SUIT
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2020

B. Moran
H. Tschofenig
Arm Limited

March 09, 2020

Strong Assertions of IoT Network Access Requirements
draft-moran-suit-mud-00

Abstract

The Manufacturer Usage Description (MUD) specification describes the access and network functionality required a device to properly function. The MUD description has to reflect the software running on the device and its configuration. Because of this, the most appropriate entity for describing device network access requirements is the same as the entity developing the software and its configuration.

A network presented with a MUD file by a device allows detection of misbehavior by the device software and configuration of access control.

This document defines a way to link a SUIT manifest to a MUD file offering a stronger binding between the two.

Table of Contents

1. Introduction

Under [RFC8520], devices report a URL to a MUD manager in the network. RFC 8520 envisions different approaches for conveying the information from the device to the network such as:

- DHCP,

- IEEE802.1AB Link Layer Discovery Protocol (LLDP), and

- IEEE 802.1X whereby the URL to the MUD file would be contained in the certificate used in an EAP method.

The MUD manager then uses the the URL to fetch the MUD file, which contains access and network functionality required a device to properly function.

The MUD manager must trust the service from which the URL is fetched and to return an authentic copy of the MUD file. This concern may be mitigated using the optional signature reference in the MUD file. The MUD manager must also trust the device to report a correct URL. In case of DHCP and LLDP the URL is unprotected. When the URL to the MUD file is included in a certificate then it is authenticated and integrity protected. A certificate created for use with network access authentication is typically not signed by the entity that wrote the software and configured the device, which leads to conflation of local network access rights with rights to assert all network access requirements.

There is a need to bind the entity that creates the software and configuration to the MUD file because only that entity can attest the network access requirements of the device. This specification defines an extension to the SUIT manifest to include a MUD file (per reference or by value). When combining a manufacturer usage description with a manifest used for software/firmware updates (potentially augmented with attestation) then a network operator can get more confidence in the description of the access and network functionality required a device to properly function.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Architecture

The intended workflow is as follows:

- At the time of onboarding, devices report their manifest in use to the MUD manager.

- If the SUIT_MUD_container has been severed, the suit-reference-uri can be used to retrieve the complete manifest.

- The manifest authenticity is verified by the MUD manager, which enforces that the MUD file presented is also authentic and as intended by the device software vendor.

- Each time a device is updated, rebooted, or otherwise substantially changed, it will execute an attestation.

  - Among other claims in the Entity Attestation Token (EAT) [I-D.ietf-rats-eat], the device will report its software digest(s), configuration digest(s), primary manifest URI, and primary manifest digest to the MUD manager.

  - The MUD manager can then validate these attestation reports in order to check that the device is operating with the expected version of software and configuration.

  - Since the manifest digest is reported, the MUD manager can look up the corresponding manifest.

- If the MUD manager does not already have a full copy of the manifest, it can be acquired using the reference URI.

- Once a full copy of the manifest is provided, the MUD manager can verify the device attestation report and apply any appropriate policy as described by the MUD file.

4. Extensions to SUIT

To enable strong assertions about the network access requirements that a device should have for a particular software/configuration pair, we include the ability to add MUD files to the SUIT manifest. However, there are also circumstances in which a device should allow the MUD to be changed without a firmware update. To enable this, we add a MUD url to SUIT along with a subject-key identifier, according to [RFC7093], mechanism 4 (the keyIdentifier is composed of the hash of the DER encoding of the SubjectPublicKeyInfo value).

The following CDDL describes the extension to the SUIT_Manifest structure:

```
? suit-manifest-mud => SUIT_Digest
```

The SUIT_Envelope is also amended:

```
? suit-manifest-mud => bstr .cbor SUIT_MUD_container

SUIT_MUD_container = {
    ? suit-mud-url => #6.32(tstr),
    ? suit-mud-ski => SUIT_Digest,
    ? suit-mud-file => bstr
}
```

The MUD file is included verbatim within the bstr. No limits are placed on the MUD file: it may be any RFC8520-compliant file.

5. Security Considerations

This specification links MUD files to other IETF technologies, particularly to SUIT manifests, for improving security protection and ease of use. By including MUD files (per reference or by value) in SUIT manifests an extra layer of protection has been created and synchronization risks can be minimized. If the MUD file and the software/firmware loaded onto the device gets out-of-sync a device may be firewalled and, with firewalling by networks in place, the device may stop functioning.

6. IANA Considerations

suit-manifest-mud must be added as an extension point to the SUIT manifest registry.

7. Normative References

[I-D.ietf-rats-eat]
Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-03 (work in progress), February 2020.

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., and H. Birkholz, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", draft-ietf-suit-manifest-03 (work in progress), February 2020.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7093] Turner, S., Kent, S., and J. Manger, "Additional Methods for Generating Key Identifiers Values", RFC 7093, DOI 10.17487/RFC7093, December 2013, <https://www.rfc-editor.org/info/rfc7093>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <https://www.rfc-editor.org/info/rfc8520>.

Authors' Addresses

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@arm.com

SUIT
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2021

B. Moran
H. Tschofenig
Arm Limited

November 02, 2020

Strong Assertions of IoT Network Access Requirements
draft-moran-suit-mud-01

Abstract

The Manufacturer Usage Description (MUD) specification describes the access and network functionality required a device to properly function. The MUD description has to reflect the software running on the device and its configuration. Because of this, the most appropriate entity for describing device network access requirements is the same as the entity developing the software and its configuration.

A network presented with a MUD file by a device allows detection of misbehavior by the device software and configuration of access control.

This document defines a way to link a SUIT manifest to a MUD file offering a stronger binding between the two.

Table of Contents

1. Introduction

Under [RFC8520], devices report a URL to a MUD manager in the network. RFC 8520 envisions different approaches for conveying the information from the device to the network such as:

- DHCP,

- IEEE802.1AB Link Layer Discovery Protocol (LLDP), and

- IEEE 802.1X whereby the URL to the MUD file would be contained in the certificate used in an EAP method.

The MUD manager then uses the the URL to fetch the MUD file, which contains access and network functionality required a device to properly function.

The MUD manager must trust the service from which the URL is fetched and to return an authentic copy of the MUD file. This concern may be mitigated using the optional signature reference in the MUD file. The MUD manager must also trust the device to report a correct URL. In case of DHCP and LLDP the URL is unprotected. When the URL to the MUD file is included in a certificate then it is authenticated and integrity protected. A certificate created for use with network access authentication is typically not signed by the entity that wrote the software and configured the device, which leads to conflation of local network access rights with rights to assert all network access requirements.

There is a need to bind the entity that creates the software and configuration to the MUD file because only that entity can attest the network access requirements of the device. This specification defines an extension to the SUIT manifest to include a MUD file (per reference or by value). When combining a manufacturer usage description with a manifest used for software/firmware updates (potentially augmented with attestation) then a network operator can get more confidence in the description of the access and network functionality required a device to properly function.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Architecture

The intended workflow is as follows:

- At the time of onboarding, devices report their manifest in use to the MUD manager.

- If the SUIT_MUD_container has been severed, the suit-reference-uri can be used to retrieve the complete manifest.

- The manifest authenticity is verified by the MUD manager, which enforces that the MUD file presented is also authentic and as intended by the device software vendor.

- Each time a device is updated, rebooted, or otherwise substantially changed, it will execute an attestation.

  - Among other claims in the Entity Attestation Token (EAT) [I-D.ietf-rats-eat], the device will report its software digest(s), configuration digest(s), primary manifest URI, and primary manifest digest to the MUD manager.

  - The MUD manager can then validate these attestation reports in order to check that the device is operating with the expected version of software and configuration.

  - Since the manifest digest is reported, the MUD manager can look up the corresponding manifest.

- If the MUD manager does not already have a full copy of the manifest, it can be acquired using the reference URI.

- Once a full copy of the manifest is provided, the MUD manager can verify the device attestation report and apply any appropriate policy as described by the MUD file.

4. Extensions to SUIT

To enable strong assertions about the network access requirements that a device should have for a particular software/configuration pair, we include the ability to add MUD files to the SUIT manifest. However, there are also circumstances in which a device should allow the MUD to be changed without a firmware update. To enable this, we add a MUD url to SUIT along with a subject-key identifier, according to [RFC7093], mechanism 4 (the keyIdentifier is composed of the hash of the DER encoding of the SubjectPublicKeyInfo value).

The following CDDL describes the extension to the SUIT_Manifest structure:

```
? suit-manifest-mud => SUIT_Digest
```

The SUIT_Envelope is also amended:

```
? suit-manifest-mud => bstr .cbor SUIT_MUD_container

SUIT_MUD_container = {
    ? suit-mud-url => #6.32(tstr),
    ? suit-mud-ski => SUIT_Digest,
    ? suit-mud-file => bstr
}
```

The MUD file is included verbatim within the bstr. No limits are placed on the MUD file: it may be any RFC8520-compliant file.

5. Security Considerations

This specification links MUD files to other IETF technologies, particularly to SUIT manifests, for improving security protection and ease of use. By including MUD files (per reference or by value) in SUIT manifests an extra layer of protection has been created and synchronization risks can be minimized. If the MUD file and the software/firmware loaded onto the device gets out-of-sync a device may be firewalled and, with firewalling by networks in place, the device may stop functioning.

6. IANA Considerations

suit-manifest-mud must be added as an extension point to the SUIT manifest registry.

7. Normative References

[I-D.ietf-rats-eat]
Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-04 (work in progress), August 2020.

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., Birkholz, H., and K. Zandberg, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", draft-ietf-suit-manifest-09 (work in progress), July 2020.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7093] Turner, S., Kent, S., and J. Manger, "Additional Methods for Generating Key Identifiers Values", RFC 7093, DOI 10.17487/RFC7093, December 2013, <https://www.rfc-editor.org/info/rfc7093>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <https://www.rfc-editor.org/info/rfc8520>.

Authors' Addresses

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@arm.com

anima Working Group
Internet-Draft
Intended status: Standards Track
Expires: 28 January 2021

M. Richardson
Sandelman Software Works
J. Yang
Huawei Technologies Co., Ltd.
27 July 2020

Security and Operational considerations for manufacturer installed keys and anchors
draft-richardson-secdispatch-idevid-considerations-02

Abstract

This document provides a nomenclature to describe ways in which manufacturers secure private keys and public trust anchors in devices.

RFCEDITOR: please remove this paragraph. This work is occuring in https://github.com/mcr/idevid-security-considerations

Table of Contents

1. Introduction

An increasing number of protocols derive a significant part of their security by using trust anchors that are installed by manufacturers. Disclosure of the list of trust anchors does not usually cause a problem, but changing them in any way does. This includes adding, replacing or deleting anchors.

Many protocols also leverage manufacturer installed identities. These identities are usually in the form of [ieee802-1AR] Initial Device Identity certificates (IDevID). The identity has two components: a private key that must remain under the strict control of a trusted part of the device, and a public part (the certificate), which (ignoring, for the moment, personal privacy concerns) may be freely disclosed.

There also situations where identities which are tied up in the provision of symmetric shared secret. A common example is the SIM card ([_3GPP.51.011]), which now comes as a virtual SIM, but which is usually not provisioned at the factory. The provision of an initial, per-device default password also falls into the category of symmetric shared secret.

It is further not unusual for many devices (particularly smartphones) to also have one or more group identity keys. This is used in, for instance, in [fidotechnote] to make claims about being a particular model of phone (see [I-D.richardson-rats-usecases]). The keypair that does this is loaded into large batches of phones for privacy reasons.

The trust anchors are used for a variety of purposes. The following uses are specifically called out:

* to validate the signature on a software update (as per [I-D.ietf-suit-architecture]).

* to verify the end of a TLS Server Certificate, such as when setting up an HTTPS connection.

* to verify the [RFC8366] format voucher that provides proof of an ownership change

Device identity keys are used when performing enrollment requests (in [I-D.ietf-anima-bootstrapping-keyinfra], and in some uses of [I-D.ietf-emu-eap-noob]. The device identity certificate is also used to sign Evidence by an Attesting Environment (see [I-D.ietf-rats-architecture]).

These security artifacts are used to anchor other chains of information: an EAT Claim as to the version of software/firmware running on a device (XXX and [I-D.birkholz-suit-coswid-manifest]), an EAT claim about legitimate network activity (via [I-D.birkholz-rats-mud], or embedded in the IDevID in [RFC8520]). Known software versions lead directly to vendor/distributor signed Software Bill of Materials (SBOM), such as those described by [I-D.ietf-sacm-coswid] and the NTIA/SBOM work [ntiasbom] and CISQ/OMG SBOM work underway [cisqsbom].

In order to manage risks and assess vulnerabilities in a Supply Chain, it is necessary to determine a degree of trusthworthiness in each device. A device may mislead audit systems as to its provenance, about its software load or even about what kind of device it is. (see [RFC7168] for a humourous example). In order to properly assess the security of a Supply Chain it is necessary to understand the kinds and severity of the threats which a device has been designed to resist. To do this, it is necessary to understand the ways in which the different trust anchors and identities are initially provisioned, are protected, and are updated.

To do this, this document details the different trust anchors (TrA) and identities (IDs) found in typical devices. The privacy and integrity of the TAs and IDs is often provided by a different, superior artifacts. This relationship is examined.

While many might desire to assign numerical values to different mitigation techniques in order to be able to rank them, this document does not attempt to do, as there are too many other (mostly human) factors that would come into play. Such an effort is more properly in the purvue of a formal ISO9001 process such as ISO14001.

1.1. Terminology

This document is not a standards track document, and it does not make use of formal requirements language.

This section will be expanded to include needed terminology as required.

The words Trust Anchor are contracted to TrA rather than TA, in order not to confuse with [I-D.ietf-teep-architecture]'s "Trusted Application".

