Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 22 January 2026

F. Denis
Individual Contributor

21 July 2025

The DNS Stamps Specification
draft-denis-dns-stamps-00

Abstract

This document specifies DNS Stamps, a compact format that encodes the information needed to connect to DNS resolvers. DNS Stamps encode all necessary parameters including addresses, hostnames, cryptographic keys, and protocol-specific configuration into a single string using a standard URI format. The specification supports multiple secure DNS protocols including DNSCrypt, DNS-over-HTTPS (DoH), DNS-over-TLS (DoT), DNS-over-QUIC (DoQ), and Oblivious DoH.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Domain Name System Working Group mailing list (namedroppers@nic.ddn.mil), which is archived at nicfs.nic.ddn.mil:~/namedroppers/*.Z.

Source for this draft and an issue tracker can be found at https://github.com/DNSCrypt/draft-denis-dns-stamps.

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 22 January 2026.

Table of Contents

1. Introduction

The Domain Name System (DNS) has evolved significantly from its original design as specified in [RFC1035]. While traditional DNS operates over unencrypted UDP and TCP connections on port 53, modern DNS deployments increasingly use encrypted transports to provide confidentiality and integrity. These secure protocols include DNSCrypt [I-D.draft-denis-dprive-dnscrypt], DNS-over-TLS (DoT) [RFC7858], DNS-over-HTTPS (DoH) [RFC8484], DNS-over-QUIC (DoQ) [RFC9250], and Oblivious DNS-over-HTTPS [ODOH].

Each secure DNS protocol requires different configuration parameters. DNSCrypt needs a provider public key and provider name in addition to server addresses. DoH requires HTTPS endpoints and paths. DoT and DoQ need TLS configuration including certificate validation parameters. This diversity in configuration requirements creates significant challenges for both users and applications attempting to configure secure DNS resolvers.

Current approaches to DNS configuration suffer from several limitations. Operating system interfaces typically support only IP addresses for DNS servers, providing no mechanism to specify encryption protocols or authentication parameters. Application-specific configuration files lack standardization, making it difficult to share configurations across different DNS client implementations. Manual configuration is error-prone, particularly when dealing with cryptographic parameters like public keys or certificate hashes. There is no standard way to distribute complete resolver configurations that would enable users to easily switch between different secure DNS providers.

DNS Stamps address these challenges by encoding all parameters required to connect to a DNS resolver into a single, compact string using a URI format. This approach enables simple sharing of resolver configurations through copy-paste, QR codes, or URLs. It provides a consistent format across different client implementations, reduces configuration errors through format validation, and supports multiple protocols through a unified specification. DNS Stamps have been implemented in numerous DNS client applications and are used by several public DNS resolver operators to publish their server configurations.

The remainder of this document is organized as follows. Section 2 establishes conventions and defines the encoding primitives used throughout the specification. Section 3 provides a high-level overview of the DNS Stamps format. Section 4 details the specific format for each supported protocol. Section 5 covers operational aspects including generation, parsing, and validation. Section 6 analyzes security considerations. Section 7 discusses implementation considerations. Section 8 specifies IANA registrations. The appendices provide test vectors and examples.

2. Conventions and Definitions

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.

2.1. Terminology

This document uses the following terminology:

DNS Stamp A URI-formatted string that encodes all parameters needed to connect to a DNS resolver.

Protocol Identifier A single byte value that identifies the DNS protocol type encoded in the stamp.

Properties A 64-bit little-endian integer encoding informal properties about the DNS resolver.

2.2. Encoding Primitives

The following encoding primitives are used throughout this specification:

‖ Denotes concatenation of byte sequences.

| Denotes the bitwise OR operation.

len(x) A single byte (unsigned 8-bit integer) representing the length of x in bytes, where x is a byte sequence of maximum length 255.

vlen(x) Variable length encoding. Equal to len(x) if x is the last element of a set. Otherwise equal to (0x80 | len(x)), indicating more elements follow.

LP(x) Length-prefixed encoding, defined as len(x) ‖ x.

