Internet Engineering Task Force (IETF)
Request for Comments: 8886
Category: Informational
ISSN: 2070-1721

W. Kumari
Google
C. Doyle
Juniper Networks

September 2020

Secure Device Install

Abstract

Deploying a new network device in a location where the operator has no staff of its own often requires that an employee physically travel to the location to perform the initial install and configuration, even in shared facilities with "remote-hands" (or similar) support. In many cases, this could be avoided if there were an easy way to transfer the initial configuration to a new device while still maintaining confidentiality of the configuration.

This document extends existing vendor proprietary auto-install to provide limited confidentiality to initial configuration during bootstrapping of the device.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8886.

Table of Contents

1. Introduction

In a growing, global network, significant amounts of time and money are spent deploying new devices and "forklift" upgrading existing devices. In many cases, these devices are in shared facilities (for example, Internet Exchange Points (IXP) or "carrier-neutral data centers"), which have staff on hand that can be contracted to perform tasks including physical installs, device reboots, loading initial configurations, etc. There are also a number of (often proprietary) protocols to perform initial device installs and configurations. For example, many network devices will attempt to use DHCP [RFC2131] or DHCPv6 [RFC8415] to get an IP address and configuration server and then fetch and install a configuration when they are first powered on.

The configurations of network devices contain a significant amount of security-related and proprietary information (for example, RADIUS [RFC2865] or TACACS+ [TACACS] secrets). Exposing these to a third party to load onto a new device (or using an auto-install technique that fetches an unencrypted configuration file via TFTP [RFC1350]) or something similar is an unacceptable security risk for many operators, and so they send employees to remote locations to perform the initial configuration work; this costs time and money.

There are some workarounds to this, such as asking the vendor to preconfigure the device before shipping it; asking the remote support to install a terminal server; providing a minimal, unsecured configuration and using that to bootstrap to a complete configuration; etc. However, these are often clumsy and have security issues. As an example, in the terminal server case, the console port connection could be easily snooped.

An ideal solution in this space would protect both the confidentiality of device configuration in transit and the authenticity (and authorization status) of configuration to be used by the device. The mechanism described in this document only addresses the former and makes no effort to do the latter, with the device accepting any configuration file that comes its way and is

encrypted to the device's key (or not encrypted, as the case may be). Other solutions (such as Secure Zero Touch Provisioning (SZTP) [RFC8572], Bootstrapping Remote Secure Key Infrastructures (BRSKI) [BRSKI], and other voucher-based methods) are more fully featured but also require more complicated machinery. This document describes something much simpler, at the cost of only providing limited protection.

This document layers security onto existing auto-install solutions (one example of which is [Cisco_AutoInstall]) to provide a method to initially configure new devices while maintaining (limited) confidentiality of the initial configuration. It is optimized for simplicity, for both the implementor and the operator. It is explicitly not intended to be a fully featured system for managing installed devices nor is it intended to solve all use cases; rather, it is a simple targeted solution to solve a common operational issue where the network device has been delivered, fiber has been laid (as appropriate), and there is no trusted member of the operator's staff to perform the initial configuration. This solution is only intended to increase confidentiality of the information in the configuration file and does not protect the device itself from loading a malicious configuration.

This document describes a concept and some example ways of implementing this concept. As devices have different capabilities and use different configuration paradigms, one method will not suit all, and so it is expected that vendors will differ in exactly how they implement this.

This solution is specifically designed to be a simple method on top of exiting device functionality. If devices do not support this new method, they can continue to use the existing functionality. In addition, operators can choose to use this to protect their configuration information or can continue to use the existing functionality.

The issue of securely installing devices is in no way a new issue nor is it limited to network devices; it occurs when deploying servers, PCs, Internet of Things (IoT) devices, and in many other situations. While the solution described in this document is obvious (encrypt the config, then decrypt it with a device key), this document only discusses the use for network devices, such as routers and switches.

2. Overview

Most network devices already include some sort of initial bootstrapping logic (sometimes called 'autoboot' or 'autoinstall'). This generally works by having a newly installed, unconfigured device obtain an IP address for itself and discover the address of a configuration server (often called 'next-server', 'siaddr', or 'tftp-server-name') using DHCP or DHCPv6 (see [RFC2131] and [RFC8415]). The device then contacts this configuration server to download its initial configuration, which is often identified using the device's serial number, Media Access Control (MAC) address, or similar. This document extends this (vendor-specific) paradigm by allowing the configuration file to be encrypted.

