無線経由のプロファイルの配布と構成

Developer

目次

図、リスト

はじめに

構成プロファイルはXML形式のファイルで、構成情報をiOSベースのデバイスに配布するために使います。多数のデバイスの構成作業を一括して行う、独自の電子メール設定やネットワーク設定をまとめて実施する、多数のデバイスの認証を行う、などの目的に有用です。

iOS構成プロファイルには次のような多数の設定項目があります。

パスコードポリシー

デバイスの機能制限（カメラを無効にするなど）

Wi-Fiの設定

VPNの設定

電子メールサーバの設定

Exchangeの設定

LDAPディレクトリサービスの設定

CalDAVカレンダーサービスの設定

ウェブクリップ

資格情報とその鍵

携帯電話網に関する高度な設定

注意: 構成プロファイルはプロパティリスト形式で、データ値はBase64でエンコードして格納します。`.plist`形式の読み書きはどのようなXMLライブラリでも可能です。

このプロファイルの内容について詳しくは、『*Configuration Profile Reference*』を参照してください。

構成プロファイルの配布方法は4通りあります。

- 物理的にデバイスを接続
- 電子メールで配布
- ウェブページ上で配布
- Over-the-Air構成を利用（この資料で解説する方法）

iOSは、暗号化されたプロファイルと暗号化されていないプロファイルの両方をサポートしています。暗号化されたプロファイルは、データの整合性を保証し、ポリシーに関する機密情報を外部から見られないよう保護します。暗号化を施した構成プロファイルには、デバイスのID証明書に対応する公開鍵を使って署名します。公開鍵は2通りのやり方で入手できます。USB経由で、iPhone Configuration Utility（iPCU）が動作しているコンピュータに接続する方法と、Over-the-Air登録を利用する方法です。

各デバイスを1台のコンピュータに接続して構成プロファイルを配布する方法が現実的に可能であれば、これをデバイスごとに用意し、iPhone Configuration Utility（iPCU）で暗号化することも可能です。その後、プロファイルを更新する際には、電子メールやウェブページを介して安全に配布できます（プロファイルを暗号化する必要がなければ、デバイスに接続せずにiPCUを使っても構いません）。

このような手作業による方法が要件に適うならば、この資料の代わりに、『*Enterprise Deployment*』サブカテゴリに属する、「iPhone Configuration Utility」その他の資料を参照してください。

事業用として大量配備するデバイスを設定する、より簡便な方法もありますが、その場合、手順を自動化するのが望ましいかも知れません。

iOSのOver-the-Airを基盤とする登録/設定機能を利用すれば、企業内で自動的かつ安全にデバイスの設定ができます。情報部門の視点で見ると、信頼できるユーザのみが企業内サービスを利用し、あらかじめ定められたポリシーに適うよう、それぞれのデバイスを適切に設定できることになります。構成プロファイルは暗号化やロックが可能であり、設定の不正消去や改竄、漏洩を防止できます。

多くの地域に分散している企業の場合、iOSベースのデバイスを物理的にiPhone Configuration Utilityホストに接続しなくてもよい、という点で、Over-the-Airによるプロファイル配布サービスはさらに重要になります。

この資料で説明するプロファイルサービスは、要求に応じて動的に構成プロファイルを生成し、それをデバイスがダウンロードする方式です。デバイスは登録URLを保持しているので、以後、構成が期限切れになったり、VPN接続に失敗したりしても、サーバから構成プロファイルを取得、更新できます。

この資料ではOver-the-Airによる登録の手順を解説します。管理者は登録URLを、電子メールやSMSでユーザに伝え、登録手続きをするよう指示します。構成プロファイルのインストールにユーザが同意すると、単一のセッション内で自動的に、デバイスが登録、設定されます。

この書類の構成

この資料では、独自のプロファイルに暗号化を施し、無線接続を介してiOSベースのデバイスに配布するための、サーバを設置する手順を解説します。

- “Over-the-Air（無線接続）を介したプロファイル配布の考え方”（8 ページ）では、無線接続を介した登録やプロファイル配布に関する、用語と基本的なセキュリティ概念について説明します。
- “無線接続を介して登録および設定をおこなうプロファイルサーバの構築”（13 ページ）ではプロファイルサーバの参照実装にもとづき、デバイスの認証から登録、プロファイルの配布まで、時系列に沿って順に解説します。
- “構成プロファイルの例”（29 ページ）ではプロファイルの実例と、これを生成するコードを示します。

この資料は、Rubyプログラミング、XML、プロパティリスト、iPhone Configuration Utility、OpenSSLについて、基本的な知識がある読者を想定しています。