VLP(x1, x2, ...xn) Variable-length-prefixed set encoding, defined as vlen(x1) ‖ x1 ‖ vlen(x2) ‖ x2 ... ‖ vlen(xn) ‖ xn. For a single-element set, VLP(x) == LP(x).

[x] Denotes that x is optional and may be omitted.

base64url(x) The URL-safe base64 encoding of x as specified in Section 5 of [RFC4648], without padding characters.

3. DNS Stamps Format Overview

This section provides a high-level overview of the DNS Stamps format before detailing specific protocol encodings.

3.1. URI Structure

A DNS Stamp is a URI [RFC3986] with the following format:

sdns://base64url(payload)

The stamp begins with the scheme sdns:// followed by a base64url-encoded payload. The payload is a byte sequence that encodes all parameters needed to connect to the DNS resolver.

3.2. Payload Structure

The general structure of the payload is:

protocol_identifier ‖ protocol_specific_data

The payload always begins with a single-byte protocol identifier that determines how to interpret the remaining bytes. The base64url encoding is applied to the entire payload as a single operation after concatenating all components.

3.3. Protocol Identifiers

The following protocol identifiers are defined:

Value	Protocol	Description
0x00	Plain DNS	Traditional unencrypted DNS
0x01	DNSCrypt	DNSCrypt protocol
0x02	DNS-over-HTTPS	DNS queries over HTTPS
0x03	DNS-over-TLS	DNS queries over TLS
0x04	DNS-over-QUIC	DNS queries over QUIC
0x05	Oblivious DoH Target	Target server for Oblivious DoH
0x81	Anonymized DNSCrypt Relay	Relay for DNSCrypt anonymization
0x85	Oblivious DoH Relay	Relay for Oblivious DoH

Table 1

Protocol identifiers in the range 0x80-0xFF are reserved for relay/ proxy protocols that forward queries to other servers.

3.4. Properties Field

Several stamp types include a properties field, which is a 64-bit little-endian integer. Each bit in this field represents a property of the resolver:

Bit	Property	Description
0	DNSSEC	The server validates DNSSEC signatures
1	No Logs	The server does not keep query logs
2	No Filter	The server does not filter or block domains
3-63	Reserved	Must be set to zero

Table 2

When encoding, undefined property bits MUST be set to zero. When decoding, undefined property bits MUST be ignored to allow future extensions.

4. Protocol-Specific Stamp Formats

This section specifies the exact format for each supported protocol type. Each format is presented with its structure, field descriptions, and encoding requirements.

4.1. Plain DNS Stamps

Plain DNS stamps encode parameters for traditional unencrypted DNS resolvers.

4.1.1. Format

payload = 0x00 ‖ props ‖ LP(addr)

4.1.2. Fields

0x00 Protocol identifier for plain DNS.

props Properties field (8 bytes, little-endian).

addr IP address and optional port as a string. IPv6 addresses MUST be enclosed in square brackets. Default port is 53.

4.1.3. Address Format

* IPv4: 192.0.2.1 or 192.0.2.1:5353

* IPv6: [2001:db8::1] or [2001:db8::1]:5353

4.2. DNSCrypt Stamps

DNSCrypt stamps encode parameters for DNSCrypt servers.

4.2.1. Format

payload = 0x01 ‖ props ‖ LP(addr) ‖ LP(pk) ‖ LP(provider_name)

4.2.2. Fields

0x01 Protocol identifier for DNSCrypt.

props Properties field (8 bytes, little-endian).

addr IP address and optional port. IPv6 addresses MUST be enclosed in square brackets. Default port is 443.

pk Provider's Ed25519 public key (exactly 32 bytes, raw binary format).

provider_name DNSCrypt provider name (e.g., 2.dnscrypt-cert.example.com).

4.2.3. Requirements

* The public key MUST be exactly 32 bytes.

* The provider name MUST be a valid DNS name.

* The provider name MUST NOT include a terminating period.

