A Deep Dive into macOS MDM

(and how it can be compromised)

by Jesse Endahl & Max Bélanager

August 2018

Version 1.0

About the authors **4**

Acknowledgements **4**

Summary **4**

Basics **5**

What is MDM? 5

Setup process 5

What is DEP? 6

Setup process 6

What are configuration profiles? 7

Overview **8**

Entities 8

Protocols & Authentication 9

MDM 9

MDM network communication 11

MDM authentication 12

DEP 13

DEP Reseller API 13

DEP “cloud service” API 15

DEP internal API 16

DEP authentication 16

DEP Reseller API Authentication 16

DEP “cloud service” API Authentication 17

DEP internal API Authentication 18

SCEP 18

SCEP Authentication 18

APNs 19

Establishment of trust 20

MDM 20

Establishment of trust between MDM vendor and Apple 20

Establishment of trust between Customer, MDM vendor, and Apple 20

DEP 21

Establishment of trust between Reseller and Apple 21

Establishment of trust between Reseller and Customer 21

Establishment of trust between MDM Vendor and Apple 22

Establishment of trust between Device and Apple 22

Establishment of trust between Device and MDM Vendor 22

Putting it all together: Device bootstrap overview (DEP + MDM) 25

Deep Dive **25**

Architecture 26

ConfigurationProfiles.framework 26

Step 4: Retrieving the activation record 27

Step 5: Retrieving the activation profile 29

Step 6: Installing the activation profile 30

Step 7: Listening for commands 33

Vulnerability: InstallApplication **33**

Fix: InstallEnterpriseApplication **37**

Conclusion **38**

Takeaways 38

MDM Vendor Product Security Checklist 38

Recommendations for Apple 39

Appendix **40**

Trust hierarchy on macOS 40

Authentication methods 40

List of macOS binaries related to MDM 41

List of Apple server URLs 45

openssl output 47

About the authors

Jesse Endahl

Jesse Endahl is co-founder, CPO, and CSO at Fleetsmith. He previously worked at Dropbox, where he spent a year as an IT Engineer and two and a half years as an Infrastructure Security Engineer. He has spoken on security at conferences such as BSides SF, Google Cloud Next , and HashiConf. Jesse studied Political Economy & Urbanization at the University of California, Berkeley, and is a classically trained vocalist.

Max Bélanger

Max Bélanger is a strategic advisor at Dropbox. He joined the company in 2010 as one of its first engineering interns and helped build many of Dropbox's desktop features, including Finder integration and the Dropbox Badge. He most recently served as architect for Dropbox's desktop products. Max studied Software Engineering at the University of Ottawa, Canada.

Acknowledgements

First and foremost we'd like to Fleetsmith cofounder Stevie Hryciw for his amazing work on the early research that became that basis for this project, and who was the first at Fleetsmith to confirm the vulnerability discussed below.

Second, we'd like to thank Victor Vrantchan (@groob), Pepijn Bruienne (@bruienne), Michael Lynn (@mikeymikey), and Jesse Petersen (@jessecpeterson) for their research on DEP & MDM, as well as their open source work and contributions to the Mac security community. This research and whitepaper builds on the incredible work they done over the years.

Lastly, we'd like to thank Apple for their quick reaction and courteous response, and for the great work their security engineering team is doing to to continually improve platform security for both iOS & macOS.

Summary

This whitepaper walks through the various stages of bootstrapping a new device via DEP and MDM on macOS. We start with an introduction to basic concepts and terminology, then give an overview of entities, protocols, authentication, and bootstrapping of trust before moving into a deep dive. The deep dive covers the later steps of the process and details the binaries involved

in DEP and MDM, the IPC flows on the device, and evaluation of network trust (TLS) at each stage. We then demonstrate how a nation-state actor could exploit a vulnerability in the bootstrapping process, such that a user could unwrap a brand-new Mac, and the attacker could root it out of the box the first time it connects to WiFi. Next, we walk through how Apple implemented a new MDM command as a mitigation for the attack. Finally, we describe our takeaways and formulate recommendations based on our observations.

Basics

What is MDM?

MDM stands for **Mobile Device Management**. This acronym can be particularly confusing because it can refer to a product category, general product functionality, or support for Apple's specific MDM protocol. When we refer to MDM, we will implicitly refer to Apple's MDM protocol as defined in the MDM Protocol Reference documentation unless indicated otherwise.[1]

In Apple's own words, the "[...] protocol provides a way for system administrators to send device management commands to managed iOS devices running iOS 4 and later, macOS devices running macOS v10.7 and later, and Apple TV devices running iOS 7 (Apple TV software 6.0) and later. Through an MDM service, an IT administrator can inspect, install, or remove profiles; remove passcodes; and begin secure erase on a managed device".[2]

MDM "allows businesses to securely configure and manage scaled iPhone, iPad, Apple TV, and Mac deployments across their organizations [...] Using MDM, IT departments can enroll iOS devices in an enterprise environment, wirelessly configure and update settings, monitor compliance with corporate policies, and even remotely wipe or lock managed devices".[3]

Setup process

The process required to set up MDM with Apple can be broken into the following 4 high-level steps:

1. A customer choosing and configuring a product that has implemented support for the MDM protocol, such as Fleetsmith. We will refer to third-party implementers of the MDM protocol as MDM vendors.
2. Establishment of (cryptographic) trust between the MDM vendor, Apple, and the customer (typically an organization).
3. Enrollment of the devices into MDM.

---

[1] https://developer.apple.com/enterprise/documentation/MDM-Protocol-Reference.pdf
[2] Page 7, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.
[3] Page 70, iOS Security Guide, January 2018.

4. Management of devices via MDM commands.

In order to complete enrollment (step 3), a device needs (at a high level):

1. an MDM enrollment Configuration Profile, and;
2. a TLS client certificate to be used as a source of device identity

The installation of the MDM enrollment Configuration Profile can be performed manually by downloading and installing the file manually. This can also be automated using DEP (see below).

What is DEP?

DEP stands for **Device Enrollment Program**. The purpose of DEP is for devices to automatically check in with an MDM server whenever they go through the initial macOS Setup Assistant. Generally, this assistant is only triggered when the device is brand new or if it's been wiped/factory reset and the operating system has been reinstalled. Specifically, DEP streamlines the retrieval of a configuration profile containing MDM enrollment information, thus allowing the automatic enrollment with a predefined MDM server.

Like MDM, the term DEP can be confusing, as it can refer to the program itself, Apple's DEP servers, whether a vendor's device management product has implemented/supports the DEP specification, as well as the state of a specific device. A given device must be "DEP eligible" to participate: this is only possible if it was purchased in association with an Apple Customer Number that is associated with the Program.

In addition to automatic enrollment into MDM, DEP also allows for customization. For example, this can be used to control which screens are shown within Setup Assistant.

Setup process

The process required to set up DEP with Apple can be broken into 8 high-level steps[4]:

1. Customer goes through the process of setting up DEP—a multistep process involving creating a new/dedicated Apple ID, providing Apple with a DUNS number, and logging in and specifying to Apple the URL of the MDM server.
2. The MDM server generates a public/private keypair.
3. The public key is made available to the customer, who downloads it.
4. The customer uploads the public key to Apple.

[4] Page 95, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.

5. Apple uses the provided public key to S/MIME encrypt the “server token” (an OAuth 1.0a token).
6. The customer uploads the encrypted server token to the MDM server.
7. The MDM server decrypts the server token.
8. The MDM server authenticates to Apple servers via OAuth.

A note on DEP eligibility on a per-device basis

macOS devices only qualify for DEP if it is purchased in association with an Apple Customer Number associated with the Device Enrollment Program.

iOS and tvOS devices can be “retroactively DEPed” using Apple Configurator.[5]

What are configuration profiles?

Configuration Profiles (also referred to as `mobileconfigs`) are Apple’s official way of setting (and enforcing) system configuration. They are used on macOS, iOS, and tvOS, although there are differences in terms of which settings can be managed on each platform. Their file extension is `.mobileconfig`, and they contain payloads formatted as XML—specifically plist format, which is a serialization format created by Apple and used extensively behind the scenes on iOS, macOS, and tvOS.

An example of a Configuration Profile looks like this:

```
<?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>ProfileDisplayName</key>
    <string>Wi-Fi</string>
    <key>ProfileIdentifier</key>
    <string>com.fleetsmith.profile.wifi</string>
    <key>ProfileInstallDate</key>
    <string>2018-07-19 23:39:40 +0000</string>
    <key>ProfileItems</key>
    <key>ProfileOrganization</key>
    <string>Fleetsmith, Inc.</string>
    <key>ProfileRemovalDisallowed</key>
    <string>true</string>
    <key>ProfileType</key>
    <string>Configuration</string>
    <key>ProfileUUID</key>
    <string>aba1e786da43235a26483cb0af2e9346a15ed6fcf1257211d8caee86fbdee996</string>
    <key>ProfileVerificationState</key>
    <string>verified</string>
    <key>ProfileVersion</key>
    <integer>1</integer>
    <array>
```

[5] https://help.apple.com/configurator/mac/2.7.1/#/cad935fc6678

```
            <dict>
                    <key>PayloadDescription</key>
                    <string>WiFi settings for: "Insanely Great Studios"</string>
                    <key>PayloadDisplayName</key>
                    <string>Wi-Fi Network</string>
                    <key>PayloadIdentifier</key>
                    <string>com.fleetsmith.profile.wifi.config.0</string>
                    <key>PayloadType</key>
                    <string>com.apple.wifi.managed</string>
                    <key>PayloadUUID</key>
                    <string>wifi-config-0</string>
                    <key>PayloadVersion</key>
                    <integer>1</integer>
                    <key>PayloadContent</key>
                    <dict>
                            <key>AutoJoin</key>
                            <true/>
                            <key>EncryptionType</key>
                            <string>WPA2</string>
                            <key>HIDDEN_NETWORK</key>
                            <false/>
                            <key>Password</key>
                            <string>********</string>
                            <key>SSID_STR</key>
                            <string>Insanely Great Studios</string>
                    </dict>
            </dict>
    </array>
</dict>
</plist>
```

This config allows a device to connect to a WiFi WPA2 network. As you can see, this config contains a password to connect to the network. Configuration profile payloads may optionally be encrypted to protect sensitive data such as this, using a public key associated with a specific device.

Apple documentation states: “Configuration profiles can be signed and encrypted to validate their origin, ensure their integrity, and protect their contents. Configuration profiles are encrypted using CMS (RFC 3852), supporting 3DES and AES-128.”[6]

Overview

Entities

- Apple
- Reseller (includes Apple Retail)
- MDM vendor
- Customer
- Device

[6] Page 70, iOS Security Guide, January 2018.

Protocols & Authentication

MDM

"The MDM protocol is built on top of HTTPS, transport layer security (TLS), and push notifications. The related MDM check-in protocol provides a way to delegate the initial registration process to a separate server."[7] In practice, MDM protocol works via a combination of APNs push notifications to the device & subsequent device check-ins to the MDM server via a RESTful API. Communication occurs between a device and a server associated with a device management product.

Similar to much of macOS (including Configuration Profiles), the MDM protocol is built on the plist format as well. In fact, Configuration Profiles can be delivered via the MDM protocol. The MDM protocol is built around the concept of commands. To install a Configuration Profile to a device, the server sends the `InstallProfile` command. A command can be sent to a device by sending a `plist`-encoded dictionary. An example of an MDM command for installing an application (using the InstallApplication command) looks like this:

```
<?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>Command</key>
    <dict>
      <key>ManagementFlags</key>
      <integer>1</integer>
      <key>ManifestURL</key>
      <string>https://link/to/manifest.plist</string>
      <key>Options</key>
      <dict></dict>
      <key>RequestType</key>
      <string>InstallApplication</string>
    </dict>
    <key>CommandUUID</key>
    <string>8dc5fba4-bccb-4541-97aa-e48a6d89f425</string>
  </dict>
</plist>
```

A device is "enrolled into MDM" by installing a Configuration Profile. An example of an actual MDM enrollment Configuration Profile looks like this:

```
<?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>ProfileDisplayName</key>
    <string>MDM Configuration</string>
    <key>ProfileIdentifier</key>
    <string>com.fleetsmith.profile.mdm</string>
```

[7] Page 7, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.

```
    <key>ProfileInstallDate</key>
    <string>2018-07-19 01:40:10 +0000</string>
    <key>ProfileItems</key>
    <array>
    <key>ProfileOrganization</key>
    <string>Fleetsmith, Inc.</string>
    <key>ProfileRemovalDisallowed</key>
    <string>true</string>
    <key>ProfileType</key>
    <string>Configuration</string>
    <key>ProfileUUID</key>
    <string>c57bb9bd-a5ab-44ea-96df-9a93145a8554</string>
    <key>ProfileVersion</key>
    <integer>1</integer>
            <dict>
                    <key>PayloadContent</key>
                    <dict>
                            <key>AccessRights</key>
                            <integer>8191</integer>
                            <key>CheckInURL</key>
                            <string>https://mdm.fleetsmith.cloud/checkin</string>
                            <key>CheckInURLPinningCertificateUUIDs</key>
                            <array>
                                    <string>19a8f95c-5e1f-4d87-a4b4-c5e0bdb5cc94</string>
                            </array>
                            <key>CheckOutWhenRremoved</key>
                            <true/>
                            <key>IdentityCertificateUUID</key>
                            <string>fd8a6b9e-0fed-406f-9571-8ec98722b713</string>
                            <key>ServerCapabilities</key>
                            <array>
                                    <string>com.apple.mdm.per-user-connections</string>
                            </array>
                            <key>ServerURL</key>
                            <string>https://mdm.fleetsmith.cloud/commands</string>
                            <key>ServerURLPinningCertificateUUIDs</key>
                            <array>
                                    <string>19a8f95c-5e1f-4d87-a4b4-c5e0bdb5cc94</string>
                            </array>
                            <key>SignMessage</key>
                            <true/>
                            <key>Topic</key>

<string>com.apple.mgmt.External.d200d2cf-048c-4ca4-882d-a8e9249e3a7f</string>
                    </dict>
                    <key>PayloadDisplayName</key>
                    <string></string>
                    <key>PayloadIdentifier</key>
                    <string>com.fleetsmith.profile.mdm.mdm</string>
                    <key>PayloadType</key>
                    <string>com.apple.mdm</string>
                    <key>PayloadUUID</key>
                    <string>629d68f6-ef7e-4c62-a025-9e0ba261a37a</string>
                    <key>PayloadVersion</key>
                    <integer>1</integer>
            </dict>
            <dict>
                    <key>PayloadContent</key>
                    <dict/>
                    <key>PayloadDisplayName</key>
                    <string></string>
                    <key>PayloadIdentifier</key>
                    <string>com.fleetsmith.profile.mdm.scep</string>
                    <key>PayloadType</key>
                    <string>com.apple.security.scep</string>
                    <key>PayloadUUID</key>
```

```
                <string>fd8a6b9e-0fed-406f-9571-8ec98722b713</string>
                <key>PayloadVersion</key>
                <integer>1</integer>
        </dict>
        <dict>
                <key>PayloadContent</key>
                <dict/>
                <key>PayloadDisplayName</key>
                <string></string>
                <key>PayloadIdentifier</key>
                <string>com.fleetsmith.profile.mdm.rootca</string>
                <key>PayloadType</key>
                <string>com.apple.security.pem</string>
                <key>PayloadUUID</key>
                <string>19a8f95c-5e1f-4d87-a4b4-c5e0bdb5cc94</string>
                <key>PayloadVersion</key>
                <integer>1</integer>
        </dict>
    </array>
</dict>
</plist>
```

There are 3 `PayloadTypes` being delivered in the above Configuration Profile:

- `com.apple.security.pem`
- `com.apple.mdm`
- `com.apple.security.scep`

The first installs new root certificates, since Fleetsmith has a private Certificate Authority built into the product. The second tells the device how to reach the MDM server, and tells the device to TLS pin to the certs we just installed. The third tells the client that it should use SCEP to generate a CSR and submit it to Fleetsmith, so that it can receive a TLS client certificate. The TLS client certificate will be used to authenticate to the MDM server. SCEP is discussed in more detail in the Protocol section of this whitepaper.

MDM network communication

After a device is enrolled, Apple describes the normal communication flow as such:

- "The server (at some point in the future) sends out a push notification to the device."
- "The device polls the server for a command in response to the push notification."
- "The device performs the command."
- "The device contacts the server to report the result of the last command and to request the next command."[8]

A more detailed version of the steps above is as follows:

[8] Page 14, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.

1. “the MDM server sends a notification through the APNs gateway to the device. The message sent with the push notification is JSON-formatted and must contain the `PushMagic` string as the value of the `mdm` key.”[9]
2. “the device responds to this push notification by contacting the MDM server using HTTP `PUT` over TLS (SSL).”[10]
3. “The server responds by sending the next command that the device should perform by enclosing it in the HTTP reply.[11]
4. “The device performs the command and sends its reply in another HTTP `PUT` request to the MDM server.”[12]
5. “The MDM server can then reply with the next command or end the connection by sending 200 status (`OK`) with an empty response body.[13]

A note on statefulness

There is almost zero client-side tracking of client state. The client (macOS, iOS) does not do anything to “help” you if your approach to device management is that of desired state/idempotency. The server is responsible for keeping track of the majority of the client’s state. This differs dramatically from the world of configuration management tooling, where many tools allow the administrator to specify the desired state of a client from the server, and the client handles keeping track of its own state, and ensures that it matches what was defined by the server. In short, it is clear that MDM was designed at a time when imperative workflows were the norm, as opposed to declarative.

MDM authentication

There are multiple pieces of information used for identifying a device when it comes MDM:

- The device TLS client certificate
- A push notification device token, which also contains within it, a PushMagic string

It’s up to the MDM server to “pair” the device certificate and push notification device token. Only the device certificate is used for authentication.

The MDM server must authenticate the client by ensuring its client certificate is issued by the expected CA, is still valid, etc. It is up to the MDM vendor to decide where to implement TLS client certficiate verification. It could be done at the “edge” at a load balancer and/or web servers such as nginx, Apache, or HAProxy and/or at the application layer.

---

[9] Page 18, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.
[10] Page 18, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.
[11] Page 19, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.
[12] Page 19, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.
[13] Page 19, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.

DEP

There are three separate DEP protocols:

- one for resellers and distributors
- one for MDM vendors
- one that devices call into at boot

For the sake of clarity, we'll refer to the three APIs as:

- DEP reseller API (used by resellers/distributors)
- DEP “cloud service” API (used by MDM vendors)
- DEP internal API (used by macOS)

macOS retrieves the same information that’s populated by the MDM vendor via the cloud service API, but uses a separate Apple protocol in the process (internal DEP API).

All three DEP APIs utilize JSON and have similar schemas.

Though the server URLs are different, the second two APIs share a server certificate (it contains multiple CNs within its server certificate). The DEP servers running the API used by resellers and distributors have a server certificate issued by a separate CA.

DEP Reseller API

A RESTful API is made available resellers and distributors to perform operations related to DEP and physical device inventory, such as device sales and returns. Technically there are three APIs under this one framework:

- Bulk Enroll Devices API
- Check Transaction Status API
- Show Order Details API

Apple documentation defines the purposes of these APIs as such:

1. **Bulk Enroll Devices** - this API call will allow the posting of your customer’s device orders to DEP.
2. **Check Transaction Status** - this API is used in conjunction with the Bulk Enroll Devices API to query the system for the status of the transaction that occurred in the past.

3. **Show Order Details** - this API is used to determine the current enrollment status of devices from a given order number. This API is meant to be used for auditing & reconciliation purposes.

The “Bulk Enroll Devices” API is most interesting, and the one worth discussing. This is how devices that have been purchased by a customer get added to DEP. Put differently, this is how a device makes its way into the Apple Business Manager web portal that customers are exposed to.

There are multiple types of API calls that can be made depending on the order type: Order (OR), Return (RE), Override (OV), and Void (VD).

See the Bulk Enrollment Devices API order matrix (below) for a full list.