関連項目

詳しくは以下のウェブページを参照してください。

Cisco：Digital Certificates PKI for IPSec VPNs（PDF）

Wikipedia：Public key infrastructure

IETF SCEPプロトコル仕様

iOSベースのデバイスに関するその他の情報や資料（企業向け）として、『*Configuration Profile Reference*』などをhttp://www.apple.com/iphone/business/で公開しています。その付録では、独自ツールの開発者を対象に、`.mobileconfig`ファイルの形式を説明しています。

Over-the-Air（無線接続）を介したプロファイル配布の考え方

無線接続を介した登録と設定の手続きは、認証、登録、デバイス設定という3段階から成ります。以下、それぞれについて解説します。

手順の概要を“ディレクトリサービス”に示します。

図 1-1　デバイスの登録手続き

第1段階：認証

認証には2つの目的があります。ひとつは、登録要求が正当なユーザによるものであることの確認です。もうひとつはデバイスに関する情報の収集で、次の証明書登録手続きで使います。

サーバはデバイス登録の際に、ユーザやデバイス、あるいはその両方に対して認証を求めることがあります。

ユーザの認証は、当該ユーザが登録用のURLにアクセスした時点で実施します。HTTPの一部を成す認証スキーム（ベーシック認証、NTMLなど）、CGIスクリプトで実装した独自の認証スキームなど、何を使って認証しても構いません。さらに、ダイジェスト認証と、認可済みユーザのリストを管理するCGIを併用するなど、混成スキームも使用可能です。

また、必要ならば、認可済みデバイスのリストと照合することも考えられます。

認証の手順は次のようになります。

1. ユーザはルートURL（`/`）にアクセスします。先頭ページである旨のメッセージが現れますが、これを用意するのは“先頭ページ（/）URLのハンドラ”（16 ページ）で説明するハンドラです。
2. ユーザは（必要ならば）認証局のURL（`/CA`）にアクセスし、ルート証明書を取得します。サーバ側でこの処理をおこなうのは、“ルート証明書（/CA）URLのハンドラ”（17 ページ）に示すコードです。ただし、これが必要なのは、自己署名ルート認証局証明書を使う場合のみです。
3. ユーザは登録URL（`/enroll`）にアクセスし、登録手続きを開始します。このときユーザは、HTTPベーシック認証（この例の場合）や既存のディレクトリサービスにより、自分自身を認証するよう求められます。

4. サーバ側の登録ハンドラ（"登録（/enroll）URLのハンドラ"（17 ページ）を参照）は、当該ユーザによるデバイスの登録を許可するかどうか判断します。許可する場合は、「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 （19 ページ）を参照）。

   このペイロード（`.mobileconfig`）には、デバイスに応じた追加情報の要求が入っており、デバイスはこの次の手順で応答することになります。

   ペイロードに`Challenge`（誰何）トークンを入れておいて、要求と元のユーザとを、サーバが対応づけて管理できるようにすることもあります。こうしておけば、ユーザごとに設定手順を変えることも可能です。トークンの値は、検証が可能で、しかも容易に推測できないものでなければなりません。たとえば、乱数を生成し、ユーザのログイン証明書と対応づけてデータベースに格納しておく、という方式が考えられます。この機能の実装詳細はサイトによって異なります。

   サービスが要求できるデバイス属性としては、iOSの版番号、Wi-FiデバイスID（MACアドレス）、製品の型（たとえばiPhone 3GSならば`iPhone2,1`）、電話機の識別番号（IMEI、International Mobile Equipment Identity）、SIMカードの固有番号（ICCID、Integrated Circuit Card ID）などがあります。

   「プロファイルサービス」ペイロードの例が、『*Enterprise Deployment Guide*』の「Sample Phase 1 Server Response」に載っています。

第2段階：証明書の登録（X.509 IDとSCEP）

第2段階では、デバイスが認証局にアクセスして署名済みのX.509 ID証明書を取得し、これを暗号化に使います。

IDを取得するため、デバイスはまず非対称暗号の公開鍵と秘密鍵を生成し、キーチェーンに登録します。このキーチェーンの秘密鍵は、特定のデバイスしか読めません。

デバイスは次に、公開鍵を認証局（CA、Certificate Authority）に送り、署名済みのX.509証明書を得ます。この証明書とデバイス側に保存しておいた秘密鍵の組が、IDになります。

このやり取りのためには、iOSがSCEP（Simple Certificate Enrollment Protocol）に対応している必要があります。SCEPは通信プロトコルのひとつで、非公開の認証局が証明書を配布する、フロントエンドとしての手段を提供します。多くの認証局がSCEPに対応しているほか、SCEPに対応した認証局を実現する、完全オープンソースのソフトウェアも存在します。

フロントエンドサービスを設置することにより、誰何（challenge）トークンを使ったアクセス制御が可能になります。実用上、これは認証トークン（使い切りパスワード、あるいはユーザ/デバイス情報を収容する署名/暗号化済みBLOB）でもあって、これにより証明書の自動発行が可能になります。

