment Guide』の複数のサンプルを各所に追加し、全体を通してiOSの名前を変更しました。
2010-06-04	付属ファイルとして全スクリプトを追加しました。
2010-03-24	オンザフライでプロファイルを生成し、無線でそれらをiPhoneデバイスに配信するサーバのビルド方法を説明した新規文書。

書類の改訂履歴