4.3. DNS-over-HTTPS Stamps

DoH stamps encode parameters for DNS-over-HTTPS servers.

4.3.1. Format

```
payload = 0x02 ‖ props ‖ LP(addr) ‖ VLP(hash1, ..., hashn) ‖
          LP(hostname) ‖ LP(path) [ ‖ VLP(bootstrap1, ..., bootstrapn) ]
```

4.3.2. Fields

0x02 Protocol identifier for DNS-over-HTTPS.

props Properties field (8 bytes, little-endian).

addr IP address of the server. May be empty string if hostname resolution is required.

hashi SHA256 digests of certificates in the validation chain (each exactly 32 bytes).

hostname Server hostname with optional port. Default port is 443.

path Absolute URI path (e.g., /dns-query).

bootstrapi Optional IP addresses for resolving hostname.

4.3.3. Requirements

* Certificate hashes MUST be exactly 32 bytes each.

* The hostname MUST NOT be percent-encoded or punycode-encoded.

* The path MUST start with “/” .

* Bootstrap addresses follow the same format as addr.

4.4. DNS-over-TLS Stamps

DoT stamps encode parameters for DNS-over-TLS servers.

4.4.1. Format

```
payload = 0x03 ‖ props ‖ LP(addr) ‖ VLP(hash1, ..., hashn) ‖
          LP(hostname) [ ‖ VLP(bootstrap1, ..., bootstrapn) ]
```

4.4.2. Fields

0x03 Protocol identifier for DNS-over-TLS.

Other fields have the same meaning as DoH stamps, except:

* Default port is 853.

* No path field is included.

4.5. DNS-over-QUIC Stamps

DoQ stamps encode parameters for DNS-over-QUIC servers.

4.5.1. Format

```
payload = 0x04 ‖ props ‖ LP(addr) ‖ VLP(hash1, ..., hashn) ‖
          LP(hostname) [ ‖ VLP(bootstrap1, ..., bootstrapn) ]
```

4.5.2. Fields

0x04 Protocol identifier for DNS-over-QUIC.

Other fields have the same meaning as DoT stamps.

4.6. Oblivious DoH Target Stamps

ODoH target stamps encode parameters for Oblivious DoH target servers.

4.6.1. Format

payload = 0x05 ‖ props ‖ LP(hostname) ‖ LP(path)

4.6.2. Fields

0x05 Protocol identifier for Oblivious DoH targets.

props Properties field (8 bytes, little-endian).

hostname Server hostname with optional port. Default port is 443.

path Absolute URI path.

4.7. Anonymized DNSCrypt Relay Stamps

DNSCrypt relay stamps encode parameters for anonymization relays.

4.7.1. Format

payload = 0x81 ‖ LP(addr)

4.7.2. Fields

0x81 Protocol identifier for DNSCrypt relays.

addr IP address and port. Port specification is mandatory.

4.8. Oblivious DoH Relay Stamps

ODoH relay stamps encode parameters for Oblivious DoH relays.

4.8.1. Format

```
payload = 0x85 ‖ props ‖ LP(addr) ‖ VLP(hash1, ..., hashn) ‖
          LP(hostname) ‖ LP(path) [ ‖ VLP(bootstrap1, ..., bootstrapn) ]
```

4.8.2. Fields

0x85 Protocol identifier for ODoH relays.

Other fields have the same meaning as DoH stamps.

5. Usage and Operations

This section describes how to generate, parse, and validate DNS stamps in practice.

5.1. Generating DNS Stamps

To generate a DNS stamp:

1. Select the appropriate protocol identifier.
2. Encode the properties field as 8 bytes in little-endian format.
3. Encode each parameter using the specified length-prefixing.
4. Concatenate all components in the specified order.
5. Apply base64url encoding to the complete payload.
6. Prepend "sdns://" to create the final stamp.

5.1.1. Implementation Requirements

Implementations generating DNS stamps MUST:

* Validate that all parameters meet format requirements.
* Ensure strings are valid UTF-8.
* Set undefined property bits to zero.
* Include all mandatory fields for the protocol type.
* Generate stamps that can be parsed by compliant implementations.