This document defines a number of hyphenated terms, and they are summarized here:

device-generated : a private or symmetric key which is generated on

the device

infrastructure-generated : a private or symmetric key which is generated by some system, likely located at factory that built the device

mechanically-installed : when a key or certificate is programmed into flash by out-of-band mechanism like JTAG

mechanically-transfered : when a key or certificate is transfered into a system via private interface, such as serial console, JTAG managed mailbox, or other physically private interface

network-transfered : when a key or certificate is transfered into a system using a network interface which would be available after the device has shipped. This applies even if the network is physically attached using a bed-of-nails.

device/infrastructure-co-generated : when a private or symmetric key is derived from a secret previously synchronized between the silicon vendor and the factory using a common algorithm.

2. Applicability Model

There is a wide variety of devices to which this analysis can apply. (See [I-D.bormann-lwig-7228bis]) This document will use a J-group class C13 as a sample. This class is sufficiently large to experience complex issues among multiple CPUs, packages and operating systems, but at the same time, small enough that this class is often deployed in single-purpose IoT-like uses. Devices in this class often have Secure Enclaves (such as the "Grapeboard"), and can include silicon manufacturer controlled processors in the boot process (the Raspberry PI boots under control of the GPU).

Almost all larger systems (servers, laptops, desktops) include a Baseboard Management Controller (BMC), which ranges from a M-Group Class 3 MCU, to a J-Group Class 10 CPU (see, for instance [openbmc] which uses a Linux kernel and system inside the BMC). As the BMC usually has complete access to the main CPU's memory, I/O hardware and disk, the boot path security of such a system needs to be understood first as being about the security of the BMC.

2.1. A reference manufacturing/boot process

In order to provide for immutability and privacy of the critical TANs and IDs, many CPU manufacturers will provide for some kind of private memory area which is only accessible when the CPU is in certain priviledged states. See the Terminology section of [I-D.ietf-teep-architecture], notably TEE, REE, and TAM, and also section 4, Architecture.

The private memory that is important is usually non-volatile and rather small. It may be located inside the CPU silicon die, or it may be located externally. If the memory is external, then it is usually encrypted by a hardware mechanism on the CPU, with only the key kept inside the CPU.

The entire mechanism may be external to the CPU in the form of a hardware-TPM module, or it may be entirely internal to the CPU in the form of a firmware-TPM. It may use a custom interface to the rest of the system, or it may implement the TPM 1.2 or TPM 2.0 specifications. Those details are important to performing a full evaluation, but do not matter that to this model (see initial-enclave-location below).

During the manufacturing process, once the components have been soldered to the board, the system is usually put through a system-level test. This is often done on as a "bed-of-nails" test [BedOfNails], where the board has key points attached mechanically to a test system. A [JTAG] process tests the System Under Test, and then initializes some firmware into the still empty flash storage. It is now common for a factory test image to be loaded first: this image will include code to initialize the private memory key described above, and will include a first-stage bootloader and some kind of (primitive) Trusted Application Manager (TAM). Embedded in the stage one bootloader will be a Trust Anchor that is able to verify the second-stage bootloader image.

After the system has undergone testing, the factory test image is erased, leaving the first-stage bootloader. One or more second-stage bootloader images is installed. The production image may be installed at that time, or if the second-stage bootloader is able to install it over the network, it may be done that way instead.

There are many variations of the above process, and this section is not attempting to be prescriptive, but to be provide enough illustration to motivate subsequent terminology.

There process may be entirely automated, or it may be entirely driven by humans working in the factory.

Or a combination of the above.

These steps may all occur on an access-controlled assembly line, or the system boards may be shipped from one place to another (maybe another country) before undergoing testing.

Some systems are intended to be shipped in a tamper-proof state, but it is usually not desireable that bed-of-nails testing be possible without tampering, so the initialization process is usually done prior to rendering the system tamper-proof.

Quality control testing may be done prior to as well as after the application of tamper-proofing, as systems which do not pass inspection may be reworked to fix flaws, and this should ideally be impossible once the system has been made tamper-proof.

3. Types of Trust Anchors

Trust Anchors are fundamentally public keys. They are used to validate other digitally signed artifacts. Typically, these are chains of PKIX certificates leading to an End-Entity certificate (EE).

The chains are usually presented as part of an externally provided object, with the term "externally" to be understood as being as close as untrusted flash, to as far as objects retrieved over a network.

There is no requirement that there be any chain at all: the trust anchor can be used to validate a signature over a target object directly.

The trust anchors are often stored in the form of self-signed certificates. The self-signature does not offer any cryptographic assurange, but it does provide a form of error detection, providing verification against non-malicious forms of data corruption. If storage is at a premium (such as inside-CPU non-volatile storage) then only the public key itself need to be stored. For a 256-bit ECDSA key, this is 32-bytes of space.

When evaluating the degree of trust for each trust anchor there are four aspects that need to be determined:

* can the trust anchor be replaced or modified?

* can additional trust anchors be added?

* can trust anchors be removed?

* how is the private key associated with the trust anchor stored?

The first three things are device specific properties of how the integrity of the trust anchor is maintained.

The fourth property has nothing to do with the device, but has to do with the reputation and care of the entity that maintains the private key.

Different anchors have different purposes.

These are:

3.1. Secured First Boot Trust Anchor

This anchor is part of the first-stage boot loader, and it is used to validate a second-stage bootloader which may be stored in external flash.

3.2. Software Update Trust Anchor

This anchor is used to validate the main application (or operating system) load for the device.

It can be stored in a number of places. First, it may be identical to the Secure Boot Trust Anchor.

Second, it may be stored in the second-stage bootloader, and therefore it's integrity is protected by the Secured First Boot Trust Anchor.

Third, it may be stored in the application code itself, where the application validates updates to the application directly (update in place), or via a double-buffer arrangement. The initial (factory) load of the application code initializes the trust arrangement.

In this situation the application code is not in a secured boot situation, as the second-stage bootloader does not validate the application/operating system before starting it, but it may still provide measured boot mechanism.

3.3. Trusted Application Manager anchor

This anchor is part of a [I-D.ietf-teep-architecture] Trusted Application Manager. Code which is signed by this anchor will be given execution privileges as described by the manifest which accompanies the code. This privilege may include updating anchors.

3.4. Public WebPKI anchors

These anchors are used to verify HTTPS certificates from web sites. These anchors are typically distributed as part of desktop browsers, and via desktop operating systems.

The exact set of these anchors is not precisely defined: it is usually determined by the browser vendor (e.g., Mozilla, Google, Apple, Safari, Microsoft), or the operating system vendor (e.g., Apple, Google, Microsoft, Ubuntu). In most cases these vendors look to the CA/Browser Forum ([CABFORUM]) for inclusion criteria.

3.5. DNSSEC root

This anchor is part of the DNS Security extensions. It provides an anchor for securing DNS lookups. Secure DNS lookups may be important in order to get access to software updates. This anchor is now scheduled to change approximately every 3 years, with the new key announced several years before it is used, making it possible to embed a keys that will be valid for up to five years.

This trust anchor is typically part of the application/operating system code and is usually updated by the manufacturer when they do updates. However, a system which is connected to the Internet may update the DNSSEC anchor itself through the mechanism described in [RFC5011].

There are concerns that there may be a chicken and egg situation for devices have remained in a powered off state (or disconnected from the Internet) for some period of years. That upon being reconnected, that the device would be unable to do DNSSEC validation. This failure would result in them being unable to obtain operating system updates that would then include the updates to the DNSSEC key.

3.6. What else?

TBD?

4. Types of Identities

Identities are installed during manufacturing time for a variety of purposes.

Identities require some private component. Asymmetric identities (e.g., RSA, ECDSA, EdDSA systems) require a co-responding public component, usually in the form of a certificate signed by a trusted third party.

The process of making this coordinated key pair and then installing it into the device is called identity provisioning.

4.1. Manufacturer installed IDevID certificates

[ieee802-1AR] defines a category of certificates that are to installed by the manufacturer, which contain at the least, a device unique serial number.

A number of protocols depend upon this certificate.

* [I-D.ietf-anima-bootstrapping-keyinfra] introduces a mechanism for new devices (called pledges) to be onboarded into a network without intervention from an expert operator. A number of derived protocols such as {{I-D.

* [I-D.ietf-rats-architecture] depends upon a key provisioned into the Attesting Environment to sign Evidence.

* [I-D.ietf-suit-architecture] may depend upon a key provisioned into the device in order to decrypt software updates.

* TBD

4.1.1. Operational Considerations for Manufacturer IDevID Public Key Infrastructure

The manufacturer has the responsability to provision a keypair into each device as part of the manufacturing process. There are a variety of mechanisms to accomplish this, which this document will overview.

There are three fundamental ways to generate IDevID certificates for devices:

1. generating a private key on the device, creating a Certificate Signing Request (or equivalent), and then returning a certificate to the device.

2. generating a private key outside the device, signing the certificate, and the installing both into the device.

3. deriving the private key from a previously installed secret seed, that is shared with only the manufacturer

There is a fourth situation where the IDevID is provided as part of a Trusted Platform Module (TPM), in which case the TPM vendor may be making the same tradeoffs.

The document [I-D.moskowitz-ecdsa-pki] provides some practical instructions on setting up a reference implementation for ECDSA keys using a three-tier mechanism.

This document recommends the use of ECDSA keys for the root and intermediate CAs, but there may be operational reasons why an RSA intermediate CA will be required for some legacy TPM equipment.

4.1.2. Key Generation process

4.1.2.1. On-device private key generation

Generating the key on-device has the advantage that the private key never leaves the device. The disadvantage is that the device may not have a verified random number generator. [factoringrsa] is an example of this scenario!

There are a number of options of how to get the public key securely from the device to the certification authority.

This transmission must be done in an integral manner, and must be securely associated with the assigned serial number. The serial number goes into the certificate, and the resulting certificate needs to be loaded into the manufacturer's asset database. This asset database needs to be shared with the MASA.

One way to do the transmission is during a factory Bed of Nails test (see [BedOfNails]) or Boundary Scan. When done via a physical connection like this, then this is referred to as a _device-generated_ / _mechanically-transfered_ .

There are other ways that could be used where a certificate signing request is sent over a special network channel when the device is powered up in the factory. This is referered to as the _device-generated_ / _network-transfered_ method.

Regardless of how the certificate signing request is sent from the device to the factory, and the certificate is returned to the device, a concern from production line managers is that the assembly line may have to wait for the certification authority to respond with the certificate.

After the key generation, the device needs to set a flag such that it no longer generates a new key, or will accept a new IDevID via the factory connection. This may be a software setting, or could be as dramatic as blowing a fuse.

The risk is that if an attacker with physical access is able to put the device back into an unconfigured mode, then the attacker may be able to substitute a new certificate into the device. It is difficult to construct a rational for doing this, unless the network initialization also permits an attacker to load or replace trust anchors at the same time.

Because the key is generated inside the device, it is assumed that the device can never be convinced to disclose the private key.

4.1.2.2. Off-device private key generation

Generating the key off-device has the advantage that the randomness of the private key can be better analyzed. As the private key is available to the manufacturing infrastructure, the authenticity of the public key is well known ahead of time.

If the device does not come with a serial number in silicon, then one should be be assigned and placed into a certificate. The private key and certificate could be programmed into the device along with the initial bootloader firmware in a single step.

Aside from the the change of origin for the randomness, a major advantage of this mechanism is that it can be done with a single write to the flash. The entire firmware of the device, including configuration of trust anchors and private keys can be loaded in a single write pass. Given some pipelining of the generation of the keys and the creation of certificates, it may be possible to install unique identities without taking any additional time.

The major downside to generating the private key off-device is that it could be seen by the manufacturing infrastructure. It could be compromised by humans in the factory, or the equipment could be compromised. The use of this method increases the value of attacking the manufacturing infrastructure.

If keys are generated by the manufacturing plant, and are immediately installed, but never stored, then the window in which an attacker can gain access to the private key is immensely reduced.

As in the previous case, the transfer may be done via physical interfaces such as bed-of-nails, giving the _infrastructure-generated_ / _mechanically-transfered_ method.

There is also the possibility of having a _infrastructure-generated_ / _network-transfered_ key. There is a support for "server-generated" keys in [RFC7030], [I-D.gutmann-scep], and [RFC4210]. All methods strongly recommend encrypting the private key for transfer.

This is difficult to comply with as there is not yet any private key material in the device, so in many cases it will not be possible to encrypt the private key.

4.1.2.3. Key setup based on 256-bit secret seed

A hybrid of the previous two methods leverages a symmetric key that is often provided by a silicon vendor to OEM manufacturers.

Each CPU (or a Trusted Execution Environment [I-D.ietf-teep-architecture], or a TPM) is provisioned at fabrication time with a unique, secret seed, usually at least 256-bits in size.

This value is revealed to the OEM board manufacturer only via a secure channel. Upon first boot, the system (probably within a TEE, or within a TPM) will generate a key pair using the seed to initialize a Pseudo-Random-Number-Generator (PRNG). The OEM, in a separate system, will initialize the same PRNG and generate the same key pair. The OEM then derives the public key part, signs it and turns it into a certificate. The private part is then destroyed, ideally never stored or seen by anyone. The certificate (being public information) is placed into a database, in some cases it is loaded by the device as its IDevID certificate, in other cases, it is retrieved during the onboarding process based upon a unique serial number asserted by the device.

This method appears to have all of the downsides of the previous two methods: the device must correctly derive its own private key, and the OEM has access to the private key, making it also vulnerable. The secret seed must be created in a secure way and it must also be communicated securely.

There are some advantages to the OEM however: the major one is that the problem of securely communicating with the device is outsourced to the silicon vendor. The private keys and certificates may be calculated by the OEM asynchronously to the manufacturing process, either done in batches in advance of actual manufacturing, or on demand when an IDevID is demanded. Doing the processing in this way permits the key derivation system to be completely disconnected from any network, and requires placing very little trust in the system assembly factory. Operational security such as often incorrectly presented fictionalized stories of a "mainframe" system to which only physical access is permitted begins to become realistic. That trust has been replaced with a heightened trust placed in the silicon (integrated circuit) fabrication facility.