API Structure					Order Type Data Requirements			
					Order (OR)	Return (RE)	Override (OV)	Void (VD)
HTTP Header	Content-Type				Required	Required	Required	Required
	Content-Encoding				Optional	Optional	Optional	Optional
	Accept-Encoding				Optional	Optional	Optional	Optional
Body	Request Context	Ship-To			Required	Required	Required	Required
		Language Code			Required	Required	Required	Required
		Time Zone			Required	Required	Required	Required
	DEP Reseller ID				Required	Required	Required	Required
	Transaction ID				Required	Required	Required	Required
	Orders	Order Number			Required	Required	Required	Required
		Order Date			Required	Required	Required	Required
		Order Type			Required	Required	Required	Required
		Customer DEP ID			Required	Required	Required	Required
		PO Number			Optional	Optional	Optional	Optional
		Deliveries	Delivery Number		Required	Required	Required	Do not include
			Ship-Date		Required	Required	Required	Do not include
			Devices	Serial Number	Required	Required	Required	Do not include
				Asset Tag	Optional	Optional	Optional	Do not include

14

An example of a POST to the Bulk Enroll Devices API looks like this:

```
{
    "requestContext": {
        "shipTo": "0000055555",
        "timeZone": "420",
        "langCode": "en"
    },
    "transactionId": "TRID_2525088",
    "depResellerId": "ID16B64D90",
    "orders": [
        {
            "orderNumber": "OR_2525088",
            "orderDate": "2014-10-20T01:23:11Z",
            "orderType": "OR",
            "customerId": “CUST_A",
            "poNumber": "PO_2525088",
            "deliveries": [
                {
```

[14] https://applecareconnect.apple.com/api-docs/depuat/html/images/order_type_matrix.png

```
                    "deliveryNumber": "DEL_2525088",
                    "shipDate": "2014-10-23T01:24:11Z",
                    "devices": [
                        {
                            "deviceId": "C3LD3525DCP9",
                            "assetTag": "A_test1"
                        },
          {
                            "deviceId": "C3LD3526DCP9",
                            "assetTag": "A_test2"
                        },
          {
                            "deviceId": "C3LD3527DCP9",
                            "assetTag": "A_test3"
                        },
          {
                            "deviceId": "C3LD3528DCP9",
                            "assetTag": "A_test4"
                        }
                    ]
                }
            ]
        }
    ]
}
```

The other APIs are not worth additional discussion here—see the "API Documentation & Process Flows" section of the documentation for more information.[15]

DEP "cloud service" API

The DEP API that MDM vendors utilize is referred to as the "cloud service API" by Apple. The MDM Protocol Reference defines it as an API that "provides profile management and mapping. With this API, you can create profiles, update profiles, delete profiles, obtain a list of devices, and associate those profiles with specific devices."[16]

Confusingly, the JSON payloads are referred to as "profiles," but have no relation to Configuration Profiles.

All DEP devices receive a JSON payload ("DEP profile") at first boot. It's delivered by Apple, but the contents are populated ahead of time by the MDM server. It includes important information such as the URL of the MDM server, any pinned certificates, and which screens should be skipped in the macOS Setup Assistant.

An example of a request sent by an MDM server, to create a DEP profile on Apple's servers, looks like this:[17]

---

[15] https://applecareconnect.apple.com/api-docs/depuat/html/WSImpManual.html
[16] Page 9, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.
[17] Pages 121–122, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.

```
POST /profile HTTP/1.1
User-Agent:ProfileManager-1.0
X-Server-Protocol-Version:2
Content-Type: application/json;charset=UTF8
Content-Length: 350
X-ADM-Auth-Session: 87a235815b8d6661ac73329f75815b8d6661ac73329f815
{
   "profile_name": "Test Profile",
   "url":"https://mdm.acmeinc.com/getconfig",
   "is_supervised":false,
   "allow_pairing":true,
   "is_mandatory":false,
   "await_device_configured":false,
   "is_mdm_removable":false,
   "department": "IT Department",
   "org_magic": "913FABBB-0032-4E13-9966-D6BBAC900331",
   "support_phone_number": "1-555-555-5555",
   "support_email_address": "org-email@example.com",
   "anchor_certs":[
       "..."
   ],
   "supervising_host_certs:[
       "..."
    ],
   "skip_setup_items":[
      "Location",
      "Restore",
      "Android",
      "AppleID",
      "TOS",
      "Siri",
      "Diagnostics",
      "Biometric",
      "Payment",
      "Zoom",
      "FileVault"
   ],
    "devices":["C8TJ500QF1MN", "B7CJ500QF1MA"]
}
```

DEP internal API

Little is known about this API. We briefly touch on it in the deep dive section, but it was not a focus of our research.

DEP authentication

Different types of authentication are utilized depending on which stage of DEP we're looking at, and which API/server endpoints are involved.

DEP Reseller API Authentication

Resellers use one of the three APIs discussed above depending on what goal they're trying to achieve, but authentication is the same for all three.

All three APIs require a client certificate to authenticate. A note can be found on the pages dedicated to each API, which states:

"This API requires certificate based authentication. The API request should be made with the appropriate SSL certificate for the particular Sold-To. ACC will validate the certificate against the Sold-To, and on successful validation, the enrollment process will continue."[18]

The documentation for reseller integration with DEP states:

“You will need to have a client certificate installed on your servers in order to communicate with APIs. Certificates expire every 2 years. Approximately 2 months prior to expiration, Apple will send a reminder to create a new certificate.”[19] The CN field of the certificate must be in the following format:

```
GRX-<10DigitsoldTo>.ACC1914.Prod.AppleCare
```

As hinted above, the Sold-To number must be in a 10-digit format. The documentation states:

“The leading zeros are important and the soldTo should always be 10 digits. For example, if your soldTo is 0000098765, the value should be:

```
GRX-0000098765.ACC1914.Test.AppleCare
```

for test and

```
GRX-0000098765.ACC1914.Prod.AppleCare
```

for production.”[20]

DEP “cloud service” API Authentication

MDM vendors use the cloud service API for the sake of syncing devices and/or updating their DEP profile. This API uses OAuth 1.0a for authentication.

---

18 https://applecareconnect.apple.com/api-docs/depuat/html/WSReference.html?user=reseller&id=1111&lang=EN

19 https://applecareconnect.apple.com/api-docs/depuat/html/WSFaq.html?user=reseller&id=1111&lang=EN

20 https://applecareconnect.apple.com/api-docs/depuat/html/WSFaq.html?user=reseller&id=1111&lang=EN

DEP internal API Authentication

DEP device to Apple
Devices calling into Apple to retrieve their DEP profile authenticate with some concept of device identity. There is no public documentation about how this works, and this portion of bootstrapping process is relatively unexplored, but it was not a focus of our research.

DEP device to MDM vendor

The first time a DEP device calls into an MDM server, it authenticates using a signed plist containing metadata about itself, which is then verified by the MDM server. See the section "Establishment of trust between Device and MDM Vendor" below for a detailed explanation.

SCEP

SCEP is a fairly old protocol that was created before TLS and HTTPS were widespread—the first RFC draft was published in the year 2000.[21] Its purpose is similar to the (much more modern) ACME protocol[22] created by the Let's Encrypt project. It's meant to be a way for a client to send a Certificate Signing Request (CSR) for the purpose of being granted a certificate. In lieu of TLS/HTTPS, PKCS#7 signed data is relied upon to ensure message integrity.

The SCEP RFC does not dictate that communication must occur over HTTPS. Although this shouldn't matter due to the use of PKCS#7 signing of messages, HTTPS should be used as a defense-in-depth measure.

In the case of MDM, SCEP is used to request and grant individual client certificates to devices. One major security benefit to SCEP is that the private key associated with the CSR and certificate is generated on-device, is never sent over the network, and remains secret from the server. Not all vendors implement support for SCEP, instead choosing to generate private keys server side and distribute them to clients. This is a less secure approach.

SCEP Authentication

1. Client fetches `GetCACert`
2. Client receives response:

"The response is simply the binary-encoded CA certificate (X.509). The client needs to validate that the CA certificate is trusted through an examination of the fingerprint/hash. This has to be

[21] https://tools.ietf.org/html/draft-nourse-scep-00
[22] https://tools.ietf.org/id/draft-ietf-acme-acme-09.html

done via an out-of-band method (a phone call to a system administrator or pre-configuration of the fingerprint within the trustpoint)."[23]

3. The client signs an "envelope" of data using (usually) a self-signed certificate, and sends it to the SCEP server. Within the envelope, there are two things, one nested inside the other.

- First, we have the encrypted data and the public key of the CA that was used to encrypt the data.
- Second, we have the actual contents of the encrypted data. The encrypted data is the actual CSR (Certificate Signing Request), which includes:
    - the requested "Subject Name" within the certificate
    - the public key of the device submitting the CSR (the client)
    - any requested extensions, such as Key Usages (which specify what the certificate can be used for) or other names that the certificate should be valid for, aside from the main one ("Subject Alternate Names").

The end result looks like this: envelope → encrypted data (CSR).

The SCEP protocol includes a field called a Challenge Password. The name is unfortunate, since the value of this field is just an arbitrary string. Unfortunately, vendors who do implement support for SCEP are left to make their own decisions about how to use the SCEP Challenge Password field, if at all (it's marked as optional in the RFC). The good news is that it can be used to include data to be evaluated by the server in the process of deciding whether to grant the requested certificate to the client. Fleetsmith uses this field to include an HMAC, for example, allowing the server to perform validation that a specific SCEP CSR ties back to a valid request from an authenticated user.

APNs

APNs is used for a variety of purposes. It is well documented due to its widespread use on iOS in everything from user facing notifications to being a fundamental building block for iMessage.

In the world of MDM, "APNs is used to wake the device so it can communicate directly with its MDM solution over a secured connection. No confidential or proprietary information is transmitted via APNs."[24]

---

[23] https://www.cisco.com/c/en/us/support/docs/security-vpn/public-key-infrastructure-pki/116167-technote-scep-00.html#anc2

[24] Page 70, iOS Security Guide, January 2018.

Establishment of trust

MDM

Establishment of trust between MDM vendor and Apple

MDM vendors must enroll in an Apple Enterprise Developer account (different than a standard Apple Developer account), after which they must submit a request to Apple to be granted an MDM signing certificate, which is necessary in order for an MDM vendor to be able to sign APNS CSRs (see MDM section in "establishment of trust" above). After the request is submitted, a verification process takes where the requester must explain their use-case/need for this special signing certificate. In Fleetsmith's experience, this meant explaining to an Apple representative that we were a new vendor in the market, building a new product with support for the MDM protocol and feature-set. It's also possible for companies to request an MDM signing certificate for internal enterprise use. This would be necessary if the company was intending to use an open-source MDM solution such as micromdm.[25]

Establishment of trust between Customer, MDM vendor, and Apple

The establishment of trust for MDM is based on a cross-signing process that works as follows:

1. MDM vendor generates a Push Certificate Request plist, and makes it available to customer for download. Must contain:
    a. a complete certificate chain all the way back to a recognized root certificate (including MDM signing certificate):
        i. MDM signing certificate
        ii. WWDR intermediate certificate
        iii. Apple Inc. root certificate
    b. A customer-specific CSR (Certificate Signing Request), signed with the private key of the MDM vendor's MDM Signing Cert, using the SHA1WithRSA signing algorithm.
2. Customer logs in to https://identity.apple.com/pushcert using a verified Apple ID and uploads the Push Certificate Request to the Apple Push Certificates Portal.
3. Apple signs CSR and returns certificate to customer.
4. Customer uploads APNs Certificate for MDM to MDM product..

The total number of steps to setup MDM are greater due to steps involving accepting legal terms and creating an Apple ID, etc. The steps listed here are the only those that establish trust cryptographically.[26]

---

[25] https://github.com/micromdm/micromdm

[26] *For a more detailed description of this processes, see "MDM Vendor CSR Signing Overview," page 218, MDM Protocol Reference.*

DEP

Establishment of trust between Reseller and Apple

Apple requires both resellers and distributors to sign legal agreements before they can participate in DEP. Furthermore, they issue client certificates to resellers, which are required in order to authenticate to the DEP resellers APIs.

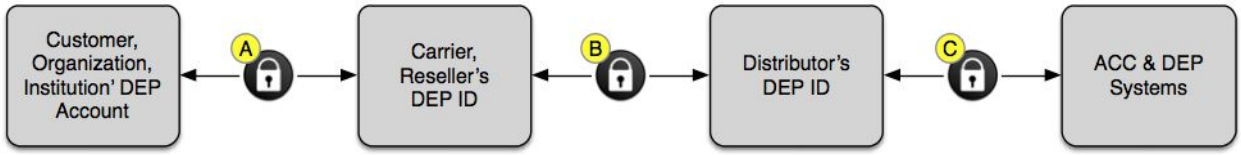
Establishment of trust between Reseller and Customer

The customer must add reseller's "SoldTo" account number within the Apple Business Manager Portal. "A reseller is only able to associate devices with a customer after that customer has "whitelisted" the reseller by adding the seller's SoldTo number"[27]

The documentation states:[28]

"

A. The reseller must provide their DEP Reseller ID to the customer. With this ID, the customer must whitelist the DEP reseller ID with their DEP account.
B. Apple will link the reseller DEP ID with the distributor's DEP ID upon the reseller and the distributor's signing the DEP agreement.
C. Apple will check that the Customer ID, DEP reseller IDs for both the reseller and the distributor are linked in ACC and the DEP back end systems. A certificate containing the distributor's Sold-to, and the Ship-to passed in the API will validate the API connection.



[29]

---

[27] https://applecareconnect.apple.com/api-docs/depuat/html/WSImpManual.html?user=reseller&id=1111&lang=EN

[28] https://applecareconnect.apple.com/api-docs/depuat/html/WSImpManual.html?user=reseller&id=1111&lang=EN

[29] https://applecareconnect.apple.com/api-docs/depuat/html/images/whitelist_05_disti.png

Establishment of trust between MDM Vendor and Apple

In order for the MDM vendor to sync device records and set "DEP profiles" for devices, the vendor must authenticate with an OAuth token. In order to be granted that OAuth token, the vendor must go through the following process:

1. MDM vendor generates a keypair, then encrypts request with customer-specific public key.
2. Customer uploads public key to Apple, Apple uses it to encrypt object, and returns to customer.
3. Customer uploads encrypted object back to MDM vendor.
4. MDM vendor decrypts object, revealing auth token. Uses it to authenticate to Apple service.

Establishment of trust between Device and Apple

As mentioned in the "DEP internal API Authentication" section, devices calling into the DEP internal API authenticate themselves with their device identity. Exactly how Apple bootstraps device identity within macOS, or how it is verified server side, is relatively unexplored, and the MDM Protocol Reference documentation does not address it. Our "deep dive" section briefly touches on this, but this was not a focus in our research.

Establishment of trust between Device and MDM Vendor

After the device has received its "DEP profile" JSON payload from the DEP internal API, the device proceeds to a POST CMS-signed (PKCS#7), DER-encoded payload plist containing metadata about the device to the MDM vendor. The device information included in that plist is reflected in the following table (reproduced from documentation):[30]

Field	Type	Content
`UDID`	String	The device's UDID.
`SERIAL`	String	The device's serial number.
`PRODUCT`	String	The device's product type, e.g. "MacBook9,1".
`VERSION`	String	The OS version installed on the device, e.g. 17G65.
`IMEI`	String	The device's IMEI (if available).
`MEID`	String	The device's MEID (if available).

[30] Page 129, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.

LANGUAGE	String	The user's currently-selected language, e.g. “en”

An example of the payload is the following:

```
<?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>LANGUAGE</key>
        <string>en_US</string>
        <key>PRODUCT</key>
        <string>MacBook10,1</string>
        <key>SERIAL</key>
        <string>XXXXXXXXXXXX</string>
        <key>UDID</key>
        <string>00000000-0000-0000-0000-000000000000</string>
        <key>VERSION</key>
        <string>17G65</string>
</dict>
</plist>
```

The request payload is signed using the device identity certificate (as documented), and the following intermediate certificates are included:

Certificate	Expiry
Apple iPhone Device CA	**Wed Apr 16 15:54:46 2014** This certificate is now expired, but is still in use.
Apple iPhone Certification Authority	**Tue Apr 12 10:43:28 2022**
Apple Root CA	**Fri Feb 9 13:40:36 2035**

Strangely, one of the certificates used for this purpose is expired. This is not documented in the current version of the MDM Protocol Reference. Thankfully this is documented in the deprecated “Over-the-Air Profile Delivery and Configuration” reference. The information we’re interested can be found in the paragraph under in following section:[31]

Creating a Profile Server for Over-The-Air Enrollment and Configuration > Profile Service Handlers > Phase 1: Authentication > Profile Service Payload > Listing 2–5 (`profile_service_payload` function)

31

https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/iPhoneOTAConfiguration/profile-service/profile-service.html#//apple_ref/doc/uid/TP40009505-CH2-SW4

The documentation states:

“The payload content provides a URL where the device should send its identification (using HTTP POST), along with a list of attributes that the server expects the device to provide (software version, IMEI, and so on).”[32]

“In response, the device sends back the list of requested attributes along with their values. If the server sent a Challenge value in its request, the device also includes this value along with the requested device attributes. Finally, to prove it is an iOS-based device, the device signs this identification with its device certificate. This response is sent to the handler for the /profile URL.”[33]

“Validate that the device certificate is issued from “Apple iPhone Device CA”, which has the following Base64 encoded PEM data:[34]

```
-----BEGIN CERTIFICATE-----
MIIDaTCCAlGgAwIBAgIBATANBgkqhkiG9w0BAQUFADB5MQswCQYDVQQGEwJVUzET
MBEGA1UEChMKQXBwbGUgSW5jLjEmMCQGA1UECxMdQXBwbGUgQ2VydGlmaWNhdGlv
biBBdXRob3JpdHkxLTArBgNVBAMTJEFwcGxlIGlQaG9uZSBDZXJ0aWZpY2F0aW9u
IEF1dGhvcml0eTAeFw0wNzA0MTYyMjU0NDZaFw0xNDA0MTYyMjU0NDZaMFoxCzAJ
BgNVBAYTAlVTMRMwEQYDVQQKEwpBcHBsZSBJbmMuMRUwEwYDVQQLEwxBcHBsZSBp
UGhvbmUxHzAdBgNVBAMTFkFwcGxlIGlQaG9uZSBEZXZpY2UgQ0EwgZ8wDQYJKoZI
hvcNAQEBBQADgY0AMIGJAoGBAPGUSsnquloYYK3Lok1NTlQZaRdZB2bLl+hmmkdf
Rq5nerVKc1SxywT2vTa4DFU4ioSDMVJl+TPhl3ecK0wmsCU/6TKqewh0lOzBSzgd
Z04IUpRai1mjXNeT9KD+VYW7TEaXXm6yd0UvZ1y8Cxi/WblshvcqdXbSGXH0KWO5
JQuvAgMBAAGjgZ4wgZswDgYDVR0PAQH/BAQDAgGGMA8GA1UdEwEB/wQFMAMBAf8w
HQYDVR0OBBYEFLL+ISNEhpVqedWBJo5zENinTI50MB8GA1UdIwQYMBaAFOc0Ki4i
3jlga7SUzneDYS8xoHw1MDgGA1UdHwQxMC8wLaAroCmGJ2h0dHA6Ly93d3cuYXBw
bGUuY29tL2FwcGxlY2EvaXBob25lLmNybDANBgkqhkiG9w0BAQUFAAOCAQEAd13P
Z3pMViukVHe9WUg8Hum+0I/0kHKvjhwVd/IMwGlXyU7DhUYWdja2X/zqj7W24Aq5
7dEKm3fqqxK5XCFVGY5HI0cRsdENyTP7lxSiiTRYj2mlPedheCn+k6T5y0U4Xr40
FXwWb2nWqCF1AgIudhgvVbxlvqcxUm8Zz7yDeJ0JFovXQhyO5fLUHRLCQFssAbf8
B4i8rYYsBUhYTspVJcxVpIIltkYpdIRSIARA49HNvKK4hzjzMS/OhKQpVKw+OCEZ
xptCVeN2pjbdt9uzi175oVo/u6B2ArKAW17u6XEHIdDMOe7cb33peVI6TD15W4MI
pyQPbp8orlXe+tA8JA==
-----END CERTIFICATE-----
```

Apple addresses the fact that the Apple iPhone Device CA certificate is expired in a single line, stating:

---

32 https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/iPhoneOTAConfiguration/profile-service/profile-service.html#//apple_ref/doc/uid/TP40009505-CH2-SW4

33 https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/iPhoneOTAConfiguration/profile-service/profile-service.html#//apple_ref/doc/uid/TP40009505-CH2-SW4

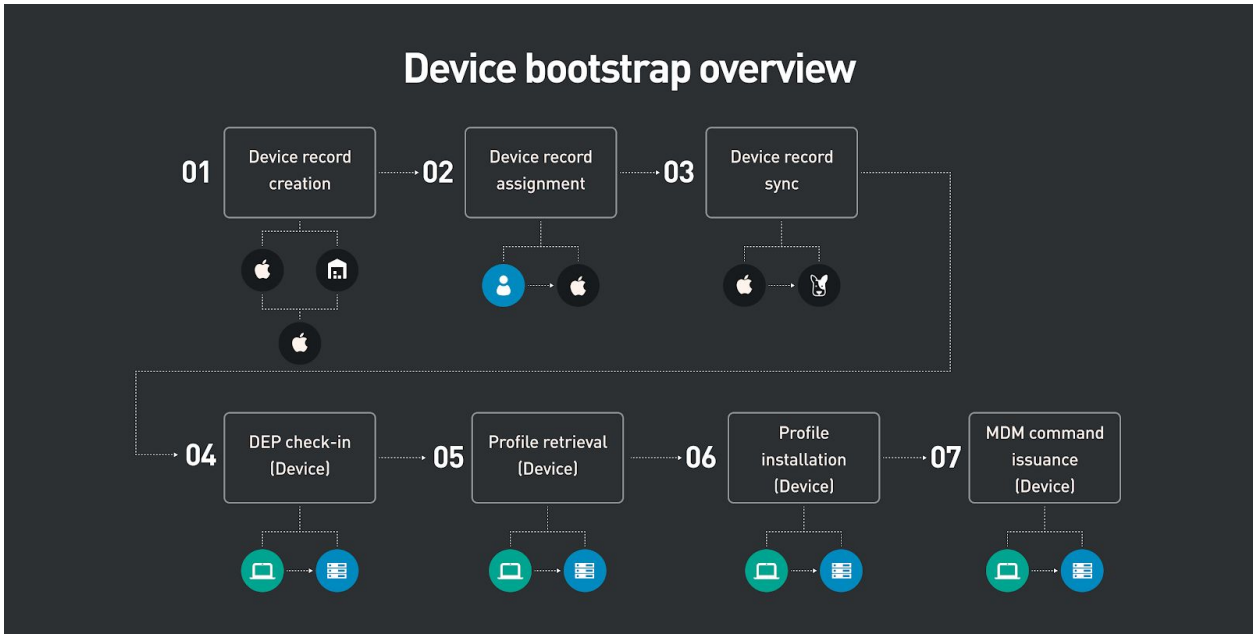
34 https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/iPhoneOTAConfiguration/profile-service/profile-service.html#//apple_ref/doc/uid/TP40009505-CH2-SW4

"WARNING: When device certificates signed "Apple iPhone Device CA" are evaluated their validity dates should be ignored."[35]

Since the deprecated/archived "Over-the-Air Profile Delivery and Configuration" documentation is the only information Apple has published about this CA, it is unknown why an expired CA continues to be used.