This document uses the serial number of the device as a unique device identifier for simplicity; some vendors may not want to implement the system using the serial number as the identifier for business reasons (a competitor or similar could enumerate the serial numbers and determine how many devices have been manufactured). Implementors are free to choose some other way of generating identifiers (e.g., a Universally Unique Identifier (UUID) [RFC4122]), but this will likely make it somewhat harder for operators to use (the serial number is usually easy to find on a device).

2.1. Example Scenario

Operator_A needs another peering router, and so they order another router from Vendor_B to be drop-shipped to the facility. Vendor_B begins assembling the new device and tells Operator_A what the new device's serial number will be (SN:17894321). When Vendor_B first installs the firmware on the device and boots it, the device generates a public-private key pair, and Vendor_B publishes the public key on its key server (in a public key certificate, for ease of use).

While the device is being shipped, Operator_A generates the initial device configuration and fetches the certificate from Vendor_B key servers by providing the serial number of the new device. Operator_A then encrypts the device configuration and puts this encrypted configuration on a (local) TFTP server.

When the device arrives at the Point of Presence (POP), it gets installed in Operator_A's rack and cabled as instructed. The new device powers up and discovers that it has not yet been configured. It enters its autoboot state and begins the DHCP process. Operator_A's DHCP server provides it with an IP address and the address of the configuration server. The router uses TFTP to fetch its configuration file. Note that all of this is existing functionality. The device attempts to load the configuration file. As an added step, if the configuration file cannot be parsed, the device tries to use its private key to decrypt the file and, assuming it validates, proceeds to install the new, decrypted configuration.

Only the "correct" device will have the required private key and be able to decrypt and use the configuration file (see Security Considerations (Section 7)). An attacker would be able to connect to the network and get an IP address. They would also be able to retrieve (encrypted) configuration files by guessing serial numbers (or perhaps the server would allow directory listing), but without the private keys, an attacker will not be able to decrypt the files.

3. Vendor Role

This section describes the vendor's roles and provides an overview of what the device needs to do.

3.1. Device Key Generation

Each device requires a public-private key pair and for the public part to be published and retrievable by the operator. The cryptographic algorithm and key lengths to be used are out of the scope of this document. This section illustrates one method, but, as with much of this document, the exact mechanism may vary by vendor. Enrollment over Secure Transport [RFC7030] and possibly the Simple Certificate Enrollment Protocol [RFC8894] are methods that vendors may want to consider.

During the manufacturing stage, when the device is initially powered on, it will generate a public-private key pair. It will send its unique device identifier and the public key to the vendor's directory server [RFC5280] to be published. The vendor's directory server should only accept certificates that are from the manufacturing facility and that match vendor-defined policies (for example, extended key usage and extensions). Note that some devices may be constrained and so may send the raw public key and unique device identifier to the certificate publication server, while more capable devices may generate and send self-signed certificates. This communication with the directory server should be integrity protected and should occur in a controlled environment.

This reference architecture needs a serialization format for the key material. Due to the prevalence of tooling support for it on network devices, X.509 certificates are a convenient format to exchange public keys. However, most of the metadata that would be used for revocation and aging will not be used and should be ignored by both the client and server. In such cases, the signature on the certificate conveys no value, and the consumer of the certificate is expected to pin the end-entity key fingerprint (versus using a PKI and signature chain).

3.2. Directory Server

The directory server contains a database of certificates. If newly manufactured devices upload certificates, the directory server can simply publish these; if the devices provide the raw public keys and unique device identifier, the directory server will need to wrap these in a certificate.

The customers (e.g., Operator_A) query this server with the serial number (or other provided unique identifier) of a device and retrieve the associated certificate. It is expected that operators will receive the unique device identifier (serial number) of devices when they purchase them and will download and store the certificate. This means that there is not a hard requirement on the reachability of the directory server.