登録の手順は次のようになります。

1. ユーザは第1段階で受け取った構成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、要求された属性の値を調べ、誰何（challenge）があった場合はこれに対する応答（response）を追加し、デバイス組み込みのID（Appleが発行した証明書）を使って応答に署名し、HTTP POSTによってプロファイル配布サービスに返送します。

   > **注意:** デバイスが既に登録済みで、新たに構成プロファイルを要求するだけであれば、認証局から以前（第3段階で）取得した証明書を使って要求に署名します。

   この例の場合、デバイスは応答を、`/profile`というURLに向けて送信します。

   この時点でのプロファイルの例が、『*Enterprise Deployment Guide*』の「Sample Phase 2 Device Response」に載っています。

3. サーバ側のプロファイル要求ハンドラ（"プロファイル要求（/profile）URLのハンドラ"（20 ページ）を参照）は、構成プロファイルを送り返して、デバイスに対し、SCEPによって登録するべき旨を伝えます（"第2段階：証明書の登録（X.509 IDとSCEP）"（9 ページ）を参照）。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が、『*Enterprise Deployment Guide*』の「Sample Phase 3 Server Response With SCEP Specifications」に載っています。

4. デバイスはSCEPによって登録をおこない、その結果、有効なID証明書がデバイスにインストールされます。

第3段階：デバイス設定と暗号化済みプロファイル

無線接続を介したプロファイル配布および設定の第3段階は、実際のプロファイル配布です。サーバはこの段階で、個々のデバイスに応じてカスタマイズした構成プロファイルを送信します。

環境によっては、企業の設定や方針を覗き見されないよう、確実に保護することが重要です。iOSはその目的で、ある特定のデバイスしか読み取れないよう、構成プロファイルを暗号化できるようになっています。

通常の構成プロファイルとまったく同じですが、デバイスのX.509 IDに対応する公開鍵を使ってペイロードを暗号化するのです。

さらに、攻撃者が内容を改竄できないよう、サービスが署名を施します。暗号化と署名のために、iOSはCMS（Cryptographic Message Syntax）を利用します。S/MIMEでも使われている標準規格です。ペイロードの暗号化にはPKCS#7のデータ形式「enveloped-data」型を使います。同様に、構成プロファイルの署名には、データ形式「signed-data」型を使います。

デバイス設定の手順は次のとおりです。

1. デバイスは/profileのハンドラに対し、署名を施した要求を再度送って、確定プロファイルを求めます（署名には先の手順で取得したID証明書を使用）。

   この段階でのプロファイル例が、『*Enterprise Deployment Guide*』の「Sample Phase 4 Device Response」に載っています。

2. サーバ側のプロファイル要求ハンドラ（“プロファイル要求（/profile）URLのハンドラ”（20 ページ）を参照）は、確定プロファイルに暗号化を施してデバイスに送信します。

登録後

デバイスは、暗号化を施した確定プロファイルを受け取り、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接続に失敗したりすると、自動的に再設定の処理が始まります。

構成プロファイルにより適用された設定を、デバイス側で変更することはできません。そのためには更新された構成プロファイルをインストールする必要があります。

構成プロファイルを更新しても、ユーザ側に、自動的に送信されるわけではありません。有効期限切れになる前に更新したければ、自動処理に任せず個別に配布するか、ユーザに再登録するよう要請する必要があります（VPNの認証に失敗したときにも、デバイスは新規プロファイルを取得しようと試みます）。

更新したプロファイルの配布には、電子メールの添付書類、またはウェブページを利用します。プロファイルを更新するためには、次の条件を満たしていなければなりません。

- プロファイルのIDが合致していること。

  このIDについて詳しくは、『*Enterprise Deployment Guide*』の「General Settings」を参照してください。

- 元のプロファイルに署名が施されている場合、新しいプロファイルにも同じ者が署名していること。

プロファイル作成時に指定した「General Settings」ペイロードによっては、ユーザがプロファイルを削除できることもあります。

- プロファイルに削除用パスワードの設定があれば、「Remove」をタップするとパスワード入力を求められます。
- 一方、削除不可と指定されている場合、「Remove」ボタンは現れません。

Important: 構成プロファイルを削除すると、これに結びついている情報もすべて消えてしまいます。該当する情報としては、ポリシー、デバイスに保存されているExchangeアカウントデータ、VPN設定、証明書、メールメッセージなどがあります。

プロファイルのセキュリティ設定について詳しくは、『*Enterprise Deployment Guide*』の「General Settings」を参照してください。

無線接続を介して登録および設定をおこなうプロファイルサーバの構築

プロファイルサーバを構築するためには、いくつかの作業が必要です。

1. 通信基盤を設定する。“通信基盤を設定する”（13 ページ）を参照。
2. サーバのSSL証明書を取得する。“SSL証明書を取得する”（14 ページ）を参照。
3. テンプレート構成プロファイルを作成する。“テンプレート構成プロファイルを作成する”（15ページ）を参照。
4. サーバ側のプログラムを実装する。コード例の一部を“サーバを起動する”（15 ページ）および“プロファイルサービスのハンドラ”（16 ページ）に掲載。
5. 動作環境に応じた認証の仕組みを追加する。
6. サービス機能の動作をテストする