Putting it all together: Device bootstrap overview (DEP + MDM)

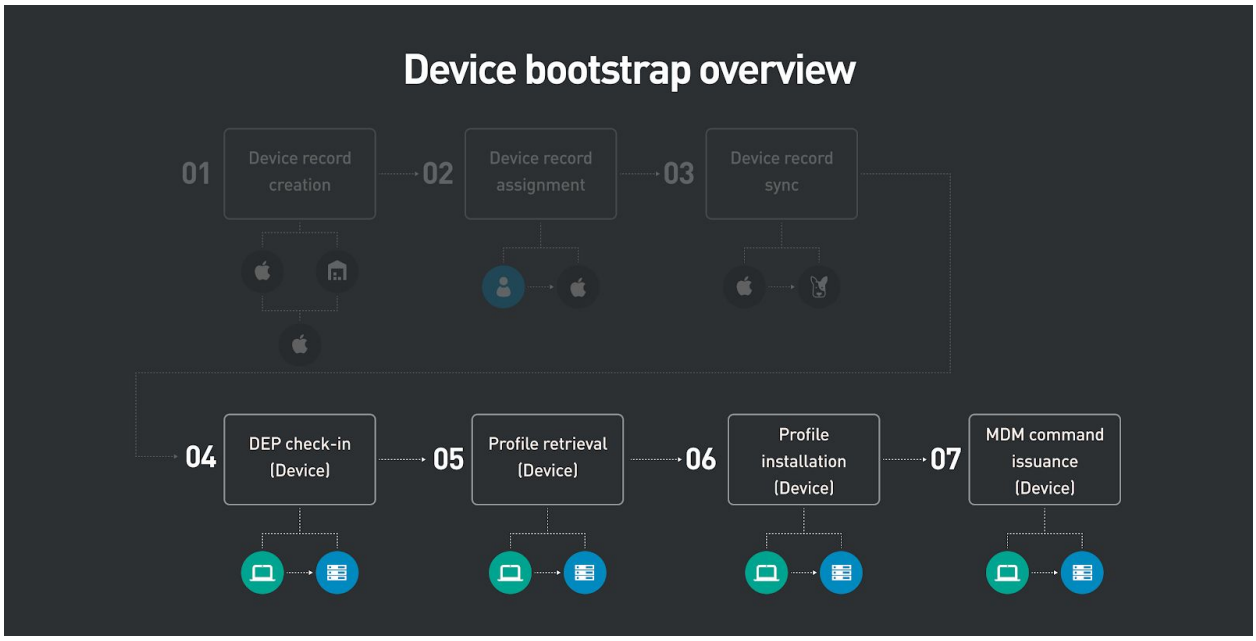
1. **Device record creation**: Apple or Apple Authorized Reseller creates device record via DEP Reseller API.
2. **Device record assignment**: Customer assigns to chosen MDM server in Apple Business Manager portal.
3. **Device record sync**: MDM vendor pulls new device records from Apple DEP servers, and creates "DEP profiles" for each device.
4. **DEP check-in**: device authenticates to Apple's DEP servers using proprietary device identity, retrieves its DEP profile (or, the "activation record").
5. **Profile retrieval**: device authenticates to MDM vendor servers with device identity, to retrieve a configuration profile (including root certs payload, MDM enrollment payload and SCEP payload, or more).
6. **Profile installation**: includes requesting and retrieving the client certificate via SCEP.
7. **MDM command issuance**: Device authenticates to MDM server using client certificate, and starts receiving MDM commands via APNS.



35

https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/iPhoneOTAConfiguration/profile-service/profile-service.html#//apple_ref/doc/uid/TP40009505-CH2-SW4

For our deep dive, we will focus on the portion from device DEP check-in to it receiving MDM commands (*steps 4 – 7*).



Deep Dive

The latter parts of the bootstrap process involve the device itself. These steps are complex, as they are implemented by the operating system through various components interacting over the network with multiple services and using a wide range of protocols. We will refer to these steps as the **enrollment process**.

In the following set of examples, we will explore what happens after a user first boots a MacBook that was previously enrolled DEP and configured for MDM enrollment. We will discuss, in detail, how a user is first presented with the option to approve remote configuration and what happens when this configuration is accepted.

Architecture

Upon first boot, the user is presented with the Setup Assistant. This application triggers the enrollment process but does not implement it. Rather, the code is made available through a framework library intended for internal components. This allows much of the DEP and/or MDM enrollment flow to be triggered from multiple locations.
The architecture of the key operating system components can be visualized with the following diagram:

Note that this architecture diagram is not complete, does not cover all possible communication channels, excludes networking and is only known to be accurate as of macOS High Sierra 10.13.6.

ConfigurationProfiles.framework

This private Objective-C library provides simple, high-level management of configuration profiles on macOS. It is used by various user-facing operating system components, and as such, we consider it to be the "entry point" to the enrollment process. As it is technically responsible for processing configuration profiles, it is responsible for dealing with a variety of payload types, and it sports a plugin-based architecture for doing so. It delegates work to various daemons and services to accomplish its work:

- **`ManagedClient.app`**: A LaunchDaemon (with an auxiliary LaunchAgent), this core component is responsible for implementing much of the functionality exposed by ConfigurationProfiles.framework.
    - It is so-named because it also implements the older MCX/Managed Client/Open Directory-based client management system.
- **`cloudconfigurationd`**: A LaunchDaemon, it is responsible for early-stage communication with Apple servers for retrieving DEP configuration.
- **`mdmclient`**: A LaunchDaemon or LaunchAgent (depending on the context), it is responsible for implementing the MDM protocol and executing MDM commands.

We will describe the various steps of the enrollment process through the lens of this framework, by mapping each step to its exported function and describing its operation.

Step 4: Retrieving the activation record

This step begins as soon as the device establishes a connection to the internet and its purpose is to perform the DEP "check-in". This amounts to retrieving the device's "activation record". This record very similar—if not identical—to the (somewhat confusingly named) "DEP profiles" MDM vendors can configure using the public DEP web services (see particularly the https://mdmenrollment.apple.com/profile endpoint)[36].

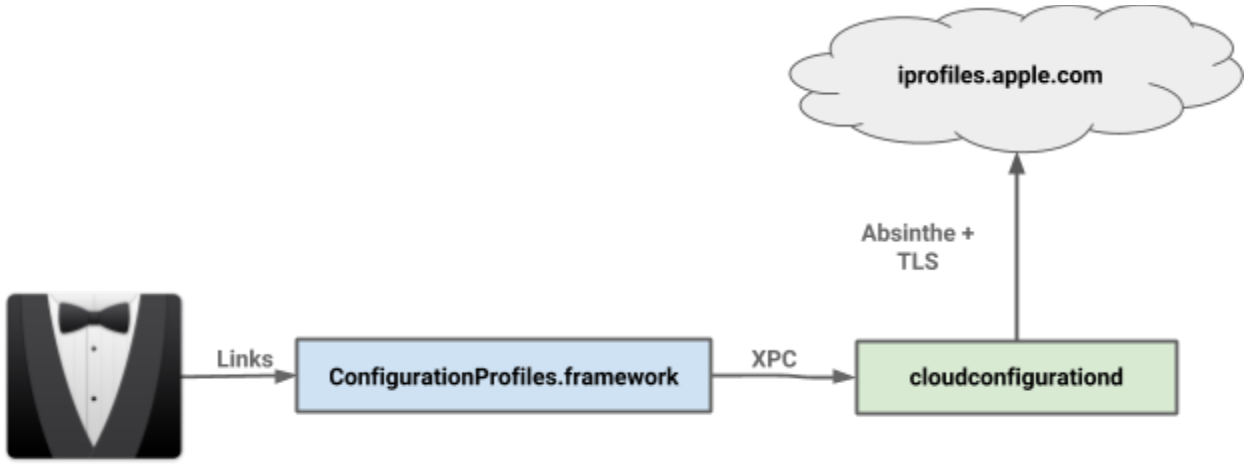
[36] Page 129, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.

An activation record is encoded (at this stage) in JSON. A simple activation record may contain the following:

```
{
  "org": {
    "name": "Fleetsmith, Inc.",
    "address": "...",
    "phone": "...",
    "email": "...",
  },
  "config": {
    "is-supervised": true,
    "allow-pairing": true,
    "is-mandatory": true,
    "await-device-configured": true,
    "is-mdm-removable": false,
    "is-multi-user": false,
    "url": "...",
    "anchor-certs": [...],
    "skip-buddy-items": [],
    "auto-advance-setup": false
  }
}
```

This maps to the **`CPFetchActivationRecord`** and **`CPGetActivationRecord`** (for caching) functions in the Configuration Profiles framework. The **`CPFetchActivationRecord`** function delegates execution to **`cloudconfigurationd`** using *XPC* through the following interface:

```
@protocol MCTeslaConfigurationFetchInterface <NSObject>
- (void)fetchConfigurationWithCompletionBlock:(void (^)(BOOL, NSDictionary *, NSError
*))completionBlock;
@end
```



The object implementing the fetching process for the activation record, **`MCTeslaConfigurationFetcher`**, is a simple state machine that performs the following steps (approximately):

1. Fetching the initial configuration “bag” (optional):
    - This step does not appear to be enabled by default, and appears to allow Apple to override the various URLs used in the later steps below.

2. Setting up Absinthe:
    - To retrieve the activation record, this daemon encodes data in a rather peculiar way. Internally named “Absinthe”, we’ve surmised this scheme is designed to:
        - uniquely identify a device when communicating with Apple.
        - encrypt messages on a per-device basis.
    - As this encoding will be necessary to identify the device, the first steps in this process are dedicated to initializing this scheme.

    a. **Fetching the certificate**:
        - An HTTP GET request to https://iprofiles.apple.com/resource/certificate.cer is performed.

    b. **Initializing Absinthe** (`NACInit`):
        - This function combines the certificate retrieved prior with other various pieces of device-specific information (e.g. it retrieves the serial number using IOKit).
        - This returns an opaque byte string (NSData).

    c. **Fetching the session**:
        - An HTTP POST request to https://iprofiles.apple.com/session is performed, containing the results of the above initialization.
        - This returns a Base64-encoded, opaque byte string.

    d. **Establishing an Absinthe session** (`NACKeyEstablishment`):
        - This function uses the results of the above request to “finish” initialization.
        - The session is is now considered ready, and will be used to sign future requests.

    - *Note*: Exploring “Absinthe” in more detail is not the focus of this document and is left as an exercise to other researchers.

3. Fetching the record:
    - The daemon is now ready to query the **iprofiles** server for the activation record.
    - The request payload is a simple JSON dictionary:

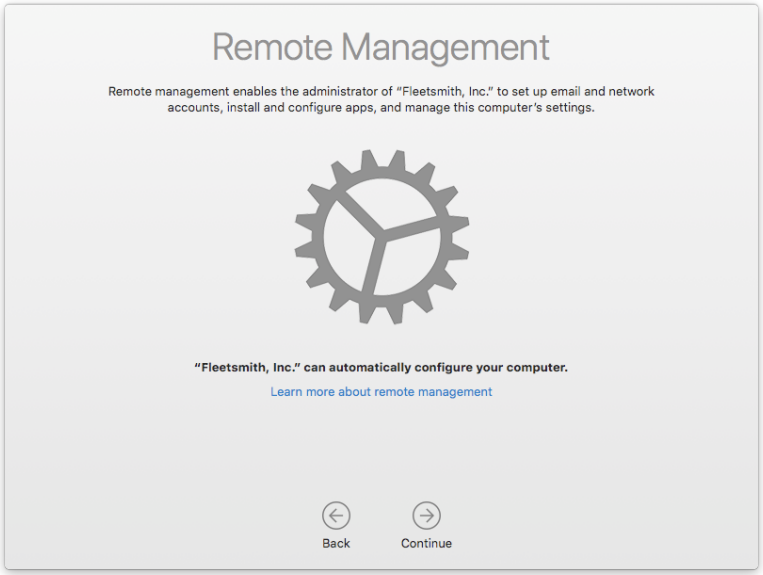
```
{
    “action”: “RequestProfileConfiguration”,
    “sn”: “<serial number>”
}
```

    - The request payload is encrypted using the Absinthe session (`NACSign`) before it is sent.
    - An HTTP POST request to https://iprofiles.apple.com/macProfile is performed, containing the encrypted payload.

- This returns a JSON dictionary representing the activation record.
- After validating it, the dictionary is returned to the caller.

Step 5: Retrieving the activation profile

If the activation record indicates the device is enrolled in DEP, the user is later presented with the option to enable remote configuration:



In this step, the device retrieves the **activation profile**. This must be a valid **configuration profile**; this is publicly documented.[37]

To retrieve the profile, some of the fields configured within the DEP profile by the MDM vendor are now used (via the activation record):

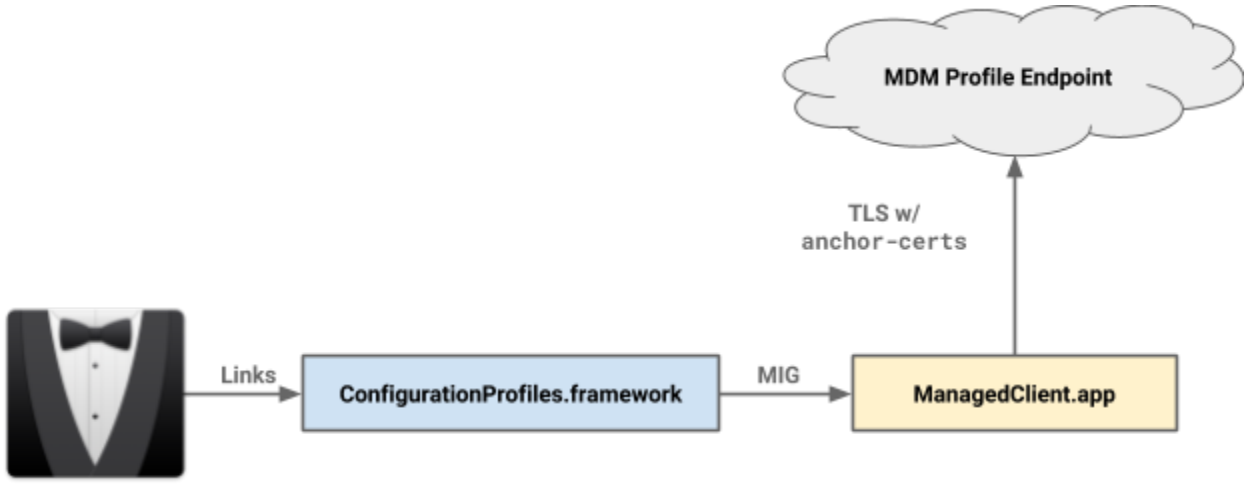
- `url`: Providing a URL allows an MDM vendor to supply an OTA configuration profile.
- `anchor_certs`: Well-behaved MDM servers should also provide anchor certificates to ensure this request's trust is properly evaluated.

This step begins once the user approves by clicking "Next". This maps to the **`CPGetActivationProfile`** function in the Configuration Profiles framework.

This function delegates the actual retrieval of the profile to **`ManagedClient.app`** via a *MIG* endpoint named **`mcxSvr_cloudconfiguration`**.

---

[37] Configuration Profile Reference, Apple, Inc., 2018-07-16 revision.



The exact details of how this request is performed are publicly documented by Apple[38] and are described above.

This (very specific) intermediate certificate chain can be retrieved using the private `SecCertificateCopyiPhoneDeviceCAChain(void)` function in `Security.framework`.

Step 6: Installing the activation profile

Once the activation profile is retrieved, it is installed similarly to any other configuration profile.

Apple supports other means of installing configuration profiles, which is made easy by their flexible payload structures. As mentioned earlier, a profile can be installed by the user (by manually downloading and installing one) or even through MDM at a later date. Thus, an **activation profile** is in reality a **configuration profile** that happens to be delivered by DEP.

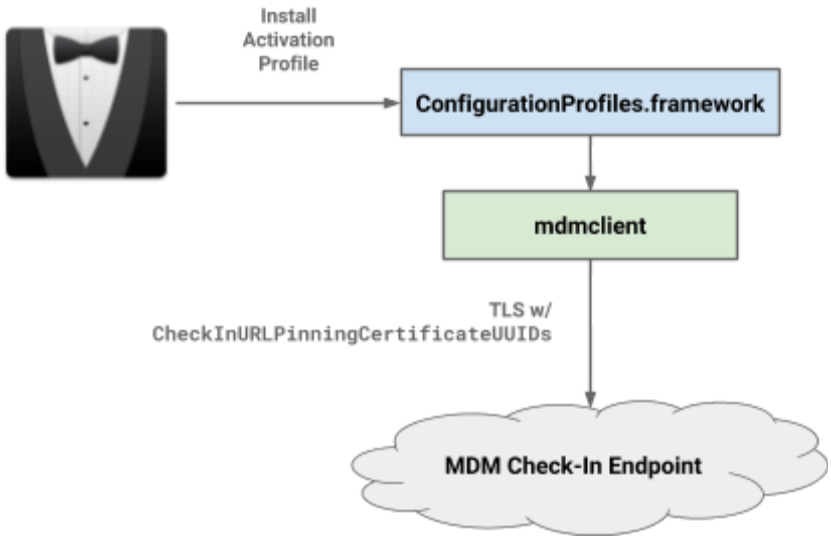
As a configuration profile can contain multiple payloads, each payload must be evaluated and installed. For the purposes of this example, we will focus on the MDM-related payloads.

This step begins, when driven by the Setup Assistant, immediately after the activation profile is retrieved. This maps to the **`CPInstallActivationProfile`** function in the Configuration Profiles framework.

This function delegates much of its work to the **`mdmclient`** daemon via XPC (and through a simple shim named `MCXToolsInterface`). This daemon supports a variety of requests over the following interface:

```
@protocol XPC_MDM_PrivateProtocol
- (void)processRequest:(NSDictionary *)arg1 withReply:(void (^)(NSDictionary *))arg2;
@end
```

[38] Page 129, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.



In this case, the framework issues an **`InstallActivationProfile`** request to the daemon, including a serialized version of the profile. For these purposes, the activation profile is represented using an instance of the **`CPProfile`** class in `ConfigurationProfiles.framework`.

The installation process for **`InstallActivationProfile`** is roughly as follows:

1. DEP-specific information in the profile and its retrieval is processed.
    a. For example, its origin as a DEP profile is asserted.
2. The profile is then installed normally by invoking the `InstallProfile` command (the remaining steps below comprise this command).
3. The profile's payloads are then each validated, then installed:
    a. To achieve this, both **`ManagedClient.app`** and **`ConfigurationProfiles.framework`** sport a plugin-based architecture.
    b. Various XPC services (in the case of the framework) and/or PlugIns (`.profileDomainPlugin` in the case of the application) are used to install specific payload types.
4. The SCEP payload (**`com.apple.security.scep`**) is used to install certificates on the device, such as an additional trusted root CA.
    a. This is necessary for MDM vendors that wish to allow mutual TLS authentication with managed devices by installing vendor-generated client certificates on the device.
    b. This payload is managed by `CertificateService.xpc`, which is responsible for installing it.
    c. This payload is installed using **`SCEP.framework`**, a simple private implementation of the SCEP. This framework is able to abstract away of the use

of SCEP to produce **`Security.framework`** objects, which are simpler to manipulate.