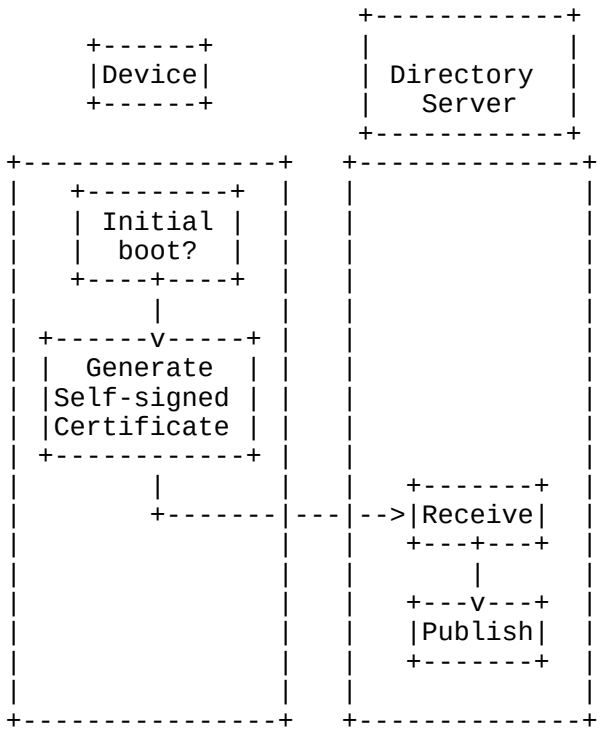

Figure 1: Initial Certificate Generation and Publication

4. Operator Role

4.1. Administrative

When purchasing a new device, the accounting department will need to get the unique device identifier (e.g., serial number) of the new device and communicate it to the operations group.

4.2. Technical

The operator will contact the vendor’s publication server and

download the certificate (by providing the unique device identifier of the device). The operator fetches the certificate using a secure transport that authenticates the source of the certificate, such as HTTPS (confidentiality protection can provide some privacy and metadata-leakage benefit but is not key to the primary mechanism of this document). The operator will then encrypt the initial configuration (for example, using S/MIME [RFC8551]) using the key in the certificate and place it on their configuration server.

See Appendix A for examples.

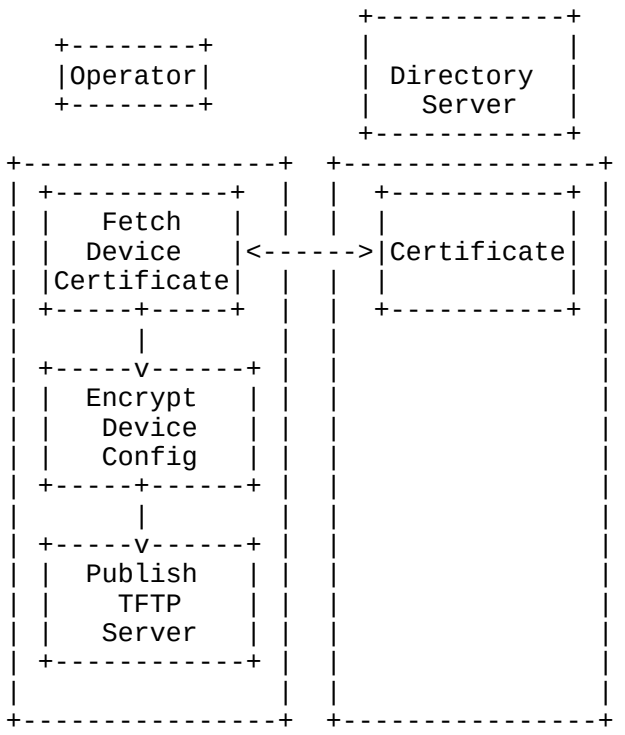

Figure 2: Fetching the Certificate, Encrypting the Configuration, and Publishing the Encrypted Configuration

4.3. Example Initial Customer Boot

When the device is first booted by the customer (and on subsequent boots), if the device does not have a valid configuration, it will use existing auto-install functionality. As an example, it performs DHCP Discovery until it gets a DHCP offer including DHCP option 66 (Server-Name) or 150 (TFTP server address), contacts the server listed in these DHCP options, and downloads its configuration file. Note that this is existing functionality (for example, Cisco devices fetch the config file named by the Bootfile-Name DHCP option (67)).

After retrieving the configuration file, the device needs to determine if it is encrypted or not. If it is not encrypted, the existing behavior is used. If the configuration is encrypted, the process continues as described in this document. If the device has been configured to only support encrypted configuration and determines that the configuration file is not encrypted, it should abort. The method used to determine if the configuration is encrypted or not is implementation dependent; there are a number of (obvious) options, including having a magic string in the file header, using a file name extension (e.g., config.enc), or using specific DHCP options.

If the file is encrypted, the device will attempt to decrypt and parse the file. If able, it will install the configuration and start using it. If it cannot decrypt the file or if parsing the

configuration fails, the device will either abort the auto-install process or repeat this process until it succeeds. When retrying, care should be taken to not overwhelm the server hosting the encrypted configuration files. It is suggested that the device retry every 5 minutes for the first hour and then every hour after that. As it is expected that devices may be installed well before the configuration file is ready, a maximum number of retries is not specified.