以下の各節では、プロファイル配布サービスを実現するソースコードの一部を、例として示します。

通信基盤を設定する

無線接続を介した登録と設定の機能を実装するためには、認証、ディレクトリ、証明書の各サービスを統合する必要があります。いずれも標準的なウェブサービスとして設置できますが、重要なシステムのいくつかは前もって準備しておかなければなりません。

ディレクトリサービス

ユーザ認証には、HTTPのベーシック認証を利用する方法、既存のディレクトリサービスに認証機能を統合する方法が考えられます。いずれにしても、登録を要求してきたユーザの権限を確認する、ウェブベースの認証機構を用意しなければなりません。

証明書サービス

登録手続きの過程で、標準的なX.509 ID証明書をiOSユーザに送信する必要があります。そのためには、SCEP（Simple Certificate Enrollment Protocol）に従って、CA（認証局、Certificate Authority）がデバイス証明書を発行しなければなりません。

Cisco IOSやMicrosoft Server 2003（証明書サービス用のアドオンを組み込み）は、いずれもSCEPに対応しています。ほかにも、Verisign、Entrust、RSAなど、SCEPに対応したPKIサービスが多数あります。PKIやSCEP、および関連するトピックについては、“はじめに”（5 ページ）の“関連項目”（7 ページ）節を参照してください。

プロファイルサービス

この手続きを実装するためには、プロファイルサービスを開発しなければなりません。これはHTTPベースのデーモンで、手続きの期間中を通してiOSベースのデバイス接続を管理し、構成プロファイルを生成してユーザに配布し、付随してユーザ証明書を検証する、といった機能が必要です。

プロファイルサービスが提供するべき、重要な機能を以下に示します。

- HTTPSセッションに対応した、ユーザがアクセス可能なウェブサイトを運用する
- ユーザから送られてくる要求を、ウェブベースの認証機能（ベーシック認証、あるいはディレクトリサービスに統合されたもの）により認証する
- 手続きの段階に応じて、必要な構成プロファイル（XML形式）を生成する
- 公開鍵暗号の手法により、構成プロファイルに署名と暗号化を施す
- 手続きの各段階を通して、ユーザの操作を追跡する（タイムスタンプ、ログなど）
- 認証局やディレクトリサービスとの接続を管理する

SSL証明書を取得する

プロファイルサービスを準備する第1段階として、ウェブサーバのSSL証明書を取得または生成します。プロファイルサーバの運用に当たり、iOSベースの各デバイスが、安全な方法でサーバに接続できる必要があるからです。一番簡単なのは、iOSが信頼している公開CAからSSL証明書を取得する、という方法でしょう。該当する公開CAの一覧が「iOS 3.0: List of Available Trusted Root Certificates」に載っています。

あるいは、独自にルート証明書を生成して自己署名する、という方法もあります。ただし、この証明書を信頼するか、各ユーザが確認を求められることになります。

テンプレート構成プロファイルを作成する

プロファイルサービスは、テンプレート構成プロファイルをデバイスに合わせて修正することにより、構成プロファイルを生成します。したがって、あらかじめこのテンプレートを作成し、ファイルとして保存しておかなければなりません。この作業はiPhone Configuration Utilityを利用すれば簡単です。

この構成プロファイルには、全般的な設定に加え、企業としてユーザに強制するポリシーも定義します。企業が所有する機器の場合、各ユーザがデバイスから削除できないよう、プロファイルをロックするべきでしょう。

プロファイルについて詳しくは、『*Enterprise Deployment Guide*』の「Configuration Profile Format」を参照してください。

サーバを起動する

SSL証明書が手に入ったので、プロファイルサービスを提供するウェブサーバと、SCEPに対応した認証局に、証明書を発行できるよう必要な設定を施します。

初期化関数（`init`）の中で、HTTPサーバの証明書とSSLの秘密鍵を読み込みます。この鍵と証明書は、組にしていつでも再利用できるよう、直近に発行した証明書のシリアル番号とともに、ディスクに保存してあります。この関数の実装例をリスト 2-1に示します。

リスト 2-1　ウェブサーバを起動するコード例

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

この例は、`8443`番ポートで待ち受けるようにサーバを起動するので、rootとして実行する必要はありません。`:DocumentRoot`には、プロファイルサービスディレクトリ以下に空のディレクトリを作り、そのパスを設定します。

さらに、SSLを有効にし、`SSLCertificate`および`SSLPrivateKey`の値として、SSL証明書と秘密鍵（“SSL証明書を取得する”（14 ページ）で取得したもの）をそれぞれ設定します。

