Network Working Group
INTERNET-DRAFT
Intended Status: Request for Comments
Expires: July 2021

C. Edge

11 March 2021

AppConfig for Mobile Applications
draft-edge-appconfig-00.txt

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

****

This is a DRAFT edition of this statement of the Application Configuration protocol (AppConfig). Comments are sought on this document for consideration and possibly incorporated in the final edition. Comments are especially sought from those actually developing MDM and MDM solutions, and particular vendors and potential vendors of applications.

The period for comments is 90 days ending April 30, 2021, at which time a revised edition will be issued with a new RFC number.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

The list of current Internet-Drafts can be accessed at
https://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at
https://www.ietf.org/shadow.html

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.
****

Abstract

Many Service Providers offer application configuration options that manage the state of applications on mobile devices. AppConfig is used to distribute settings to applications upon installation or updated arbitrarily via a trusted Mobile Device Management (MDM) service. This document defines the specification by allowing a standardized format for data to stream into applications via XML (RFC 3470) or JSON interfaces (RFC 4627).

Table of Contents

1. Abstract

Many Service Providers offer application configuration options that manage the state of applications on mobile devices. AppConfig is used to distribute settings to applications upon installation or updated arbitrarily via a trusted Mobile Device Management (MDM) service. This document defines the specification by allowing a standardized format for data to stream into applications via XML (RFC 3470) or JSON interfaces (RFC 4627).

2. Introduction

AppConfig describes a method by which mobile application vendors can deploy applications with a payload of standard Extensible Markup Language (XML) or JavaScript Object Notation (JSON) that configures settings for application. Settings can include any input streamed to the device over the Internet, leveraging frameworks from an existing Mobile Device Management (MDM) solution for authorization to stream settings into apps at deployment or check-in time.

This provides an improvement to developing independent proprietary software development kits (SDKs) to enable configuration and management features of apps through a device management solution. The AppConfig standard allows developers to implement a consistent set of attributes to deploy apps through device management platforms alongside their configuration flow for user credentials for authentication as well as with custom settings or controls in the apps. Standardization also provides multi-platform compatibility and allows for data coming in from app stores and custom-deployed apps concurrently.

2.1 Conventions Used in This Document

> The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

The grammatical rules in this document are to be interpreted as described in RFC 4234.

2.2 Terminology

Mobile Device Management (MDM) describes a management system used to deploy management commands, apps, and profiles to mobile devices.

An object is an unordered collection of zero or more name/value pairs, where a name is a string and a value is a boolean, date, float, floatArray, integer, integerArray, string, or stringArray defined as follows:

* boolean: a key containing a 0 for a false state and a 1 for a true state. * date: a key that represents a date represented in the ISO-8601 format (e.g. 2019-10-25T21:34:30Z). * float: a decimal value of an integer (e.g. 0.1 or 19.0000333). * floatArray: an array of float objects. * integer is a whole number (e.g. 1 or 64999). * integerArray: an array of zero or more values of whole numbers. * string: A string is a sequence of zero or more Unicode characters * stringArray is an ordered sequence of zero or more values of Unicode strings.

The terms "object" and "array" are derived from the conventions of JavaScript.

3. Specification of Requirements

The problems that are to be solved in the AppConfig standard are mainly three:

* Standardize how, when, and why setting objects are deployed to mobile devices * Define the address space to be used so settings are interoperable between application vendors * Define authentication and authorization mechanisms for mobile and web apps

The aim of the AppConfig study is to define a plan that solves all these problems as a whole and not each of them separately.

The general requirements that we underline for this transition are:

- Transparency to MDM and application developers - Flexibility: Simplify the suitability to new communication technology and to topology changes due to new services provided or to different users needs.

This specification is designed for use with HTTP or HTTP over TLS. The use of AppConfig over any protocol other than HTTP or HTTP over TLS is out of scope.

3a. Requirements
There are a number of requirements to using AppConfig.
Some are optional, based on the adherence of a particular vendor or implementer to the standard, and their security posture.

The device an application is being installed on MUST be enrolled (joined) to an existing MDM.

The application (binary) configured by AppConfig MUST be installed by an MDM in order to prevent potentially malicious activity.

The MDM server MUST support AppConfig and provide feedback.

The MDM server SHOULD support dynamic updating of an application installed using AppConfig.

An implementation may set limits on the size of texts that it accepts.
An implementation may set limits on the maximum depth of nesting.
An implementation may set limits on the range of numbers.
An implementation may set limits on the length and character contents of strings.