Note that the device only needs to be able to download the configuration file; after the initial power on in the factory, it never needs to access the Internet, vendor, or directory server. The device (and only the device) has the private key and so has the ability to decrypt the configuration file.

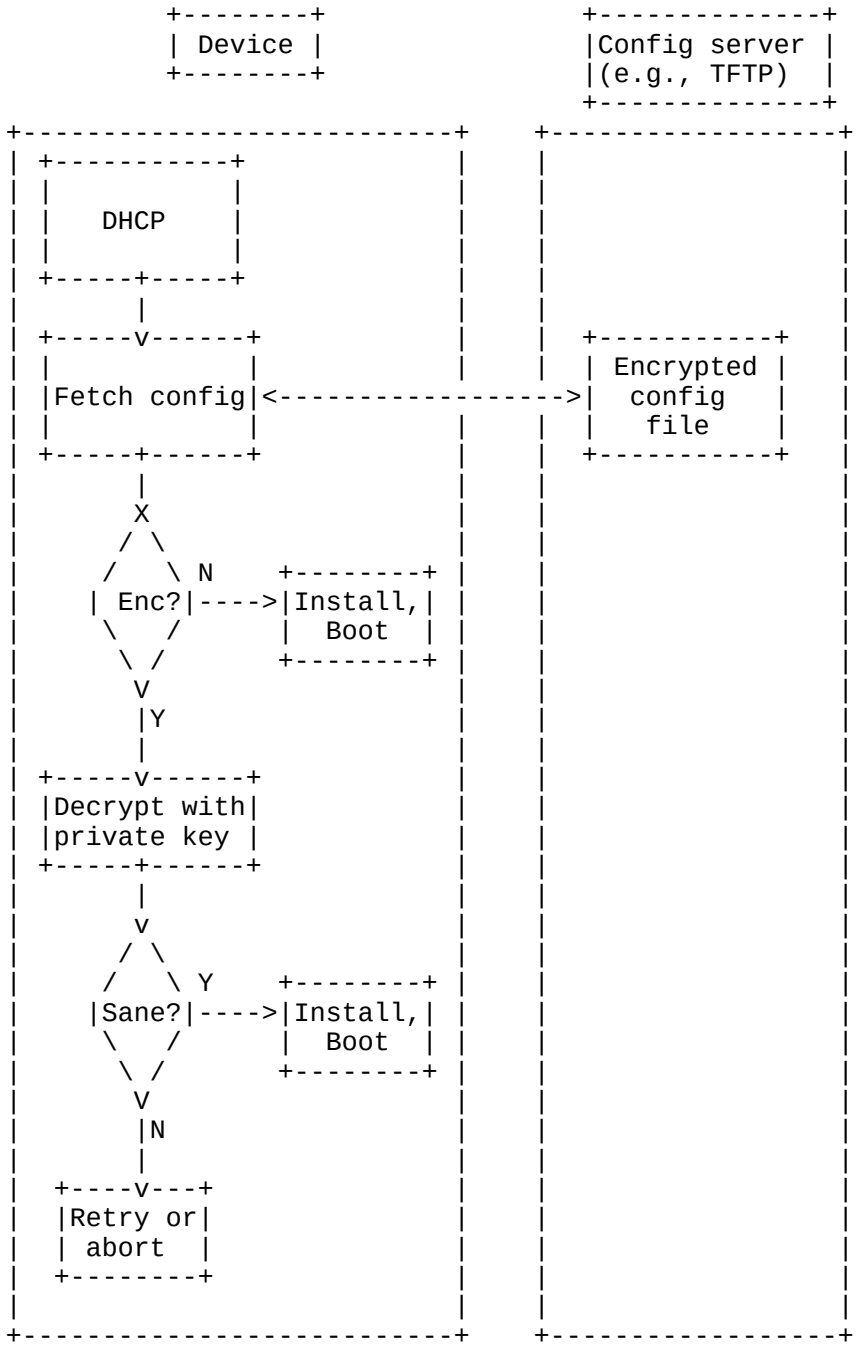

Figure 3: Device Boot, Fetch, and Install Configuration File

5. Additional Considerations

5.1. Key Storage

Ideally, the key pair would be stored in a Trusted Platform Module (TPM) on something that is identified as the "router" -- for example, the chassis/backplane. This is so that a key pair is bound to what humans think of as the "device" and not, for example, (redundant) routing engines. Devices that implement IEEE 802.1AR [IEEE802-1AR] could choose to use the Initial Device Identifier (IDevID) for this purpose.