さらに、クライアント証明書による認証は無効にしなければなりません。クライアント側デバイスにはまだ、検証可能なIDがないからです。

プロファイルサービスのハンドラ

基本的な設定のウェブサーバを用意した後、登録および配送の処理をおこなうハンドラを記述する必要があります。

第1段階：認証

先頭ページ（/）URLのハンドラ

新規ユーザがサイトにアクセスして最初に目にする、ルートレベル（/）のページです。このページのハンドラをリスト 2-2に示します。

リスト 2-2　先頭ページ（/）URL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

}
```

上掲の自己署名証明書を使っている場合、ユーザがこのページにアクセスした時点で、Safariの画面には、このサーバのSSL証明書を信頼するか否か訊ねる表示が現れます。信頼すると回答すれば、当該ページが現れることになります。しかし登録には、これだけでは不充分です。

サイト証明書が自己署名か否かにかかわらず、SCEPサービスによる登録処理では、独自に設置した認証局のルート証明書もデバイスが信頼する、すなわち、この証明書をデバイスの「信頼するアンカ」リストに追加する必要があります。そのためには、正しいMIME型の証明書を渡すURLのハンドラを実装しなければなりません。

ルート証明書（/CA）URLのハンドラ

先頭ページに/CAへのリンクを設置して、独自に設置した認証局のルート証明書を、ユーザがデバイスの「信頼するアンカ」リストに追加できるようにします。これはSCEPによる登録手続きで必要になります。

iOS側のSafariは、当該URLからルート証明書を取得すると、デバイスの「信頼するアンカ」リストに追加してよいか、ユーザに許可を求めます（このページにはSSLによる接続を強制するべきです）。

リスト 2-3に、ルート証明書を送信するハンドラのコード例を示します。

リスト 2-3　ルート証明書（/CA）URL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
}
```