4. AppConfig Interaction Procedures AppConfig leverages existing Mobile Device Management (MDM) technology to run commands on mobile computing devices. Those commands install an app, alongside a profile that configures settings for the app. Those settings are then interpreted by an agent on the device that process the app installation as well as the application of the settings file. Responses are then sent using the AppConfig Application Feedback protocol. The settings and responses are standardized in the AppConfig protocol. The MDM then uses the XML to define and build the User Interface on the admin console. The administrator then uses that console to edit the desired settings and once complete uses the MDM to push the settings to devices per the needs of the platform accepting the management command.

4.1 Overview The AppConfig standard communications leverage an existing Mobile Device Management (MDM) framework to send settings into apps. Developers create an XML document per the Managed App Configuration spec for a given app. The XML document is loaded into the MDM and the MDM provides the translation to the platform-specific needs to leverage endpoints available per-platform. Key-value pairs and other elements are then sent over existing frameworks per the documentation of each platform adopting the specification.

4.2 Details The flow of information can be seen in Figure 4-2. Here, the following steps are taken:

* A: The MDM sends a push notification indicating that a command is waiting to be performed. * B: The client responds to the push notification, checking in and downloading the app to be installed. * C: The MDM sends the bundleID or the URL to a self-published app according to the type of app. * D: The client device downloads the app from the online store or the URL provided. * E: Once the app is installed, the managed app configuration profile is installed.

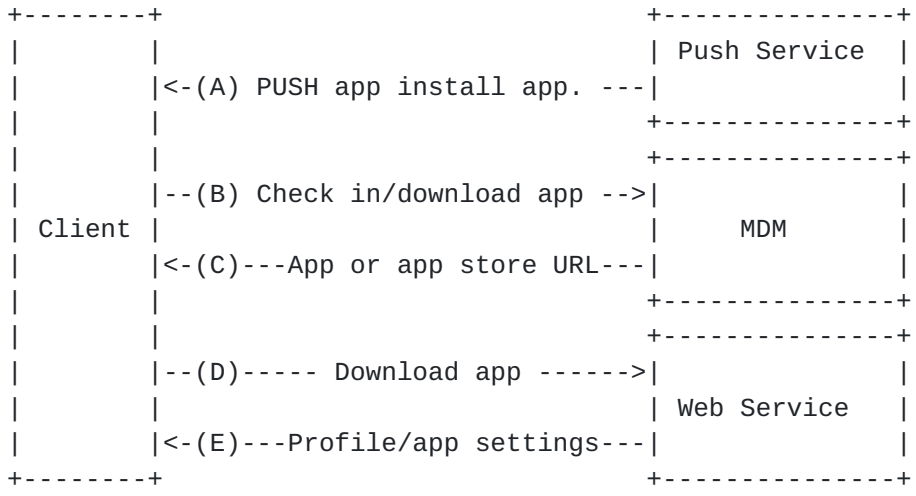

Figure 4-2: AppConfig MDM Interaction Subsequent updates are then handled by repeating steps A, B, and E. Each step in the process is encrypted, but the data at rest on devices may or may not be encrypted.

5. AppConfig Standards The AppConfig Standards define the information being send to devices. These involve profiles that contain a number of elements, defined in the following sections. These are comprised of a profile that is interpreted by a process running on the device, per the specifications of each vendor.

5.1 App Configuration Container Elements Apps have a number of settings that can be configured, including email addresses, user names, ports, paths, etc. Vendors and administrators also need to limit certain features. These are often configured at the initial installation of the app or ad hoc following that installation. Application Programming Interfaces (APIs) establish a standard for vendors to develop applications to control those objects. The AppConfig protocol allows these objects to be configured by an MDM in a standardized fashion. Applications then interpret the elements of each object that are provided in the form of key-value pairs.

The managedApplicationConfiguration is the root element and contains applications to be managed in the form of a bundleID. The bundleId is a string element that specifies the bundle ID of the app that the XML document is associated with. These are portrayed in the XML as follows:

<bundleId>com.myCompanyName.mAppName</bundleId>

These contain a number of key-value pairs that contain a dictionary of settings (dict for short), a version key, and a presentation key. The Version key is an integer that is 1-5 digits long. For example:

<version>2</version>

The dictionary contains a number of keyName attributes. This is the name of the key in the key-value pair that the MDM service sends to the app. The app uses this name to look up the configuration setting in the managed app configuration dictionary (e.g. in iOS this is called NSDictionary).