The downsides of this method to the OEM are: they must be supplied by a trusted silicon fabrication system, which must communicate the set of secrets seeds to the OEM in batches, and they OEM must store and care for these keys very carefully. There are some operational advantages to keeping the secret seeds around in some form, as the same secret seed could be used for other things. There are some significant downsides to keeping that secret seed around.

5. Public Key Infrastructures (PKI)

[RFC5280] describes the format for certificates, and numerous mechanisms for doing enrollment have been defined (including: EST: [RFC7030], CMP: [RFC4210], SCEP [I-D.gutmann-scep]).

[RFC5280] provides mechanisms to deal with multi-level certification authorities, but it is not always clear what operating rules apply.

PKIs can suffer two kinds of failures: 1. disclosure of a private key. 2. loss of a private key.

A PKI which discloses one or more private certification authority keys is no longer secure. An attacker can create new identities, and forge certificates connecting existing identities to attacker controlled public/private keypairs. This can permit the attacker to impersonate the specific device.

If the PKI uses Certificate Revocation Lists (CRL)s, then an attacker can also revoke existing identities.

In the other direction, a PKI which loses access to a private key can no longer function. Existing identities may continue to function, unless CRLs or OCSP is in use, in which case, the inability to sign a fresh CRL or OCSP response will result in all identities becoming invalid.

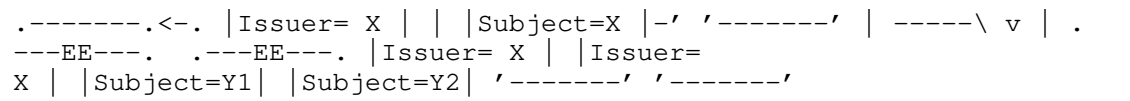
This section details some nomenclature about the structure of certification authorities.

5.1. Number of levels of certification authorities

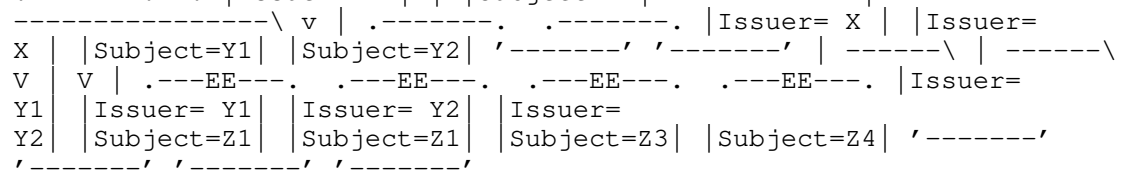
The certification authority (CA) starts with a Trust Anchor (TA). This is counted as the zeroth level of the authority.

In the degenerate case of a self-signed certificate, then this a level zero PKI.

The Trust Anchor signs one or more certificates. When this first level authority trusts only End-Entity (EE) certificates, then this is a level one PKI.



When this first level authority signs intermediate certification authorities, and those certification authorities sign End-Entity certificates, then this is a level two PKI.



In general, when arranged as a tree, with the End-Entity certificates at the bottom, and the Trust Anchor at the top, then the level is where the deepest EE certificates are, counting from zero.

It is quite common to have a two-level PKI, where the root of the CA is stored in a Hardware Security Module, while the level one intermediate CA is available online.

5.2. Protection of CA private keys

The private key for the certification authorities must be protected from disclosure. The strongest protection is afforded by keeping them in a offline device, passing Certificate Signing Requests (CSR)s to the offline device by human process.

For examples of extreme measures, see [kskceremony]. There is however a wide spectrum of needs, as exampled in [rootkeyceremony]. The SAS70 audit standard is usually used as a basis for the Ceremony, see [keyceremony2].

This is inconvenient, and may involve latencies of days, possibly even weeks to months if the offline device is kept in a locked environment that requires multiple keys to be present.

There is therefore a tension between protection and convenience. This is often accomplished by having some levels of the PKI be offline, and some levels of the PKI be online.

There is usually a need to maintain backup copies of the critical keys. It is often appropriate to use secret splitting technology such as Shamir Secret Sharing among a number of parties [shamir79] This mechanism can be setup such that some threshold k (less than the total n) of shares are needed in order to recover the secret.

5.3. Supporting provisioned anchors in devices

IDevID-type Identity (or Birth) Certificates which are provisioned into devices need to be signed by a certification authority maintained by the manufacturer. The manufacturer needs to maintain availability of this PKI.

Trust anchors which are provisioned in the devices will have corresponding private keys maintained by the manufacturer. The trust anchors will often anchor a PKI which is going to be used for a particular purpose. There will be End-Entity (EE) certificates of this PKI which will be used to sign particular artifacts (such as software updates), or communications protocols (such as TLS connections). The private key associated with these EE certificates are not stored in the device, but are maintained by the manufacturer. These need even more care than the private keys stored in the devices, as compromise of the software update key compromises all of the devices, not just a single device.

6. Evaluation Questions

This section recaps the set of questions that may need to be answered. This document does not assign valuation to the answers.

6.1. Integrity and Privacy of on-device data

initial-enclave-location : Is the location of the initial software trust anchor internal to the CPU package?

initial-enclave-integrity-key : If the first-stage bootloader is external to the CPU, and it is integrity protected, where is the key used to check the integrity?

initial-enclave-privacy-key : If the first-stage data is external to the CPU, is it encrypted?

first-stage-initialization : The number of people involved in the

first stage initialization. An entirely automated system would have a number zero. A factory with three 8 hour shifts might have a number that is a multiple of three. A system with humans involved may be subject to bribery attacks, while a system with no humans may be subject to attacks on the system which are hard to notice.

first-second-stage-gap : If a board is initialized with a first-stage bootloader in one location (factory), and then shipped to another location, there may situations where the device can not be locked down until the second step.

6.2. Integrity and Privacy of device identify infrastructure

For IDevID provisioning, which includes a private key and matching certificate installed into the device, the associated public key infrastructure that anchors this identity must be maintained by the manufacturer.

identity-pki-level : how deep are the IDevID certificates that are issued?

identity-time-limits-per-intermediate : how long is each intermediate CA maintained before before a new intermediate CA key is generated? There may be no time limit, only a device count limit.

identity-number-per-intermediate : how many identities are signed by a particular intermediate CA before it is retired? There may be no numeric limit, only a time limit.

identity-anchor-storage : how is the root CA key stored? How many people are needed to recover it?

6.3. Integrity and Privacy of included trust anchors

For each trust anchor (public key) stored in the device, there will be an associated PKI. For each of those PKI the following questions need to be answered.

pki-level : how deep is the EE that will be evaluated

pki-level-locked : (a boolean) is the level where the EE cert will be found locked by the device, or can levels be added or deleted by the PKI operator without code changes to the device.

pki-breadth : how many different EE certificates exist in this PKI

pki-lock-policy : can any EE certificate be used with this trust anchor to sign? Or, is there some kind of policy OID or Subject restriction? Are specific intermediate CAs needed that lead to the EE?

pki-anchor-storage: how is the root CA stored? How many people are needed to recover it?

7. Privacy Considerations

many yet to be detailed

8. Security Considerations

This entire document is a security considerations.

9. IANA Considerations

This document makes no IANA requests.

10. Acknowledgements

Robert Martin of MITRE provides some guidance about citing the SBOM efforts.

11. Changelog

12. References

12.1. Normative References

[BCP14] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[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>.

[I-D.moskowitz-ecdsa-pki] Moskowitz, R., Birkholz, H., Xia, L., and M. Richardson, "Guide for building an ECC pki", Work in Progress, Internet-Draft, draft-moskowitz-ecdsa-pki-08, 14 February 2020, <http://www.ietf.org/internet-drafts/draft-moskowitz-ecdsa-pki-08.txt>.

[ieee802-1AR]
IEEE Standard, ., "IEEE 802.1AR Secure Device Identifier", 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

12.2. Informative References

[I-D.richardson-anima-masa-considerations]
Richardson, M. and W. Pan, "Operatonal Considerations for Voucher infrastructure for BRSKI MASA", Work in Progress, Internet-Draft, draft-richardson-anima-masa-considerations-04, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-richardson-anima-masa-considerations-04.txt>.

[I-D.ietf-anima-bootstrapping-keyinfra]
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-41, 8 April 2020, <http://www.ietf.org/internet-drafts/draft-ietf-anima-bootstrapping-keyinfra-41.txt>.

[RFC5011] StJohns, M., "Automated Updates of DNS Security (DNSSEC) Trust Anchors", STD 74, RFC 5011, DOI 10.17487/RFC5011, September 2007, <https://www.rfc-editor.org/info/rfc5011>.

[RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <https://www.rfc-editor.org/info/rfc8366>.

[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>.

[I-D.gutmann-scep]
Gutmann, P., "Simple Certificate Enrolment Protocol", Work in Progress, Internet-Draft, draft-gutmann-scep-16, 27 March 2020, <http://www.ietf.org/internet-drafts/draft-gutmann-scep-16.txt>.

[RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <https://www.rfc-editor.org/info/rfc4210>.

[_3GPP.51.011]
3GPP, "Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface", 3GPP TS 51.011 4.15.0, 15 June 2005, <http://www.3gpp.org/ftp/Specs/html-info/51011.htm>.

[BedOfNails]
Wikipedia, ., "Bed of nails tester", July 2020, <https://en.wikipedia.org/wiki/In-circuit_test#Bed_of_nails_tester>.

[pelionfcu]
ARM Pelion, "Factory provisioning overview", 28 June 2020, <https://www.pelion.com/docs/device-management-provision/1.2/introduction/index.html>.

[factoringrsa]
"Factoring RSA keys from certified smart cards: Coppersmith in the wild", 16 September 2013, <https://core.ac.uk/download/pdf/204886987.pdf>.

[RambusCryptoManager]
Qualcomm press release, "Qualcomm Licenses Rambus CryptoManager Key and Feature Management Security Solution", 2014, <https://www.rambus.com/qualcomm-licenses-rambus-cryptomanager-key-and-feature-management-security-solution/>.

[kskceremony]
Verisign, "DNSSEC Practice Statement for the Root Zone ZSK Operator", 2017, <https://www.iana.org/dnssec/dps/zsk-operator/dps-zsk-operator-v2.0.pdf>.

[rootkeyceremony]
Community, "Root Key Ceremony, Cryptography Wiki", April 2020, <https://cryptography.fandom.com/wiki/Root_Key_Ceremony>.

[keyceremony2]
Digi-Sign, "SAS 70 Key Ceremony", April 2020, <http://www.digi-sign.com/compliance/key%20ceremony/index>.

[shamir79] Shamir, A., "How to share a secret.", 1979, <https://www.cs.jhu.edu/~sdoshi/crypto/papers/shamirturing.pdf>.

[nistsp800-57]
NIST, "SP 800-57 Part 1 Rev. 4 Recommendation for Key Management, Part 1: General", 1 January 2016, <https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-4/final>.

[fidotechnote]
FIDO Alliance, ., "FIDO TechNotes: The Truth about Attestation", July 2018, <https://fidoalliance.org/fido-technotes-the-truth-about-attestation/>.

[ntiasbom] NTIA, ., "NTIA Software Compoment Transparency", n.d., <https://www.ntia.doc.gov/SoftwareTransparency>.

[cisqsbom] CISQ/Object Management Group, ., "TOOL-TO-TOOL SOFTWARE BILL OF MATERIALS EXCHANGE", July 2020, <https://www.it-cisq.org/software-bill-of-materials/index.htm>.

[openbmc] Linux Foundation/OpenBMC Group, ., "Defining a Standard Baseboard Management Controller Firmware Stack", July 2020, <https://www.openbmc.org/>.

[JTAG] IEEE Standard, ., "1149.7-2009 - IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture", 2009, <https://ieeexplore.ieee.org/document/5412866>.

[rootkeyrollover]
ICANN, ., "Proposal for Future Root Zone KSK Rollovers", 2019, <https://www.icann.org/en/system/files/files/proposal-future-rz-ksk-rollovers-01nov19-en.pdf>.

[CABFORUM] CA/Browser Forum, ., "CA/Browser Forum Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates, v.1.2.2", October 2014, <https://cabforum.org/wp-content/uploads/BRv1.2.2.pdf>.

[I-D.richardson-rats-usecases]
Richardson, M., Wallace, C., and W. Pan, "Use cases for Remote Attestation common encodings", Work in Progress, Internet-Draft, draft-richardson-rats-usecases-07, 9 March 2020, <http://www.ietf.org/internet-drafts/draft-richardson-rats-usecases-07.txt>.

[I-D.ietf-suit-architecture]
Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", Work in Progress, Internet-Draft, draft-ietf-suit-architecture-11, 27 May 2020, <http://www.ietf.org/internet-drafts/draft-ietf-suit-architecture-11.txt>.

[I-D.ietf-emu-eap-noob]
Aura, T. and M. Sethi, "Nimble out-of-band authentication for EAP (EAP-NOOB)", Work in Progress, Internet-Draft, draft-ietf-emu-eap-noob-02, 12 July 2020, <http://www.ietf.org/internet-drafts/draft-ietf-emu-eap-noob-02.txt>.

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-05, 10 July 2020, <http://www.ietf.org/internet-drafts/draft-ietf-rats-architecture-05.txt>.

[I-D.birkholz-suit-coswid-manifest]
Birkholz, H., "A SUIT Manifest Extension for Concise Software Identifiers", Work in Progress, Internet-Draft, draft-birkholz-suit-coswid-manifest-00, 17 July 2018, <http://www.ietf.org/internet-drafts/draft-birkholz-suit-coswid-manifest-00.txt>.

