DNSOP Working Group
Internet-Draft
Intended status: Standards Track
Expires: 3 September 2026

J. Stenstam
E. Bergstr旦m
L. Fernandez
The Swedish Internet Foundation

2 March 2026

Automating DNS Delegation Management via DDNS
draft-ietf-dnsop-delegation-mgmt-via-ddns-01

Abstract

Delegation information (i.e. the NS RRset, possible glue, possible DS records) should always be kept in sync between child zone and parent zone. However, in practice that is not always the case.

When the delegation information is not in sync the child zone is usually working fine, but without the amount of redundancy that the zone owner likely expects to have. Hence, should any further problems ensue it could have catastrophic consequences.

The DNS name space has lived with this problem for decades and it never goes away. Or, rather, it will never go away until a fully automated mechanism for how to keep the information in sync automatically is deployed.

This document proposes such a mechanism based on DNS Dynamic Updates (DDNS) secured with SIG(0) signatures, sent from the child to the parent across the zone cut. The target of the update is discovered via the DSYNC record defined in [RFC9859].

TO BE REMOVED: This document is being collaborated on in Github at: https://github.com/johanix/draft-ietf-dnsop-delegation-mgmt-via-ddns (https://github.com/johanix/draft-ietf-dnsop-delegation-mgmt-via-ddns). The most recent working version of the document, open issues, etc, should all be available there. The authors (gratefully) accept pull requests.

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 3 September 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

For DNSSEC signed child zones with TLD registries as parents there is an emerging trend towards running so-called "CDS scanners" and "CSYNC scanners" by the parent.

These scanners detect publication of CDS records (the child signalling a desire for an update to the DS RRset in the parent) and/or a CSYNC record (the child signalling a desire for an update to the NS RRset or, possibly, in-bailiwick glue in the parent).

The scanners have a number of drawbacks, including being inefficient and slow. They are only applicable to DNSSEC-signed child zones and they add significant complexity to the parent operations. But given that, they do work.

[RFC9859] proposes a method to alleviate the inefficiency and slowness of scanners. But the DNSSEC requirement and the complexity remain.

This document proposes an alternative mechanism to automate the updating of delegation information in the parent zone for a child zone based on DNS Dynamic Updates secured with SIG(0) signatures.

This alternative mechanism shares the property of being efficient and provides rapid convergence (similar to generalized notifications in conjunction with scanners). Furthermore, it has the advantages of not requiring any scanners in the parent at all and also not being dependent on the child (and parent) being DNSSEC-signed.

Knowledge of DNS NOTIFY [RFC1996] and DNS Dynamic Updates [RFC2136] and [RFC3007] is assumed. DNS SIG(0) transaction signatures are documented in [RFC2931]. In addition this Internet-Draft borrows heavily from the thoughts and problem statement from [RFC9859].

1.1. Requirements Notation

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.

1.2. Terminology

SIG(0) An asymmetric signing algorithm that allows the recipient to only need to know the public key to verify a signature created by the sender’s private key.

UPDATE Receiver The entity that receives and processes DNS UPDATE messages on behalf of the parent zone. This may be the parent primary name server or a separate dedicated service.

Bootstrap The process of establishing initial trust in a SIG(0) public key. A key that has been received but not yet validated is "known"; once validated it is promoted to "trusted".

2. Is there a Use Case?

Because of the drawbacks of CDS and CSYNC scanners they are unlikely to be able to fully automate the maintenance of delegation information in all parent zones. The primary reasons are the hard requirement on DNSSEC in the child zones and the complexity cost of operating the scanner infrastructure. In practice, scanners are likely mostly realistic for parent zones that are operated by well-resourced registries.

All the parts of the DNS name space where the parent is smaller and more resource constrained would be able to automate the delegation management via this mechanism without the requirement of operating scanners. Also all parts of the name space where there are child zones that are not DNSSEC-signed would be able to use this.

Obviously, also well-resourced parent zones with DNSSEC-signed child zones would be able to use this DNS UPDATE-based mechanism, but in those cases scanners plus generalized notifications would also work.

3. DNS NOTIFY versus DNS UPDATE

DNS NOTIFY and DNS UPDATE messages share several properties and are used to address similar issues.

3.1. Similarities between NOTIFY and UPDATE

Both NOTIFY and UPDATE are "push" rather than "pull" messages and therefore very efficient.

Both NOTIFY and UPDATE are messages, in DNS packet format. They are used by one party (the sender) to inform another party (the recipient) that some piece of DNS information has changed and that as a consequence of this change the recipient of the message may want to also make a change to its DNS data.