The name of the keys must:

* Contain only alphanumeric characters, spaces,_(underscore), and -(dash). * Be unique within the dict element (e.g. keyName="allowAdvancedOptions").

Keys also have a default value and a constraint. These, respectively, allow a developer to:

* Send data into the app container as the value in a key-value pair if the MDM administrator makes no change to the user interface control. * Define the allowable settings for the app. * Display data in the user interface control for the configuration setting.

Constraint is the allowable settings in a given field, using the following options:

* nullable: Set to true to allow the value in the key-value pair to be null. The default value is false. * min: The lower bound for the value. If the min attribute is not specified, the value has no lower bound. * max: The upper bound for the value. If the max attribute is not specified, the value has no upper bound. * pattern: A regular expression to use to validate a string value of a key.

Furter allowable elements that SHOULD be used include the following:

* Key: Unique identifier for the key within the app that is being defined. This is then handled as a parameter passed into apps. The key is a maximum 100 characters. * Data: Optional machine-readable value provided by the MDM to allow managing the setting defined in the key. For example, an MDM admin could send a domain name into an app so a user does not need to provide the domain name. Maximum 1000 characters. * Description: Optional description of what the key does, which can be displayed in a management platform, feed, or app. * Version: The version number should be an integer that increases sequentially so the management platform has a trigger to send a new payload to client devices.

5.2 Existing Key Standards Keys are camel-cased where each word in a key is separated by an underscore. Keys are used to perform one of four actions:

* Backend Configuration: Configures the app to connect to a multi-tenant environment. * User Configuration: Configures the user credentials or sub-tenant * App Configuration Settings: Configures the default or additional settings. Sent as an array of objects. * App Security Restrictions: Configures various features of an app. Examples provided, but extensible.

5.2.1 Backend Configuration Performs the connection to a web service for an app. Keys include:

* Server_URL: A string containing the URL to access a given web service. * Port: The port to connect to the URL of the web service. * Use_SSL: Boolean that enables an SSL-based connection to the web service. * Tenant_ID: A string that enables a tenant ID for a web service.

5.2.2 User Configuration Configures settings for a user of an app on a device. Keys include:

* Username: A string that represents the unique name used for a tenant to access a web service over the URL provided. * Email: A string that represents an email address when used in addition or to replace a username. * Domain: A domain name that houses a username and/or email address within a tenant, if required.

Note: The underscore separated words are due to the prevalence of acronyms in the strings used.

5.2.3 App Configuration MDM Service Settings

Where appropriate, developers SHOULD use the MDM service device variables as the default value for key-value pairs where appropriate. These include the following identifiers:

* iccid: Integrated Circuit Card Identifier (e.g. 89014104254287052057 * imei: International Mobile Equipment Identity (e.g. 01 342300 291808 3) * imsi: International Mobile Subscriber Identity (e.g. 310150123456789) * meid: Mobile Equipment Identifier (e.g. A0123456789012) * model: The model of the device (e.g. iPhone 10) * phone_Number: The phone number associated with the device * serial_Number: The vendor-supplied serial number of the device * udid: The Unique Device Identifier (e.g. c752e7052fe5e5ca8166e408c4b48573b5b5bd82) * wifi_Mac_Address: Wi-Fi MAC Address (e.g. 30:f7:c5:87:e8:78)

5.2.4 Global App Configuration Settings Organizations CAN implement granular app configurations. The client will interpret these as how the app is displayed on the device:

* Application_Name: The name of the application as displayed on the device. * Application_Icon: The icon file to be used in the form of a URI to the icon file itself (the format of which can be different per platform and device used. * Arbitrary application keys: Standard key value pairs that control various settings of a given app, defined previously in this document.

5.2.5 App Security Restrictions Organizations CAN implement granular security and data loss protection in applications deployed to users. This prevents sensitive data from leaving the control of the company and prevents applications from being used in unintended ways. These security features can be managed in an object-oriented fashion, based on the login built into the MDM. Applications currently have some capabilities to restrict various features through the use of the existing appconfig.org spec or through an SDK; however, this document codifies the use and provides an extensible framework in the form of a namespace schema to be applied to additional capabilities.