ユーザは、信頼するウェブサーバ（HTTPS）からルート証明書をダウンロードし、これをクリックして登録手続きを進めます。

登録（/enroll）URLのハンドラ

リスト 2-4に、先頭ページに設置した/enrollへのリンクを処理するハンドラの例を示します。

リスト 2-4　登録（/enroll）URL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

}
```

上掲のハンドラは、ごく簡単な認証処理でユーザを特定します。ユーザは、ユーザ名「apple」とパスワードを、HTTPベーシック認証を経た接続を介して送信します。実際のサーバではこの部分に、ディレクトリサービスその他のアカウント管理システムを組み込んでください。

このハンドラは応答のMIME型として「application/x-apple-aspen-config」を設定しているので、iOS上のSafariは応答を構成プロファイルとして認識できます。

profile_service_payload関数（“「プロファイルサービス」ペイロード”（18 ページ）を参照）は、電話機に対して自分自身をプロファイルサービスに登録するよう指示する、特別な設定を生成します。リテラル文字列"signed-auth-token"は、ユーザの証明書を検証した認証サービスから得た、認可トークンに置き換えてください。

最後にOpenSSL::PKCS7.signでプロファイルに署名を施し、デバイスに送信します。

> **セキュリティに関する注記:** このやり取りにはHTTPSを使って、ユーザ名、パスワード、署名済み認可トークンを保護します。したがって、SSL証明書で署名しても、セキュリティが強化されるわけではありません。もっとも、プロファイルサービスが別のSSL証明書を使っており、別のHTTPSサーバ上で稼働しているならば、これにも意味があります。

「プロファイルサービス」ペイロード

（登録可能である旨が確立した）デバイスに送信される第1のペイロードは、「プロファイルサービス」ペイロードです。/enrollのハンドラから、profile_service_payload(req, "signed-auth-token")を呼び出すことにより送信します（リスト 2-4）.

「プロファイルサービス」ペイロードの例が、『*Enterprise Deployment Guide*』の「Sample Phase 1 Server Response」に載っています。

リスト 2-5　profile_service_payload関数

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

    payload['PayloadType'] = "Profile Service" # do not modify
    payload['PayloadIdentifier'] = "com.acme.mobileconfig.profile-service"

    # strings that show up in UI, customisable
    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",              # e.g. iPhone1,1 or iPod2,1
        "SERIAL",               # The device's serial number
        "MEID",                 # The device's Mobile Equipment Identifier
        "IMEI"
=end
        ];
    if (challenge && !challenge.empty?)
        payload_content['Challenge'] = challenge
    end

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

この関数はまず、general_payloadを呼び出すことにより、版番号と組織（サーバが同じである限り値は変化しない）を設定し、プロファイルのUUIDを通知するテンプレートペイロードを得ます。

ペイロードの内容（payload_content）として、デバイスがIDを（HTTP POSTで）送る送信先URLと、デバイスに対して値を返すよう要求する属性（ソフトウェアの版番号、IMEIなど）の一覧を指定します。

（ユーザ認証を表す）認可トークンが引数として渡された場合（リスト2-4（18ページ）を参照）、このトークンをChallenge属性として追加します。

デバイスは応答として、要求された属性とその値のリストを送り返します。サーバからの要求にChallengeの値があった場合は、要求されたデバイス属性とともに、この値もリストに入れます。最後に、自分自身がiOSベースのデバイスであることを示すため、デバイス証明書を使ってIDに署名を施します。この応答の送り先は、プロファイル要求（/profile）URLのハンドラです。

警告: デバイス証明書を使って署名した「Apple iPhone Device CA」を評価する際、有効日は無視されます。

値に関する注記: ペイロード型（PayloadType）を変更してはなりません。電話機の側は、"Profile Service"というリテラル文字列があると想定しています。

ID（PayloadIdentifier）は、逆DNS形式の適切なIDに変更してください。また、どのプロファイルサービスについても、IDは同じでなければなりません。

表示名（PayloadDisplayName）や説明（PayloadDescription）の値は、次に何が起こるのか説明する文言として画面に表示されます。

第2段階：証明書の登録

プロファイル要求（/profile）URLのハンドラ

プロファイル要求（/profile）URLのハンドラは、2回呼び出されます。1回目はデバイス認証要求を送信するためで、SCEPによる登録が許可される前に起こります。2回目はSCEPの手順後で、確定プロファイルをデバイスに配送するのが目的です。

このハンドラでプロファイルサーバは、PKCS#7で署名したデータペイロードをデバイスから受信し、中身を取り出して検証します。このプロファイルの例が、『*Enterprise Deployment Guide*』の「Sample Phase 2 Device Response」に載っています。

以下、コードを追っていく便宜のため、/profileのハンドラをいくつかに分割して掲載します。第1の部分をリスト2-6に示します。

リスト 2-6　プロファイル要求（`/profile`）URLのハンドラ（1/7）

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

    # verify CMS blob, but don't check signer certificate
    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記:** この参照実装では誰が署名したかを検査していません。実際には、デバイス認証局からルート認証局まで、中間に介在する認証局も含む信頼ストア、およびプロファイルサービスIDを発行するために使う階層と照合して検査するべきです。

デバイスが要求に署名する際、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った（すなわち、このデバイスを登録したことがある）のであれば、第1の分岐を実行します（リスト 2-7を参照）。この場合、更新済みの構成プロファイルを暗号化して発行するか、あるいはここで実装してあるように、もう一度登録するようデバイスに指示します。テストの目的で、プロファイルを取得したことがあるデバイスは、再登録を要することにしてあります。

リスト 2-7　プロファイル要求（`/profile`）URLのハンドラ（2/7）

```
    # this should be checking whether the signer is a cert we issued
    #
    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
```

この時点で、登録したことがあるクライアントは、再登録するよう、登録ページにリダイレクトされています。

もう一方の分岐に制御が渡った場合、プロパティのリスト、または確定プロファイルを求める新しい要求を受け取っていることになります。

リスト2-8で、暗号化を施したプロファイルが生成されます。ここは第3段階（デバイスの設定）に当たるので、コード例のみ示し、詳しくは“プロファイル要求（/profile）URLのハンドラ（再掲）”（24 ページ）で解説します。

リスト 2-8　プロファイル要求（/profile）URL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に示すのは、デバイスが自身のIDを送信した場合の処理です。応答が有効なデバイス証明書を使って署名されているか検証した上で、属性をパースします。

リスト 2-9　プロファイル要求（/profile）URL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
```

続くリスト 2-10は、=beginと=endで囲んでコメントアウトしてあります。特定のデバイスに対してのみプロファイルを発行するよう（UDID（Unique Device ID）を調べて）制限し、さらに、Challengeが以前発行したChallengeの値と一致するかどうか検証する方法を示しています。

実稼働環境ではこれを、サイトの特性に応じた処理コードで置き換えてください。ディレクトリサービスに問い合わせて認可トークンを確認する処理、認可済みUDID値を登録したデータベースに問い合わせて組織が所有するデバイスかどうか確認する処理などが考えられます。

リスト 2-10　プロファイル要求（`/profile`）URLのハンドラ（5/7）

```
=begin
        # Limit issuing of profiles to one device and validate challenge
        if device_attributes['UDID'] == "213cee5cd11778bee2cd1cea624bcc0ab813d235"
 &&
            device_attributes['CHALLENGE'] == "signed-auth-token"
        end
=end
```

続いてリスト2-11では、デバイスに送信するペイロードを取得します。登録手続きの進め方を伝えるペイロードです。詳しくは`encryption_cert_payload`関数の説明を参照してください。

リスト 2-11　プロファイル要求（`/profile`）URLのハンドラ（6/7）

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

最後に、何も送信するべきものがなければ例外を起こし、その結果、HTTP要求は失敗します。そうでなければ、構成プロファイルに署名を施して返します。この部分のコードをリスト 2-12　（23 ページ）に示します。