A NOTIFY (as per [RFC1996]) is only a hint and the recipient may ignore it. But if the recipient does listen to the NOTIFY it should make its own lookups to verify what has changed and whether that should trigger any changes in the DNS data provided by the recipient.

3.2. Differences between NOTIFY and UPDATE

The difference between the UPDATE and the NOTIFY is that the UPDATE contains the exact change that should (in the opinion of the sender) be applied to the recipient’s DNS data. Furthermore, for secure Dynamic Updates, the message also contains proof why the update should be trusted (in the form of a digital signature by a key that the recipient trusts).

In this document the term "Dynamic Update" or "DNS UPDATE" implies secure dynamic update. Furthermore this document implies that the signature algorithms are always based on asymmetric crypto keys, using the same algorithms as are being used for DNSSEC. I.e. by using the right algorithm the resulting signatures will be as strong as DNSSEC-signatures.

DNS UPDATEs can be used to update any information in a zone (subject to the policy of the recipient). But in the special case where the data that is updated is the delegation information for a child zone and it is sent across a zone cut (i.e. the child sends it to the parent), it acts as a glorified generalized NOTIFY.

The DNS UPDATE in this case is essentially a message that says:

"the delegation information for this child zone has changed; here is the exact change; here is the proof that the change is authentic, please verify this signature"

4. Updating Delegation Information via DNS UPDATEs

This is not a new idea. There is lots of prior art and prior documents, including the expired I-D.andrews-dnsop-update-parent-zones-04.

The functionality to update delegation information in the parent zone via DNS UPDATE has been available for years in at least one DNS implementation (BIND9). However, while DNS UPDATE is used extensively inside organisations it has not seen much use across organisational boundaries. And zone cuts, almost by definition, usually cut across such boundaries.

When sending a DNS UPDATE it is necessary to know where to send it. Inside an organisation this information is usually readily available. But outside the organisation it is not. And even if the sender would know where to send the update, it is not at all clear that the destination is reachable to the sender (the parent primary is likely to be protected by firewalls and other measures).

This constitutes a problem for using DNS UPDATES across zone cuts.

Another concern is that traditionally DNS UPDATEs are sent to a primary nameserver, and if the update signature verifies the update is automatically applied to the DNS zone. This is often not an acceptable mechanism. The recipient may, for good reason, require additional policy checks and likely an audit trail. Finally, the change should in many cases not be applied to the running zone but rather to some sort of provisioning system or a database.

This creates another problem for using DNS UPDATEs for managing delegation information.

Both problems are addressed by the target location mechanism defined in [RFC9859], described in the next section.

5. Locating the Target for a generalized NOTIFY and/or DNS UPDATE

Section 3 of [RFC9859] defines a new RR type, DSYNC, that has the following format:

```
{qname}   IN  DSYNC  {RRtype} {scheme} {port} {target}
```

where {target} is the domain name of the recipient of the NOTIFY message. {RRtype} is typically "CDS" or "CSYNC" in the case where delegation information should be updated (there are also other uses of generalized notifications).

Finally, {scheme} is an 8 bit unsigned integer to indicate the type of notification mechanism to use. Scheme=1 is defined as "NOTIFY" (and the presentation format is also "NOTIFY"), as in "send a generalized NOTIFY to {target} on port {port}".

This document proposes the definition of a new {scheme} for the same record that is used for generalized NOTIFY. Scheme=TBD is here defined as "UPDATE", as in "send a DNS UPDATE to {target} on port {port}". When parsing or presenting DNS zone data the 8 bit unsigned integer TBD should be replaced by the string "UPDATE", as in the examples below.

Apart from defining a new scheme to specify the mechanism "UPDATE" (rather than the mechanism "NOTIFY") this document does not say anything about what Qname to look up or what RR type. The UPDATE mechanism should use exactly the same method of locating the target of the UPDATE as is used for generalized NOTIFY.

This lookup addresses the first issue with using DNS UPDATE across organizational boundaries.

Example 1: a parent zone announces support for DNS UPDATE as a mechanism for delegation synchronization for all child zones:

```
_dsync.parent.  IN DSYNC ANY UPDATE 5302 ddns-receiver.parent.
```

Example 2: a parent zone announces support for different DNS UPDATE targets on a per-child basis:

```
childA._dsync.parent.  IN DSYNC ANY UPDATE 5302 ddns-receiver.registrar1.
childB._dsync.parent.  IN DSYNC ANY UPDATE 5302 ddns-receiver.registrar3.
childC._dsync.parent.  IN DSYNC ANY UPDATE 5302 ddns-receiver.registrar2.
```

The DSYNC RRset is looked up, typically by the child primary name server or by a separate agent for the child, at the time that the delegation information for the child zone changes in some way that would prompt an update in the parent zone. When the {scheme} is "UPDATE" (i.e. the number 2 in the wire protocol) the interpretation is:

Send a DNS UPDATE to the IP address for the name {target} on port 5302, where {target} is the domain name in the right-hand side of the DSYNC record that matches the qname in the DNS query.