As with App Configuration Container Elements, some keys are built in, in order to provide compatibility with earlier features and options. These include the following keys and corresponding capabilities:

* App_Security_Passcode: Used to set a pincode, fingerprint and/or facial recognition key in an application. * App_Security_Managed_Open_In: Globally restricts all data deployed into a device via an MDM provider from being moved between other objects (e.g. apps, email accounts, etc). * App_Security_Prevent_App_Backup: Interpreted by the mobile device platform to prevent backups to cloud services. * App_Security_Disable_Screen_Capture: Disables the ability for the device to perform a screen capture. * App_Security_Enforce_App_Encryption: Force device passcode security control in MDM; enforces the native data protection encryption. * App_Security_Remotely_Wipe_App: Enables the ability to remotely wipe the app from a device. * App_Security_Disable_Copy_Paste: Disable the ability to copy and paste from within the app to another app.

Standard configuration keys for enterprise apps are included in this section of the document; however, each developer can name each key as per their logic, provided the keys do not conflict in namespace with keys provided in the specification. Managed Configuration specification files are then made publicly available to download and consume and when an application is added. The location of the specification file is provided to allow for a consistent display in the MDM and the app.

5.2.6 App User Variables A number of uses for app configuration involve the expansion of user-oriented variables. Those are supported based on the users found in the database of the MDM, and/or through a directory service. The built-in options for applications to leverage include the following: * cn: Common Name (CN) attribute extracted from the distinguished name. * displayName: The name displayed in the directory service or interface. * Dn: Distinguished name when used with a common LDAP interface. * emailAddressDomain: The domain portion of the email address. * emailAddressLocalPart: The local portion of the email address. * emailAddress: The full email address for a user. * firstName: The first name of a user. * lastName: The late name or names of a user. * Locale: The language preference for a user. * ou: The Organizational Unit (OU) attribute extracted from the distinguished name. * sAMAccountName: The Microsoft sAMAccountName attribute, for use when Active Directory is the source. * username: The Login ID in the form of an email address. * upn: The Microsoft userPrincipalName attribute (for use when using Active Directory as the source).

Federated Identity Management solutions and specs vary from this and integration through those standards are defined later in this document.

5.3 Localization Localized strings in the XML document define descriptive text for containers, as it is displayed for additional languages, to render in the administrative console of an MDM product. Localized strings are represented in language elements that contain a string for each language-region code supported. For example:

The following elements contain language elements:

* Field: The field element defines the admin console UI control for a configuration setting * Description: Elements that render descriptive text about the UI control. * name elements: Specify the name of a group of elements provided as fields (or the field to provide the setting for). * language: Defines the language in the form of a code or where needed language-region code. * type: * selected: Boolean operator that enables the dictionary. * value: The value as output based on the language selected. * defaultLocale: The default language value to be used. * fieldGroup: Defines multiple key elements in a grouping. * option: Defines a list of elements, each SHOULD specify localized option names for an option in environments using select or multiselect UI controls.

For example, to define en-US and es-ES, use the following key structure represents keyNames startTime that is Start Time in en-US but then rendered as the Hora de Inicio translation in Spanish and then with descriptive text for each as used in a description field but useable as more:

```
<presentation defaultLocale="en-US"> <field keyName="startTime"
type="input"> <label> <language value="en-US">Start Time</language>
<language value="es-ES">Hora de Inicio</language> </label> <description>
<language value="en-US">The time of day to collect data</language>
<language value="en-US">La hora del dia para recoger los
datos</language> </description> </field> </presentation>
```

This XML files creates a standard scheme to document the accepted configs and values that your app supports. Many MDM vendors support automatically parsing these files in the MDM admin console.

5.5 Considerations The application can be a public app in an application store or may be an internally developed app that has been properly signed.

According to the vendor implementation, data can be extracted from devices. Therefore, do not use AppConfig Container Elements to store private data such as certificates or passwords. Instead refer to Single Sign-On Container Elements later in this section.

Operating system vendors typically SHOULD restrict a device to being enrolled in a single MDM at a time and therefore only one MDM can write to a given AppConfig domain at a time.

The MDM system MUST detect and take remediation action on a device that has been compromised (e.g. due to vulnerability or being jailbroken) and that may then expose the managed configurations.

Each MDM has the option to setup AppConfig and corresponding services differently. Contact the MDM vendor for documentation specific to their system if needed.

Sensitive information such as passwords or certificates should not be sent to the device using this approach.

5.2 App Tunnel Container Elements

Some applications CAN require access to a given web service that sits behind a firewall. In those cases, the application MAY require a secure application tunnel to connect between an application on a device and the network hosting a server. This might be to access a Federated Identity Provider (IDP) using SAML or OAuth, or for an application to authenticate into a private on premises service.