[I-D.birkholz-rats-mud]
Birkholz, H., "MUD-Based RATS Resources Discovery", Work in Progress, Internet-Draft, draft-birkholz-rats-mud-00, 9 March 2020, <http://www.ietf.org/internet-drafts/draft-birkholz-rats-mud-00.txt>.

[RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <https://www.rfc-editor.org/info/rfc8520>.

[I-D.ietf-sacm-coswid]
Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", Work in Progress, Internet-Draft, draft-ietf-sacm-coswid-15, 1 May 2020, <http://www.ietf.org/internet-drafts/draft-ietf-sacm-coswid-15.txt>.

[RFC7168] Nazar, I., "The Hyper Text Coffee Pot Control Protocol for Tea Efflux Appliances (HTCPCP-TEA)", RFC 7168, DOI 10.17487/RFC7168, April 2014, <https://www.rfc-editor.org/info/rfc7168>.

[I-D.bormann-lwig-7228bis] Bormann, C., Ersue, M., Keranen, A., and C. Gomez, "Terminology for Constrained-Node Networks", Work in Progress, Internet-Draft, draft-bormann-lwig-7228bis-06, 9 March 2020, <http://www.ietf.org/internet-drafts/draft-bormann-lwig-7228bis-06.txt>.

[I-D.ietf-teep-architecture] Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-12, 13 July 2020, <http://www.ietf.org/internet-drafts/draft-ietf-teep-architecture-12.txt>.

Authors' Addresses

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Jie Yang
Huawei Technologies Co., Ltd.

Email: jay.yang@huawei.com

Independent Stream
Internet-Draft
Intended status: Informational
Expires: 21 April 2021

M. Schanzenbach
GNUnet e.V.
C. Grothoff
Berner Fachhochschule
B. Fix
GNUnet e.V.

18 October 2020

The GNU Name System
draft-schanzen-gns-02

Abstract

This document contains the GNU Name System (GNS) technical specification.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on 21 April 2021.

Copyright Notice

Table of Contents

1. Introduction

The Domain Name System (DNS) is a unique distributed database and a vital service for most Internet applications. While DNS is distributed, it relies on centralized, trusted registrars to provide globally unique names. As the awareness of the central role DNS plays on the Internet rises, various institutions are using their power (including legal means) to engage in attacks on the DNS, thus threatening the global availability and integrity of information on the Internet.

DNS was not designed with security as a goal. This makes it very vulnerable, especially to attackers that have the technical capabilities of an entire nation state at their disposal. This specification describes a censorship-resistant, privacy-preserving and decentralized name system: The GNU Name System (GNS). It is designed to provide a secure alternative to DNS, especially when censorship or manipulation is encountered. GNS can bind names to any kind of cryptographically secured token, enabling it to double in some respects as even as an alternative to some of today's Public Key Infrastructures, in particular X.509 for the Web.

This document contains the GNU Name System (GNS) technical specification of the GNU Name System [GNS], a fully decentralized and censorship-resistant name system. GNS provides a privacy-enhancing alternative to the Domain Name System (DNS). The design of GNS incorporates the capability to integrate and coexist with DNS. GNS is based on the principle of a petname system and builds on ideas from the Simple Distributed Security Infrastructure (SDSI), addressing a central issue with the decentralized mapping of secure identifiers to memorable names: namely the impossibility of providing a global, secure and memorable mapping without a trusted authority. GNS uses the transitivity in the SDSI design to replace the trusted root with secure delegation of authority thus making petnames useful to other users while operating under a very strong adversary model.

This document defines the normative wire format of resource records, resolution processes, cryptographic routines and security considerations for use by implementors. GNS requires a distributed hash table (DHT) for record storage. Specification of the DHT is out of scope of 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 [RFC2119].

2. Zones

A zone in GNS is defined by a zone type "ztype" that identifies a cryptosystem and a public/private key pair "(d,zk)", where "d" is the private key and "zk" the corresponding public key in the public key cipher identified by the "ztype". The contents of a zone are cryptographically signed before being published a distributed hash table (DHT). Records are grouped by their label and encrypted (Section 6.3) using an encryption key derived from the label and the zone public key. Instead of the zone private key "d", the signature MUST be created using a blinded public/private key pair "d'" and "zk'". This blinding is realized using a hierarchical deterministic key derivation (HDKD) scheme. Such a scheme allows the deterministic derivation of keys from the original public and private zone keys using "label" values. Specifically, the zone owner can derive private keys "d'", and a resolver to derive the corresponding public keys "zk'". Using different "label" values in the derivation results in different keys. Without knowledge of the "label" values, the different derivations are unlinkable both to the original key and to each other. This prevents zone enumeration and requires knowledge of both "zk" and the "label" to confirm affiliation with a specific zone. At the same time, the blinded "zk'" provides nodes with the ability to verifiy the integrity of the published information without disclosing the originating zone.

The following variables are associated with a zone in GNS:

ztype is the unique type of the zone type as registered in the GNUnet Assigned Numbers Authority [GANA]. The zone type determines which cryptosystem is used for the asymmetric and symmetric key operations of the zone. A 32-bit number.

d is the private zone key. The specific format depends on the zone type.

zk is the public zone key. The specific format depends on the zone type.

zid is the unique public identifier of a zone. It consists of the "ztype" and the public zone key "zk".

zTLD is a string which encodes the "ztype" as well as the zone key "zk" into a domain name. The "zTLD" is used as a globally unique reference to a specific namespace in the process of name resolution.

The "zid" wire format is defined as follows:

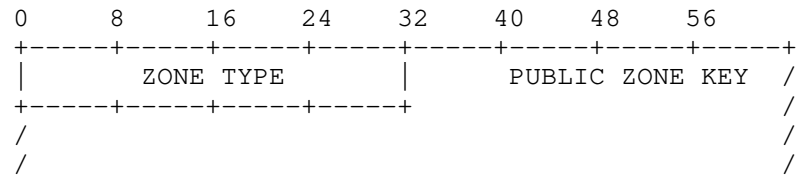


Figure 1

For the string representation of the "zid", we use a base-32 encoding "StringEncode". However, instead of following [RFC4648] we base our character map on the optical character recognition friendly proposal of Crockford [CrockfordB32]. The only difference to Crockford is that the letter "U" decodes to the same base-32 value as the letter "V" (27).

zkl := <StringEncode(zid)>

If "zkl" is less than 63 characters, it is also the "zTLD". If the resulting "zkl" should be longer than 63 characters, the string must be divided into smaller labels separated by the label separator ".". Here, the most significant bytes of the "zid" must be contained in the rightmost label of the resulting string and the least significant bytes in the leftmost label of the resulting string. For example, assuming a "zkl" of 130 characters, the encoding would be:

zTLD := zkl[126:129].zkl[63:125].zkl[0:62]

3. Zone Types

A zone type identifies a family of eight functions:

Private-KeyGen() -> d is a function to generate a fresh private key "d".

Public-KeyGen(d) -> zk is a function to derive a public key "zk" from a private key "d".

HDKD-Private(d,label) -> d' is an HDKD function which blinds a private zone key "d" using "label", resulting in another private key which can be used to create cryptographic signatures.

S-Encrypt(zk,label,nonce,expiration,rdata) -> bdata is a deterministic symmetric encryption function which encrypts the record data "rdata" based on key material derived from "zk", "label", "nonce" and "expiration". A deterministic encryption scheme is required to improve performance by leveraging caching features of DHTs.

Sign(d',bdata) -> sig is a function to sign "bdata" using the (blinded) private key "d'", yielding an unforgable cryptographic signature "sig".

HDKD-Public(zk,label) -> zk' is a HDKD function which blinds a public zone key "zk" using "label". "zk" and "zk'" must be unlinkable. Furthermore, blinding "zk" with different values for "label" must result in unlinkable different resulting values for "zk'".

Verify(zk',bdata,sig) -> valid is a function to verify the signature "sig" was created by the a private key "d'" derived from "d" and "label" if "zk'" was derived from the corresponding to "zk := Public-Keygen(d)" and "label". The function returns "true" if the signature is valid, and otherwise "false".

S-Decrypt(zk,label,nonce,expiration,bdata) -> rdata is a symmetric encryption function which decrypts the encrypted record data "bdata" based on key material derived from "zk", "label", "nonce" and "expiration".

Zone types are identified by a 32-bit resource record type number. Resource record types are discussed in the next section.

4. Resource Records

A GNS implementor MUST provide a mechanism to create and manage resource records for local zones. A local zone is established by selecting a zone type and creating a zone key pair. Implementations SHOULD select a secure zone type automatically and not leave the zone type selection to the user. Records may be added to each zone, hence a (local) persistency mechanism for resource records and zones must be provided. This local zone database is used by the GNS resolver implementation and to publish record information.

A GNS resource record holds the data of a specific record in a zone. The resource record format is defined as follows:

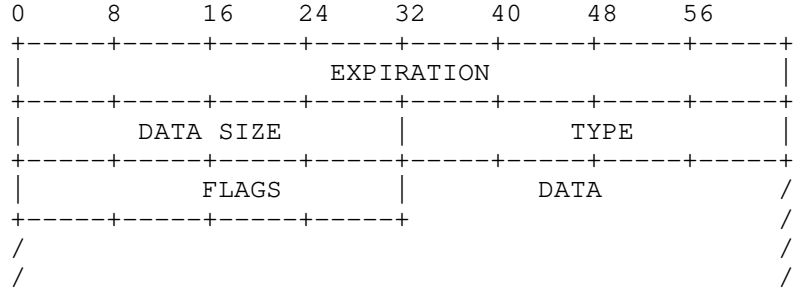


Figure 2

where:

EXPIRATION denotes the absolute 64-bit expiration date of the record. In microseconds since midnight (0 hour), January 1, 1970 in network byte order.

DATA SIZE denotes the 32-bit size of the DATA field in bytes and in network byte order.

TYPE is the 32-bit resource record type. This type can be one of the GNS resource records as defined in Section 4 or a DNS record type as defined in [RFC1035] or any of the complementary standardized DNS resource record types. This value must be stored in network byte order. Note that values below 2^16 are reserved for allocation via IANA ([RFC6895]), while values above 2^16 are allocated by the GNUnet Assigned Numbers Authority [GANA].

FLAGS is a 32-bit resource record flags field (see below).

DATA the variable-length resource record data payload. The contents are defined by the respective type of the resource record.

Flags indicate metadata surrounding the resource record. A flag value of 0 indicates that all flags are unset. The following illustrates the flag distribution in the 32-bit flag value of a resource record:

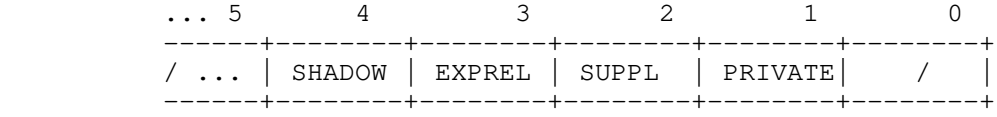


Figure 3

where:

SHADOW If this flag is set, this record should be ignored by resolvers unless all (other) records of the same record type have expired. Used to allow zone publishers to facilitate good performance when records change by allowing them to put future values of records into the DHT. This way, future values can propagate and may be cached before the transition becomes active.

EXPREL The expiration time value of the record is a relative time

(still in microseconds) and not an absolute time. This flag should never be encountered by a resolver for records obtained from the DHT, but might be present when a resolver looks up private records of a zone hosted locally.

SUPPL This is a supplemental record. It is provided in addition to the other records. This flag indicates that this record is not explicitly managed alongside the other records under the respective name but may be useful for the application. This flag should only be encountered by a resolver for records obtained from the DHT.

PRIVATE This is a private record of this peer and it should thus not be published in the DHT. Thus, this flag should never be encountered by a resolver for records obtained from the DHT. Private records should still be considered just like regular records when resolving labels in local zones.

5. Record Types

A registry of GNS Record Types is described in Section 12. The registration policy for this registry is "First Come First Served", as described in [RFC8126].

5.1. PKEY

In GNS, a delegation of a label to a zone of type "PKEY" is represented through a PKEY record. The PKEY number is a zone type and thus also implies the cryptosystem for the zone that is being delegated to. A PKEY resource record contains the public key of the zone to delegate to. A PKEY record MUST be the only record under a label. No other records are allowed. A PKEY DATA entry has the following format:

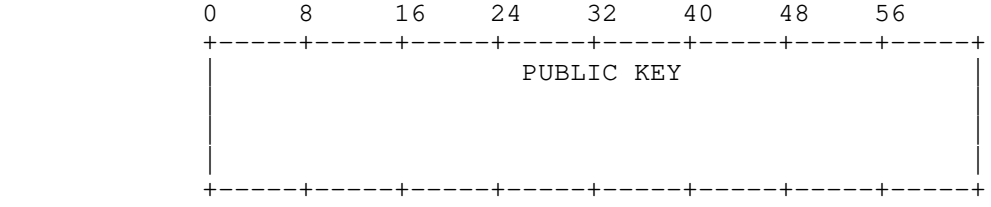


Figure 4

where:

PUBLIC KEY A 256-bit ECDSA zone key.

For PKEY zones the zone key material is derived using the curve parameters of the twisted edwards representation of Curve25519 [RFC7748] (a.k.a. edwards25519) with the ECDSA scheme ([RFC6979]). Consequently , we use the following naming convention for our cryptographic primitives for PKEY zones:

d is a 256-bit ECDSA private zone key.

zk is the ECDSA public zone key corresponding to d. It is defined in [RFC6979] as the curve point d*G where G is the group generator of the elliptic curve. The public key is used to uniquely identify a GNS zone and is referred to as the "zone key".

p is the prime of edwards25519 as defined in [RFC7748], i.e. 2^255 - 19.