5. The MDM payload (**`com.apple.mdm`**) is used to enable the use of the MDM protocol.
    a. This is used to configure the MDM enrollment for the device, including the device-specific certificate, the MDM server URLs and the APNs topic used for push notifications[39].
    b. This payload is managed by `MDMService.xpc`, which (interestingly) issues installation commands *back to* **`mdmclient`** itself (this reflects the recursive nature of configuration profiles).
    c. The installation of an MDM payload amounts to what is publicly documented as the "check-in protocol"[40]:
        i. The device contacts the MDM server at the check-in url (via the `CheckInURL` property, if it is provided, otherwise `ServerURL`).
        ii. Well-behaved MDM vendors can (and should) provide anchor certificates via the profile and matching identifiers in the MDM payload to ensure the check-in URL's trust is validated properly (via the `CheckInURLPinningCertificateUUIDs` property).
        iii. Internally, the `MDMConnectionHelper` object is responsible for communications in the check-in process, and will make use of the URL and of the pinning UUIDs to securely contact the vendor.

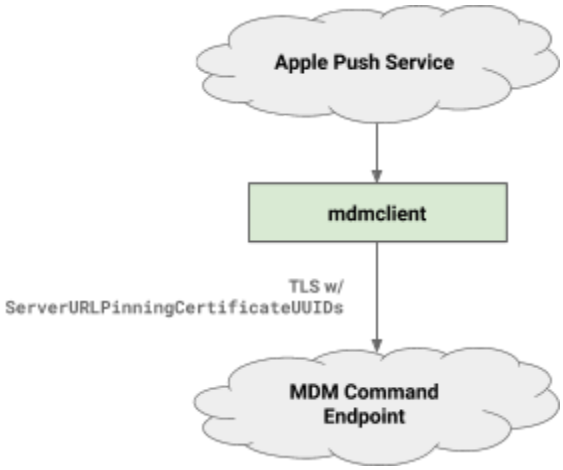
            This is achieved using a rather standard `NSURLSession` object and its `didReceiveChallenge` delegate method.

Step 7: Listening for commands

Once the activation profile is installed, the device is ready to receive instructions from the MDM server.

This step begins automatically after the installation of the activation profile. It is implemented in **`mdmclient`**, along with the rest of the MDM protocol.

[39] Page 16, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.
[40] Page 10, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.



Generally, it executes the following loop, as is publicly documented:[41]

1. The server sends a push notification using APNs to the device.
2. The device contacts the server to receive instructions.
    a. The device contacts the MDM server using the primary URL (via the `ServerURL` property).
    b. Well-behaved MDM vendors can (and should) provide anchor certificates via the MDM payload to ensure the URL's trust is validated properly (via the `ServerURLPinningCertificateUUIDs` property).
3. The device performs the command.

Vulnerability: InstallApplication

We will recapitulate this flow from the perspective of an MDM vendor. So far, what can an MDM vendor do to prevent malicious actors from impersonating it?

- **Step 1**: This step is primarily under Apple's control.
- **Step 2**: The vendor can provide anchor certificates in the DEP profile for when the device retrieves the MDM enrollment configuration profile.
- **Step 3**: The vendor can assign a device an identifying certificate via the SCEP payload of the configuration profile, and refer to this certificate in the MDM payload. It is then used as the client certificate when performing MDM check-in.
- **Step 4**: The vendor can provide anchor certificates for the MDM protocol via various payloads of the configuration profile, and refer to these certificates in the MDM payload. They are then used to evaluate trust when performing MDM check-in, as well as later requests of the MDM protocol.

[41] Page 14, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.

This leaves the various commands which a vendor can issue to a device. These are highly varied and wide-ranging in purpose and scope.

One of the most popular commands provided by the MDM protocol is **`InstallApplication`**[42] . It allows an MDM vendor to install an application package (`.pkg`) on the device, as well as later manage its life cycle. The package must signed with an "Developer ID Installer Certificate" and must be a "distribution" style package. Normal flat packages may be converted to a distribution package by issuing the `productbuild` tool:[43]

```
productbuild --package someFlatPkg.pkg myNewPkg.pkg
```

The `InstallApplication` command works by providing a manifest URL returning an XML file containing the details of the package (the format for such packages matches what Xcode provides in some scenarios, and is publicly documented[44]). The MDM vendor may or may not be hosting this URL. Upon retrieval of the manifest, the system installs the application described therein, downloading any packages and assets as need be.

A manifest is encoded as an XML property-list (or `.plist`). A simple manifest may contain the following:

```
<?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>items</key>
        <array>
            <dict>
                <key>assets</key>
                <array>
                    <dict>
                        <key>kind</key>
                        <string>software-package</string>
                        <key>md5-size</key>
                        <integer></integer>
                        <key>md5s</key>
                        <array>
                            <string>...</string>
                        </array>
                        <key>url</key>
                        <string>https://link/to/pkg</string>
                    </dict>
                </array>
            </dict>
        </array>
    </dict>
</plist>
```

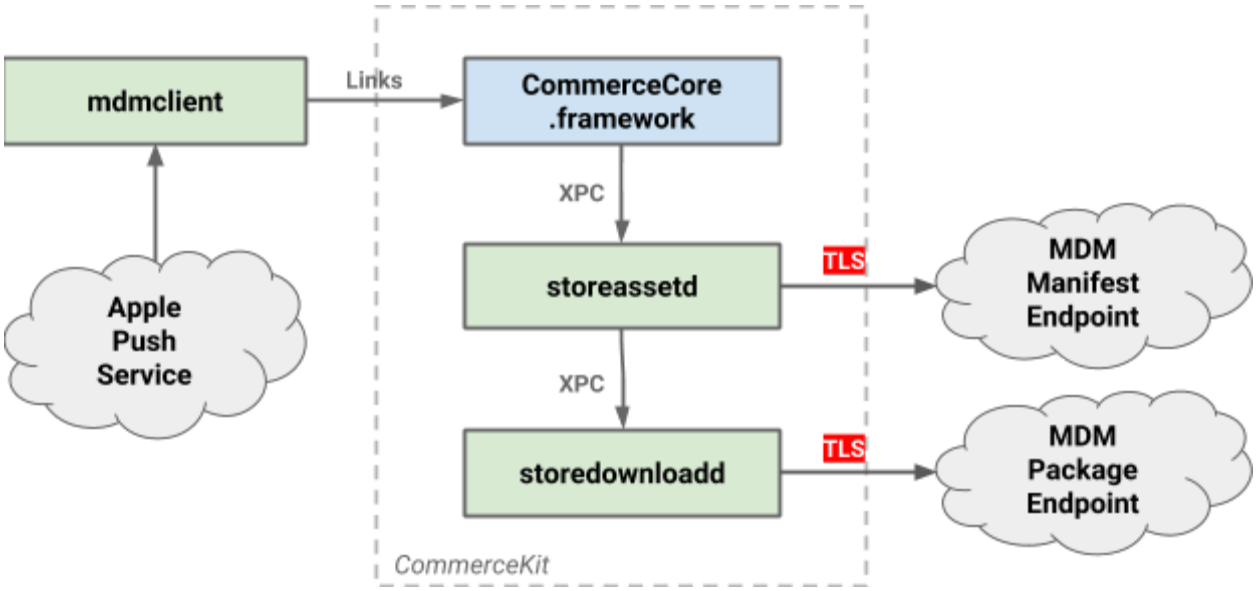
[42] Page 10, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.

[43] https://github.com/micromdm/micromdm/wiki/distributing-packages-with-InstallApplication#creating-and-signing-packages

[44] iOS Deployment Reference, https://help.apple.com/deployment/ios/#/apd11fd167c4, Apple, Inc., retrieved on 2018-07-18.

However, the manifest URL's trust evaluation is not under the control of **`mdmclient`**. To process a manifest, the daemon leverages the Mac App Store infrastructure's private APIs.

The following is an updated diagram that illustrates the architecture in play when installing an application via InstallApplication, as may occur as part of first boot:



To trigger the download and installation of an application's manifest, **`mdmclient`** calls the **`CKMDMProcessManifestAtURL`** function in the private **`CommerceCore.framework`**. We will take a closer look at this function's implementation.

To do its work, this function establishes a connection to the `storelegacy` daemon (a LaunchAgent) over XPC (via port `com.apple.storeagent-xpc`). This daemon is (unsurprisingly) is a part of the `CommerceKit` framework (as is `CommerceCore`).

This message is directly handled by the `_handleMDMMessage` function, but this binary is largely a shim from what appears to be a legacy API to a more modern version built over `StoreFoundation.framework`. This framework consists of multiple services, each with its own interface and implemented in its own daemon process.

In this case, `storelegacy` routes the request to the "asset" service, implemented in **`storeassetd`** (a LaunchAgent and/or LaunchDaemon, depending on the context). This service exposes the following XPC interface (simplified for our purposes):

```
@protocol ISAssetService
- (void)processMDMManifestAtURL:(NSURL *)url options:(NSDictionary *)options
replyBlock:(void (^)(BOOL, NSArray *, NSError *))reply;
```

```
@end
```

To respond to the call to **`processMDMManifestAtURL`**, `storeassetd` creates an instance of the **`MDMManifestCheckOperation`** class. This object is responsible for installing MDM-provided manifests, in the following process:

1. The manifest is downloaded synchronously from the provided URL:
    a. This is achieved using helper classes for networking shared with other Store daemons. In this case, the **`ISURLRequest`** and **`ISURLOperation`** classes are used (from `StoreFoundation.framework`).
    b. These objects make rather standard use of `NSURLConnection` and its delegates.
    c. In macOS 10.13.5 and below, these classes do not evaluate trust. They rely on the system to do so using standard means.
2. The manifest is processed:
    a. The downloaded manifest is parsed, its properties evaluated.

The request is now dispatched (provided the manifest was valid) to the “download” service, implemented in `storedownloadd` (a LaunchAgent and/or LaunchDaemon). This service exposes the following XPC interface (simplified for our purposes):

```
@protocol ISDownloadService
- (void)performDownload:(SSDownload *)download withOptions:(unsigned long long)options
replyBlock:(void (^)(unsigned long long, NSError *))reply;
@end
```

From our observations, the download (and subsequent installation) largely follows the same code paths as when a user regularly installs an application from the Mac App Store.

Since there is no indication that server trust is evaluated beyond simple means, **it is possible to for a man-in-the-middle attack to compromise the retrieval of the manifest itself**. This could allow an attacker to make a device install another application during the Setup Assistant process.

We theorize this is partly caused by **`mdmclient`** and `StoreFoundation` being a part of two different operating system components with different/separate threat models. As the Store is primarily concerned with downloading and providing Apple-blessed packages (which are code-signed) from Apple’s servers (seemingly in the majority of cases), it is possible that pinning-based trust evaluation was considered to add little value. However, in the case of MDM-based installations, this can create a vulnerability.

Fix: InstallEnterpriseApplication

We disclosed the issue to Apple shortly after discovering it. Based on our feedback, a fix was quickly implemented in the form of a new MDM command: **`InstallEnterpriseApplication`**, which is now documented publicly[45].

This command (available as of macOS 10.13.6) allows MDM vendors to provide specific certificates to pin the request to the `ManifestURL` (using the new `ManifestURLPinningCerts` property of said command). It is up to the MDM vendor to implement this, but this serves as an adequate solution to this problem. We will take a closer look at how the vulnerability was addressed.

In newer versions the affected libraries, the **`CKMDMProcessManifestAtURL`** function's second parameter is now an **`NSDictionary`** that can contain the following keys:

- **`_CKMDMManifestOptionPinCertificates`**: An array of DER-encoded certificates which are used to pin the manifest request. This matches the `ManifestURLPinningCerts` property.
- **`_CKMDMManifestOptionPinningRevocationCheckRequired`**: A boolean indicating whether revocation checks should be performed. This matches the `PinningRevocationCheckRequired` property.

As before, the object managing the process in `storeassetd` is **`MDMManifestCheckOperation`** but this object's implementation has been altered. Its `_fetchManifestAtURL` method now makes use of these options to configure the `ISURLOperation`:

```
[operation setRequireExtendedValidationCertificate:0x0];
certs = [[self options] objectForKeyedSubscript:_CKMDMManifestOptionPinCertificates];
[operation setAnchorCertificates:certs];
revocation_check = [[self options]
objectForKeyedSubscript:_CKMDMManifestOptionPinningRevocationCheckRequired];
[operation setAnchorRevocationCheck:[revocation_check boolValue]];
```

Further, the `StoreFoundation` classes responsible for networking are now capable of evaluating server trust. Specifically, `ISURLOperation`'s delegate now implements the standard `willSendRequestForAuthenticationChallenge`:

1. The underlying `SecTrustRef` for the challenge is retrieved via:

   ```
   [[challenge protectionSpace] serverTrust]
   ```

2. The certificates are parsed from the options using `SecCertificateCreateWithData`.
3. The certificates are added as anchors to the `SecTrustRef` (using `SecTrustSetAnchorCertificates`).

[45] Page 57, MDM Protocol Reference, Apple, Inc., 2018-07-16 revision.

4. The policies for the `SecTrustRef` are set to perform a revocation check if the options dictionary so indicated that option (using `SecPolicyCreateRevocation` and `SecTrustSetPolicies`).

Conclusion

Takeaways

- Various OS components have different threat models, so bugs can appear at their intersection.
- As MDM has a wide surface area of interaction within the operating system, this increases the risk of vulnerabilities.
- With macOS 10.13.6 and 10.14, all communications between a Mac device and an MDM vendor can be pinned except SCEP.
- Security is often dependent on the MDM vendor. And vendors can do more. Which brings us to...

MDM Vendor Product Security Checklist

Based on our observations, we make the following recommendations to other MDM vendors:

- **Pin everywhere**
  - DEP profile
  - MDM enrollment payload
  - InstallEnterpriseApplication
  - Inside agent binary (if one exists)
- **SCEP**
  - Use it! Don't generate private keys server side and then distribute them to clients (at least 1 vendor does this).
  - Use the SCEP Challenge Password field—make sure it's more than just a text string. HMAC is a good approach.
- **Configuration Profiles**
  - All should be signed.
  - All that contain sensitive data should be signed (e.g. using an Apple Developer Certificate) and encrypted (e.g. using the device's public key).
- **Encrypt sensitive customer data at rest (e.g. WiFi passwords).**

In addition to vendors doing more, Apple has more control than anyone to improve the security of DEP & MDM.

Recommendations for Apple

Make DEP & MDM **secure by design**, regardless of which product/vendor a customer chooses:

- Fully document the security model for DEP & MDM, including the role of the Apple iPhone Device CA.
- Renew the expired iPhone Device CA cert built into macOS (used for device identity), so that the chain can be properly validated.
- Require all MDM vendors comply with an MDM security checklist similar to the one proposed here within 12—18 months.
- Make factory installed OS version and build number available via DEP “cloud service” API, so MDM can show that info in-product, allowing customers to know whether new devices are vulnerable.
- Enforce (in the OS) that any Configuration Profile that contains a sensitive value (password, private key, etc.) must be both signed and encrypted as a defense-in-depth measure.
- Unify SCEP implementations across macOS & iOS, and document Apple’s implementation (current docs just point at the RFC).

Appendix

Trust hierarchy on macOS

***with* T2 chip**

- MDM vendor servers
- MDM vendor CAs
- DEP servers
- Commercial CAs
- macOS binaries involved in MDM & DEP
- macOS
- Firmware
- SEP
- Crypto engine chip
- Apple Root CAs[46]
    - Apple X86 Secure Boot Root CA - G1
    - T8012Mac-TssLive-ManifestKeyGlobal-RevB-DataCenter

***without* T2 chip**

- MDM vendor servers
- MDM vendor CAs
- DEP servers
- Commercial CAs
- macOS binaries involved in MDM & DEP
- macOS

Authentication methods

SCEP

- Challenge Password field[47]

DEP

- TLS client certificates (issued by Apple) to talk to DEP reseller APIs
- OAuth tokens (MDM vendor servers to Apple)
- Device identity certificate (device to Apple)

---

[46] http://michaellynn.github.io/2018/07/27/booting-secure/

[47] https://www.cisco.com/c/en/us/support/docs/security-vpn/public-key-infrastructure-pki/116167-technote-scep-00.html

MDM

- TLS client certificates

List of macOS binaries related to MDM

mdmclient
Full path: /usr/libexec/mdmclient

Entitlements:

```
<?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>com.apple.ManagedClient.cloudconfigurationd-access</key>
<true/>
<key>com.apple.accounts.inactive.fullaccess</key>
<true/>
<key>com.apple.avfoundation.allow-system-wide-context</key>
<true/>
<key>com.apple.avfoundation.allows-access-to-device-list</key>
<true/>
<key>com.apple.avfoundation.allows-set-output-device</key>
<true/>
<key>com.apple.logd.admin</key>
<true/>
<key>com.apple.private.AuthorizationServices</key>
<array>
<string>com.apple.trust-settings.user</string>
<string>system.install.apple-software</string>
<string>system.install.apple-software.standard-user</string>
</array>
<key>com.apple.private.accounts.allaccounts</key>
<true/>
<key>com.apple.private.admin.writeconfig</key>
<true/>
<key>com.apple.private.appstored</key>
<array>
<string>upp</string>
</array>
<key>com.apple.private.aps-client-cert-access</key>
<true/>
<key>com.apple.private.aps-connection-initiate</key>
<string>com.apple.mgmt.</string>
<key>com.apple.private.bmk.allow</key>
<true/>
<key>com.apple.private.commerce</key>
<array>
<string>Accounts</string>
</array>
<key>com.apple.private.configurationprofiles.readwrite</key>
<true/>
<key>com.apple.private.coreservices.canmanagebackgroundtasks</key>
<true/>
<key>com.apple.private.dark-wake-push</key>
<true/>
<key>com.apple.private.efilogin-helper</key>
<true/>
<key>com.apple.private.findmymacd.spi</key>
```

```
<true/>
<key>com.apple.private.iaaccounts</key>
<true/>
<key>com.apple.private.iokit.interactive-push</key>
<true/>
<key>com.apple.private.managedclient.mdmclient-private</key>
<true/>
<key>com.apple.private.managedclient.profiledomainservice.host</key>
<true/>
<key>com.apple.private.networkextension.configuration-may-skip-prompt</key>
<true/>
<key>com.apple.private.notificationcenter-system</key>
<array>
<dict>
<key>identifier</key>
<string>com.apple.mdmclient</string>
</dict>
<dict>
<key>identifier</key>
<string>com.apple.mdmclient.cloudconfig</string>
</dict>
</array>
<key>com.apple.private.push-to-wake</key>
<true/>
<key>com.apple.private.security.allow-migration</key>
<true/>
<key>com.apple.private.tcc.allow</key>
<array>
<string>kTCCServiceAddressBook</string>
<string>kTCCServiceCalendar</string>
<string>kTCCServiceReminders</string>
<string>kTCCServiceAppleEvents</string>
</array>
<key>com.apple.rootless.storage.ConfigurationProfilesPrivate</key>
<true/>
<key>com.apple.wifi.associate</key>
<true/>
<key>com.apple.wifi.scan</key>
<true/>
<key>com.apple.wifi.set_power</key>
<true/>
<key>keychain-access-groups</key>
<array>
<string>apple</string>
<string>appleaccount</string>
</array>
</dict>
</plist>
```

ManagedClient

Full path: /System/Library/CoreServices/ManagedClient.app/Contents/MacOS/ManagedClient

Entitlements:

```
<?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>com.apple.ManagedClient.cloudconfigurationd-access</key>
<true/>
<key>com.apple.keystore.config.set</key>
<true/>
```

```
<key>com.apple.locationd.authorizeapplications</key>
<true/>
<key>com.apple.private.accounts.allaccounts</key>
<true/>
<key>com.apple.private.admin.writeconfig</key>
<true/>
<key>com.apple.private.aps-client-cert-access</key>
<true/>
<key>com.apple.private.aps-connection-initiate</key>
<true/>
<key>com.apple.private.bmk.allow</key>
<true/>
<key>com.apple.private.managedclient.mdmclient-private</key>
<true/>
<key>com.apple.private.opendirectoryd.modify_uuid</key>
<true/>
<key>com.apple.private.opendirectoryd.securetoken</key>
<true/>
<key>com.apple.private.security.storage.Safari</key>
<true/>
<key>com.apple.private.security.storage.universalaccess</key>
<true/>
<key>com.apple.rootless.storage.ConfigurationProfilesPrivate</key>
<true/>
<key>com.apple.security.device.bluetooth</key>
<true/>
<key>com.apple.security.system-groups</key>
<array>
<string>systemgroup.com.apple.powerlog</string>
</array>
<key>com.apple.wifi.associate</key>
<true/>
<key>com.apple.wifi.scan</key>
<true/>
<key>com.apple.wifi.set_power</key>
<true/>
<key>keychain-access-groups</key>
<array>
<string>apple</string>
</array>
</dict>
</plist>
```

cloudconfigurationd
Full path: /usr/libexec/cloudconfigurationd
Entitlements: None

storeassetd
Full path:

/System/Library/PrivateFrameworks/CommerceKit.framework/Versions/A/Resources/storeassetd