5.2. Key Replacement

It is anticipated that some operator may want to replace the (vendor-provided) keys after installing the device. There are two options when implementing this: a vendor could allow the operator's key to completely replace the initial device-generated key (which means that, if the device is ever sold, the new owner couldn't use this technique to install the device), or the device could prefer the operator's installed key. This is an implementation decision left to the vendor.

5.3. Device Reinstall

Increasingly, operations are moving towards an automated model of device management, whereby portions of the configuration (or the entire configuration) are programmatically generated. This means that operators may want to generate an entire configuration after the device has been initially installed and ask the device to load and use this new configuration. It is expected (but not defined in this document, as it is vendor specific) that vendors will allow the operator to copy a new, encrypted configuration (or part of a configuration) onto a device and then request that the device decrypt and install it (e.g., 'load replace <filename> encrypted'). The operator could also choose to reset the device to factory defaults and allow the device to act as though it were the initial boot (see Section 4.3).

6. IANA Considerations

This document has no IANA actions.

7. Security Considerations

This reference architecture is intended to incrementally improve upon commonly accepted "auto-install" practices used today that may transmit configurations unencrypted (e.g., unencrypted configuration files that can be downloaded connecting to unprotected ports in data centers, mailing initial configuration files on flash drives, or emailing configuration files and asking a third party to copy and paste them over a serial terminal) or allow unrestricted access to these configurations.

This document describes an object-level security design to provide confidentiality assurances for the configuration stored at rest on the configuration server and for configuration while it is in transit between the configuration server and the unprovisioned device, even if the underlying transport does not provide this security service.

The architecture provides no assurances about the source of the encrypted configuration or protect against theft and reuse of devices.

An attacker (e.g., a malicious data center employee, person in the supply chain, etc.) who has physical access to the device before it is connected to the network or who manages to exploit it once installed may be able to extract the device private key (especially

if it is not stored in a TPM), pretend to be the device when connecting to the network, and download and extract the (encrypted) configuration file.

An attacker with access to the configuration server (or the ability to route traffic to configuration server under their control) and the device's public key could return a configuration of the attacker's choosing to the unprovisioned device.

This mechanism does not protect against a malicious vendor. While the key pair should be generated on the device and the private key should be securely stored, the mechanism cannot detect or protect against a vendor who claims to do this but instead generates the key pair off device and keeps a copy of the private key. It is largely understood in the operator community that a malicious vendor or attacker with physical access to the device is largely a "Game Over" situation.

Even when using a secure bootstrap mechanism, security-conscious operators may wish to bootstrap devices with a minimal or less-sensitive configuration and then replace this with a more complete one after install.

8. Informative References

[BRSKI] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-bootstrapping-keyinfra-44, 21 September 2020, <https://tools.ietf.org/html/draft-ietf-anima-bootstrapping-keyinfra-44>.

[Cisco_AutoInstall] Cisco Systems, Inc., "Using AutoInstall to Remotely Configure Cisco Networking Devices", Configuration Fundamentals Configuration Guide, Cisco IOS Release 15M&T, January 2018, <https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/fundamentals/configuration/15mt/fundamentals-15-mt-book/cf-autoinstall.html>.

[IEEE802-1AR] IEEE, "IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity", IEEE Std 802-1AR, June 2018, <https://standards.ieee.org/standard/802_1AR-2018.html>.

[RFC1350] Sollins, K., "The TFTP Protocol (Revision 2)", STD 33, RFC 1350, DOI 10.17487/RFC1350, July 1992, <https://www.rfc-editor.org/info/rfc1350>.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <https://www.rfc-editor.org/info/rfc2131>.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <https://www.rfc-editor.org/info/rfc2865>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key

Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

[RFC8415] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <https://www.rfc-editor.org/info/rfc8415>.

[RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <https://www.rfc-editor.org/info/rfc8551>.

[RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <https://www.rfc-editor.org/info/rfc8572>.

[RFC8894] Gutmann, P., "Simple Certificate Enrolment Protocol", RFC 8894, DOI 10.17487/RFC8894, September 2020, <https://www.rfc-editor.org/info/rfc8894>.