G is the group generator (X(P),Y(P)) of edwards25519 as defined in [RFC7748].

L is the prime-order subgroup of edwards25519 in [RFC7748].

The "zid" of a PKEY is 32 + 4 bytes in length. This means that a "zTLD" will always fit into a single label and does not need any further conversion.

Given a label, the output d' of the HDKD-Private(d,label) function for zone key blinding is calculated as follows for PKEY zones:

```
zk := d * G
PRK_h := HKDF-Extract ("key-derivation", zk)
h := HKDF-Expand (PRK_h, label | "gns", 512 / 8)
d' := h * d mod L
```

Equally, given a label, the output zk' of the HDKD-Public(zk,label) function is calculated as follows for PKEY zones:

```
PRK_h := HKDF-Extract ("key-derivation", zk)
h := HKDF-Expand (PRK_h, label | "gns", 512 / 8)
zk' := h mod L * zk
```

The PKEY cryptosystem uses a hash-based key derivation function (HKDF) as defined in [RFC5869], using HMAC-SHA512 for the extraction phase and HMAC-SHA256 for the expansion phase. "PRK_h" is key material retrieved using an HKDF using the string "key-derivation" as salt and the public zone key "zk" as initial keying material. "h" is the 512-bit HKDF expansion result. The expansion info input is a concatenation of the label and string "gns". "label" is a UTF-8 string under which the resource records are published.

We point out that the multiplication of "zk" with "h" is a point multiplication, while the multiplication of "d" with "h" is a scalar multiplication.

The Sign() and Verify() functions for PKEY zones are implemented using 512-bit ECDSA deterministic signatures as specified in [RFC6979].

The S-Encrypt() and S-Decrypt() functions use AES in counter mode as defined in [MODES] (CTR-AES-256):

```
RDATA := CTR-AES256(K, IV, BDATA)
BDATA := CTR-AES256(K, IV, RDATA)
```

The key "K" and counter "IV" are derived from the record "label" and the zone key "zk" as follows:

```
PRK_k := HKDF-Extract ("gns-aes-ctx-key", zk)
PRK_n := HKDF-Extract ("gns-aes-ctx-iv", zk)
K := HKDF-Expand (PRK_k, label, 256 / 8);
NONCE := HKDF-Expand (PRK_n, label, 32 / 8)
```

HKDF is a hash-based key derivation function as defined in [RFC5869]. Specifically, HMAC-SHA512 is used for the extraction phase and HMAC-SHA256 for the expansion phase. The output keying material is 32 octets (256 bits) for the symmetric key and 4 octets (32 bits) for the nonce. The symmetric key "K" is a 256-bit AES [RFC3826] key.

The nonce is combined with a 64-bit initialization vector and a 32-bit block counter as defined in [RFC3686]. The block counter begins with the value of 1, and it is incremented to generate subsequent portions of the key stream. The block counter is a 32-bit integer value in network byte order. The initialization vector is the expiration time of the resource record block in network byte order. The resulting counter ("IV") wire format is as follows:

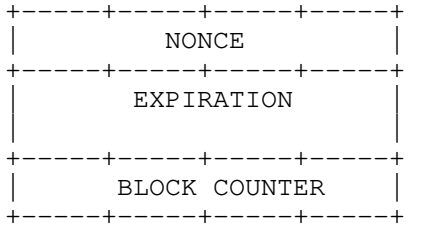


Figure 5

5.2. EDKEY

In GNS, a delegation of a label to a zone of type "EDKEY" is represented through a EDKEY record. The EDKEY number is a zone type and thus also implies the cryptosystem for the zone that is being delegated to. An EDKEY resource record contains the public key of the zone to delegate to. A EDKEY record MUST be the only record under a label. No other records are allowed. A EDKEY DATA entry has the following format:

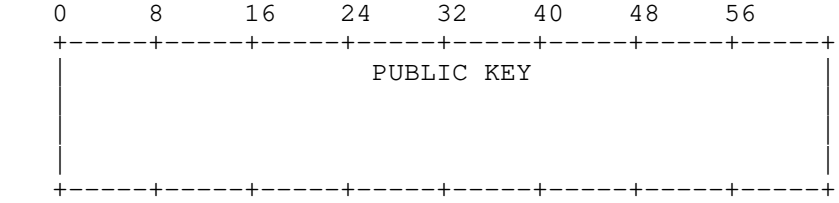


Figure 6

where:

PUBLIC KEY A 256-bit EdDSA zone key.

For EDKEY zones the zone key material is derived using the curve parameters of the twisted edwards representation of Curve25519 [RFC7748] (a.k.a. edwards25519) with the Ed25519-SHA-512 scheme [ed25519]. Consequently , we use the following naming convention for our cryptographic primitives for EDKEY zones:

d is a 256-bit EdDSA private zone key.

a is is an integer derived from "d" using the SHA512 hash function as defined in [ed25519].

zk is the EdDSA public zone key corresponding to "d". It is defined in [ed25519] as the curve point "a*G" where "G" is the group generator of the elliptic curve and "a" is an integer derived from "d" using the SHA512 hash function. The public key is used to uniquely identify a GNS zone and is referred to as the "zone key".

p is the prime of edwards25519 as defined in [RFC7748], i.e. 2^255 - 19.

G is the group generator (X(P),Y(P)) of edwards25519 as defined in [RFC7748].

L is the prime-order subgroup of edwards25519 in [RFC7748].

The "zid" of an EDKEY is 32 + 4 bytes in length. This means that a "zTLD" will always fit into a single label and does not need any further conversion.

The "EDKEY" HDKD instantiation is based on [Tor224]. Given a label, the output of the HDKD-Private function for zone key blinding is calculated as follows for EDKEY zones:

```
zk := a * G
PRK_h := HKDF-Extract ("key-derivation", zk)
h := HKDF-Expand (PRK_h, label | "gns", 512 / 8)
h[31] &= 7
a1 := a / 8 /* 8 is the cofactor of Curve25519 */
a2 := h * a1 mod L
a' = a2 * 8 /* 8 is the cofactor of Curve25519 */
```

Equally, given a label, the output of the HDKD-Public function is calculated as follows for PKEY zones:

```
PRK_h := HKDF-Extract ("key-derivation", zk)
h := HKDF-Expand (PRK_h, label | "gns", 512 / 8)
h[31] &= 7  // Implies h mod L == h
zk' := h * zk
```

We note that implementors must employ a constant time scalar multiplication for the constructions above. Also, implementors must ensure that the private key "a" is an ed25519 private key and specifically that "a[0] & 7 == 0" holds.

The EDKEY cryptosystem uses a hash-based key derivation function (HKDF) as defined in [RFC5869], using HMAC-SHA512 for the extraction phase and HMAC-SHA256 for the expansion phase. "PRK_h" is key material retrieved using an HKDF using the string "key-derivation" as salt and the public zone key "zk" as initial keying material. "h" is the 512-bit HKDF expansion result. The expansion info input is a concatenation of the label and string "gns". The result of the HKDF must be clamped. "a" is the 256-bit integer corresponding to the 256-bit private zone key "d". "label" is a UTF-8 string under which the resource records are published.

We point out that the multiplication of "zk" with "h" is a point multiplication, while the division and multiplication of "a" and "a1" with the cofactor are integer operations.

Signatures for EDKEY zones using the derived private key "a'" are NOT compliant with [ed25519]. Instead, signatures MUST be generated as follows for any given message M and deterministic random-looking "r":

```
R := r * G
S := r + SHA512(R, zk', M) * a' mod L
```

A signature (R,S) is valid if the following holds:

```
SB == R + SHA512(R, zk', M) * A'
```

The S-Encrypt() and S-Decrypt() functions use AES in galois counter mode as defined in [GCM] (GCM-AES-256):

```
RDATA := GCM-AES-256(K, IV, BDATA)
BDATA := GCM-AES-256(K, IV, RDATA) = CIPHERTEXT | GCM_TAG
```

The result of the GCM encryption function is the encrypted ciphertext concatenated with the 128-bit GCM authentication tag "GCM_TAG". Accordingly, the length of BDATA equals the length of the RDATA plus the 16 octets of the authentication tag.

The key "K" and counter "IV" are derived from the record "label" and the zone key "zk" as follows:

```
PRK_k := HKDF-Extract ("gns-aes-ctx-key", zk)
PRK_n := HKDF-Extract ("gns-aes-ctx-iv", zk)
K := HKDF-Expand (PRK_k, label, 256 / 8);
IV := HKDF-Expand (PRK_n, label, 96 / 8)
```

HKDF is a hash-based key derivation function as defined in [RFC5869]. Specifically, HMAC-SHA512 is used for the extraction phase and HMAC-SHA256 for the expansion phase. The output keying material is 32 octets (256 bits) for the symmetric key and 12 octets (96 bits) for the IV. The symmetric key "K" is a 256-bit AES [RFC3826] key. No additional authenticated data (AAD) is used.

5.3. GNS2DNS

It is possible to delegate a label back into DNS through a GNS2DNS record. The resource record contains a DNS name for the resolver to continue with in DNS followed by a DNS server. Both names are in the format defined in [RFC1034] for DNS names. A GNS2DNS DATA entry has the following format:

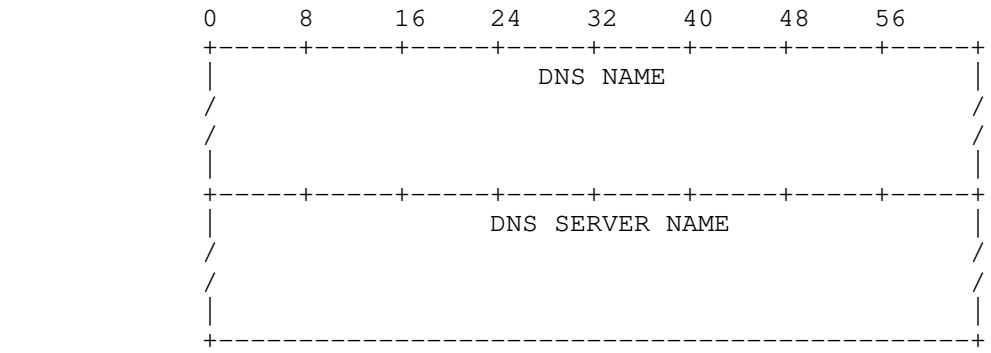


Figure 7

where:

DNS NAME The name to continue with in DNS (0-terminated).

DNS SERVER NAME The DNS server to use. May be an IPv4/IPv6 address in dotted decimal form or a DNS name. It may also be a relative GNS name ending with a "+" top-level domain. The value is UTF-8 encoded (also for DNS names) and 0-terminated.

5.4. LEHO

Legacy hostname records can be used by applications that are expected to supply a DNS name on the application layer. The most common use case is HTTP virtual hosting, which as-is would not work with GNS names as those may not be globally unique. A LEHO resource record is expected to be found together in a single resource record with an IPv4 or IPv6 address. A LEHO DATA entry has the following format:

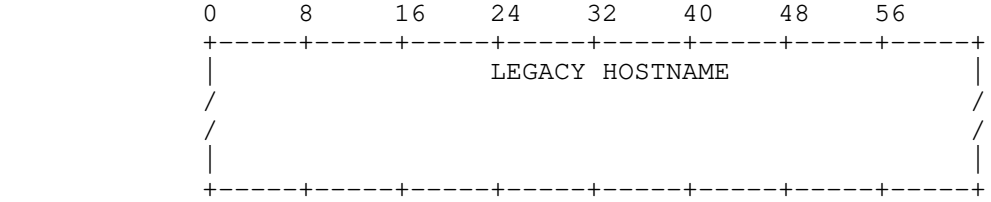


Figure 8

where:

LEGACY HOSTNAME A UTF-8 string (which is not 0-terminated) representing the legacy hostname.

NOTE: If an application uses a LEHO value in an HTTP request header (e.g. "Host:" header) it must be converted to a punycode representation [RFC5891].

5.5. NICK

Nickname records can be used by zone administrators to publish an indication on what label this zone prefers to be referred to. This is a suggestion to other zones what label to use when creating a delegation record (Section 3) containing this zone's public zone key. This record SHOULD only be stored under the empty label "@" but MAY be returned with record sets under any label as a supplemental record. Section 8.2.6 details how a resolver must process supplemental and non-supplemental NICK records. A NICK DATA entry has the following format:

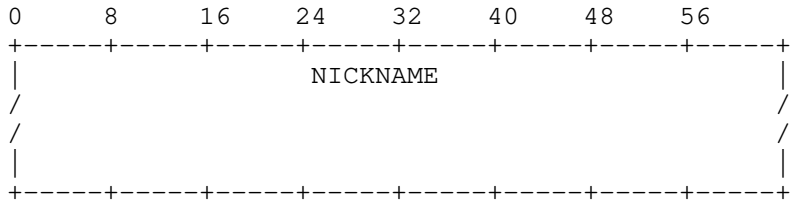


Figure 9

where:

NICKNAME A UTF-8 string (which is not 0-terminated) representing the preferred label of the zone. This string MUST NOT include a "." character.

5.6. BOX

In GNS, every "." in a name delegates to another zone, and GNS lookups are expected to return all of the required useful information in one record set. This is incompatible with the special labels used by DNS for SRV and TLSA records. Thus, GNS defines the BOX record format to box up SRV and TLSA records and include them in the record set of the label they are associated with. For example, a TLSA record for "_https._tcp.example.org" will be stored in the record set of "example.org" as a BOX record with service (SVC) 443 (https) and protocol (PROTO) 6 (tcp) and record TYPE "TLSA". For reference, see also [RFC2782]. A BOX DATA entry has the following format:

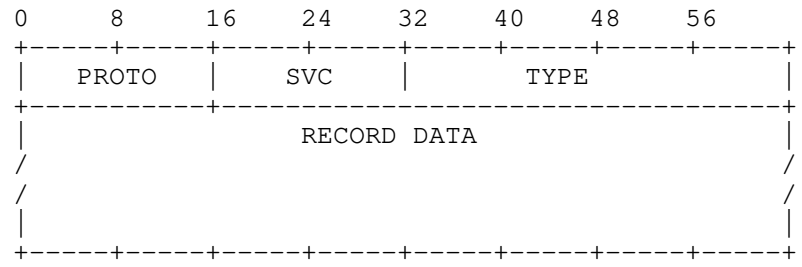


Figure 10

where:

PROTO the 16-bit protocol number, e.g. 6 for tcp. In network byte order.