6. Limitation of Scope for the Proposed Mechanism

DNS UPDATE is in wide use all over the world, for all sorts of purposes. It is not in wide use across organizational boundaries. This document only addresses the specific case of a child zone that makes a change in its DNS delegation information that will require an update of the corresponding information in the parent zone. This includes:

* changes to the NS RRset
* changes to glue (if any)
* changes to the DS RRset (if any)
* changes to the KEY RRset (only used for validation of UPDATE messages)

Only for those specific cases is the described mechanism proposed.

7. The DNS UPDATE Receiver

While the simplest design is to send the DNS UPDATEs to the primary name server of the parent it will in most cases be more interesting to send them to a separate UPDATE Receiver. To separate the primary name server from the UPDATE Receiver, use a {target} with addresses separate from the addresses of the primary name server.

7.1. Processing the UPDATE in the DNS UPDATE Receiver

The receiver of the DNS UPDATE messages should implement a suitably strict policy for what updates are accepted (typically only allowing updates to the NS RRset, glue and DS RRset).

Furthermore, it is strongly recommended that the policy is further tightened by only allowing updates to the delegation information of a child zone with the exact same name as the name of the SIG(0) key that signed the UPDATE request. I.e. an UPDATE request for the delegation information for the zone child.parent. should only be processed if it is signed by a SIG(0) key with the name child.parent. and the signature verifies correctly.

Once the DNS UPDATE message has been verified to be correctly signed by a known and trusted key with the correct name and also adhere to the update policy it should be subjected to the same set of correctness tests as a CDS/CSYNC scanner would have performed. If these requirements are also fulfilled the change may be applied to the parent zone in whatever manner the parent zone is maintained (as a text file, data in a database, via an API, etc).

8. Interpretation of the response to the DNS UPDATE

All DNS transactions are designed as a pair of messages and this is true also for DNS UPDATE. The interpretation of the different responses to DNS UPDATE are fully documented in [RFC2136], section 2.2.

8.1. RCODE NOERROR

A response with rcode=0 ("NOERROR") should be interpreted as a confirmation that the DNS UPDATE has been received and accepted. I.e. the change to the parent DNS data should be expected to be published in the parent zone at some future time.

8.2. RCODE REFUSED

A response with rcode=5 ("REFUSED") should be interpreted as a permanent signal that DNS UPDATEs are not supported by the receiver. This would indicate a parent misconfiguration, as the UPDATE should not be sent unless the parent has announced support for DNS UPDATE via publication of an appropriate target location record.

8.3. RCODE BADKEY

A response with rcode=17 ("BADKEY") should be interpreted as a definitive statement that the DNS UPDATE Receiver does not have access to the public SIG(0) key needed for signature verification. In this case the child should fall back to bootstrap of the SIG(0) public key into the DNS UPDATE Receiver, see below.

8.4. No response to a DNS UPDATE

The case of no response is more complex, as it is not possible to know whether the DNS UPDATE actually reached the receiver (or was lost in transit) or the response was not sent (or lost in transit).

For this reason it is suggested that a lack of response is left as implementation dependent. That way the implementation has sufficient freedom to choose a sensible approach. Eg. if the sender of the DNS UPDATE message (perhaps the primary name server of the child zone)

only serves a single child, then resending the DNS UPDATE once or twice may be ok (to ensure that the lack of response is not due to packets being lost in transit). On the other hand, if the sender serves a large number of child zones below the same parent zone, then it may already know that the receiver for the DNS UPDATEs is not responding for any of the child zones, and then resending the update immediately is likely pointless.

9. Management of SIG(0) Public Keys

Only the child should have access to the SIG(0) private key. The corresponding SIG(0) public key should preferably be published in DNS, but it doesn't have to be. The SIG(0) public key only needs to be available to the parent DNS UPDATE Receiver. Keeping all the public SIG(0) keys for different child zones in some sort of database is perfectly fine.

9.1. Bootstrapping the SIG(0) Public Key Into the DNS UPDATE Receiver

Bootstrap of the child public SIG(0) key to be trusted by the UPDATE Receiver may be done either manually or automatically. Manually may in various cases be the preferred method, especially in the case of non-registry parents with a small number of child delegations.