Entitlements:

```
<?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>com.apple.developer.notificationcenter-identifiers</key>
    <array>
            <dict>
                    <key>NSUserNotificationAlertStyle</key>
                    <string>alert</string>
                    <key>_NSUserNotificationHideFromPrefs</key>
                    <true/>
                    <key>identifier</key>
                    <string>com.apple.appstore</string>
            </dict>
    </array>
    <key>com.apple.private.AuthorizationServices</key>
    <array>
            <string>system.install.app-store-software</string>
            <string>system.install.apple-software</string>
            <string>system.install.app-store-software.standard-user</string>
            <string>system.install.apple-software.standard-user</string>
            <string>system.install.software.mdm-provided</string>
    </array>
    <key>com.apple.private.adid</key>
    <true/>
    <key>com.apple.private.bookkit</key>
    <true/>
    <key>com.apple.private.domain-extension</key>
    <true/>
    <key>com.apple.private.fpsd.client</key>
    <true/>
    <key>com.apple.private.launchpad.list</key>
    <true/>
    <key>com.apple.private.managedclient.configurationprofiles</key>
    <true/>
    <key>com.apple.private.mas-receipt-installer</key>
    <true/>
    <key>com.apple.private.notificationcenter-system</key>
    <array>
            <dict>
                    <key>_NSUserNotificationHideFromPrefs</key>
                    <true/>
                    <key>identifier</key>
                    <string>com.apple.storeassetd</string>
            </dict>
    </array>
    <key>com.apple.private.rtcreportingd</key>
    <true/>
    <key>com.apple.private.securityd.stash</key>
    <true/>
    <key>com.apple.private.storeinstallagent.install-request</key>
    <true/>
    <key>com.apple.private.storeinstalld.install-request</key>
    <true/>
    <key>com.apple.security.temporary-exception.apple-events</key>
    <array>
            <string>com.apple.loginwindow</string>
    </array>
</dict>
</plist>
```

storedownloadd

Full path:
/System/Library/PrivateFrameworks/CommerceKit.framework/Versions/A/Resources/storedownloadd

Entitlements:

```
<?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>com.apple.duet.activityscheduler.allow</key>
    <true/>
    <key>com.apple.private.AuthorizationServices</key>
    <array>
            <string>system.install.app-store-software</string>
            <string>system.install.apple-software</string>
            <string>system.install.app-store-software.standard-user</string>
            <string>system.install.apple-software.standard-user</string>
            <string>system.install.software.mdm-provided</string>
    </array>
    <key>com.apple.private.adid</key>
    <true/>
    <key>com.apple.private.bookkit</key>
    <true/>
    <key>com.apple.private.domain-extension</key>
    <true/>
    <key>com.apple.private.fpsd.client</key>
    <true/>
    <key>com.apple.private.launchservices.cansetapplicationstrusted</key>
    <true/>
    <key>com.apple.private.librarian.container-proxy</key>
    <true/>
    <key>com.apple.private.mas-display-ui</key>
    <true/>
    <key>com.apple.private.mas-receipt-installer</key>
    <true/>
    <key>com.apple.private.rtcreportingd</key>
    <true/>
    <key>com.apple.private.securityd.stash</key>
    <true/>
    <key>com.apple.security.temporary-exception.apple-events</key>
    <array>
            <string>com.apple.loginwindow</string>
    </array>
</dict>
</plist>
```

List of Apple server URLs

MDM (APNs)

https://api.push.apple.com — actual server

https://identity.apple.com/pushcert — customer portal

DEP ↔ MDM server

https://mdmenrollment.apple.com

DEP ↔ Resellers

Bulk Enroll Devices API:

Dev/Test environment:
https://acc-ipt.apple.com/enroll-service/1.0/bulk-enroll-devices

Joint UAT environment:
https://api-applecareconnect-ept.apple.com/enroll-service/1.0/bulk-enroll-devices
(even soldToId)
https://api-applecareconnect-ept2.apple.com/enroll-service/1.0/bulk-enroll-devices
(odd soldToId)

Check Transaction Status API:

Dev/Test environment:
https://acc-ipt.apple.com/enroll-service/1.0/check-transaction-status

Joint UAT environment:
https://api-applecareconnect-ept.apple.com/enroll-service/1.0/check-transaction-status
(even soldToId)
https://api-applecareconnect-ept2.apple.com/enroll-service/1.0/check-transaction-status
(odd soldToId)

Show Order Details API:

Dev/Test environment:
https://acc-ipt.apple.com/enroll-service/1.0/show-order-details

Joint UAT environment:
https://api-applecareconnect-ept.apple.com/enroll-service/1.0/show-order-details
(even soldToId)
https://api-applecareconnect-ept2.apple.com/enroll-service/1.0/show-order-details
(odd soldToId)

DEP ↔ Devices

Domain: iprofiles.apple.com

openssl output

api-applecareconnect.apple.com

```
openssl s_client -showcerts -servername api-applecareconnect.apple.com -connect
api-applecareconnect.apple.com:443
```

```
CONNECTED(00000006)
depth=2 C = US, O = "VeriSign, Inc.", OU = VeriSign Trust Network, OU = "(c) 2006 VeriSign,
Inc. - For authorized use only", CN = VeriSign Class 3 Public Primary Certification
Authority - G5
verify error:num=19:self signed certificate in certificate chain
verify return:0
---
Certificate chain
 0
s:/1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=California/businessCategory=Private
Organization/serialNumber=C0806592/C=US/postalCode=95014/ST=California/L=Cupertino/street=1
Infinite Loop/O=Apple Inc./OU=IS&T/CN=acc-nwk-waf.apple.com
   i:/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA -
G3
-----BEGIN CERTIFICATE-----
MIIIKTCCBxGgAwIBAgIQK7EZ0BU6rCrmv5vJ3kIrCzANBgkqhkiG9w0BAQsFADB3
MQswCQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAd
BgNVBAsTFlN5bWFudGVjIFRydXN0IE5ldHdvcmsxKDAmBgNVBAMTH1N5bWFudGVj
IENsYXNzIDMgRVYgU1NMIENBIC0gRzMwHhcNMTcwOTAyMDAwMDAwWhcNMTkwOTAz
MjM1OTU5WjCCAQgxEzARBgsrBgEEAYI3PAIBAxMCVVMxGzAZBgsrBgEEAYI3PAIB
AgwKQ2FsaWZvcm5pYTEdMBsGA1UEDxMUUHJpdmF0ZSBPcmdhbml6YXRpb24xETAP
BgNVBAUTCEMwODA2NTkyMQswCQYDVQQGEwJVUzEOMAwGA1UEEQwFOTUwMTQxEzAR
BgNVBAgMCkNhbGlmb3JuaWExEjAQBgNVBAcMCUN1cGVydGlubzEYMBYGA1UECQwP
MSBJbmZpbml0ZSBMb29wMRMwEQYDVQQKDApBcHBsZSBJbmMuMQ0wCwYDVQQLDARJ
UyZUMR4wHAYDVQQDDBVhY2MtbndrLXdhZi5hcHBsZS5jb20wggEiMA0GCSqGSIb3
DQEBAQUAA4IBDwAwggEKAoIBAQDhSbLdlJxMeXsavZifsQ0NqldDpUJwonzvhkWJ
0izpe9c+e+LWIDFHm/QgYQY0tJicP3bGA5FN6feR5YACaeYPpn/Rmc23A7YCRFA1
ZMQHA8Cnum9Y2oXnbQhfHuwsmmC7E9Nf5rs0dR2lMLz+EW6DUWS7f/gUzc3fTk3Z
aoMJfXqcQggDF+5YgCY4Mx8h8XmKbqEIrd7SWINrWXCvzasJJPinXTSOZYHFQZB6
/pKssTdOOUq7Uyw60A6tla4qN79qIsQEX1bz5sOItFVmlC3O6jojubsra3833vCq
fZOniPa2POofFOZZb7IxTUGL6YYY5IayKeq9KpnrYhxdPns3AgMBAAGjggQcMIIE
GDCCAT8GA1UdEQSCATYwggEyghVhY2MtbndrLXdhZi5hcHBsZS5jb22CEWFjYy1u
d2suYXBwbGUuY29tghphcHBsZWNhcmVjb25uZWN0LmFwcGxlLmNvbYIZYXBpLWFj
Yy1ud2std2FmLmFwcGxlLmNvbYIVYXBpLWFjYy1ud2suYXBwbGUuY29tgh5hcGkt
YXBwbGVjYXJlY29ubmVjdC5hcHBsZS5jb22CFWFjYy1tZG4td2FmLmFwcGxlLmNv
bYIRYWNjLW1kbi5hcHBsZS5jb22CG2FwcGxlY2FyZWNvbm5lY3QyLmFwcGxlLmNv
bYIZYXBpLWFjYy1tZG4td2FmLmFwcGxlLmNvbYIVYXBpLWFjYy1tZG4uYXBwbGUu
Y29tgh9hcGktYXBwbGVjYXJlY29ubmVjdDIuYXBwbGUuY29tMAkGA1UdEwQCMAAw
DgYDVR0PAQH/BAQDAgWgMB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjBv
BgNVHSAEaDBmMFsGC2CGSAGG+EUBBxcGMEwwIwYIKwYBBQUHAgEWF2h0dHBzOi8v
ZC5zeW1jYi5jb20vY3BzMCUGCCsGAQUFBwICMBkMF2h0dHBzOi8vZC5zeW1jYi5j
b20vcnBhMAcGBWeBDAEBMB8GA1UdIwQYMBaAFAFZq+fdOgtZpmRj1s8gB1fVkedq
MCsGA1UdHwQkMCIwIKAeoByGGmh0dHA6Ly9zci5zeW1jYi5jb20vc3IuY3JsMFcG
CCsGAQUFBwEBBEswSTAfBggrBgEFBQcwAYYTaHR0cDovL3NyLnN5bWNkLmNvbTAm
BggrBgEFBQcwAoYaaHR0cDovL3NyLnN5bWNiLmNvbS9zci5jcnQwggF/BgorBgEE
AdZ5AgQCBIIBbwSCAWsBaQB2AN3rHSt6DU+mIIuBrYFocH4ujp0B1VyIjT0RxM22
7L7MAAABXkRpMI0AAAQDAEcwRQIhANQ3rUNr0camAniOF+rt+iBUH6b0G/t26/N5
OY4R0QCFAiAZqpkbXiZ3a/Ds4o5ahvbTkIIxAvUiHfK2SlXip7pfhAB3AKS5CZC0
GFgUh7sTosxncAo8NZgE+RvfuON3zQ7IDdwQAAABXkRpMMUAAAQDAEgwRgIhAN8B
VYRYsV4uLakwGqz04dJZmRYEAdneaQQFmkiMK0cBAiEA6JARU+4NBNKbZ4UW/XvP
```

```
2cTIs9Au7NsSBq8pDHkLA4AAdgDuS723dc5guuFCaR+r4Z5mow9+X7By2IMAxHuJ
eqj9ywAAAV5EaTKNAAAEAwBHMEUCIQDCRpQQGISCBhUgtwGeq04s2JNdf5H4K0K6
DlAfgkYxFgIgAgohNt7d7XpEDS13ffk6FMa3ttocAP4QNROzR+PakTMwDQYJKoZI
hvcNAQELBQADggEBAJ3f5NRLBSj6ib0yZgvnErhUhPB+tkT+BXRgeQuKzN2wRTuv
TLnoZhhvhQX1JJo2cCKRHoaQi/RktN1Bx0b5xQkoBWTHe2NVu3kdFtvwVbB6I0ud
fPLtO95F+rA2ZFZblV9RUM2m0Nu2C46PfqxZ3AMZmSfY9GH7MRQ+t+pji9ADauWr
1fxS+uErvcrYxgRUxIqJWgB5JaV4kUnjkl4W5ENnTgMG2bUDqfZA0gEy3t7bf1kU
+GLA3gf6Yioa4y1sdTGEssDo2QCelB/nplQUd5URKjZ5eVrUFMCpT4joNLFdq0g4
EjP4Pbx8j2C3BHb/QiKv7vXndj6WzWZisFSDUHc=
-----END CERTIFICATE-----
 1 s:/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA -
G3
   i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For
authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority - G5
-----BEGIN CERTIFICATE-----
MIIFKzCCBBOgAwIBAgIQfuFKb2/v8tN/P61lTTratDANBgkqhkiG9w0BAQsFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
aG9yaXR5IC0gRzUwHhcNMTMxMDMxMDAwMDAwWhcNMjMxMDMwMjM1OTU5WjB3MQsw
CQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAdBgNV
BAsTFlN5bWFudGVjIFRydXN0IE5ldHdvcmsxKDAmBgNVBAMTH1N5bWFudGVjIENs
YXNzIDMgRVYgU1NMIENBIC0gRzMwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQDYoWV0I+grZOIy1zM3PY71NBZI3U9/hxz4RCMTjvsR2ERaGHGOYBYmkpv9
FwvhcXBC/r/6HMCqo6e1cej/GIP23xAKE2LIPZyn3i4/DNkd5y77Ks7Imn+Hv9hM
BBUyydHMlXGgTihPhNk1++OGb5RT5nKKY2cuvmn2926OnGAE6yn6xEdC0niY4+wL
pZLct5q9gGQrOHw4CVtm9i2VeoayNC6FnpAOX7ddpFFyRnATv2fytqdNFB5suVPu
IxpOjUhVQ0GxiXVqQCjFfd3SbtICGS97JJRL6/EaqZvjI5rq+jOrCiy39GAI3Z8c
zd0tAWaAr7MvKR0juIrhoXAHDDQPAgMBAAGjggFdMIIBWTAvBggrBgEFBQcBAQQj
MCEwHwYIKwYBBQUHMAGGE2h0dHA6Ly9zMi5zeW1jYi5jb20wEgYDVR0TAQH/BAgw
BgEB/wIBADBlBgNVHSAEXjBcMFoGBFUdIAAwUjAmBggrBgEFBQcCARYaaHR0cDov
L3d3dy5zeW1hdXRoLmNvbS9jcHMwKAYIKwYBBQUHAgIwHBoaaHR0cDovL3d3dy5z
eW1hdXRoLmNvbS9ycGEwMAYDVR0fBCkwJzAloCOgIYYfaHR0cDovL3MxLnN5bWNi
LmNvbS9wY2EzLWc1LmNybDAOBgNVHQ8BAf8EBAMCAQYwKQYDVR0RBCIwIKQeMBwx
GjAYBgNVBAMTEVN5bWFudGVjUEtJLTEtNTMzMB0GA1UdDgQWBBQBWavn3ToLWaZk
Y9bPIAdX1ZHnajAfBgNVHSMEGDAWgBR/02Wnwt3su/AwCfNDOfoCrzMxMzANBgkq
hkiG9w0BAQsFAAOCAQEAQgFVe9AWGl1Y6LubqE3X89frE5SG1n8hC0e8V5uSXU8F
nzikEHzPg74GQ0aNCLxq1xCm+quvL2GoY/Jl339MiBKIT7Np2f8nwAqXkY9W+4nE
qLuSLRtzsMarNvSWbCAI7woeZiRFT2cAQMgHVHQzO6atuyOfZu2iRHA0+w7qAf3P
eHTfp61Vt19N9tY/4IbOJMdCqRMURDVLtt/JYKwMf9mTIUvunORJApjTYHtcvNUw
LwfORELEC5n+5p/8sHiGUW3RLJ3GlvuFgrsEL/digO9i2n/2DqyQuFa9eT/ygG6j
2bkPXToHHZGThkspTOHcteHgM52zyzaRS/6htO7w+Q==
-----END CERTIFICATE-----
 2 s:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For
authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority - G5
   i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For
authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority - G5
-----BEGIN CERTIFICATE-----
MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjNzGs7SjANBgkqhkiG9w0BAQUFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
aG9yaXR5IC0gRzUwNDYxMTA4MDAwMDAwWhcNMzYwNzE2MjM1OTU5WjCByjEL
MAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQLExZW
ZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJpU2ln
biwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxWZXJp
U2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0aG9y
aXR5IC0gRzUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCvJAgIKXo1
nmAMqudLO07cfLw8RRy7K+D+KQL5VwijZIUVJ/XxrcgxiV0i6CqqpkKzj/i5Vbex
t0uz/o9+B1fs70PbZmIVYc9gDaTY3vjgw2IIPVQT60nKWVSFJuUrjxuf6/WhkcIz
SdhDY2pSS9KP6HBRTdGJaXvHcPaz3BJ023tdS1bTlr8Vd6Gw9KIl8q8ckmcY5fQG
BO+QueQA5N06tRn/Arr0PO7gi+s3i+z016zy9vA9r911kTMZHRxAy3QkGSGT2RT+
rCpSx4/VBEnkjWNHiDxpg8v+R70rfk/Fla4OndTRQ8Bnc+MUCH7lP59zuDMKz10/
NIeWiu5T6CUVAgMBAAGjgbIwga8wDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8E
```

```
BAMCAQYwbQYIKwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEwHzAH
BgUrDgMCGgQUj+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28udmVy
aXNpZ24uY29tL3ZzbG9nby5naWYwHQYDVR0OBBYEFH/TZafC3ey78DAJ80M5+gKv
MzEzMA0GCSqGSIb3DQEBBQUAA4IBAQCTJEowX2LP2BqYLz3q3JktvXf2pXkiOOzE
p6B4Eq1iDkVwZMXnl2YtmAl+X6/WzChl8gGqCBpH3vn5fJJaCGkgDdk+bW48DW7Y
5gaRQBi5+MHt39tBquCWIMnNZBU4gcmU7qKEKQsTb47bDN0lAtukixlE0kF6BWlK
WE9gyn6CagsCqiUXObXbf+eEZSqVir2G3l6BFoMtEMze/aiCKm0oHw0LxOXnGiYZ
4fQRbxC1lfznQgUy286dUV4otp6F01vvpX1FQHKOtw5rDgb7MzVIcbidJ4vEZV8N
hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
-----END CERTIFICATE-----
---
Server certificate
subject=/1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=California/businessCategory=P
rivate
Organization/serialNumber=C0806592/C=US/postalCode=95014/ST=California/L=Cupertino/street=1
Infinite Loop/O=Apple Inc./OU=IS&T/CN=acc-nwk-waf.apple.com
issuer=/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA
- G3
---
Acceptable client certificate CA names
/UID=identity:idms.group.729488/CN=GRX.ACC1914Prod.AppleCare/OU=management:idms.group.72948
8/O=ELP/DC=Certificate Manager
/CN=Apple Corporate External Authentication CA 1/OU=Certification Authority/O=Apple
Inc./C=US
/CN=Apple Corporate Root CA/OU=Certification Authority/O=Apple Inc./C=US
---
SSL handshake has read 5817 bytes and written 495 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID: 3989BAD2C770674535A8954D6406BBD2A36EF5BDA84A1838E5F21F6D39504755
    Session-ID-ctx:
    Master-Key:
E34CEDB61BD36B4C666B764AE51AF05014E9EA2ABAF51D9A8FBB50F09D8C56E194CB5E0469B1AF5E6880D293E69
ED946
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - a0 1d d8 49 59 c6 76 61-aa a6 d6 05 70 01 c8 3c   ...IY.va....p..<
    0010 - 9b 4f 18 1f 9d 0c 21 0d-3d 8b cc 87 ba 11 8f e4   .O....!.=.......
    0020 - 8d 08 28 0c 49 dc 39 54-9a 8a 10 c6 d4 b9 7e 57   ..(.I.9T......~W
    0030 - 49 06 69 01 8b a6 e5 fb-40 e4 32 0e 3f 22 7a 3b   I.i.....@.2.?"z;
    0040 - bc 10 ba cc 19 8d ab af-c6 b1 61 1d ea 48 47 27   ..........a..HG'
    0050 - 79 66 0f 7b ac 27 84 ea-2f ff e3 19 dc ea 9d 46   yf.{.'../......F
    0060 - d5 ad c7 28 cc 45 d1 d5-f5 04 80 21 85 08 59 ab   ...(.E.....!..Y.
    0070 - b9 ff 47 b3 24 9e ee f1-da c9 4b 54 80 55 cd 58   ..G.$.....KT.U.X
    0080 - 49 ef 31 08 72 03 7c 79-83 ef 2c 1a f2 41 66 46   I.1.r.|y..,..AfF
    0090 - 08 d5 cf 49 16 c6 dd 42-56 56 54 ee 67 0e 7c 73   ...I...BVVT.g.|s
    00a0 - 64 98 5a 96 b5 22 56 fe-20 58 e6 f2 65 46 bb 82   d.Z.."V. X..eF..
    00b0 - af b7 9f 24 6e 51 92 e0-c4 63 f3 52 e1 5a 4e ef   ...$nQ...c.R.ZN.
    00c0 - 9f ac 46 03 00 41 b5 4f-4b a5 6f fe 38 59 ae f2   ..F..A.OK.o.8Y..

    Start Time: 1533060433
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
```