Application tunneling works by connecting an application o a VPN using built-in per-app VPN functionality. The Per-App VPN might be hosted by the MDM or leverage existing infrastructure. The Per-App VPN functionality is deployed using a profile and the device then connects to the service; the application vendor developing the app then has no requirement to develop a specific option in the app.

The App Tunnel settings are global on the device. Keys made available to enable this functionality SHOULD include the following:

* UserDefinedName: The name displayed in the VPN settings on the device. * VPNType: Defines the settings available for the payload, including L2TP, PPTP, IPSec, IKEv2, AlwaysOn, and VPN. * VPNSubType: Defines the vendor bundle identifier of the VPN plugin used (e.g. com.cisco.anyconnect.applevpn.plugin). * ProviderBundleIdentifier: app-proxy or packet-tunnel to define the option in the vendor * OnDemandEnabled: Boolean that enables the VPN connection when the app is opened. * OnDemandRules: Array defining the options for the on-demand connection, with options definable per device manufacturer and per VPNType.

The user of a given app then substantiates the connection to the VPN automatically when the App is opened.

5.3 Single Sign-On Container Elements

Application developers SHOULD provide a mechanism for users to log into applications using credentials derived from a Federated Identity Provider (IdP). This means supporting a SAML or OAuth that SHOULD allow for multiple authentication flows. The interpretation of required attributes, by application, can then be deployed using an existing technology such as FIDO AppID, Webauthn to embed a screen at initial launch, or a standard flow using a supported app.

The applications backend service must support identity federation to an organization identity provider (IDP) via a standard such as SAML, OAuth or other protocols.

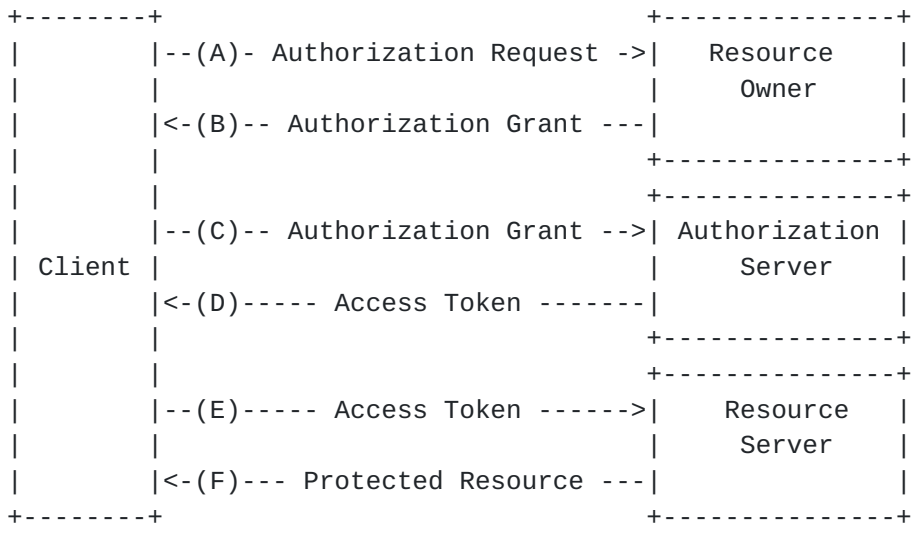

Figure 1: Abstract Protocol Flow

Using built-in frameworks mean that when the app is launched, the user will see the identity provider login screen and once the user is authenticated, any future apps that leverage the same identity provider will be able to detect the existing authenticated session and will not need to prompt the user to login again. This standard does not include any specifics for performing these actions other than to indicate that data at rest on devices is often in an unencrypted state and so credentials and/or tokens should not live in the application container or a globally accessible unencrypted location on the device filesystem.

6. Application Feedback

Mobile Device Management (MDM) providers SHOULD provide an application_feedback REST endpoint to receive a response from the application should the application have information to send to the MDM. The API endpoint available listens for a PUT that contains a keyed app state. That app state contains the status of the apps installed on managed devices and leverages existing encryption for a MDM the device is joined to. As an example a MDM can listen for a custom key that indicates the setup of a given app is complete.