SVC the 16-bit service value of the boxed record, i.e. the port number. In network byte order.

TYPE is the 32-bit record type of the boxed record. In network byte order.

RECORD DATA is a variable length field containing the "DATA" format of TYPE as defined for the respective TYPE in DNS.

5.7. VPN

A VPN DATA entry has the following format:

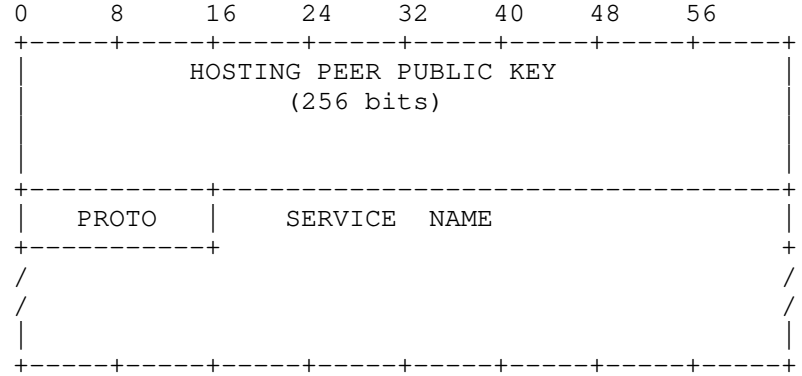


Figure 11

where:

HOSTING PEER PUBLIC KEY is a 256-bit EdDSA public key identifying the peer hosting the service.

PROTO the 16-bit protocol number, e.g. 6 for TCP. In network byte order.

SERVICE NAME a shared secret used to identify the service at the hosting peer, used to derive the port number requird to connect to the service. The service name MUST be a 0-terminated UTF-8 string.

6. Publishing Records

GNS resource records are published in a distributed hash table (DHT). We assume that a DHT provides two functions: GET(key) and PUT(key,value). In GNS, resource records are grouped by their respective labels, encrypted and published together in a single resource records block (RRBLOCK) in the DHT under a key "q": PUT(q, RRBLOCK). The key "q" which is derived from the zone key "zk" and the respective "label" of the contained records.

6.1. DHT Key Derivations

Given a label, the DHT key "q" is derived as follows:

q := SHA512 (HDKD-Public(zk, label))

label is a UTF-8 string under which the resource records are published.

zk is the public zone key.

q Is the 512-bit DHT key under which the resource records block is published. It is the SHA512 hash over the derived public zone key.

6.2. Resource Records Block

GNS records are grouped by their labels and published as a single block in the DHT. The grouped record sets MAY be paired with any number of supplemental records. Supplemental records must have the supplemental flag set (See Section 4). The contained resource records are encrypted using a symmetric encryption scheme. A GNS implementation must publish RRBLOCKs in accordance to the properties and recommendations of the underlying DHT. This may include a periodic refresh publication. A GNS RRBLOCK has the following format:

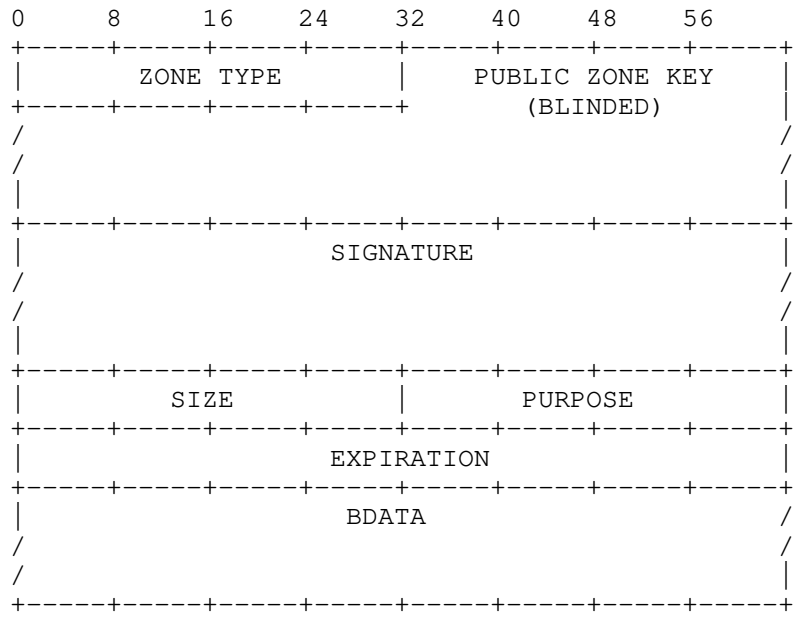


Figure 12

where:

ZONE TYPE is the 32-bit zone type.

ZONE PUBLIC KEY is the blinded public zone key "HDKD-Public(zk, label)" to be used to verify SIGNATURE.

SIGNATURE The signature is computed over the data following the PUBLIC KEY field. The signature is created using the Sign() function of the cryptosystem of the zone and the derived private key "HDKD-Private(d, label)" (see Section 3).

SIZE A 32-bit value containing the length of the signed data following the PUBLIC KEY field in network byte order. This value always includes the length of the fields SIZE (4), PURPOSE (4) and EXPIRATION (8) in addition to the length of the BDATA. While a 32-bit value is used, implementations MAY refuse to publish blocks beyond a certain size significantly below 4 GB. However, a minimum block size of 62 kilobytes MUST be supported.

PURPOSE A 32-bit signature purpose flag. This field MUST be 15 (in network byte order).

EXPIRATION Specifies when the RRBLOCK expires and the encrypted

block SHOULD be removed from the DHT and caches as it is likely stale. However, applications MAY continue to use non-expired individual records until they expire. The value MUST be set to the expiration time of the resource record contained within this block with the smallest expiration time. If a records block includes shadow records, then the maximum expiration time of all shadow records with matching type and the expiration times of the non-shadow records is considered. This is a 64-bit absolute date in microseconds since midnight (0 hour), January 1, 1970 in network byte order.

BDATA The encrypted resource records with a total size of SIZE - 16.

6.3. Record Data Encryption and Decryption

A symmetric encryption scheme is used to encrypt the resource records set RDATA into the BDATA field of a GNS RRBLOCK. The wire format of the RDATA looks as follows:

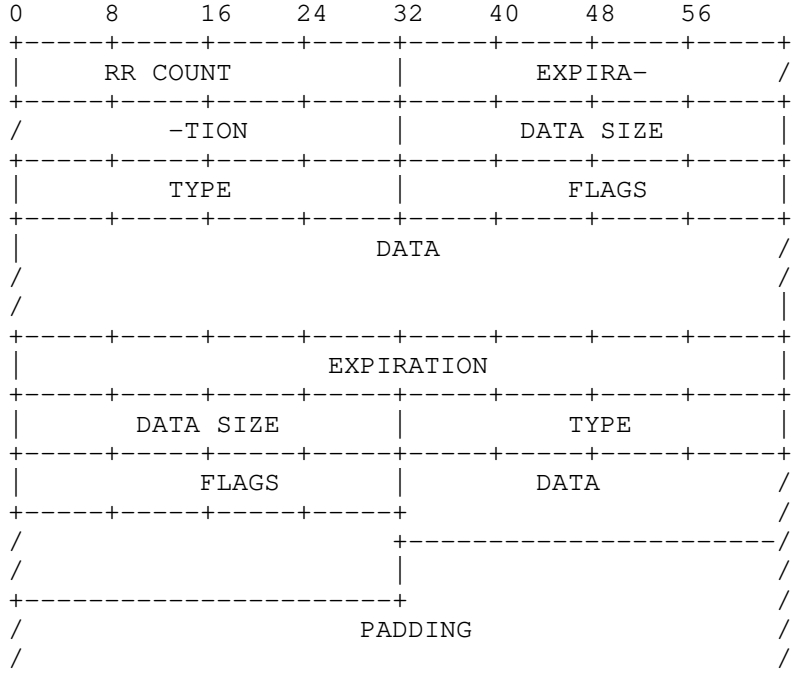


Figure 13

where:

RR COUNT A 32-bit value containing the number of variable-length

resource records which are following after this field in network byte order.

EXPIRATION, DATA SIZE, TYPE, FLAGS and DATA These fields were defined in the resource record format in Section 4. There MUST be a total of RR COUNT of these resource records present.

PADDING The padding MUST contain the value 0 in all octets. The padding MUST ensure that the size of the RDATA WITHOUT the RR COUNT field is a power of two. As a special exception, record sets with (only) a zone delegation record type are never padded. Note that a record set with a delegation record MUST NOT contain other records.

7. Internationalization and Character Encoding

All labels in GNS are encoded in UTF-8 [RFC3629]. This does not include any DNS names found in DNS records, such as CNAME records, which are internationalized through the IDNA specifications [RFC5890].

8. Name Resolution

Names in GNS are resolved by recursively querying the DHT record storage. In the following, we define how resolution is initiated and each iteration in the resolution is processed.

GNS resolution of a name must start in a given starting zone indicated using a zone public key. Details on how the starting zone may be determined is discussed in Section 10.

When GNS name resolution is requested, a desired record type MAY be provided by the client. The GNS resolver will use the desired record type to guide processing, for example by providing conversion of VPN records to A or AAAA records, if that is desired. However, filtering of record sets according to the required record types MUST still be done by the client after the resource record set is retrieved.

8.1. Recursion

In each step of the recursive name resolution, there is an authoritative zone zk and a name to resolve. The name may be empty. Initially, the authoritative zone is the start zone. If the name is empty, it is interpreted as the apex label "@".

From here, the following steps are recursively executed, in order:

1. Extract the right-most label from the name to look up.

2. Calculate q using the label and zk as defined in Section 6.1.

3. Perform a DHT query GET(q) to retrieve the RRBLOCK.

4. Verify and process the RRBLOCK and decrypt the BDATA contained in it as defined in Section 6.3.

Upon receiving the RRBLOCK from the DHT, apart from verifying the provided signature, the resolver MUST check that the authoritative zone key was used to sign the record: The derived zone key "h*zk" MUST match the public key provided in the RRBLOCK, otherwise the RRBLOCK MUST be ignored and the DHT lookup GET(q) MUST continue.

8.2. Record Processing

Record processing occurs at the end of a single recursion. We assume that the RRBLOCK has been cryptographically verified and decrypted. At this point, we must first determine if we have received a valid record set in the context of the name we are trying to resolve:

1. Case 1: If the remainder of the name to resolve is empty and the record set does not consist of a delegation, CNAME or DNS2GNS record, the record set is the result and the recursion is concluded.

2. Case 2: If the name to be resolved is of the format "_SERVICE._PROTO" and the record set contains one or more matching BOX records, the records in the BOX records are the result and the recusion is concluded (Section 8.2.4).

3. Case 3: If the remainder of the name to resolve is not empty and does not match the "_SERVICE._PROTO" syntax, then the current record set MUST consist of a single delegation record (Section 8.2.1), a single CNAME record (Section 8.2.3), or one or more GNS2DNS records (Section 8.2.2), which are processed as described in the respective sections below. The record set may include any number of supplemental records. Otherwise, resolution fails and the resolver MUST return an empty record set. Finally, after the recursion terminates, the client preferences for the record type SHOULD be considered. If a VPN record is found and the client requests an A or AAAA record, the VPN record SHOULD be converted (Section 8.2.5) if possible.

8.2.1. Encountering zone delegation records

When the resolver encounters a record of a supported zone delegation record type (such as PKEY or EDKEY) and the remainder of the name is not empty, resolution continues recursively with the remainder of the name in the GNS zone specified in the delegation record. Implementations MUST NOT allow multiple different zone type delegations under a single label. Implementations MAY support any subset of zone types. If an unsupported zone type is encountered, resolution fails (NXDOMAIN).

If the remainder of the name to resolve is empty and we have received a record set containing only a single PKEY record, the recursion is continued with the PKEY as authoritative zone and the empty apex label "@" as remaining name, except in the case where the desired record type is PKEY, in which case the PKEY record is returned and the resolution is concluded without resolving the empty apex label.

8.2.2. GNS2DNS

When a resolver encounters one or more GNS2DNS records and the remaining name is empty and the desired record type is GNS2DNS, the GNS2DNS records are returned.

Otherwise, it is expected that the resolver first resolves the IP(s) of the specified DNS name server(s). GNS2DNS records MAY contain numeric IPv4 or IPv6 addresses, allowing the resolver to skip this step. The DNS server names may themselves be names in GNS or DNS. If the DNS server name ends in ".+", the rest of the name is to be interpreted relative to the zone of the GNS2DNS record. If the DNS server name ends in a label representation of a zone key, the DNS server name is to be resolved against the GNS zone zk.

Multiple GNS2DNS records may be stored under the same label, in which case the resolver MUST try all of them. The resolver MAY try them in any order or even in parallel. If multiple GNS2DNS records are present, the DNS name MUST be identical for all of them, if not the resolution fails and an emtpy record set is returned as the record set is invalid.

Once the IP addresses of the DNS servers have been determined, the DNS name from the GNS2DNS record is appended to the remainder of the name to be resolved, and resolved by querying the DNS name server(s). As the DNS servers specified are possibly authoritative DNS servers, the GNS resolver MUST support recursive resolution and MUST NOT delegate this to the authoritative DNS servers. The first successful recursive name resolution result is returned to the client. In addition, the resolver returns the queried DNS name as a supplemental LEHO record (Section 5.4) with a relative expiration time of one hour.