If the UPDATE Receiver only supports manual bootstrap, then that is what will happen (apart from informing the child about this policy). If the child wants to enforce manual bootstrap it needs to request this from the UPDATE Receiver.

In those cases there is by definition some mechanism in place to communicate information from the child to the parent, be it email, a web form, pieces of paper or something else. The same mechanism can be extended to also be used to communicate the initial SIG(0) public key from the child to the parent.

Regardless of whether manual bootstrap or automatic bootstrap is to be used (subject to the UPDATE Receiver policy), the bootstrap must be initiated. This is done by the child issuing a self-signed DNS UPDATE to the parent containing:

```
DEL child.parent. {ttl} ANY KEY
ADD child.parent. {ttl} IN  KEY ...
```

The first record is an instruction to delete any previous keys for this child (CLASS=ANY in a DNS UPDATE is a request to delete an entire RRset). The second is an instruction to add the new key.

When receiving such a message (where the self-signature validates) the parent UPDATE Receiver SHOULD mark that key as "known" (but not yet trusted) and then either put that key into a queue for later look up and validation of the corresponding KEY record (if supporting automatic bootstrap) or put that key into a queue for subsequent manual validation and verification.

9.1.1. Automatic Bootstrap of the SIG(0) Public Key

Automated bootstrapping is also possible, subject to the policy of the UPDATE Receiver. The basic idea is to publish the public SIG(0) key as a KEY record either at the child apex or at a special name below the name of an authoritative nameserver for the zone located in a DNSSEC- signed zone. This publication must occur prior to the initial key upload. Then the UPDATE Receiver (or an agent) may look that KEY up for subsequent validation.

9.1.2. Automatic Bootstrap When Child Zone Is DNSSEC-signed

If the child zone is DNSSEC-signed (including a signed delegation via a DS record), then the KEY record containing the SIG(0) public key located at the zone apex can be looked up and validated by the DNS UPDATE Receiver.

```
child.parent. IN KEY ...
child.parent. IN RRSIG KEY ...
```

In case of validation success the key SHOULD be promoted to "trusted" by the UPDATE Receiver. At this point any old keys should be deleted.

In case of validation failure (or absence of a DNSSEC signature) the SIG(0) SHOULD NOT be promoted to the set of keys that the UPDATE Receiver trusts and any old keys MUST be kept.

9.1.3. Automatic Bootstrap When Child Nameserver Is In A DNSSEC-signed Zone

If the child zone is unsigned but at least one of the authoritative nameservers for the zone is located in a DNSSEC-signed zone (including a signed delegation via a DS record), then the KEY record containing the SIG(0) public key can be looked up and validated by the DNS UPDATE Receiver.

Example: assume that the unsigned child zone "child.parent." has two nameservers:

```
child.parent.  IN NS ns.child.parent.
child.parent.  IN NS ns.provider.example.
```

ns.child.parent. is located in the unsigned zone child.parent. and a KEY record located under that name can not be DNSSEC-validated. ns.provider.example. on the other hand is located in a DNSSEC signed zone. In this case the KEY record containing the public SIG(0) key for child.parent. may be located at the special label "_sig0key.{child}._signal.{nameserver}":

```
_sig0key.child.parent._signal.ns.provider.example. IN KEY ...
_sig0key.child.parent._signal.ns.provider.example. IN RRSIG KEY ...
```

In case of validation success the key SHOULD be promoted to "trusted" by the UPDATE Receiver. At this point any old keys should be deleted.

In case of validation failure (or absence of a DNSSEC signature) the SIG(0) SHOULD NOT be promoted to the set of keys that the UPDATE Receiver trusts and any old keys MUST be kept.

This bootstrap mechanism is identical to the method used for DNSSEC bootstrap as described in [RFC9615]. As the proof of the SIG(0) key being authentic is based on a clear DNSSEC signature chain this method is as secure as if the child zone had been signed.

9.1.4. Automatic Bootstrap When Child Zone Is Unsigned

Hence, to bootstrap the public SIG(0) key for a child zone it is possible for the parent to use a "bootstrap policy" a la:

* Look up the KEY RRset in the child zone. Compare to the child KEY received in the self-signed DNS UPDATE.

* To mitigate a possible on-path attacker, do the look up of the child KEY RRset, using TCP transport, from multiple vantage points, at multiple times. The child KEY RRset must be consistent over time and space.