6a. Application Feedback Container Elements

Components of a keyed app state are represented in the container elements of the application feedback payload. This information is sent back to the MDM by the application using the ManagedApplicationFeedback command over a standard REST interface. These elements are comprised of the following:

* Type: The type of feedback in the message. * Key: Unique identifier for the app state. Maximum 100 characters. * Message: Optional message describing the app state. Maximum 1000 characters. Note: Typically messages should be significantly shorter than this. * Data: Optional machine-readable value intended for MDMs to allow IT admins to set up alerts or filters based on the value. For example, an IT admin could set up an alert if the data field battery_percentage < 10. Maximum 1000 characters. * Severity: The severity of the app state. Allowable values are SEVERITY_ERROR and SEVERITY_INFO(default). Only set severity to SEVERITY_ERROR for genuine error conditions that an organization needs to take action to fix. * Timestamp: When a keyed app state is set, it's automatically sent with a timestamp in milliseconds since epoch. * DeleteFeedback: Removes the feedback so the feedback is not sent again.

To send a ManagedApplicationFeedback command, the server sends a dictionary containing the following keys:

* RequestType (String): Contains ManagedApplicationFeedback. * Identifiers (Array): An array of managed bundle identifiers, as strings. * DeleteFeedback (Boolean): Removes the dictionary containing the feedback once read by the MDM.

7. Security Considerations As stated in RFC 2617, the greatest sources of risks are usually found not in the core protocol itself but in policies and procedures surrounding its use. Implementers are strongly encouraged to assess how this protocol addresses their security requirements.

8. IANA Considerations

The values of the Foobar parameter are assigned by the Barfoo registry on behalf of the Rabfoo Forum. Therefore, this document has no IANA actions.

Appendix A. Acknowledgements This document is based, in part, on the work done by the Appconfig.org community.

This specification is directly based on the AppConfig.org community specification, which in turn was modeled after existing proprietary protocols from Apple and Google, and best practices that have been independently implemented by various companies.

The community specification was edited by Pepjin Bruienne, Kyle Hammond,

The editor would like to thank the following individuals for their invaluable contribution to the publication of this edition of the protocol: XXX

Appendix B. Normative References The normative references used in this document include the following:

RFC 2119: https://tools.ietf.org/html/rfc2119 RFC 8174: https://tools.ietf.org/html/rfc8174 HTTP: http://www.rfc-editor.org/info/rfc2616 HTTP over TLS: https://www.rfc-editor.org/info/rfc2818 Oauth 2: https://www.rfc-editor.org/info/rfc6749 XML: http://www.rfc-editor.org/info/rfc3470 JSON: http://www.rfc-editor.org/info/rfc4627 URI: https://tools.ietf.org/html/rfc3986 L2TP: https://tools.ietf.org/html/rfc3931 PPTP: https://tools.ietf.org/html/rfc2637 IPSec: https://tools.ietf.org/html/rfc6071 IKE: https://tools.ietf.org/html/rfc5996 ISO-8601: https://www.iso.org/iso-8601-date-and-time-format.html RFC 4234: https://tools.ietf.org/html/rfc4234 RFC 5198: https://tools.ietf.org/html/rfc5198 RFC 2617: https://tools.ietf.org/html/rfc2617

Appendix C. Informative References The informative references used in this document include the following:

https://developer.android.com/work/app-feedback/overview
https://www.appconfig.org/ios/
https://storage.googleapis.com/appconfig-media/appconfig-content/uploads
/2017/01/ManagedAppConfig.pdf
https://developer.apple.com/business/documentation/MDM-Protocol-
Reference.pdf
https://developer.apple.com/library/ios/samplecode/sc2279/Introduction/
Intro. html

Enterprise Distribution
https://developers.google.com/android/management/introduction
https://docs.aws.amazon.com/iot-device-management/index.html

Authors' Address

Charles Edge
Jamf
100 S Washington Ave 1100
Minneapolis, MN 55418 US
Phone: +1 612 581 6602
Email: krypted@jamf.com

Bob Whiteman
Apple
One Infinite Loop Cupertino, CA 95014 US Phone: +1
408 606 5775
EMail: bwhiteman@apple.com

Paul Cerat
Microsoft
One Microsoft Way
Redmond, WA 98052 US Phone: +1
425 882 8080
EMail: andrew.cerat@microsoft.com

Anuj Goyal
Google
1600 Amphitheatre Parkway
Mountain View, California
Email: goanuj@google.com

Sam Weiss
Jamf
100 S Washington Ave #1100
Minneapolis, MN 55418 US
EMail: sam.weiss@jamf.com

James Felton
Jamf
100 S Washington Ave #1100
Minneapolis, MN 55418 US
EMail: james.felton@jamf.com

Kalyan Vishnubhotla
MobileIron
490 E Middlefield Rd
Mountain View, CA

94043 US
EMail: vkalyan@mobileiron.com