GNS resolvers SHOULD offer a configuration option to disable DNS processing to avoid information leakage and provide a consistent security profile for all name resolutions. Such resolvers would return an empty record set upon encountering a GNS2DNS record during the recursion. However, if GNS2DNS records are encountered in the record set for the apex and a GNS2DNS record is expicitly requested by the application, such records MUST still be returned, even if DNS support is disabled by the GNS resolver configuration.

8.2.3. CNAME

If a CNAME record is encountered, the canonical name is appended to the remaining name, except if the remaining name is empty and the desired record type is CNAME, in which case the resolution concludes with the CNAME record. If the canonical name ends in ".+", resolution continues in GNS with the new name in the current zone. Otherwise, the resulting name is resolved via the default operating system name resolution process. This may in turn again trigger a GNS resolution process depending on the system configuration.

The recursive DNS resolution process may yield a CNAME as well which in turn may either point into the DNS or GNS namespace (if it ends in a label representation of a zone key). In order to prevent infinite loops, the resolver MUST implement loop detections or limit the number of recursive resolution steps. If the last CNAME was a DNS name, the resolver returns the DNS name as a supplemental LEHO record (Section 5.4) with a relative expiration time of one hour.

8.2.4. BOX

When a BOX record is received, a GNS resolver must unbox it if the name to be resolved continues with "_SERVICE._PROTO". Otherwise, the BOX record is to be left untouched. This way, TLSA (and SRV) records do not require a separate network request, and TLSA records become inseparable from the corresponding address records.

8.2.5. VPN

At the end of the recursion, if the queried record type is either A or AAAA and the retrieved record set contains at least one VPN record, the resolver SHOULD open a tunnel and return the IPv4 or IPv6 tunnel address, respectively. The type of tunnel depends on the contents of the VPN record data. The VPN record MUST be returned if the resolver implementation does not support setting up a tunnnel.

8.2.6. NICK

NICK records are only relevant to the recursive resolver if the record set in question is the final result which is to be returned to the client. The encountered NICK records may either be supplemental (see Section 4) or non-supplemental. If the NICK record is supplemental, the resolver only returns the record set if one of the non-supplemental records matches the queried record type.

The differentiation between a supplemental and non-supplemental NICK record allows the client to match the record to the authoritative zone. Consider the following example:

```
Query: alice.example (type=A)
Result:
A: 192.0.2.1
NICK: eve
```

Figure 14

In this example, the returned NICK record is non-supplemental. For the client, this means that the NICK belongs to the zone "alice.doe" and is published under the empty label along with an A record. The NICK record should be interpreted as: The zone defined by "alice.doe" wants to be referred to as "eve". In contrast, consider the following:

```
Query: alice.example (type=AAAA)
Result:
AAAA: 2001:DB8::1
NICK: john (Supplemental)
```

Figure 15

In this case, the NICK record is marked as supplemental. This means that the NICK record belongs to the zone "doe" and is published under the label "alice" along with an A record. The NICK record should be interpreted as: The zone defined by "doe" wants to be referred to as "john". This distinction is likely useful for other records published as supplemental.

9. Zone Revocation

Whenever a recursive resolver encounters a new GNS zone, it MUST check against the local revocation list whether the respective zone key has been revoked. If the zone key was revoked, the resolution MUST fail with an empty result set.

In order to revoke a zone key, a signed revocation object SHOULD be published. This object MUST be signed using the private zone key. The revocation object is flooded in the overlay network. To prevent flooding attacks, the revocation message MUST contain a proof of work (PoW). The revocation message including the PoW MAY be calculated ahead of time to support timely revocation.

For all occurences below, "Argon2id" is the Password-based Key Derivation Function as defined in [Argon2]. For the PoW calculations the algorithm is instantiated with the following parameters:

S The salt. Fixed 16-octet string: "GnsRevocationPow".

t Number of iterations: 3

m Memory size in KiB: 1024

T Output length of hash in bytes: 64

p Parallelization parameter: 1

v Algorithm version: 0x13

y Algorithm type (Argon2id): 2

X Unused

K Unused

The following is the message string "P" on which the PoW is calculated:

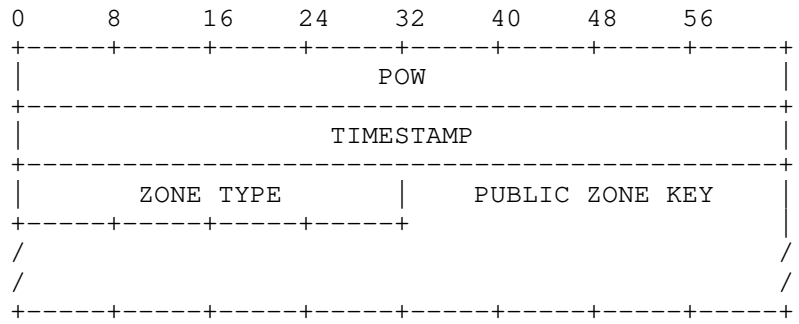


Figure 16

where:

POW A 64-bit solution to the PoW. In network byte order.

TIMESTAMP denotes the absolute 64-bit date when the revocation was computed. In microseconds since midnight (0 hour), January 1, 1970 in network byte order.

PUBLIC KEY is the 256-bit public key "zk" of the zone which is being revoked and the key to be used to verify SIGNATURE. The wire format of this value is defined in [RFC8032], Section 5.1.5.

Traditionally, PoW schemes require to find a "POW" such that at least D leading zeroes are found in the hash result. D is then referred to as the "difficulty" of the PoW. In order to reduce the variance in time it takes to calculate the PoW, we require that a number "Z" different PoWs must be found that on average have "D" leading zeroes.

The resulting proofs may then published and disseminated. The concrete dissemination and publication methods are out of scope of this document. Given an average difficulty of "D", the proofs have an expiration time of EPOCH. With each additional bit difficulty, the lifetime of the proof is prolonged for another EPOCH. Consequently, by calculating a more difficult PoW, the lifetime of the proof can be increased on demand by the zone owner.

The parameters are defined as follows:

Z The number of PoWs required is fixed at 32.

D The difficulty is fixed at 22.

EPOCH A single epoch is fixed at 365 days.

Given that proof has been found, a revocation data object is defined as follows:

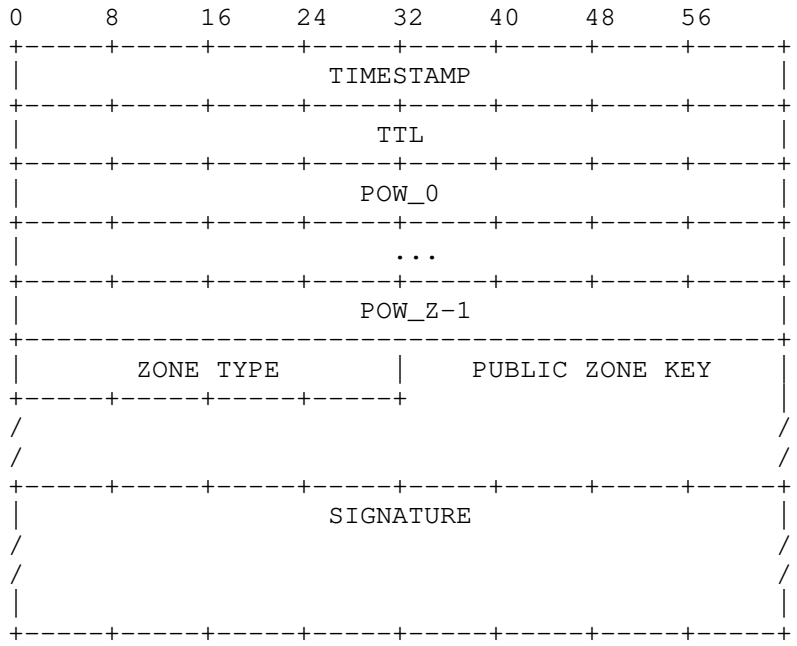


Figure 17

where:

TIMESTAMP denotes the absolute 64-bit date when the revocation was computed. In microseconds since midnight (0 hour), January 1, 1970 in network byte order. This is the same value as the timestamp used in the individual PoW calculations.

TTL denotes the relative 64-bit time to live of of the record in microseconds also in network byte order. This field is informational for a verifier. The verifier may discard revocation if the TTL indicates that it is already expired. However, the actual TTL of the revocation must be determined by examining the leading zeros in the proof of work calculation.

POW_i The values calculated as part of the PoW, in network byte order. Each POW_i MUST be unique in the set of POW values. To facilitate fast verification of uniqueness, the POW values must be given in strictly monotonically increasing order in the message.

ZONE TYPE The 32-bit zone type corresponding to the zone public key.

ZONE PUBLIC KEY is the public key "zk" of the zone which is being revoked and the key to be used to verify SIGNATURE.

SIGNATURE A signature over a timestamp and the public zone zk of the zone which is revoked and corresponds to the key used in the PoW. The signature is created using the Sign() function of the cryptosystem of the zone and the private zone key (see Section 3).

The signature over the public key covers a 32-bit pseudo header conceptually prefixed to the public key. The pseudo header includes the key length and signature purpose:

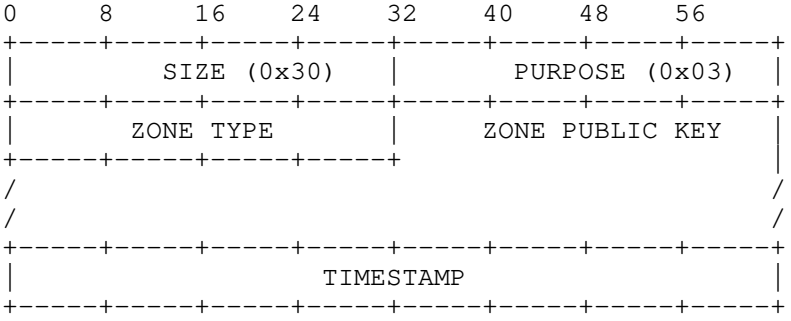


Figure 18

where:

SIZE A 32-bit value containing the length of the signed data in bytes in network byte order.

PURPOSE A 32-bit signature purpose flag. This field MUST be 3 (in network byte order).

ZONE TYPE The 32-bit zone type corresponding to the zone public key.

ZONE PUBLIC KEY / TIMESTAMP Both values as defined in the revocation data object above.

In order to verify a revocation the following steps must be taken, in order:

1. The current time MUST be between TIMESTAMP and TIMESTAMP+TTL.

2. The signature MUST match the public key.

3. The set of POW values MUST NOT contain duplicates.

4. The average number of leading zeroes resulting from the provided POW values D' MUST be greater than D.

5. The validation period (TTL) of the revocation is calculated as (D'-D) * EPOCH * 1.1. The EPOCH is extended by 10% in order to deal with unsynchronized clocks. The TTL added on top of the TIMESTAMP yields the expiration date.

10. Determining the Root Zone and Zone Governance

The resolution of a GNS name must start in a given start zone indicated to the resolver using any public zone key. The local resolver may have a local start zone configured/hard-coded which points to a local or remote start zone key. A resolver client may also determine the start zone from the suffix of the name given for resolution or using information retrieved out of band. The governance model of any zone is at the sole discretion of the zone owner. However, the choice of start zone(s) is at the sole discretion of the local system administrator or user.

This is an important distinguishing factor from the Domain Name System where root zone governance is centralized at the Internet Corporation for Assigned Names and Numbers (ICANN). In DNS terminology, GNS roughly follows the idea of a hyper-hyper local root zone deployment, with the difference that it is not expected that all deployments use the same local root zone.

In the following, we give examples how a local client resolver SHOULD discover the start zone. The process given is not exhaustive and clients MAY suppliement it with other mechanisms or ignore it if the particular application requires a different process.

GNS clients MUST first try to interpret the top-level domain of a GNS name as a zone key representation ("zTLD"). If the top-level domain is indicated to be a label representation of a public zone key with a well-defined "ztype" value, the root zone of the resolution process is implicitly given by the suffic of the name:

Example name: www.example.<zTLD>
=> Root zone: zk of type ztype
=> Name to resolve from root zone: www.example

In GNS, users MAY own and manage their own zones. Each local zone SHOULD be associated with a single GNS label, but users MAY choose to use longer names consisting of multiple labels. If the name of a locally managed zone matches the suffix of the name to be resolved, resolution SHOULD start from the respective local zone:

```
Example name: www.example.org
Local zones:
fr = (d0,zk0)
gnu = (d1,zk1)
com = (d2,zk2)
...
=> Entry zone: zk1
=> Name to resolve from entry zone: www.example
```

Finally, additional "suffix to zone" mappings MAY be configured. Suffix to zone key mappings SHOULD be configurable through a local configuration file or database by the user or system administrator. The suffix MAY consist of multiple GNS labels concatenated with a ".". If multiple suffixes match the name to resolve, the longest matching suffix MUST BE used. The suffix length of two results cannot be equal, as this would indicate a misconfiguration. If both a locally managed zone and a configuration entry exist for the same suffix, the locally managed zone MUST have priority.

```
Example name: www.example.org
Local suffix mappings:
gnu = zk0
example.org = zk1
example.com = zk2
...
=> Entry zone: zk1
=> Name to resolve from entry zone: www
```

11. Security Considerations

11.1. Cryptography

The security of cryptographic systems depends on both the strength of the cryptographic algorithms chosen and the strength of the keys used with those algorithms. The security also depends on the engineering of the protocol used by the system to ensure that there are no non-cryptographic ways to bypass the security of the overall system.