* Make on-path attacks even more difficult by adding a requirement to, in addition to TCP, also use a more secure transport, like DoT, DoH or DoQ once those become more generally available for DNS queries to authoritative servers. The child KEY RRset must be consistent over all tested transports.

* If the received KEY RRset is consistent from multiple vantage points and multiple times then it is considered authentic and promoted by the parent's UPDATE Receiver from "known" to "trusted" SIG(0) key for the child. At this point any old SIG(0) public keys for the child should be deleted.

Should a "registry" parent want to support this mechanism (as a service to its unsigned children) then a likely model is that the target of the DNS UPDATE is operated by the registrar (or possibly that the DNS UPDATE is forwarded to the registrar). The registrar performs its normal verifications of a change and then transforms the update into an EPP [RFC5730] transaction to communicate it to the registry.

This bootstrap mechanism for the case of an unsigned child is essentially identical to the "automatic DNSSEC Bootstrap via CDS" service a la [RFC8078] that multiple TLD registries provide today. It does not provide the provable authenticity of the two previous methods but it is possible to make on-path attacks arbitrarily difficult by using a larger number of repeated queries from a larger number of vantage points.

9.1.5. Publishing Supported Bootstrap Methods

As there are multiple possible methods to bootstrap the initial SIG(0) public key to become trusted by the parent it becomes important for child zone operators to be able to find out which method to use.

This document utilizes the same mechanism as used in [I-D.berra-dnsop-announce-scanner] to announce details about CDS and CSYNC scanner capabilities. This mechanism uses an SVCB record located at the target of the DSYNC record to announce capabilities. For the UPDATE Receiver the important capabilities are primarily supported bootstrap methods.

9.1.5.1. SvcParamKey "bootstrap"

The "bootstrap" SvcParamKey in the SVCB record is used to signal what mechanisms are supported for bootstrapping the trust of the child's public SIG(0) key to the UPDATE Receiver. Four mechanisms are currently identified:

* "at-apex": This is an indication that the UPDATE Receiver supports automatic bootstrap of the SIG(0) public key for signed child zones by the child publishing the DNSSEC-signed KEY record at the child zone apex.

```
child.parent.  IN KEY ...
child.parent.  IN RRSIG KEY ...
```

See Automatic Bootstrap When Child Zone Is DNSSEC-signed (Section 9.1.2)

* "at-ns": This is an indication that the UPDATE Receiver supports automatic bootstrap of the SIG(0) public key by publishing the DNSSEC-signed KEY record both at the zone apex AND at the special name "_sig0key.{child}._signal.{nameserver}." below the name of an authoritative nameserver located in a signed zone. This mechanism is similar to what is described in [RFC9615] for CDS bootstrap:

```
child.parent.     IN NS ns.provider.com.
child.parent.     IN KEY ...
_sig0key.{child}._signal.ns.provider.com.  IN KEY ...
_sig0key.{child}._signal.ns.provider.com.  IN RRSIG KEY ...
```

See Automatic Bootstrap When Child Nameserver Is In A DNSSEC-signed Zone (Section 9.1.3)

* "unsigned": This method is similar to what [RFC8078] describes for CDS bootstrapping.

```
child.parent.     IN KEY ...
```

See Automatic Bootstrap When Child Zone Is Unsigned (Section 9.1.4)

* "manual": This is an indication that the UPDATE Receiver supports some other mechanism for bootstrap of the SIG(0) public key.

The value of the "bootstrap" key is a string with one or more of the supported mechanisms, separated by ",". The mechanisms may occur in arbitrary order.

New mechanisms are likely to be defined in the future.

9.1.5.2. Complete Example

Example for a parent that operates an UPDATE Receiver that only supports the secure automatic bootstrap methods "at-apex" and "at-ns". Otherwise "manual" bootstrap is needed.

```
_dsync.parent.example.   IN  DSYNC ANY UPDATE 0 updater.parent.example.
updater.parent.example.  IN  SVCB 0 . (bootstrap="at-apex,at-ns,manual")
```

I.e. this UPDATE Receiver does not support the "unsigned" method based on [RFC8078].

9.2. Communication Between Child and Parent UPDATE Receiver

There are two cases where communication between child and parent UPDATE Receiver would benefit greatly from some additional information.

9.2.1. Communication in Case of Errors

An error response from the parent UPDATE Receiver would be improved by more detail provided via a set of new Extended DNS Error Codes [RFC8914]. In particular, it would be useful to be able to express the following "states":

* "SIG(0) key is known, but not yet trusted": indicating that bootstrap of the key is not yet complete. Waiting may resolve the issue.

* "SIG(0) key is known, but validation failed": indicating that bootstrap has failed and waiting will not resolve the issue.