リスト 2-12　プロファイル要求（`/profile`）URLのハンドラ（7/7）

```
    if !configuration || configuration.empty?
        raise "you lose"
    else
    # we're either sending a configuration to enroll the profile service cert
    # or a profile specifically for this device
    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
}
```

この関数が構成プロファイルを送信して、登録方法をデバイスに伝えると、デバイスはSCEPに従って実際にIDを登録します。次にデバイスは、同じプロファイル要求（/profile）URLにもう一度要求を送信して、確定プロファイルを取得します。

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”（25 ページ）および“「暗号化証明書」ペイロード”（25 ページ）で解説します。構成プロファイルの例が、『*Enterprise Deployment Guide*』の「Sample Phase 3 Server Response With SCEP Specifications」に載っています。

第3段階：デバイスの構成

プロファイル要求（/profile）URLのハンドラ（再掲）

リスト 2-8（22 ページ）では暗号化した構成プロファイルの生成手順を示しました。このコードは、実際には第3段階で実行するものなので、詳細の説明は保留にしていました。この節で改めて、/profileのハンドラの説明をします。

暗号化を施した構成プロファイルは、次の手順で生成します。

- 構成プロファイルを、ひと揃いの構成ペイロードを使って生成します（このペイロードの内容について詳しくは、『*Enterprise Deployment Guide*』の「Configuration Profile Format」を参照）。

  この参照実装では、デバイスによらず同じプロファイルを生成します。しかし必要ならば、Challenge情報を使って要求元ユーザを特定し、当該ユーザに特化したプロファイルを生成することも可能です。

  同様に、渡されたデバイス情報を使って、当該デバイス、あるいはある型のデバイスに特化したプロファイルを生成することもできます（たとえばiOSベースのデバイスの機種ごとに異なるプロファイル）。
- この構成プロファイルに、要求に署名したデバイスの公開鍵を使って暗号化を施します。
- 暗号化したデータのBLOBを、構成プロファイルにラップします。

この暗号化したBLOBについて詳しくは、client_cert_configuration_payloadの説明（リスト A-1（29 ページ））およびconfiguration_payloadの説明（“「構成プロファイル」ペイロード”（25 ページ））を参照してください。

リスト 2-13　プロファイル要求（/profile）URL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関数で生成）は、“「プロファイルサービス」ペイロード”（18 ページ）に示した「プロファイルサービス」ペイロードとよく似ています。違いはペイロードが伝送する内容だけです。

この段階のプロファイル例が、『*Enterprise Deployment Guide*』の「Sample Phase 4 Device Response」に載っています。

「暗号化証明書」ペイロード

リスト 2-14に「暗号化証明書」ペイロードを生成するコード例を示します。クライアントに登録手続きの方法を伝える働きがあります。

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" # do not modify

    # strings that show up in UI, customisable
    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”（26 ページ）を参照してください。

「SCEP証明書」ペイロード

scep_cert_payload関数（リスト 2-15）は、名前が表すように、証明書の登録に必要な情報をデバイスに通知する、SCEPペイロードを生成します。

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" # do not modify
```

com.apple.security.scepというペイロード型がSCEPペイロードである旨を表し、その内容（PayloadContent）としてパラメータを指定します。

```
    # strings that show up in UI, customisable
    payload['PayloadDisplayName'] = purpose
    payload['PayloadDescription'] = "Provides device encryption identity"

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

まず第一に、SCEPサービス（便宜上、ここではサンプル）のベースURLを指定します。IOS版のSCEPサーバ（http://scep-server/cgi-bin/pkiclient.exe）か、Windows版のSCEPサーバ（http://scep-server/certsrv/mscep/mscep.dll）か、によって若干違って見えます。

```
=begin
    # scep instance NOTE: required for MS SCEP servers
    payload_content['Name'] = ""
=end
```

Nameの値によって、異なる証明書発行サービスを提供することも可能です。この値は最終的なURLの一部として埋め込まれます。Windows版の場合、この設定は必須です。ただし具体的な値は何でも構いません。

```
    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（Subject）には、要求されたサブジェクトをクライアントが指定できるようにする働きがあります。この場合、プロファイルサービスが値を埋めます。サービスによっては、クライアントがこれを指定することを許可せず、Challengeを使って要求元のIDをエンコードすることもあります。

X.509の「サブジェクト」フィールドは複雑な構造ですが、ここでは配列の配列として模倣し、必要なものをすべて指定しています。キーと値の組をそれぞれ、配列として指定します。キーは第1要素として指定し、その値は、OID（たとえば「0.9.2342.19200300.100.1.25」はDC（Domain Component）を表す）または所定の省略語（CN（Common Name）、C（Country name）、ST（STate or province name）、L（Locality name）、O（Organization name）、OU（Organization Unit name）など）を表す文字列です。上掲の例は、通常ならば「/O=ACME Inc./CN={purpose} ({random UUID})」と表示されるサブジェクトです。

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

以下、単純ながら若干の考慮を要するパラメータをいくつか示します。「KeySize」は、この長さのキーペアを生成するようデバイスに要求します。指定できるのは1024ビットまたは2048ビットに限ります。2048ビットを超える長さのキーには未対応です。2048ビット長のキーは生成時のオーバーヘッドが大きいので、通常は1024ビット長を推奨します。

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

「Key Type」はRSAでなければなりません。この参照実装では（そして実際にSCEPも）RSAキーにしか対応していないからです。