api-applecareconnect-ept2.apple.com

```
openssl s_client -showcerts -servername api-applecareconnect-ept2.apple.com -connect
api-applecareconnect-ept2.apple.com:443
CONNECTED(00000005)
depth=2 C = US, O = "VeriSign, Inc.", OU = VeriSign Trust Network, OU = "(c) 2006 VeriSign,
Inc. - For authorized use only", CN = VeriSign Class 3 Public Primary Certification
Authority - G5
verify error:num=19:self signed certificate in certificate chain
verify return:0
---
Certificate chain
 0
s:/1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=California/businessCategory=Private
Organization/serialNumber=C0806592/C=US/postalCode=95014/ST=California/L=Cupertino/street=1
Infinite Loop/O=Apple Inc./OU=IS&T/CN=acc-ept-waf.apple.com
   i:/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA -
G3
-----BEGIN CERTIFICATE-----
MIIIOjCCByKgAwIBAgIQdJHLXgLoRPX/fBNi3gPsHjANBgkqhkiG9w0BAQsFADB3
MQswCQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAd
BgNVBAsTFlN5bWFudGVjIFRydXN0IE5ldHdvcmsxKDAmBgNVBAMTH1N5bWFudGVj
IENsYXNzIDMgRVYgU1NMIENBIC0gRzMwHhcNMTcwODE4MDAwMDAwWhcNMTkwODE5
MjM1OTU5WjCCAQgxEzARBgsrBgEEAYI3PAIBAxMCVVMxGzAZBgsrBgEEAYI3PAIB
AgwKQ2FsaWZvcm5pYTEdMBsGA1UEDxMUUHJpdmF0ZSBPcmdhbml6YXRpb24xETAP
BgNVBAUTCEMwODA2NTkyMQswCQYDVQQGEwJVUzEOMAwGA1UEEQwFOTUwMTQxEzAR
BgNVBAgMCkNhbGlmb3JuaWExEjAQBgNVBAcMCUN1cGVydGlubzEYMBYGA1UECQwP
MSBJbmZpbml0ZSBMb29wMRMwEQYDVQQKDApBcHBsZSBJbmMuMQ0wCwYDVQQLDARJ
UyZUMR4wHAYDVQQDDBVhY2MtZXB0LXdhZi5hcHBsZS5jb20wggEiMA0GCSqGSIb3
DQEBAQUAA4IBDwAwggEKAoIBAQDPdnI4Zr+N8WOiMJTFmT9c6IYVW68xweH9eKE1
KwykIr6rCTqimWD1IBz806jEJ/0E22kmqM5xI8PjyT3268rBIN9/pWM9IWwII4B0
y3d5NIA4UljzUXQaiuJsIgoTWBrUhRT0XNRcX/b7qv+E9OHRF1Vfx3ajM7BaWAji
+of46xJtUO5yAEF+pJNpPlysFa7+z2qFEvJtenGOuz+G4PLJLHzSsU8eYA9ekGug
b5kNQaRQxc6A9z6D/S9RVjGQxSW42F74rO3lAgpIEHJZUivEVCujXvbfw1qqVVVq
NyUJ9P2pCWpdZN0gjMK4CGAxd8hAWzXfx5oy0buqtJG4reRvAgMBAAGjggQtMIIE
KTCCAVMGA1UdEQSCAUowggFGghVhY2MtZXB0LXdhZi5hcHBsZS5jb22CEWFjYy1l
cHQuYXBwbGUuY29tgh5hcHBsZWNhcmVjb25uZWN0LWVwdC5hcHBsZS5jb22CGWFw
aS1hY2MtZXB0LXdhZi5hcHBsZS5jb22CFWFwaS1hY2MtZXB0LmFwcGxlLmNvbYIi
YXBpLWFwcGxlY2FyZWNvbm5lY3QtZXB0LmFwcGxlLmNvbYIWYWNjLWVwdDItd2Fm
LmFwcGxlLmNvbYISYWNjLWVwdDIuYXBwbGUuY29tgh9hcHBsZWNhcmVjb25uZWN0
LWVwdDIuYXBwbGUuY29tghphcGktYWNjLWVwdDItd2FmLmFwcGxlLmNvbYIWYXBp
LWFjYy1lcHQyLmFwcGxlLmNvbYIjYXBpLWFwcGxlY2FyZWNvbm5lY3QtZXB0Mi5h
cHBsZS5jb20wCQYDVR0TBAIwADAOBgNVHQ8BAf8EBAMCBaAwHQYDVR0lBBYwFAYI
KwYBBQUHAwEGCCsGAQUFBwMCMG8GA1UdIARoMGYwWwYLYIZIAYb4RQEHFwYwTDAj
BggrBgEFBQcCARYXaHR0cHM6Ly9kLnN5bWNiLmNvbS9jcHMwJQYIKwYBBQUHAgIw
GQwXaHR0cHM6Ly9kLnN5bWNiLmNvbS9ycGEwBwYFZ4EMAQEwHwYDVR0jBBgwFoAU
AVmr5906C1mmZGPWzyAHV9WR52owKwYDVR0fBCQwIjAgoB6gHIYaaHR0cDovL3Ny
LnN5bWNiLmNvbS9zci5jcmwwVwYIKwYBBQUHAQEESzBJMB8GCCsGAQUFBzABhhNo
dHRwOi8vc3Iuc3ltY2QuY29tMCYGCCsGAQUFBzAChhpodHRwOi8vc3Iuc3ltY2Iu
Y29tL3NyLmNydDCCAXwGCisGAQQB1nkCBAIEggFsBIIBaAFmAHUA3esdK3oNT6Yg
i4GtgWhwfi6OnQHVXIiNPRHEzbbsvswAAAFd9Rpa6wAABAMARjBEAiAmU56VTPXB
bm+TeiKDRF0iwZXSh8H3KrZ5ilwXZXGMrwIgRnFOho5Mb+CJ09mWIoYGiLOsq+BA
5p9bnn6Sq8qLo/4AdgCkuQmQtBhYFIe7E6LMZ3AKPDWYBPkb37jjd80OyA3cEAAA
AV31GlsdAAAEAwBHMEUCIQCtWwdEABRhoQ4u3nTcJxeuu1vRb7VgH2XeTKElWLQs
yQIgcNVdB9C+ResAtH8qXDPmB8GvUFYdO24aCLKq4AB+emgAdQDuS723dc5guuFC
aR+r4Z5mow9+X7By2IMAxHuJeqj9ywAAAV31GlzyAAAEAwBGMEQCIBCqxVMexgal
cewrTmffBWzw1vW3YmRR77liUJjnUQFIAiBNNGX0VVWANbVWt7l1k2NDuBTEMALB
FDtatljMHIIVBjANBgkqhkiG9w0BAQsFAAOCAQEAsx3fgtPgqdNrS4dEjWV3EwfI
8z2zDdiU6g2z33s0D5ehqb7Vs5m1GOdBHTaw+Ubef9OtMjE2/1QicpQgeibwbbuc
k3xgYJ3DmI1YtDUpxLjGDNoyrkPfB/yco4mBBZdUdPLHmoLLLP/OtAAdo6Txk0Qp
LtRBns03jo4Ytnm88kUHXkghuOMRm1W2WoGKGnml9Q/Dda8+63JQ/xhLj+r5WlXd
QC9Fqc1oRNEfeWRp8QOPBWGUlx5F58tRq45uFNQJL7SvOiqggzNbBx5XVZUkYDWz
+p/XBLBXzk7vJDRWivwCKREz4ySn+bYjjPpOeaOxAJNOmg2V++1kPTgiEI0JPA==
-----END CERTIFICATE-----
 1 s:/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA -
G3
```

```
   i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For
authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority - G5
-----BEGIN CERTIFICATE-----
MIIFKzCCBBOgAwIBAgIQfuFKb2/v8tN/P61lTTratDANBgkqhkiG9w0BAQsFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
aG9yaXR5IC0gRzUwHhcNMTMxMDMxMDAwMDAwWhcNMjMxMDMwMjM1OTU5WjB3MQsw
CQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAdBgNV
BAsTFlN5bWFudGVjIFRydXN0IE5ldHdvcmsxKDAmBgNVBAMTH1N5bWFudGVjIENs
YXNzIDMgRVYgU1NMIENBIC0gRzMwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQDYoWV0I+grZOIy1zM3PY71NBZI3U9/hxz4RCMTjvsR2ERaGHGOYBYmkpv9
FwvhcXBC/r/6HMCqo6e1cej/GIP23xAKE2LIPZyn3i4/DNkd5y77Ks7Imn+Hv9hM
BBUyydHMlXGgTihPhNk1++OGb5RT5nKKY2cuvmn2926OnGAE6yn6xEdC0niY4+wL
pZLct5q9gGQrOHw4CVtm9i2VeoayNC6FnpAOX7ddpFFyRnATv2fytqdNFB5suVPu
IxpOjUhVQ0GxiXVqQCjFfd3SbtICGS97JJRL6/EaqZvjI5rq+jOrCiy39GAI3Z8c
zd0tAWaAr7MvKR0juIrhoXAHDDQPAgMBAAGjggFdMIIBWTAvBggrBgEFBQcBAQQj
MCEwHwYIKwYBBQUHMAGGE2h0dHA6Ly9zMi5zeW1jYi5jb20wEgYDVR0TAQH/BAgw
BgEB/wIBADBlBgNVHSAEXjBcMFoGBFUdIAAwUjAmBggrBgEFBQcCARYaaHR0cDov
L3d3dy5zeW1hdXRoLmNvbS9jcHMwKAYIKwYBBQUHAgIwHBoaaHR0cDovL3d3dy5z
eW1hdXRoLmNvbS9ycGEwMAYDVR0fBCkwJzAloCOgIYYfaHR0cDovL3MxLnN5bWNi
LmNvbS9wY2EzLWc1LmNybDAOBgNVHQ8BAf8EBAMCAQYwKQYDVR0RBCIwIKQeMBwx
GjAYBgNVBAMTEVN5bWFudGVjUEtJLTEtNTMzMB0GA1UdDgQWBBQBWavn3ToLWaZk
Y9bPIAdX1ZHnajAfBgNVHSMEGDAWgBR/02Wnwt3su/AwCfNDOfoCrzMxMzANBgkq
hkiG9w0BAQsFAAOCAQEAQgFVe9AWGl1Y6LubqE3X89frE5SG1n8hC0e8V5uSXU8F
nzikEHzPg74GQ0aNCLxq1xCm+quvL2GoY/Jl339MiBKIT7Np2f8nwAqXkY9W+4nE
qLuSLRtzsMarNvSWbCAI7woeZiRFT2cAQMgHVHQzO6atuyOfZu2iRHA0+w7qAf3P
eHTfp61Vt19N9tY/4IbOJMdCqRMURDVLtt/JYKwMf9mTIUvunORJApjTYHtcvNUw
LwfORELEC5n+5p/8sHiGUW3RLJ3GlvuFgrsEL/digO9i2n/2DqyQuFa9eT/ygG6j
2bkPXToHHZGThkspTOHcteHgM52zyzaRS/6htO7w+Q==
-----END CERTIFICATE-----
 2 s:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For
authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority - G5
   i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For
authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority - G5
-----BEGIN CERTIFICATE-----
MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjNzGs7SjANBgkqhkiG9w0BAQUFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
aG9yaXR5IC0gRzUwHhcNMDYxMTA4MDAwMDAwWhcNMzYwNzE2MjM1OTU5WjCByjEL
MAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQLExZW
ZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJpU2ln
biwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxWZXJp
U2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0aG9y
aXR5IC0gRzUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCvJAgIKXo1
nmAMqudLO07cfLw8RRy7K+D+KQL5VwijZIUVJ/XxrcgxiV0i6CqqpkKzj/i5Vbex
t0uz/o9+B1fs70PbZmIVYc9gDaTY3vjgw2IIPVQT60nKWVSFJuUrjxuf6/WhkcIz
SdhDY2pSS9KP6HBRTdGJaXvHcPaz3BJ023tdS1bTlr8Vd6Gw9KIl8q8ckmcY5fQG
BO+QueQA5N06tRn/Arr0PO7gi+s3i+z016zy9vA9r911kTMZHRxAy3QkGSGT2RT+
rCpSx4/VBEnkjWNHiDxpg8v+R70rfk/Fla4OndTRQ8Bnc+MUCH7lP59zuDMKz10/
NIeWiu5T6CUVAgMBAAGjgbIwga8wDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8E
BAMCAQYwbQYIKwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEwHzAH
BgUrDgMCGgQUj+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28udmVy
aXNpZ24uY29tL3ZzbG9nby5naWYwHQYDVR0OBBYEFH/TZafC3ey78DAJ80M5+gKv
MzEzMA0GCSqGSIb3DQEBBQUAA4IBAQCTJEowX2LP2BqYLz3q3JktvXf2pXkiOOzE
p6B4Eq1iDkVwZMXnl2YtmAl+X6/WzChl8gGqCBpH3vn5fJJaCGkgDdk+bW48DW7Y
5gaRQBi5+MHt39tBquCWIMnNZBU4gcmU7qKEKQsTb47bDN0lAtukixlE0kF6BWlK
WE9gyn6CagsCqiUXObXbf+eEZSqVir2G3l6BFoMtEMze/aiCKm0oHw0LxOXnGiYZ
4fQRbxC1lfznQgUy286dUV4otp6F01vvpX1FQHKOtw5rDgb7MzVIcbidJ4vEZV8N
hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
-----END CERTIFICATE-----
---
Server certificate
```

```
subject=/1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=California/businessCategory=P
rivate
Organization/serialNumber=C0806592/C=US/postalCode=95014/ST=California/L=Cupertino/street=1
Infinite Loop/O=Apple Inc./OU=IS&T/CN=acc-ept-waf.apple.com
issuer=/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA
- G3
---
Acceptable client certificate CA names
/UID=identity:idms.group.729488/CN=GRX.ACC1914Prod.AppleCare/OU=management:idms.group.72948
8/O=ELP/DC=Certificate Manager
/CN=Apple Corporate External Authentication CA 1/OU=Certification Authority/O=Apple
Inc./C=US
/CN=Apple Corporate Root CA/OU=Certification Authority/O=Apple Inc./C=US
---
SSL handshake has read 5834 bytes and written 500 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID: DFCC1B9C5AF18DB7741C935DFF1C15E7379F7A35999E8C33792F9F750BC98B98
    Session-ID-ctx:
    Master-Key:
67926578D1463A91E65A5E647FABAF222B792B5C0FC7AAAFCD531FD10681C71F478D37FBE03595787D90DE1644A
0B38E
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - a4 5b 15 32 2e 35 97 07-55 cd bd 3b 1a 6f 29 71   .[.2.5..U..;.o)q
    0010 - 91 77 d3 c0 f5 3f 6f 8b-37 b0 ad 0a 05 e2 22 d0   .w...?o.7.....".
    0020 - 91 6e a4 04 e4 1b bc 39-b3 e6 68 7d ec fa e9 e1   .n.....9..h}....
    0030 - e9 08 bc d3 68 8a eb b4-8f 4c 1d 40 c0 5d 1c 15   ....h....L.@.]..
    0040 - 59 c4 61 e3 ab 7e 36 b4-09 c0 b8 62 ef 43 df ed   Y.a..~6....b.C..
    0050 - 0e 67 c6 ea 70 b0 b5 70-47 0e f2 bf c6 eb 22 64   .g..p..pG......"d
    0060 - 45 52 41 1b 60 7d 43 fc-a1 d9 3b 59 3b b6 0b da   ERA.`}C...;Y;...
    0070 - bf 78 96 a9 40 c2 c9 1d-a8 df 06 9a b2 bd 39 c0   .x..@.........9.
    0080 - 52 4b a3 58 1c fd 3a 40-cf 07 37 19 b8 74 f3 0f   RK.X..:@..7..t..
    0090 - 72 e6 f5 8e 92 ef 6e 8c-3d dc 36 13 cb 64 4c 95   r.....n.=.6..dL.
    00a0 - 1a 41 6f 4c 47 23 9e b0-7e 67 eb 44 e3 3b 18 4c   .AoLG#..~g.D.;.L
    00b0 - 63 1a 82 13 62 06 e7 9f-bf fb 7f 3a 67 a0 e9 3b   c...b......:g..;
    00c0 - 9d 01 0a 20 46 74 bb 8a-8f 6c b4 74 d4 e8 41 5c   ... Ft...l.t..A\

    Start Time: 1533308946
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
```