* "Automatic bootstrap of SIG(0) keys not supported; manual bootstrap required": indicating that while the parent does support delegation synchronization via DNS UPDATE, it only supports manual bootstrap. Note that the child should normally discover this via the SVCB "bootstrap" SvcParamKey before attempting automatic bootstrap; this error serves as a fallback for cases where that discovery was not performed or the SVCB record was not available.

9.2.2. Communication To Inquire State

Extended DNS Errors [RFC8914] provides an excellent mechanism for adding more detail to error responses. However it is, intentionally, limited to:

* returning extra information from the receiver to the original sender. It is not possible to "send" information, only "return" information.

* no information except the actual error code is meant for automatic processing. It is therefore not possible to communicate things like, eg. a KeyId via Extended DNS Errors.

The communication between child and parent would benefit from the addition of the ability to also send inquiries to the parent. For example, the child should be able to inquire about key state: "I operate under the assumption that the key {key} is known and trusted by you (the parent). Is this correct?"

[I-D.berra-dnsop-keystate] is proposed as a mechanism to improve the communication between child and parent, both in the error case and in the inquiry case. If that draft is supported then the above example would travel as "KeyState codes" in a KeyState OPT as specified in [I-D.berra-dnsop-keystate].

9.3. Mutual Authentication and the UPDATE Receiver SIG(0) Key

For the KeyState communication described above to be trustworthy, the child must be able to verify that responses from the UPDATE Receiver are authentic. Otherwise an adversary could potentially cause disruption by sending forged responses, e.g. falsely claiming that the child's key is unknown and triggering an unnecessary re-bootstrap.

For this reason the UPDATE Receiver SHOULD also maintain a SIG(0) key pair and use its private key to sign responses to the child. This enables mutual authentication: the child authenticates to the parent via its SIG(0) signature on the UPDATE, and the parent authenticates to the child via its SIG(0) signature on the response.

9.3.1. Publishing the UPDATE Receiver SIG(0) Key

The UPDATE Receiver SHOULD publish its public SIG(0) key as a KEY record at the domain name that is the {target} of the DSYNC record. Example:

```
_dsync.parent.example.      IN DSYNC ANY UPDATE 0 updater.parent.example.
updater.parent.example.     IN KEY ...
```

9.3.2. Bootstrapping the UPDATE Receiver Key to the Child

The child needs to acquire and validate the UPDATE Receiver's public SIG(0) key before it can verify signed responses. Two methods are defined:

* If the KEY record for the UPDATE Receiver is located in a DNSSEC-signed zone (which is expected in the common case where the parent zone is DNSSEC-signed), the child looks up the KEY record and performs standard DNSSEC validation. On validation success the key is trusted.

* If the KEY record is not in a DNSSEC-signed zone, manual bootstrap is required. The same out-of-band mechanisms available for bootstrapping the child key (email, web form, etc.) may be used in reverse.

9.4. Rolling the SIG(0) Key

Once the parent (or registrar) DNS UPDATE Receiver has the key, the child can update it via a DNS UPDATE just like updating the NS RRset, the DS RRset or the glue in the parent zone (assuming a suitable DNS UPDATE policy in the parent). I.e. only the initial bootstrapping of the key is an issue.

Note, however, that the alternative of re-bootstrapping (by whatever bootstrapping mechanism was used) in case of a key compromise may be a better alternative to the parent supporting key rollover for child SIG(0) keys. The decision of whether to allow SIG(0) key rollover via DNS UPDATE is left as parent-side policy.

9.5. Re-bootstrapping In Case of Errors

Sometimes things get out of sync in spite of all efforts. It could be an operator error in the parent end losing the child public key or it could be an error in the child end losing the private key. Another possibility could be a key rollover that for some reason didn’t sync correctly.

If [I-D.berra-dnsop-keystate] is supported then the child can inquire with the UPDATE Receiver about the KeyState of the SIG(0) key it would use in an UPDATE request.

In all such cases, as soon as the child becomes aware of the problem it should simply re-bootstrap by the same mechanism as used initially. The self-signed DNS UPDATE that starts the bootstrapping process contains a

DEL child.parent. {ttl} ANY KEY

and that is an instruction to the parent UPDATE Receiver to delete any SIG(0) public keys for this child (and after that start the process to validate the new key).

Note that when receiving such a self-signed DNS UPDATE the parent MUST NOT delete any old keys until the new key has been validated and therefore promoted to "trusted". This is needed and important to avoid the potential attack vector of an adversary causing a parent to invalidate the child key by just sending a self-signed bootstrap UPDATE (which will not validate, but if the old key is deleted then the harm would already be done).