```
    payload_content['Key Usage'] = 5 # digital signature (1) | key encipherment
(4)
```

「Key Usage」は、キーの使用目的をビットマスクで指定します。ビット0の値が1ならばディジタル署名を施すこと、ビット2の値が1ならば暗号化を施すことを表します。なお、MS SCEPサーバは、署名と暗号化のいずれか一方しか実行できません。

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

SCEPは、認証局の証明書を別に検証できるならば、HTTP上で動作します。この機能は、iOSが未対応であるため、（上掲のように）当面無効にしてあります。このような処理に対応するためには、登録の際、HTTPSにより電話機がダウンロードするSCEPペイロードに、フィンガープリントを追加する必要があります。

```
    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つの節から成ります。まず"構成プロファイルのペイロードを生成するコード例"（29ページ）では、基本的なプロファイルのペイロードを、プログラムで組み立てる方法を示します。次に"応答の例"（30ページ）で、SCEPの典型的な登録セッションでやり取りされる、プロパティリストの例を示します。

構成プロファイルのペイロードを生成するコード例

リスト A-1に例示した構成プロファイルのペイロードには、イントラネット上のサイトを指すウェブクリップが含まれています。電話機はこのペイロードにもとづき、SSLクライアント認証証明書（保護されたアセットにアクセスするために必要）を登録できます。

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" # do not modify

    # strings that show up in UI, customisable
    webclip_payload['PayloadDisplayName'] = "ACME Inc."
    webclip_payload['PayloadDescription'] = "Creates a link to the ACME intranet
on the home screen"

    # allow user to remove webclip
    webclip_payload['IsRemovable'] = true

    # the link
    webclip_payload['Label'] = "ACME Inc."
```

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

ウェブクリップは、ユーザがくだんのURLに着目するよう、アイコンを生成します。この例では、ユーザによるウェブクリップの削除を許可します。

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

クライアント証明書の登録は、SCEPペイロードを生成することによりおこないます。暗号化したペイロードを復号するためのペイロードとよく似た形です。現実的な実装では、キーの使い方やポリシー、サブジェクトの別名（SAN、Subject Alternative Names）を指定するパラメータを追加して、登録されたIDと、ユーザやその能力とを、サーバが容易に照合できるようにするのが普通です。

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


end
```

この関数は最後に、ペイロードの「生の」配列をダンプします。呼び出し元はこれを構成プロファイルにラップし、署名を施します（リスト 2-8 （22 ページ）を参照）。

作成可能なペイロードの型について詳しくは、『*Configuration Profile Reference*』（developer.apple.com/library/ios/featuredarticles/iPhoneConfigurationProfileRef/Introduction/Introduction.htmlで配布）を参照してください。

応答の例

この節では、無線接続を介した登録および設定の各段階で現れる、プロファイルの例を示します。いずれも抜粋であり、実際の要件は、この例とは違っているでしょう。構文についてはこの付録で先に示した詳細を参照してください。各段階の説明が『*Over-the-Air Profile Delivery and Configuration*』に載っています。

第1段階：サーバの応答例

```
<?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>

              or

            <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>
```

第2段階：デバイスの応答例

```
<?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>

either:

        <string>String</string>

or:

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

    </dict>
</plist>
```

第3段階：サーバの応答例（SCEPの仕様）

```
<?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>
```

第4段階：デバイスの応答例

```
<?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>
```

書類の改訂履歴

この表は「*無線経由のプロファイルの配布と構成*」の改訂履歴です。

日付	メモ
2014-04-17	Apple iPhone Device CAの評価に関する警告を追加しました。
2010-08-03	『Enterprise Deployment Guide』に載っている例を織り込み、全体にわたってiOSの名前を変更しました。
2010-06-04	完全なスクリプトをコンパニオンファイルとして追加しました。
2010-03-24	新規資料。要求に応じてプロファイルを生成し、無線接続を介してiPhoneデバイスに配布する、サーバの構築方法を解説。

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

Apple Japan
〒106-6140 東京都港区六本木
6丁目10番1号 六本木ヒルズ
http://www.apple.com/jp/

Offline copy. Trademarks go here.