This document concerns itself with the selection of cryptographic algorithms for use in GNS. The algorithms identified in this document are not known to be broken (in the cryptographic sense) at the current time, and cryptographic research so far leads us to believe that they are likely to remain secure into the foreseeable future. However, this isn't necessarily forever, and it is expected that new revisions of this document will be issued from time to time to reflect the current best practices in this area.

GNS PKEY zone keys use ECDSA over Curve25519. This is an unconventional choice, as ECDSA is usually used with other curves. However, traditional ECDSA curves are problematic for a range of reasons described in the Curve25519 and EdDSA papers. Using EdDSA directly is also not possible, as a hash function is used on the private key which destroys the linearity that the GNU Name System depends upon. We are not aware of anyone suggesting that using Curve25519 instead of another common curve of similar size would lower the security of ECDSA. GNS uses 256-bit curves because that way the encoded (public) keys fit into a single DNS label, which is good for usability.

In terms of crypto-agility, whenever the need for an updated cryptographic scheme arises to, for example, replace ECDSA over Curve25519 for PKEY records it may simply be introduced through a new record type. Such a new record type may then replace the delegation record type for future records. The old record type remains and zones can iteratively migrate to the updated zone keys.

In order to ensure ciphertext indistinguishability, care must be taken with respect to the initialization vector in the counter block. In our design, the IV is always the expiration time of the record block. For blocks with relative expiration times it is implicitly ensured that each time a block is published into the DHT, its IV is unique as the expiration time is calculated dynamically and increases monotonically. For blocks with absolute expiration times, the implementation MUST ensure that the expiration time is modified when the record data changes. For example. the expiration time may be increased by a single microsecond.

11.2. Abuse mitigation

GNS names are UTF-8 strings. Consequently, GNS faces similar issues with respect to name spoofing as DNS does for internationalized domain names. In DNS, attackers may register similar sounding or looking names (see above) in order to execute phishing attacks. GNS zone administrators must take into account this attack vector and incorporate rules in order to mitigate it.

Further, DNS can be used to combat illegal content on the internet by having the respective domains seized by authorities. However, the same mechanisms can also be abused in order to impose state censorship, which ist one of the motivations behind GNS. Hence, such a seizure is, by design, difficult to impossible in GNS. In particular, GNS does not support WHOIS ([RFC3912]).

11.3. Zone management

In GNS, zone administrators need to manage and protect their zone keys. Once a zone key is lost it cannot be recovered. Once it is compromised it cannot be revoked (unless a revocation message was pre-calculated and is still available). Zone administrators, and for GNS this includes end-users, are required to responsibly and dilligently protect their cryptographic keys. Offline signing is in principle possible, but GNS does not support separate zone signing and key-signing keys (as in [RFC6781]) in order to provide usable security.

Similarly, users are required to manage their local root zone. In order to ensure integrity and availability or names, users must ensure that their local root zone information is not compromised or outdated. It can be expected that the processing of zone revocations and an initial root zone is provided with a GNS client implementation ("drop shipping"). Extension and customization of the zone is at the full discretion of the user.

11.4. Impact of underlying DHT

This document does not specifiy the properties of the underlying distributed hash table (DHT) which is required by any GNS implementation. For implementors, it is important to note that the properties of the DHT are directly inherited by the GNS implementation. This includes both security as well as other non-functional properties such as scalability and performance. Implementors should take great care when selecting or implementing a DHT for use in a GNS implementation. DHTs with strong security and performance guarantees exist [R5N]. It should also be taken into consideration that GNS implementations which build upon different DHT overlays are unlikely to be interoperable with each other.

11.5. Revocations

Zone administrators are advised to pre-generate zone revocations and securely store the revocation information in case the zone key is lost, compromised or replaced in the furture. Pre-calculated revocations may become invalid due to expirations or protocol changes such as epoch adjustments. Consequently, implementors and users must make precautions in order to manage revocations accordingly.

Revocation payloads do NOT include a ’new’ key for key replacement. Inclusion of such a key would have two major disadvantages:

If revocation is used after a private key was compromised, allowing key replacement would be dangerous: if an adversary took over the private key, the adversary could then broadcast a revocation with a key replacement. For the replacement, the compromised owner would have no chance to issue even a revocation. Thus, allowing a revocation message to replace a private key makes dealing with key compromise situations worse.

Sometimes, key revocations are used with the objective of changing cryptosystems. Migration to another cryptosystem by replacing keys via a revocation message would only be secure as long as both cryptosystems are still secure against forgery. Such a planned, non-emergency migration to another cryptosystem should be done by running zones for both ciphersystems in parallel for a while. The migration would conclude by revoking the legacy zone key only once it is deemed no longer secure, and hopefully after most users have migrated to the replacement.

12. GANA Considerations

GANA [GANA] is requested to create an "GNU Name System Record Types" registry. The registry shall record for each entry:

* Name: The name of the record type (case-insensitive ASCII string, restricted to alphanumeric characters

* Number: 32-bit, above 65535

* Comment: Optionally, a brief English text describing the purpose of the record type (in UTF-8)

* Contact: Optionally, the contact information of a person to contact for further information

* References: Optionally, references describing the record type (such as an RFC)

The registration policy for this sub-registry is "First Come First Served", as described in [RFC8126]. GANA is requested to populate this registry as follows:

Number	Name	Contact	References	Description
65536	PKEY	N/A	[This.I-D]	GNS zone delegation
65537	NICK	N/A	[This.I-D]	GNS zone nickname
65538	LEHO	N/A	[This.I-D]	GNS legacy hostname
65539	VPN	N/A	[This.I-D]	VPN resolution
65540	GNS2DNS	N/A	[This.I-D]	Delegation to DNS
65541	BOX	N/A	[This.I-D]	Boxed record

Figure 19

GANA is requested to amend the "GNUnet Signature Purpose" registry as follows:

Purpose	Name	References	Description
3	GNS_REVOCATION	[This.I-D]	GNS zone key revocation
15	GNS_RECORD_SIGN	[This.I-D]	GNS record set signature

Figure 20

13. Test Vectors

The following represents a test vector for a record set with a DNS record of type "A" as well as a GNS record of type "PKEY" under the label "test".

Zone private key (d, little-endian, with ztype prepended):
0001000020110ab2
807d702f7b86dc30
6c37e8e2e0a5dbb2
7ae934727d9ca07d
69c73579

Zone identifier (zid):
0001000063db8bf0
44212617ce5db4fc
7c06fb9e35b2e177
3b4b76c05b42a1e7
17d018c6

Encoded zone identifier (zkl = zTLD):
000G0033VE5Z0H114RBWWQDMZHY0DYWY6PSE2XSV9DVC0PT2M7KHFM0RRR

Label: test
RRCOUNT: 2

```
Record #0
EXPIRATION: 1602865000130231
DATA_SIZE: 4
TYPE: 1
FLAGS: 0
DATA:
01020304

Record #1
EXPIRATION: 1602865000130231
DATA_SIZE: 32
TYPE: 65536
FLAGS: 2
DATA:
00010000796f4a8b
66d7780f62f46604
24c750295f31674d
052a4989cf0779a7

RDATA:
0005b1cc16f4a6b7
0000000400000001
0000000001020304
0005b1cc16f4a6b7
0000002000010000
0000000200010000
796f4a8b66d7780f
62f4660424c75029
5f31674d052a4989
cf0779a700000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000

BDATA:
4f20986bde1fbbed
b57196c1c23e35e9
f1ee62207de81297
0c2b370a9980042f
e8296cdd8ca66d69
11ebb2f3b2550959
7cb781ef56ac07d1
7c5dd0903bb94c67
c07e100079f59db3
3363fe110f435838
```

```
ef482e60b527f553
2ee435e4c0525439
3965d3dbe72e7c92
9bb4172b3bda7270
06c33578682cb212
23ac2cf389a4fbab
bb8cb55e

RRBLOCK:
000100007bc2eb40
ef056b05a5a84c35
241ca7190284a4a4
f5afdae14e8b784c
4b516dd6082d7969
2d2bbcb1328bc1df
270b2c02693bdaa9
f4d496dd850068d4
3a471fac0156b902
3536e54960fac47b
58762d82c5ad8e7f
34a121819c7ca75d
64c78d3a00000094
0000000f0005b1cc
16f4a6b74f20986b
de1fbbedb57196c1
c23e35e9f1ee6220
7de812970c2b370a
9980042fe8296cdd
8ca66d6911ebb2f3
b25509597cb781ef
56ac07d17c5dd090
3bb94c67c07e1000
79f59db33363fe11
0f435838ef482e60
b527f5532ee435e4
c05254393965d3db
e72e7c929bb4172b
3bda727006c33578
682cb21223ac2cf3
89a4fbabbb8cb55e
```

The following is an example revocation for a zone:

```
Zone private key (d, little-endian scalar, with ztype prepended):
00010000a065bf68
07cb3d90d10394a9
a56693e07087ad35
24f8e303931d4ade
946dc447

Zone identifier (zid):
00010000d06ab6d9
14e8a8064609b2b3
cb661c586042adcb
0dc5faeb61994d25
5ebdca72

Encoded zone identifier (zkl = zTLD):
000G006GDAVDJ578N034C2DJPF5PC72RC11AVJRDRQXEPRCS9MJNXFEAE8

Difficulty (5 base difficulty + 2 epochs): 7

Proof:
0005b13f536e2b0e
0000395d1827c000
5caaeaa2b955d82c
5caaeaa2b955da02
5caaeaa2b955daf0
5caaeaa2b955db20
5caaeaa2b955db2d
5caaeaa2b955dba1
5caaeaa2b955dba9
5caaeaa2b955dbc2
5caaeaa2b955dbc8
5caaeaa2b955dbd1
5caaeaa2b955dbf7
5caaeaa2b955dc0e
5caaeaa2b955dc54
5caaeaa2b955dc8c
5caaeaa2b955dca5
5caaeaa2b955dcb5
5caaeaa2b955dcf8
5caaeaa2b955dd47
5caaeaa2b955dd91
5caaeaa2b955dd98
5caaeaa2b955dd99
5caaeaa2b955ddc4
5caaeaa2b955de7f
5caaeaa2b955de80
5caaeaa2b955de92
5caaeaa2b955ded3
```

```
5caaeaa2b955df1a
5caaeaa2b955df77
5caaeaa2b955dfdf
5caaeaa2b955e06e
5caaeaa2b955e08d
5caaeaa2b955e0c4
00010000d06ab6d9
14e8a8064609b2b3
cb661c586042adcb
0dc5faeb61994d25
5ebdca7206b11f93
41f4e1649976c421
b1efe668a44becbe
5a9f76804adb6f6e
2cd16de00d81841d
cbd135aacad3bdab
3f2209bd10d55cc1
c7aed9a9bd53a1f6
cae1789d
```

14. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/info/rfc2782>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC3686] Housley, R., "Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)", RFC 3686, DOI 10.17487/RFC3686, January 2004, <https://www.rfc-editor.org/info/rfc3686>.

[RFC3826] Blumenthal, U., Maino, F., and K. McCloghrie, "The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model", RFC 3826, DOI 10.17487/RFC3826, June 2004, <https://www.rfc-editor.org/info/rfc3826>.

[RFC3912] Daigle, L., "WHOIS Protocol Specification", RFC 3912, DOI 10.17487/RFC3912, September 2004, <https://www.rfc-editor.org/info/rfc3912>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <https://www.rfc-editor.org/info/rfc5891>.

[RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, DOI 10.17487/RFC6781, December 2012, <https://www.rfc-editor.org/info/rfc6781>.

[RFC6895] Eastlake 3rd, D., "Domain Name System (DNS) IANA Considerations", BCP 42, RFC 6895, DOI 10.17487/RFC6895, April 2013, <https://www.rfc-editor.org/info/rfc6895>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[GANA] GNUnet e.V., "GNUnet Assigned Numbers Authority (GANA)", April 2020, <https://gana.gnunet.org/>.

[GNS] Wachs, M., Schanzenbach, M., and C. Grothoff, "A Censorship-Resistant, Privacy-Enhancing and Fully Decentralized Name System", 2014, <https://doi.org/10.1007/978-3-319-12280-9_9>.

[R5N] Evans, N. S. and C. Grothoff, "R5N: Randomized recursive routing for restricted-route networks", 2011, <https://doi.org/10.1109/ICNSS.2011.6060022>.

[Argon2] Biryukov, A., Dinu, D., Khovratovich, D., and S. Josefsson, "The memory-hard Argon2 password hash and proof-of-work function", March 2020, <https://datatracker.ietf.org/doc/draft-irtf-cfrg-argon2/>.

[MODES] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Methods and Techniques", December 2001, <https://doi.org/10.6028/NIST.SP.800-38A>.

[GCM] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", November 2007, <https://doi.org/10.6028/NIST.SP.800-38D>.

[CrockfordB32] Douglas, D., "Base32", March 2019, <https://www.crockford.com/base32.html>.

[Tor224] Goulet, D., Kadianakis, G., and N. Mathewson, "Next-Generation Hidden Services in Tor", November 2013, <https://gitweb.torproject.org/torspec.git/tree/proposals/224-rend-spec-ng.txt#n2135>.

[ed25519] Bernstein, D., Duif, N., Lange, T., Schwabe, P., and B. Yang, "High-Speed High-Security Signatures", 2011, <http://link.springer.com/chapter/10.1007/978-3-642-23951-9_9>.

Authors' Addresses

Martin Schanzenbach
GNUnet e.V.
Boltzmannstrasse 3
85748 Garching
Germany

Email: schanzen@gnunet.org

Christian Grothoff
Berner Fachhochschule
Hoeheweg 80
CH-2501 Biel/Bienne
Switzerland

Email: grothoff@gnunet.org

Bernd Fix
GNUnet e.V.
Boltzmannstrasse 3
85748 Garching
Germany

Email: fix@gnunet.org