10. Scalability Considerations

The primary existing mechanism for automatic synchronization of DNS delegation information is based on parent-side "scanning" of the child zones for CDS and/or CSYNC RRsets, performing DNSSEC validation on the result and then, in the CSYNC case, based on the result, issue and validate a potentially large number of additional DNS queries, all of which must be DNSSEC validated. This makes a CDS/CSYNC scanner for a parent with a significant number of delegations a complex and resource consuming service. Among the issues are rate-limiting by large DNS operators and inherent difficulties in issuing millions of recursive DNS queries where all received data must be validated.

By comparison, the DNS UPDATE based mechanism for automatic synchronization shifts most of the effort to the child side. It is the child that is responsible for detecting the need to update the delegation information in the parent zone (which makes sense as it is the child that has made a change and therefore, in many cases, already "knows"). It is the child rather than the parent that computes what records should be added or removed. All of this is good for scalability.

Furthermore, the information collection and validation effort for the UPDATE Receiver is restricted to validation of a single DNS message, using a SIG(0) key that the UPDATE Receiver already has.

Hence, as the data collection and validation is much simplified the task of the UPDATE Receiver is mostly focused on the policy issues of whether to approve the UPDATE or not (i.e. the same process that a CDS and/or CSYNC scanner follows).

11. Security Considerations

Any fully automatic mechanism to update the contents of a DNS zone opens up a potential vulnerability should the mechanism not be implemented correctly.

In this case the definition of "correct" is a question for the receiver of the DNS UPDATE. The receiver should verify the authenticity of the DNS UPDATE and then do the same checks and verifications as a CDS or CSYNC scanner does. The difference from the scanner is only in the validation: single SIG(0) signature by a key that the UPDATE Receiver trusts vs DNSSEC signatures that chain back to a DNSSEC trust anchor that the validator trusts.

Another issue of concern is whether a parent-side service that provides support for changes to child delegation information via DNS UPDATE is open for potential denial-of-service attacks. The answer is likely no, as it is possible to have a very strict rate-limiting policy based on the observation that no child zone should have a legitimate need to change its delegation information frequently.

Furthermore, as the location of the UPDATE Receiver can be separated from any parent name server even in the worst case the only service that can be subject to an attack is the UPDATE Receiver itself, which is a service that previously did not exist.

12. IANA Considerations

IANA is requested to assign a new "scheme" value to the registry for "DSYNC Location of Synchronization Endpoints" as follows:

Reference (this document)

RRtype	Scheme	Purpose	Reference
ANY	TBD	Delegation management	(this document)

Table 1

13. Acknowledgements

* Peter Thomassen and I together came up with the location mechanism for the generalized notifications, which this draft relies upon.

* Mark Andrews provided the initial inspiration for writing some code to experiment with the combination of the location mechanism from the generalised notifications with DNS UPDATEs across zone cuts.

* Stefan Ubbink helped me realize the need to also cater for the case of re-bootstrapping if or when things got out of sync for some reason.

14. References

14.1. Normative References

[RFC1996] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, DOI 10.17487/RFC1996, August 1996, <https://www.rfc-editor.org/rfc/rfc1996>.

[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/rfc/rfc2119>.

[RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <https://www.rfc-editor.org/rfc/rfc2136>.

[RFC2931] Eastlake 3rd, D., "DNS Request and Transaction Signatures (SIG(0)s)", RFC 2931, DOI 10.17487/RFC2931, September 2000, <https://www.rfc-editor.org/rfc/rfc2931>.

[RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", RFC 3007, DOI 10.17487/RFC3007, November 2000, <https://www.rfc-editor.org/rfc/rfc3007>.

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, RFC 5730, DOI 10.17487/RFC5730, August 2009, <https://www.rfc-editor.org/rfc/rfc5730>.

[RFC8078] Gudmundsson, O. and P. Wouters, "Managing DS Records from the Parent via CDS/CDNSKEY", RFC 8078, DOI 10.17487/RFC8078, March 2017, <https://www.rfc-editor.org/rfc/rfc8078>.

[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/rfc/rfc8174>.

[RFC8914] Kumari, W., Hunt, E., Arends, R., Hardaker, W., and D. Lawrence, "Extended DNS Errors", RFC 8914, DOI 10.17487/RFC8914, October 2020, <https://www.rfc-editor.org/rfc/rfc8914>.

[RFC9615] Thomassen, P. and N. Wisiol, "Automatic DNSSEC Bootstrapping Using Authenticated Signals from the Zone's Operator", RFC 9615, DOI 10.17487/RFC9615, July 2024, <https://www.rfc-editor.org/rfc/rfc9615>.