iprofiles.apple.com

```
openssl s_client -showcerts -servername iprofiles.apple.com -connect
iprofiles.apple.com:443
CONNECTED(00000005)
depth=2 C = US, O = "VeriSign, Inc.", OU = VeriSign Trust Network, OU = "(c) 2006 VeriSign,
Inc. - For authorized use only", CN = VeriSign Class 3 Public Primary Certification
Authority - G5
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
```

```
 0
s:/1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=California/businessCategory=Private
Organization/serialNumber=C0806592/C=US/postalCode=95014/ST=California/L=Cupertino/street=1
Infinite Loop/O=Apple Inc./OU=Delivery Site Reliability Engineering/CN=deploy.apple.com
   i:/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA -
G3
-----BEGIN CERTIFICATE-----
MIIIMjCCBxqgAwIBAgIQEbL7W0NFW+xs/GVe+3nGXjANBgkqhkiG9w0BAQsFADB3
MQswCQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAd
BgNVBAsTFlN5bWFudGVjIFRydXN0IE5ldHdvcmsxKDAmBgNVBAMTH1N5bWFudGVj
IENsYXNzIDMgRVYgU1NMIENBIC0gRzMwHhcNMTcxMTI5MDAwMDAwWhcNMTkxMTI5
MjM1OTU5WjCCASQxEzARBgsrBgEEAYI3PAIBAxMCVVMxGzAZBgsrBgEEAYI3PAIB
AgwKQ2FsaWZvcm5pYTEdMBsGA1UEDxMUUHJpdmF0ZSBPcmdhbml6YXRpb24xETAP
BgNVBAUTCEMwODA2NTkyMQswCQYDVQQGEwJVUzEOMAwGA1UEEQwFOTUwMTQxEzAR
BgNVBAgMCkNhbGlmb3JuaWExEjAQBgNVBAcMCUN1cGVydGlubzEYMBYGA1UECQwP
MSBJbmZpbml0ZSBMb29wMRMwEQYDVQQKDApBcHBsZSBJbmMuMS4wLAYDVQQLDCVE
ZWxpdmVyeSBTaXRlIFJlbGlhYmlsaXR5IEVuZ2luZWVyaW5nMRkwFwYDVQQDDBBk
ZXBsb3kuYXBwbGUuY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA
xb1F7AMPY4L2yE8l7RE47Qu+R4ztjYYtkAc99lVfOHU1UNGunGF8YotbbZ6Njxjm
V/hpT1asIEW50K4is7LN+LoRMC9XMDLag3rS+pMMMZJOoeU9v8qtLzXx8/AQvMcv
lvRIfZGKXlMmQifLkc2UJ0fQrSr47XiYYssxh+k4ysBoeRUIMQuU38XM2cZfmgIr
SVD7d8npDasHsrcinkihrJt0e9fYJydC+BYu3r4EJkUfJ4ZzpUcvsrl+j4yVbDSf
J8tXpulbqpxJDQBH69Z4kdsO7eHJhUq+9obQ4m4tfgClEWA/TcuFh8uT2uqx0ALy
tZtXPfWBmsLsw4HcRZfyVwIDAQABo4IECTCCBAUwggEvBgNVHREEggEmMIIBIoIQ
dm9sdW1lLmFwcGxlLmNvbYITaXByb2ZpbGVzLmFwcGxlLmNvbYIXbWRtZW5yb2xs
bWVudC5hcHBsZS5jb22CEGRlcGxveS5hcHBsZS5jb22CEHNjaG9vbC5hcHBsZS5j
b22CE3dzLnNjaG9vbC5hcHBsZS5jb22CFWJldGEuc2Nob29sLmFwcGxlLmNvbYIa
ZGV2aWNlZW5yb2xsbWVudC5hcHBsZS5jb22CFndzLWNsYXNzcm9vbS5hcHBsZS5j
b22CF2Z1dHVyZS5zY2hvb2wuYXBwbGUuY29tghV3cy5idXNpbmVzcy5hcHBsZS5j
b22CFmVlc3VwcG9ydGFwcC5hcHBsZS5jb22CFGVlLXN1cHBvcnQuYXBwbGUuY29t
MAkGA1UdEwQCMAAwDgYDVR0PAQH/BAQDAgWgMB0GA1UdJQQWMBQGCCsGAQUFBwMB
BggrBgEFBQcDAjBvBgNVHSAEaDBmMFsGC2CGSAGG+EUBBxcGMEwwIwYIKwYBBQUH
AgEWF2h0dHBzOi8vZC5zeW1jYi5jb20vY3BzMCUGCCsGAQUFBwICMBkMF2h0dHBz
Oi8vZC5zeW1jYi5jb20vcnBhMAcGBWeBDAEBMB8GA1UdIwQYMBaAFAFZq+fdOgtZ
pmRj1s8gB1fVkedqMCsGA1UdHwQkMCIwIKAeoByGGmh0dHA6Ly9zci5zeW1jYi5j
b20vc3IuY3JsMFcGCCsGAQUFBwEBBEswSTAfBggrBgEFBQcwAYYTaHR0cDovL3Ny
LnN5bWNkLmNvbTAmBggrBgEFBQcwAoYaaHR0cDovL3NyLnN5bWNiLmNvbS9zci5j
cnQwggF8BgorBgEEAdZ5AgQCBIIBbASCAWgBZgB2AN3rHSt6DU+mIIuBrYFocH4u
jp0B1VyIjT0RxM227L7MAAABYAkoddAAAAQDAEcwRQIgA5tUOcBruEXxW75C1rFK
oKs4nujsi9lVSX7vAnbqxLQCIQDnegLziXLHJ2Z+4MoHX1teG8gT69r2dZ4rQHdk
1y21pgB1AKS5CZC0GFgUh7sTosxncAo8NZgE+RvfuON3zQ7IDdwQAAABYAkod2cA
AAQDAEYwRAIgUiT6IduC5nv/Z3pu4szv838glw2hLx6LEBnWs4mBXGcCIF7IV6B1
NNLOHwNFXf/z5XuXdEEqBqpm4eJvHaM+8TtJAHUA7ku9t3XOYLrhQmkfq+GeZqMP
fl+wctiDAMR7iXqo/csAAAFgCSh2qAAABAMARjBEAiARpU7hg6yaRE5x9skgyMY3
8L9O2c56NDPYpwWeXLqlGAIgIF/yjdzP4B3CSHn7KeyzCPYHrFK0062dlSyxJJjy
6BUwDQYJKoZIhvcNAQELBQADggEBAGMd2BMnFMhKsgvFJyJxLKxGdidPHzDGOdqs
SbqkYWpDGDT3IelCWkzSgbalmAv7V7n/4aBX5/SK+pmrCMWJODJWRyziiUZv58zl
N87fiC1Hb7eI3Ld+ws8qv31o3GmRwciZn1ZoSDSiIwOYa7EntBex/gdMPJyiJLhb
hOVbr0QALwc/23byBCdBb5XCedcu+6pr5c0nfEIfgKuN4TI6eYp+8asquBtZCw2q
vAXeuCdqjDZHVBUq+n37ybQLnpTlFyQQx71iAQAVWZjmb0PZkWCjzAy45C4Hl76H
sWlQuE04fb+4XWm+PNX9HCLvKNZTW3abS15NJ28fwnnQF9dwXcY=
-----END CERTIFICATE-----
 1 s:/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA -
G3
   i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For
authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority - G5
-----BEGIN CERTIFICATE-----
MIIFKzCCBBOgAwIBAgIQfuFKb2/v8tN/P61lTTratDANBgkqhkiG9w0BAQsFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
aG9yaXR5IC0gRzUwHhcNMTMxMDMxMDAwMDAwWhcNMjMxMDMwMjM1OTU5WjB3MQsw
CQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAdBgNV
BAsTFlN5bWFudGVjIFRydXN0IE5ldHdvcmsxKDAmBgNVBAMTH1N5bWFudGVjIENs
YXNzIDMgRVYgU1NMIENBIC0gRzMwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
```

```
AoIBAQDYoWV0I+grZOIy1zM3PY71NBZI3U9/hxz4RCMTjvsR2ERaGHGOYBYmkpv9
FwvhcXBC/r/6HMCqo6e1cej/GIP23xAKE2LIPZyn3i4/DNkd5y77Ks7Imn+Hv9hM
BBUyydHMlXGgTihPhNk1++OGb5RT5nKKY2cuvmn2926OnGAE6yn6xEdC0niY4+wL
pZLct5q9gGQrOHw4CVtm9i2VeoayNC6FnpAOX7ddpFFyRnATv2fytqdNFB5suVPu
IxpOjUhVQ0GxiXVqQCjFfd3SbtICGS97JJRL6/EaqZvjI5rq+jOrCiy39GAI3Z8c
zd0tAWaAr7MvKR0juIrhoXAHDDQPAgMBAAGjggFdMIIBWTAvBggrBgEFBQcBAQQj
MCEwHwYIKwYBBQUHMAGGE2h0dHA6Ly9zMi5zeW1jYi5jb20wEgYDVR0TAQH/BAgw
BgEB/wIBADBlBgNVHSAEXjBcMFoGBFUdIAAwUjAmBggrBgEFBQcCARYaaHR0cDov
L3d3dy5zeW1hdXRoLmNvbS9jcHMwKAYIKwYBBQUHAgIwHBoaaHR0cDovL3d3dy5z
eW1hdXRoLmNvbS9ycGEwMAYDVR0fBCkwJzAloCOgIYYfaHR0cDovL3MxLnN5bWNi
LmNvbS9wY2EzLWc1LmNybDAOBgNVHQ8BAf8EBAMCAQYwKQYDVR0RBCIwIKQeMBwx
GjAYBgNVBAMTEVN5bWFudGVjUEtJLTEtNTMzMB0GA1UdDgQWBBQBWavn3ToLWaZk
Y9bPIAdX1ZHnajAfBgNVHSMEGDAWgBR/02Wnwt3su/AwCfNDOfoCrzMxMzANBgkq
hkiG9w0BAQsFAAOCAQEAQgFVe9AWGl1Y6LubqE3X89frE5SG1n8hC0e8V5uSXU8F
nzikEHzPg74GQ0aNCLxq1xCm+quvL2GoY/Jl339MiBKIT7Np2f8nwAqXkY9W+4nE
qLuSLRtzsMarNvSWbCAI7woeZiRFT2cAQMgHVHQzO6atuyOfZu2iRHA0+w7qAf3P
eHTfp61Vt19N9tY/4IbOJMdCqRMURDVLtt/JYKwMf9mTIUvunORJApjTYHtcvNUw
LwfORELEC5n+5p/8sHiGUW3RLJ3GlvuFgrsEL/digO9i2n/2DqyQuFa9eT/ygG6j
2bkPXToHHZGThkspTOHcteHgM52zyzaRS/6htO7w+Q==
-----END CERTIFICATE-----
 2 s:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For
authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority - G5
   i:/C=US/O=VeriSign, Inc./OU=Class 3 Public Primary Certification Authority
-----BEGIN CERTIFICATE-----
MIIE0DCCBDmgAwIBAgIQJQzo4DBhLp8rifcFTXz4/TANBgkqhkiG9w0BAQUFADBf
MQswCQYDVQQGEwJVUzEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xNzA1BgNVBAsT
LkNsYXNzIDMgUHVibGljIFByaW1hcnkgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkw
HhcNMDYxMTA4MDAwMDAwWhcNMjExMTA3MjM1OTU5WjCByjELMAkGA1UEBhMCVVMx
FzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQLExZWZXJpU2lnbiBUcnVz
dCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJpU2lnbiwgSW5jLiAtIEZv
ciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxWZXJpU2lnbiBDbGFzcyAz
IFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0aG9yaXR5IC0gRzUwggEi
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCvJAgIKXo1nmAMqudLO07cfLw8
RRy7K+D+KQL5VwijZIUVJ/XxrcgxiV0i6CqqpkKzj/i5Vbext0uz/o9+B1fs70Pb
ZmIVYc9gDaTY3vjgw2IIPVQT60nKWVSFJuUrjxuf6/WhkcIzSdhDY2pSS9KP6HBR
TdGJaXvHcPaz3BJ023tdS1bTlr8Vd6Gw9KIl8q8ckmcY5fQGBO+QueQA5N06tRn/
Arr0PO7gi+s3i+z016zy9vA9r911kTMZHRxAy3QkGSGT2RT+rCpSx4/VBEnkjWNH
iDxpg8v+R70rfk/Fla4OndTRQ8Bnc+MUCH7lP59zuDMKz10/NIeWiu5T6CUVAgMB
AAGjggGbMIIBlzAPBgNVHRMBAf8EBTADAQH/MDEGA1UdHwQqMCgwJqAkoCKGIGh0
dHA6Ly9jcmwudmVyaXNpZ24uY29tL3BjYTMuY3JsMA4GA1UdDwEB/wQEAwIBBjA9
BgNVHSAENjA0MDIGBFUdIAAwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly93d3cudmVy
aXNpZ24uY29tL2NwczAdBgNVHQ4EFgQUf9Nlp8Ld7LvwMAnzQzn6Aq8zMTMwbQYI
KwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEwHzAHBgUrDgMCGgQU
j+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28udmVyaXNpZ24uY29t
L3ZzbG9nby5naWYwNAYIKwYBBQUHAQEEKDAmMCQGCCsGAQUFBzABhhhodHRwOi8v
b2NzcC52ZXJpc2lnbi5jb20wPgYDVR0lBDcwNQYIKwYBBQUHAwEGCCsGAQUFBwMC
BggrBgEFBQcDAwYJYIZIAYb4QgQBBgpghkgBhvhFAQgBMA0GCSqGSIb3DQEBBQUA
A4GBABMC3fjohgDyWvj4IAxZiGIHzs73Tvm7WaGY5eE43U68ZhjTresY8g3JbT5K
lCDDPLq9ZVTGr0SzEK0saz6r1we2uIFjxfleLuUqZ87NMwwq14lWAyMfs77oOghZ
tOxFNfeKW/9mz1Cvxm1XjRl4t7mi0VfqH5pLr7rJjhJ+xr3/
-----END CERTIFICATE-----
---
Server certificate
subject=/1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=California/businessCategory=P
rivate
Organization/serialNumber=C0806592/C=US/postalCode=95014/ST=California/L=Cupertino/street=1
Infinite Loop/O=Apple Inc./OU=Delivery Site Reliability Engineering/CN=deploy.apple.com
issuer=/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA
- G3
---
No client certificate CA names sent
---
SSL handshake has read 5161 bytes and written 472 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
```

```
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID: FAB6FEFD9213E4740C1DC0FF70B18F831C75A61E61D1F3098194164925B15C78
    Session-ID-ctx:
    Master-Key:
1EF5A1544EA079C3C118BD5912E214A1E32FD0727DD609B4D7F0CD8482E80721233D715471D9F1447A252403B18
5A720
    Start Time: 1533309561
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
```

```
openssl s_client -showcerts -servername mdmenrollment.apple.com -connect
mdmenrollment.apple.com:443
CONNECTED(00000005)
depth=1 C = US, O = Symantec Corporation, OU = Symantec Trust Network, CN = Symantec Class
3 EV SSL CA - G3
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0
s:/1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=California/businessCategory=Private
Organization/serialNumber=C0806592/C=US/postalCode=95014/ST=California/L=Cupertino/street=1
Infinite Loop/O=Apple Inc./OU=Delivery Site Reliability Engineering/CN=deploy.apple.com
   i:/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA -
G3
-----BEGIN CERTIFICATE-----
MIIIMjCCBxqgAwIBAgIQEbL7W0NFW+xs/GVe+3nGXjANBgkqhkiG9w0BAQsFADB3
MQswCQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAd
BgNVBAsTFlN5bWFudGVjIFRydXN0IE5ldHdvcmsxKDAmBgNVBAMTH1N5bWFudGVj
IENsYXNzIDMgRVYgU1NMIENBIC0gRzMwHhcNMTcxMTI5MDAwMDAwWhcNMTkxMTI5
MjM1OTU5WjCCASQxEzARBgsrBgEEAYI3PAIBAxMCVVMxGzAZBgsrBgEEAYI3PAIB
AgwKQ2FsaWZvcm5pYTEdMBsGA1UEDxMUUHJpdmF0ZSBPcmdhbml6YXRpb24xETAP
BgNVBAUTCEMwODA2NTkyMQswCQYDVQQGEwJVUzEOMAwGA1UEEQwFOTUwMTQxEzAR
BgNVBAgMCkNhbGlmb3JuaWExEjAQBgNVBAcMCUN1cGVydGlubzEYMBYGA1UECQwP
MSBJbmZpbml0ZSBMb29wMRMwEQYDVQQKDApBcHBsZSBJbmMuMS4wLAYDVQQLDCVE
ZWxpdmVyeSBTaXRlIFJlbGlhYmlsaXR5IEVuZ2luZWVyaW5nMRkwFwYDVQQDDBBk
ZXBsb3kuYXBwbGUuY29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA
xb1F7AMPY4L2yE8l7RE47Qu+R4ztjYYtkAc99lVfOHU1UNGunGF8YotbbZ6Njxjm
V/hpT1asIEW50K4is7LN+LoRMC9XMDLag3rS+pMMMZJOoeU9v8qtLzXx8/AQvMcv
lvRIfZGKXlMmQifLkc2UJ0fQrSr47XiYYssxh+k4ysBoeRUIMQuU38XM2cZfmgIr
SVD7d8npDasHsrcinkihrJt0e9fYJydC+BYu3r4EJkUfJ4ZzpUcvsrl+j4yVbDSf
J8tXpulbqpxJDQBH69Z4kdsO7eHJhUq+9obQ4m4tfgClEWA/TcuFh8uT2uqx0ALy
tZtXPfWBmsLsw4HcRZfyVwIDAQABo4IECTCCBAUwggEvBgNVHREEggEmMIIBIoIQ
dm9sdW1lLmFwcGxlLmNvbYITaXByb2ZpbGVzLmFwcGxlLmNvbYIXbWRtZW5yb2xs
bWVudC5hcHBsZS5jb22CEGRlcGxveS5hcHBsZS5jb22CEHNjaG9vbC5hcHBsZS5j
b22CE3dzLnNjaG9vbC5hcHBsZS5jb22CFWJldGEuc2Nob29sLmFwcGxlLmNvbYIa
ZGV2aWNlZW5yb2xsbWVudC5hcHBsZS5jb22CFndzLWNsYXNzcm9vbS5hcHBsZS5j
b22CF2Z1dHVyZS5zY2hvb2wuYXBwbGUuY29tghV3cy5idXNpbmVzcy5hcHBsZS5j
b22CFmVlc3VwcG9ydGFwcC5hcHBsZS5jb22CFGVlLXN1cHBvcnQuYXBwbGUuY29t
MAkGA1UdEwQCMAAwDgYDVR0PAQH/BAQDAgWgMB0GA1UdJQQWMBQGCCsGAQUFBwMB
BggrBgEFBQcDAjBvBgNVHSAEaDBmMFsGC2CGSAGG+EUBBxcGMEwwIwYIKwYBBQUH
AgEWF2h0dHBzOi8vZC5zeW1jYi5jb20vY3BzMCUGCCsGAQUFBwICMBkMF2h0dHBz
Oi8vZC5zeW1jYi5jb20vcnBhMAcGBWeBDAEBMB8GA1UdIwQYMBaAFAFZq+fdOgtZ
pmRj1s8gB1fVkedqMCsGA1UdHwQkMCIwIKAeoByGGmh0dHA6Ly9zci5zeW1jYi5j
b20vc3IuY3JsMFcGCCsGAQUFBwEBBEswSTAfBggrBgEFBQcwAYYTaHR0cDovL3Ny
LnN5bWNkLmNvbTAmBggrBgEFBQcwAoYaaHR0cDovL3NyLnN5bWNiLmNvbS9zci5j
cnQwggF8BgorBgEEAdZ5AgQCBIIBbASCAWgBZgB2AN3rHSt6DU+mIIuBrYFocH4u
jp0B1VyIjT0RxM227L7MAAABYAkoddAAAAQDAEcwRQIgA5tUOcBruEXxW75C1rFK
```

```
oKs4nujsi9lVSX7vAnbqxLQCIQDnegLziXLHJ2Z+4MoHX1teG8gT69r2dZ4rQHdk
1y21pgB1AKS5CZC0GFgUh7sTosxncAo8NZgE+RvfuON3zQ7IDdwQAAABYAkod2cA
AAQDAEYwRAIgUiT6IduC5nv/Z3pu4szv838glw2hLx6LEBnWs4mBXGcCIF7IV6B1
NNLOHwNFXf/z5XuXdEEqBqpm4eJvHaM+8TtJAHUA7ku9t3XOYLrhQmkfq+GeZqMP
fl+wctiDAMR7iXqo/csAAAFgCSh2qAAABAMARjBEAiARpU7hg6yaRE5x9skgyMY3
8L9O2c56NDPYpwWeXLqlGAIgIF/yjdzP4B3CSHn7KeyzCPYHrFK0062dlSyxJJjy
6BUwDQYJKoZIhvcNAQELBQADggEBAGMd2BMnFMhKsgvFJyJxLKxGdidPHzDGOdqs
SbqkYWpDGDT3IelCWkzSgbalmAv7V7n/4aBX5/SK+pmrCMWJODJWRyziiUZv58zl
N87fiC1Hb7eI3Ld+ws8qv31o3GmRwciZn1ZoSDSiIwOYa7EntBex/gdMPJyiJLhb
hOVbr0QALwc/23byBCdBb5XCedcu+6pr5c0nfEIfgKuN4TI6eYp+8asquBtZCw2q
vAXeuCdqjDZHVBUq+n37ybQLnpTlFyQQx71iAQAVWZjmb0PZkWCjzAy45C4Hl76H
sWlQuE04fb+4XWm+PNX9HCLvKNZTW3abS15NJ28fwnnQF9dwXcY=
-----END CERTIFICATE-----
 1 s:/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA -
G3
   i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c) 2006 VeriSign, Inc. - For
authorized use only/CN=VeriSign Class 3 Public Primary Certification Authority - G5
-----BEGIN CERTIFICATE-----
MIIFKzCCBBOgAwIBAgIQfuFKb2/v8tN/P61lTTratDANBgkqhkiG9w0BAQsFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
aG9yaXR5IC0gRzUwHhcNMTMxMDMxMDAwMDAwWhcNMjMxMDMwMjM1OTU5WjB3MQsw
CQYDVQQGEwJVUzEdMBsGA1UEChMUU3ltYW50ZWMgQ29ycG9yYXRpb24xHzAdBgNV
BAsTFlN5bWFudGVjIFRydXN0IE5ldHdvcmsxKDAmBgNVBAMTH1N5bWFudGVjIENs
YXNzIDMgRVYgU1NMIENBIC0gRzMwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQDYoWV0I+grZOIy1zM3PY71NBZI3U9/hxz4RCMTjvsR2ERaGHGOYBYmkpv9
FwvhcXBC/r/6HMCqo6e1cej/GIP23xAKE2LIPZyn3i4/DNkd5y77Ks7Imn+Hv9hM
BBUyydHMlXGgTihPhNk1++OGb5RT5nKKY2cuvmn2926OnGAE6yn6xEdC0niY4+wL
pZLct5q9gGQrOHw4CVtm9i2VeoayNC6FnpAOX7ddpFFyRnATv2fytqdNFB5suVPu
IxpOjUhVQ0GxiXVqQCjFfd3SbtICGS97JJRL6/EaqZvjI5rq+jOrCiy39GAI3Z8c
zd0tAWaAr7MvKR0juIrhoXAHDDQPAgMBAAGjggFdMIIBWTAvBggrBgEFBQcBAQQj
MCEwHwYIKwYBBQUHMAGGE2h0dHA6Ly9zMi5zeW1jYi5jb20wEgYDVR0TAQH/BAgw
BgEB/wIBADBlBgNVHSAEXjBcMFoGBFUdIAAwUjAmBggrBgEFBQcCARYaaHR0cDov
L3d3dy5zeW1hdXRoLmNvbS9jcHMwKAYIKwYBBQUHAgIwHBoaaHR0cDovL3d3dy5z
eW1hdXRoLmNvbS9ycGEwMAYDVR0fBCkwJzAloCOgIYYfaHR0cDovL3MxLnN5bWNi
LmNvbS9wY2EzLWc1LmNybDAOBgNVHQ8BAf8EBAMCAQYwKQYDVR0RBCIwIKQeMBwx
GjAYBgNVBAMTEVN5bWFudGVjUEtJLTEtNTMzMB0GA1UdDgQWBBQBWavn3ToLWaZk
Y9bPIAdX1ZHnajAfBgNVHSMEGDAWgBR/02Wnwt3su/AwCfNDOfoCrzMxMzANBgkq
hkiG9w0BAQsFAAOCAQEAQgFVe9AWGl1Y6LubqE3X89frE5SG1n8hC0e8V5uSXU8F
nzikEHzPg74GQ0aNCLxq1xCm+quvL2GoY/Jl339MiBKIT7Np2f8nwAqXkY9W+4nE
qLuSLRtzsMarNvSWbCAI7woeZiRFT2cAQMgHVHQzO6atuyOfZu2iRHA0+w7qAf3P
eHTfp61Vt19N9tY/4IbOJMdCqRMURDVLtt/JYKwMf9mTIUvunORJApjTYHtcvNUw
LwfORELEC5n+5p/8sHiGUW3RLJ3GlvuFgrsEL/digO9i2n/2DqyQuFa9eT/ygG6j
2bkPXToHHZGThkspTOHcteHgM52zyzaRS/6htO7w+Q==
-----END CERTIFICATE-----
---
Server certificate
subject=/1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=California/businessCategory=P
rivate
Organization/serialNumber=C0806592/C=US/postalCode=95014/ST=California/L=Cupertino/street=1
Infinite Loop/O=Apple Inc./OU=Delivery Site Reliability Engineering/CN=deploy.apple.com
issuer=/C=US/O=Symantec Corporation/OU=Symantec Trust Network/CN=Symantec Class 3 EV SSL CA
- G3
---
No client certificate CA names sent
---
SSL handshake has read 3922 bytes and written 476 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
```

```
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID: DADB8B92FF9EFC7A8AC29102DB17CA035B4E016A8306686433E34935E5DAB31C
    Session-ID-ctx:
    Master-Key:
E332F65CFF64206674E8B10FE05AC835C639C6AD9646096D7BA08590B5871397C43B10B5F1BA725BA959475E939
73302
    Start Time: 1533309640
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
```