[TACACS] Dahm, T., Ota, A., Medway Gash, D., Carrel, D., and L. Grant, "The TACACS+ Protocol", Work in Progress, Internet-Draft, draft-ietf-opsawg-tacacs-18, 20 March 2020, <https://tools.ietf.org/html/draft-ietf-opsawg-tacacs-18>.

Appendix A. Proof of Concept

This section contains a proof of concept of the system. It is only intended for illustration and is not intended to be used in production.

It uses OpenSSL from the command line. In production, something more automated would be used. In this example, the unique device identifier is the serial number of the router, SN19842256.

A.1. Step 1: Generating the Certificate

This step is performed by the router. It generates a key, then a Certificate Signing Request (CSR), and then a self-signed certificate.

A.1.1. Step 1.1: Generate the Private Key

```
$ openssl ecparam -out privatekey.key -name prime256v1 -genkey
$
```

A.1.2. Step 1.2: Generate the Certificate Signing Request

```
$ openssl req -new -key key.pem -out SN19842256.csr
Common Name (e.g., server FQDN or YOUR name) []:SN19842256
```

A.1.3. Step 1.3: Generate the (Self-Signed) Certificate Itself

```
$ openssl req -x509 -days 36500 -key key.pem -in SN19842256.csr
-out SN19842256.crt
```

The router then sends the key to the vendor’s key server for

publication (not shown).

A.2. Step 2: Generating the Encrypted Configuration

The operator now wants to deploy the new router.

They generate the initial configuration (using whatever magic tool generates router configs!), fetch the router's certificate, and encrypt the configuration file to that key. This is done by the operator.

A.2.1. Step 2.1: Fetch the Certificate

```
$ wget http://keyserv.example.net/certificates/SN19842256.crt
```

A.2.2. Step 2.2: Encrypt the Configuration File

S/MIME is used here because it is simple to demonstrate. This is almost definitely not the best way to do this.

```
$ openssl smime -encrypt -aes-256-cbc -in SN19842256.cfg\
   -out SN19842256.enc -outform PEM SN19842256.crt
$ more SN19842256.enc
-----BEGIN PKCS7-----
MIICigYJKoZIhvcNAQcDoIICezCCAncCAQAxggE+MIIBOgIBADAiMBUxEzARBgNV
BAMMClNOMTk4NDIyNTYCCQDJVuBlaTOb1DANBgkqhkiG9w0BAQEFAASCAQBABvM3
...
LZoq08jqlWhZZWhTKs4XPGHUdmnZRYIP8KXyEtHt
-----END PKCS7-----
```

A.2.3. Step 2.3: Copy Configuration to the Configuration Server

```
$ scp SN19842256.enc config.example.com:/tftpboot
```

A.3. Step 3: Decrypting and Using the Configuration

When the router connects to the operator's network, it will detect that it does not have a valid configuration file and will start the "autoboot" process. This is a well-documented process, but the high-level overview is that it will use DHCP to obtain an IP address and configuration server. It will then use TFTP to download a configuration file, based upon its serial number (this document modifies the solution to fetch an encrypted configuration file (ending in .enc)). It will then decrypt the configuration file and install it.

A.3.1. Step 3.1: Fetch Encrypted Configuration File from Configuration Server

```
$ tftp 2001:0db8::23 -c get SN19842256.enc
```

A.3.2. Step 3.2: Decrypt and Use the Configuration

```
$ openssl smime -decrypt -in SN19842256.enc -inform pkcs7\
   -out config.cfg -inkey key.pem
```

If an attacker does not have the correct key, they will not be able to decrypt the configuration file:

```
$ openssl smime -decrypt -in SN19842256.enc -inform pkcs7\
   -out config.cfg -inkey wrongkey.pem
Error decrypting PKCS#7 structure
140352450692760:error:06065064:digital envelope
 routines:EVP_DecryptFinal_ex:bad decrypt:evp_enc.c:592:
$ echo $?
4
```

Acknowledgments

The authors wish to thank everyone who contributed, including Benoit Claise, Francis Dupont, Mirja Kuehlewind, Sam Ribeiro, Michael Richardson, Sean Turner, and Kent Watsen. Joe Clarke also provided significant comments and review, and Tom Petch provided significant editorial contributions to better describe the use cases and clarify the scope.

Roman Danyliw and Benjamin Kaduk also provided helpful text, especially around the certificate usage and security considerations.

Authors' Addresses

Warren Kumari
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America

Email: warren@kumari.net

Colin Doyle
Juniper Networks
1133 Innovation Way
Sunnyvale, CA 94089
United States of America

Email: cdoyle@juniper.net