[RFC9859] Stenstam, J., Thomassen, P., and J. Levine, "Generalized DNS Notifications", RFC 9859, DOI 10.17487/RFC9859, September 2025, <https://www.rfc-editor.org/rfc/rfc9859>.

14.2. Informative References

[I-D.berra-dnsop-announce-scanner]
Bergstr旦m, E., Stenstam, J., and L. Fernandez, "Announce Existence of Parent CDS/CSYNC Scanner", Work in Progress, Internet-Draft, draft-berra-dnsop-announce-scanner-03, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-berra-dnsop-announce-scanner-03>.

[I-D.berra-dnsop-keystate]
Bergstr旦m, E., Fernandez, L., and J. Stenstam, "Signalling Key State Via DNS EDNS(0) OPT", Work in Progress, Internet-Draft, draft-berra-dnsop-keystate-02, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-berra-dnsop-keystate-02>.

Appendix A. Change History (to be removed before publication)

* draft-ietf-dnsop-delegation-mgmt-via-ddns-01

  Large number of editorial nits addressed (thanks to Yorgos Thessalonikefs)

  Replaced the hand-waving "Mutual Authentication" section with a concrete specification for where the UPDATE Receiver publishes its SIG(0) public key and how the child validates it (DNSSEC validation or manual bootstrap).

  Moved "Communication Between Child and Parent UPDATE Receiver" and "Mutual Authentication" to directly after the child key bootstrap sections, so that the motivation (KeyState communication) directly precedes the solution (mutual authentication via the parent's SIG(0) key).

  Removed the policy inquiry references that were made obsolete by the SVCB "bootstrap" SvcParamKey mechanism.

  Moved Terminology to the Introduction and added definitions for UPDATE Receiver and Bootstrap.

Expanded the abstract to describe the proposed mechanism.

* draft-ietf-dnsop-delegation-mgmt-via-ddns-00

  Re-submitted as a WG document after adoption.

* draft-johani-dnsop-delegation-mgmt-via-ddns-06

  Added the more secure bootstrap of the SIG(0) public key based on RFC9615 in addition to the original mechanism based on RFC8078.

  Replaced references to the draft on generalized notifications with a reference to RFC9859.

* draft-johani-dnsop-delegation-mgmt-via-ddns-05

  Fixed typos in the KeyState OPT section.

  Added a section on mutual authentication.

  Fixed spelling errors.

* draft-johani-dnsop-delegation-mgmt-via-ddns-04

  Reworked the section on automatic bootstrapping a la RFC8078.

  Added a section on re-bootstrapping the SIG(0) key with the parent after problems.

  Added text on the importance of augmenting error responses using EDE (RFC8914).

  Added text on the insufficiency of RFC8914 for child-to-parent communication and a reference to the (upcoming) draft on a new OPT code to alleviate this.

* draft-johani-dnsop-delegation-mgmt-via-ddns-03

  Update the draft based on the excellent dnsdir review of draft-ietf-dnsop-generalized-notify-01 by Patrick Mevsek.

  Expand the section on alternatives for initial bootstrap to suggest a mechanism similar to what is used for automatic bootstrap of DNSSEC signed delegations via multiple queries for child the CDS RRset.

  Added a section on scalability considerations.

Expanded the Security Considerations section with a paragraph on the potential for DDOS attacks aimed at the UPDATE Receiver.

* draft-johani-dnsop-delegation-mgmt-via-ddns-02

  Update the references to the (updated) I-D for generalized notifications.

  Remove duplicated descriptions between the two drafts. It is sufficient that the generalized notification draft describes the mechanics.

* draft-johani-dnsop-delegation-mgmt-via-ddns-01

  Change the RRtype from the original "NOTIFY" to the proposed "DSYNC"

  Expand the description of how to interpret different RCODE responses to the UPDATE. Expand the description of bootstrapping alternatives. Change the mnemonic of the RR type used from "NOTIFY" to "DSYNC" in the examples.

* draft-johani-dnsop-delegation-mgmt-via-ddns-00

  Initial public draft.

Authors' Addresses

Johan Stenstam
The Swedish Internet Foundation
Sweden
Email: johan.stenstam@internetstiftelsen.se

Erik Bergstr旦m
The Swedish Internet Foundation
Sweden
Email: erik.bergstrom@internetstiftelsen.se

Leon Fernandez
The Swedish Internet Foundation
Sweden
Email: leon.fernandez@internetstiftelsen.se