api.push.apple.com

```
openssl s_client -showcerts -servername api.push.apple.com -connect api.push.apple.com:443
CONNECTED(00000006)
depth=1 CN = Apple IST CA 2 - G1, OU = Certification Authority, O = Apple Inc., C = US
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:/CN=api.push.apple.com/OU=management:idms.group.533599/O=Apple Inc./ST=California/C=US
   i:/CN=Apple IST CA 2 - G1/OU=Certification Authority/O=Apple Inc./C=US
-----BEGIN CERTIFICATE-----
MIIGGzCCBQOgAwIBAgIIBHE7D0TJ45IwDQYJKoZIhvcNAQELBQAwYjEcMBoGA1UE
AxMTQXBwbGUgSVNUIENBIDIgLSBHMTEgMB4GA1UECxMXQ2VydGlmaWNhdGlvbiBB
dXRob3JpdHkxEzARBgNVBAoTCkFwcGxlIEluYy4xCzAJBgNVBAYTAlVTMB4XDTE4
MDEwNjAwMTQxNVoXDTIwMDIwNTAwMTQxNVowezEbMBkGA1UEAwwSYXBpLnB1c2gu
YXBwbGUuY29tMSUwIwYDVQQLDBxtYW5hZ2VtZW50OmlkbXMuZ3JvdXAuNTMzNTk5
MRMwEQYDVQQKDApBcHBsZSBJbmMuMRMwEQYDVQQIDApDYWxpZm9ybmlhMQswCQYD
VQQGEwJVUzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANBqg4kZlG/2
Dai2K6953tNNuuFWfYQjMejUR+1ne+ee8ED72Mwm52AwRuojrdhxbYhDIoj+7k40
jsM3oS5rJv+W/MWt3PmIR18UCF3izZo3lYZS1zBjtCPrcwa+Tbqwj6qtva+o+ED8
xU9znCZ/oeSqgS8jMg3wey957rpuLYFOsXMX6NCmDOk4aJf4C+ec1B+BsSIDVEQw
FyubteM0u3VnSVQPxBbI2Uh7PI5nKupIJKxB6M77zc00Q6LWkYIHK5ucGvPOrqbc
aJRWIMzeTMKcLaxvPEe00Ro5b2afnT24em/muE0+OI0ayLyRzNPpMpoO28pWnRBF
UnI4CmDDR/UCAwEAAaOCArowggK2MH4GCCsGAQUFBwEBBHIwcDA0BggrBgEFBQcw
AoYoaHR0cDovL2NlcnRzLmFwcGxlLmNvbS9hcHBsZWlzdGNhMmcxLmRlcjA4Bggr
BgEFBQcwAYYsaHR0cDovL29jc3AuYXBwbGUuY29tL29jc3AwNC1hcHBsZWlzdGNh
MmcxMDEwHQYDVR0OBBYEFJWAml5q3L4a4hdIUDZP21Uzae5UMAwGA1UdEwEB/wQC
MAAwHwYDVR0jBBgwFoAU2HqURHyQcJAWnt0XnAFEA4bWKikwgf8GA1UdIASB9zCB
9DCB8QYKKoZIhvdjZAULBDCB4jCBpAYIKwYBBQUHAgIwgZcMgZRSZWxpYW5jZSBv
biB0aGlzIGNlcnRpZmljYXRlIGJ5IGFueSBwYXJ0eSBhc3N1bWVzIGFjY2VwdGFu
Y2Ugb2YgYW55IGFwcGxpY2FibGUgdGVybXMgYW5kIGNvbmRpdGlvbnMgb2YgdXNl
IGFuZC9vciBjZXJ0aWZpY2F0aW9uIHByYWN0aWNlIHN0YXRlbWVudHMuMDkGCCsG
AQUFBwIBFi1odHRwOi8vd3d3LmFwcGxlLmNvbS9jZXJ0aWZpY2F0ZWF1dGhvcml0
eS9ycGEwNwYDVR0fBDAwLjAsoCqgKIYmaHR0cDovL2NybC5hcHBsZS5jb20vYXBw
bGVpc3RjYTJnMS5jcmwwDgYDVR0PAQH/BAQDAgWgMB0GA1UdJQQWMBQGCCsGAQUF
BwMBBggrBgEFBQcDAjB8BgNVHREEdTBzghVtci1hcGkucHVzaC5hcHBsZS5jb22C
EmFwaS5wdXNoLmFwcGxlLmNvbYIVcHYtYXBpLnB1c2guYXBwbGUuY29tghhhcGkt
Y2FycnkucHVzaC5hcHBsZS5jb22CFXN0LWFwaS5wdXNoLmFwcGxlLmNvbTANBgkq
hkiG9w0BAQsFAAOCAQEAYVE2P8QFxnfyt1sBEXtTJElbNr5p2dy8Iul/62D7csoY
1svHMYyWoaiKA75ugxNVJ7ZiSlEl7Y1u4PsZ7sox08+Tt2hfJO48GD86P5e/fGO2
lUwTpWXjczEI8LkYrou4DFsVy9YCGIn3Q4FWQE4794+JiD26kLY3go8pa8bcYShn
p6bUdjyH9YUWoGaUP6sRFaKvFtqDpp1bg1CxbG9SuaxFHu/rIMlPWyjRonLjLGau
ChTbuxV4/ptAtgeyNH5zNUhsFQnm5KhWwOJLNgQcF18FOraorktCdFoEpDOpq6x/
YU/7zvCdYhXX+6CJ71/0RHEbCkx/5rpAxxM6Ol7zXQ==
-----END CERTIFICATE-----
 1 s:/CN=Apple IST CA 2 - G1/OU=Certification Authority/O=Apple Inc./C=US
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
-----BEGIN CERTIFICATE-----
MIIEQDCCAyigAwIBAgIDAjp0MA0GCSqGSIb3DQEBCwUAMEIxCzAJBgNVBAYTAlVT
MRYwFAYDVQQKEw1HZW9UcnVzdCBJbmMuMRswGQYDVQQDExJHZW9UcnVzdCBHbG9i
```

```
YWwgQ0EwHhcNMTQwNjE2MTU0MjAyWhcNMjIwNTIwMTU0MjAyWjBiMRwwGgYDVQQD
ExNBcHBsZSBJU1QgQ0EgMiAtIEcxMSAwHgYDVQQLExdDZXJ0aWZpY2F0aW9uIEF1
dGhvcml0eTETMBEGA1UEChMKQXBwbGUgSW5jLjELMAkGA1UEBhMCVVMwggEiMA0G
CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDQk6EdR0MgFrILa+vD1bTox5jN896/
6E3p4zaAB/xFG2p8RYauVtOkCX9hDWtdflJrfbTIOcT0Zzr3g84Zb4YvfkV+Rxxn
UsqVBV3iNlGFwNRngDVvFd0+/R3S/Y80UNjsdiq+49Pa5P3I6ygClhGXF2Ec6cRZ
O0LcMtEJHdqm0UOG/16yvIzPZtsBiwKulEjzOI/96jKoCOyGl1GUJD5JSZZT6Hmh
QIHpBbuTlVH84/18EUv3ngizFUkVB/nRN6CbSzL2tcTcatH8Cu324MUpoKiLcf4N
krz+VHAYCm3H7Qz7yS0Gw4yF/MuGXNY2jhKLCX/7GRo41fCUMHoPpozzAgMBAAGj
ggEdMIIBGTAfBgNVHSMEGDAWgBTAephojYn7qwVkDBF9qn1luMrMTjAdBgNVHQ4E
FgQU2HqURHyQcJAWnt0XnAFEA4bWKikwEgYDVR0TAQH/BAgwBgEB/wIBADAOBgNV
HQ8BAf8EBAMCAQYwNQYDVR0fBC4wLDAqoCigJoYkaHR0cDovL2cuc3ltY2IuY29t
L2NybHMvZ3RnbG9iYWwuY3JsMC4GCCsGAQUFBwEBBCIwIDAeBggrBgEFBQcwAYYS
aHR0cDovL2cuc3ltY2QuY29tMEwGA1UdIARFMEMwQQYKYIZIAYb4RQEHNjAzMDEG
CCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdlb3RydXN0LmNvbS9yZXNvdXJjZXMvY3Bz
MA0GCSqGSIb3DQEBCwUAA4IBAQAWR3NvhaJi4ecqdruJlUIml7xKrKxwUzo/MYM9
PByrmuKxXRx2GqA8DHJXvtOeUODImdZY1wLqzg0pVHzN9cLGkClVo28UqAtCDTqY
bQZ4nvBqox0CCqIopI3CgUY+bWfa3j/+hQ5CKhLetbf7uBunlux3n+zUU5V6/wf0
8goUwFFSsdaOUAsamVy8C8m97e34XsFW201+I6QRoSzUGwWa5BtS9nw4mQVLunKN
QolgBGYq9P1o12v3mUEo1mwkq+YlUy7Igpnioo8jvjCDsSeL+mh/AUnoxphrEC6Y
XorXykuxx8lYmtA225aV7LaB5PLNbxt5h0wQPInkTfpU3Kqm
-----END CERTIFICATE-----
---
Server certificate
subject=/CN=api.push.apple.com/OU=management:idms.group.533599/O=Apple
Inc./ST=California/C=US
issuer=/CN=Apple IST CA 2 - G1/OU=Certification Authority/O=Apple Inc./C=US
---
Acceptable client certificate CA names
/C=US/O=Apple Inc./OU=Apple Certification Authority/CN=Apple Root CA
/C=US/O=Apple Inc./OU=Apple Worldwide Developer Relations/CN=Apple Worldwide Developer
Relations Certification Authority
/CN=Apple Application Integration 2 Certification Authority/OU=Apple Certification
Authority/O=Apple Inc./C=US
/C=US/O=Apple Inc./OU=Apple Worldwide Developer Relations/CN=Apple Worldwide Developer
Relations Certification Authority
/C=US/O=Apple Inc./OU=Apple Certification Authority/CN=Apple Application Integration
Certification Authority
---
SSL handshake has read 3905 bytes and written 483 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: 011EE0857266CA398F81F5E16FC29CA4E1FDF26D9089F00C8274A0AC3659E7D8
    Session-ID-ctx:
    Master-Key:
5860BC1043B7F8F2226EDF7F7F4B1173527AC4F192B9753BFFF6469216B95EE161E37262946B5AFBC3C6E93D59E
67F4B
    Start Time: 1533309812
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
```

api.push.apple.com:2197

```
openssl s_client -showcerts -servername api.push.apple.com -connect api.push.apple.com:2197
CONNECTED(00000006)
```

```
depth=1 CN = Apple IST CA 2 - G1, OU = Certification Authority, O = Apple Inc., C = US
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:/CN=api.push.apple.com/OU=management:idms.group.533599/O=Apple Inc./ST=California/C=US
   i:/CN=Apple IST CA 2 - G1/OU=Certification Authority/O=Apple Inc./C=US
-----BEGIN CERTIFICATE-----
MIIGGzCCBQOgAwIBAgIIBHE7D0TJ45IwDQYJKoZIhvcNAQELBQAwYjEcMBoGA1UE
AxMTQXBwbGUgSVNUIENBIDIgLSBHMTEgMB4GA1UECxMXQ2VydGlmaWNhdGlvbiBB
dXRob3JpdHkxEzARBgNVBAoTCkFwcGxlIEluYy4xCzAJBgNVBAYTAlVTMB4XDTE4
MDEwNjAwMTQxNVoXDTIwMDIwNTAwMTQxNVowezEbMBkGA1UEAwwSYXBpLnB1c2gu
YXBwbGUuY29tMSUwIwYDVQQLDBxtYW5hZ2VtZW50OmlkbXMuZ3JvdXAuNTMzNTk5
MRMwEQYDVQQKDApBcHBsZSBJbmMuMRMwEQYDVQQIDApDYWxpZm9ybmlhMQswCQYD
VQQGEwJVUzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANBqg4kZlG/2
Dai2K6953tNNuuFWfYQjMejUR+1ne+ee8ED72Mwm52AwRuojrdhxbYhDIoj+7k40
jsM3oS5rJv+W/MWt3PmIR18UCF3izZo3lYZS1zBjtCPrcwa+Tbqwj6qtva+o+ED8
xU9znCZ/oeSqgS8jMg3wey957rpuLYFOsXMX6NCmDOk4aJf4C+ec1B+BsSIDVEQw
FyubteM0u3VnSVQPxBbI2Uh7PI5nKupIJKxB6M77zc00Q6LWkYIHK5ucGvPOrqbc
aJRWIMzeTMKcLaxvPEe00Ro5b2afnT24em/muE0+OI0ayLyRzNPpMpoO28pWnRBF
UnI4CmDDR/UCAwEAAaOCArowggK2MH4GCCsGAQUFBwEBBHIwcDA0BggrBgEFBQcw
AoYoaHR0cDovL2NlcnRzLmFwcGxlLmNvbS9hcHBsZWlzdGNhMmcxLmRlcjA4Bggr
BgEFBQcwAYYsaHR0cDovL29jc3AuYXBwbGUuY29tL29jc3AwNC1hcHBsZWlzdGNh
MmcxMDEwHQYDVR0OBBYEFJWAml5q3L4a4hdIUDZP21Uzae5UMAwGA1UdEwEB/wQC
MAAwHwYDVR0jBBgwFoAU2HqURHyQcJAWnt0XnAFEA4bWKikwgf8GA1UdIASB9zCB
9DCB8QYKKoZIhvdjZAULBDCB4jCBpAYIKwYBBQUHAgIwgZcMgZRSZWxpYW5jZSBv
biB0aGlzIGNlcnRpZmljYXRlIGJ5IGFueSBwYXJ0eSBhc3N1bWVzIGFjY2VwdGFu
Y2Ugb2YgYW55IGFwcGxpY2FibGUgdGVybXMgYW5kIGNvbmRpdGlvbnMgb2YgdXNl
IGFuZC9vciBjZXJ0aWZpY2F0aW9uIHByYWN0aWNlIHN0YXRlbWVudHMuMDkGCCsG
AQUFBwIBFi1odHRwOi8vd3d3LmFwcGxlLmNvbS9jZXJ0aWZpY2F0ZWF1dGhvcml0
eS9ycGEwNwYDVR0fBDAwLjAsoCqgKIYmaHR0cDovL2NybC5hcHBsZS5jb20vYXBw
bGVpc3RjYTJnMS5jcmwwDgYDVR0PAQH/BAQDAgWgMB0GA1UdJQQWMBQGCCsGAQUF
BwMBBggrBgEFBQcDAjB8BgNVHREEdTBzghVtci1hcGkucHVzaC5hcHBsZS5jb22C
EmFwaS5wdXNoLmFwcGxlLmNvbYIVcHYtYXBpLnB1c2guYXBwbGUuY29tghhhcGkt
Y2FycnkucHVzaC5hcHBsZS5jb22CFXN0LWFwaS5wdXNoLmFwcGxlLmNvbTANBgkq
hkiG9w0BAQsFAAOCAQEAYVE2P8QFxnfyt1sBEXtTJElbNr5p2dy8Iul/62D7csoY
1svHMYyWoaiKA75ugxNVJ7ZiSlEl7Y1u4PsZ7sox08+Tt2hfJO48GD86P5e/fGO2
lUwTpWXjczEI8LkYrou4DFsVy9YCGIn3Q4FWQE4794+JiD26kLY3go8pa8bcYShn
p6bUdjyH9YUWoGaUP6sRFaKvFtqDpp1bg1CxbG9SuaxFHu/rIMlPWyjRonLjLGau
ChTbuxV4/ptAtgeyNH5zNUhsFQnm5KhWwOJLNgQcF18FOraorktCdFoEpDOpq6x/
YU/7zvCdYhXX+6CJ71/0RHEbCkx/5rpAxxM6Ol7zXQ==
-----END CERTIFICATE-----
 1 s:/CN=Apple IST CA 2 - G1/OU=Certification Authority/O=Apple Inc./C=US
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
-----BEGIN CERTIFICATE-----
MIIEQDCCAyigAwIBAgIDAjp0MA0GCSqGSIb3DQEBCwUAMEIxCzAJBgNVBAYTAlVT
MRYwFAYDVQQKEw1HZW9UcnVzdCBJbmMuMRswGQYDVQQDExJHZW9UcnVzdCBHbG9i
YWwgQ0EwHhcNMTQwNjE2MTU0MjAyWhcNMjIwNTIwMTU0MjAyWjBiMRwwGgYDVQQD
ExNBcHBsZSBJU1QgQ0EgMiAtIEcxMSAwHgYDVQQLExdDZXJ0aWZpY2F0aW9uIEF1
dGhvcml0eTETMBEGA1UEChMKQXBwbGUgSW5jLjELMAkGA1UEBhMCVVMwggEiMA0G
CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDQk6EdR0MgFrILa+vD1bTox5jN896/
6E3p4zaAB/xFG2p8RYauVtOkCX9hDWtdflJrfbTIOcT0Zzr3g84Zb4YvfkV+Rxxn
UsqVBV3iNlGFwNRngDVvFd0+/R3S/Y80UNjsdiq+49Pa5P3I6ygClhGXF2Ec6cRZ
O0LcMtEJHdqm0UOG/16yvIzPZtsBiwKulEjzOI/96jKoCOyGl1GUJD5JSZZT6Hmh
QIHpBbuTlVH84/18EUv3ngizFUkVB/nRN6CbSzL2tcTcatH8Cu324MUpoKiLcf4N
krz+VHAYCm3H7Qz7yS0Gw4yF/MuGXNY2jhKLCX/7GRo41fCUMHoPpozzAgMBAAGj
ggEdMIIBGTAfBgNVHSMEGDAWgBTAephojYn7qwVkDBF9qn1luMrMTjAdBgNVHQ4E
FgQU2HqURHyQcJAWnt0XnAFEA4bWKikwEgYDVR0TAQH/BAgwBgEB/wIBADAOBgNV
HQ8BAf8EBAMCAQYwNQYDVR0fBC4wLDAqoCigJoYkaHR0cDovL2cuc3ltY2IuY29t
L2NybHMvZ3RnbG9iYWwuY3JsMC4GCCsGAQUFBwEBBCIwIDAeBggrBgEFBQcwAYYS
aHR0cDovL2cuc3ltY2QuY29tMEwGA1UdIARFMEMwQQYKYIZIAYb4RQEHNjAzMDEG
CCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdlb3RydXN0LmNvbS9yZXNvdXJjZXMvY3Bz
MA0GCSqGSIb3DQEBCwUAA4IBAQAWR3NvhaJi4ecqdruJlUIml7xKrKxwUzo/MYM9
PByrmuKxXRx2GqA8DHJXvtOeUODImdZY1wLqzg0pVHzN9cLGkClVo28UqAtCDTqY
bQZ4nvBqox0CCqIopI3CgUY+bWfa3j/+hQ5CKhLetbf7uBunlux3n+zUU5V6/wf0
8goUwFFSsdaOUAsamVy8C8m97e34XsFW201+I6QRoSzUGwWa5BtS9nw4mQVLunKN
```

```
QolgBGYq9P1o12v3mUEo1mwkq+YlUy7Igpnioo8jvjCDsSeL+mh/AUnoxphrEC6Y
XorXykuxx8lYmtA225aV7LaB5PLNbxt5h0wQPInkTfpU3Kqm
-----END CERTIFICATE-----
---
Server certificate
subject=/CN=api.push.apple.com/OU=management:idms.group.533599/O=Apple
Inc./ST=California/C=US
issuer=/CN=Apple IST CA 2 - G1/OU=Certification Authority/O=Apple Inc./C=US
---
Acceptable client certificate CA names
/C=US/O=Apple Inc./OU=Apple Certification Authority/CN=Apple Root CA
/C=US/O=Apple Inc./OU=Apple Worldwide Developer Relations/CN=Apple Worldwide Developer
Relations Certification Authority
/CN=Apple Application Integration 2 Certification Authority/OU=Apple Certification
Authority/O=Apple Inc./C=US
/C=US/O=Apple Inc./OU=Apple Worldwide Developer Relations/CN=Apple Worldwide Developer
Relations Certification Authority
/C=US/O=Apple Inc./OU=Apple Certification Authority/CN=Apple Application Integration
Certification Authority
---
SSL handshake has read 3905 bytes and written 483 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: 660D1C58C1F899DA06CB7ECE04656720ED590C3416B89D011408513DC012EBFB
    Session-ID-ctx:
    Master-Key:
7D6884D3B2B5A1896D5E7AF9FB5E116B6F190E017214C60538E12575903B11C47ACA0102D6ECC7D92B350DF532B
B15DE
    Start Time: 1533310648
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
```