5.2. Parsing DNS Stamps

To parse a DNS stamp:

1. Verify the stamp begins with "sdns://".
2. Extract and base64url-decode the payload.
3. Read the first byte as the protocol identifier.
4. Parse remaining fields according to the protocol format.
5. Validate all fields meet requirements.

5.2.1. Error Handling

Implementations MUST detect and handle these error conditions:

* Invalid base64url encoding
* Unknown protocol identifier
* Truncated payload
* Invalid length prefixes
* Malformed fields

Implementations SHOULD provide descriptive error messages indicating the specific validation failure.

5.3. Validation Requirements

5.3.1. Length Validation

* Length prefixes MUST NOT exceed remaining payload size.
* Certificate hashes MUST be exactly 32 bytes.
* Ed25519 public keys MUST be exactly 32 bytes.
* Properties field MUST be exactly 8 bytes.

5.3.2. Format Validation

* IP addresses MUST be valid IPv4 or IPv6 addresses.
* Hostnames MUST be valid DNS names.
* Ports MUST be in the range 1-65535.
* Paths MUST begin with /.

5.3.3. Semantic Validation

* Certificate hashes SHOULD be validated against actual certificates.
* Provider names SHOULD be verified to exist in DNS.
* Bootstrap resolvers SHOULD be reachable.

5.4. Internationalization

Hostnames in DNS stamps MUST be represented in their Unicode form within the stamp payload. Implementations MUST NOT apply punycode encoding before storing hostnames in stamps. When using the hostname for actual DNS queries or TLS connections, implementations MUST apply the appropriate encoding for the protocol being used.

This approach:

* Preserves readability when stamps are decoded for display
* Avoids double-encoding issues
* Allows implementations to apply protocol-specific encoding rules

6. Security Considerations

6.1. Stamp Integrity

DNS stamps contain security-critical configuration including server addresses, cryptographic keys, and certificate hashes. The integrity of stamps is essential - a modified stamp could redirect users to malicious resolvers.

6.1.1. Threats

* *Substitution*: Replacing legitimate stamps with malicious ones
* *Modification*: Altering addresses, keys, or certificate hashes
* *Downgrade*: Replacing secure protocol stamps with insecure ones

6.1.2. Mitigations

Implementations SHOULD:

* Obtain stamps over authenticated channels (HTTPS with certificate validation)
* Verify stamps against known-good values when available
* Warn users when importing stamps from untrusted sources
* Validate all cryptographic parameters before use

6.2. Certificate Validation

For protocols using TLS (DoH, DoT, DoQ), stamps may include SHA256 hashes of certificates in the validation chain. These provide certificate pinning but require careful management.

6.2.1. Security Requirements

Implementations MUST:

* Verify at least one certificate in the chain matches a provided hash
* Follow standard certificate validation per [RFC5280]
* Check certificate validity periods and signatures
* Verify the certificate matches the specified hostname

6.2.2. Operational Considerations

Implementations SHOULD:

* Support multiple certificate hashes to enable rotation
* Provide clear errors for validation failures
* Allow optional fallback to standard WebPKI validation
* Cache certificate validation results appropriately

6.3. Privacy Considerations

DNS stamps may reveal information about resolver configuration:

* *Server Locations*: IP addresses indicate geographic regions
* *Logging Policies*: Properties flags indicate data retention
* *Query Privacy*: Bootstrap resolvers may see some queries

Users should understand the privacy implications of their chosen resolvers. Applications SHOULD display relevant properties clearly.

6.4. Implementation Security

6.4.1. Parsing Safety

Malformed stamps could trigger implementation vulnerabilities:

* *Buffer Overflows*: Validate all lengths before allocation
* *Integer Overflows*: Check length calculations
* *Resource Exhaustion*: Limit maximum stamp size

6.4.2. Cryptographic Safety

* Validate Ed25519 public keys are valid points
* Ensure certificate hashes are compared in constant time
* Use cryptographically secure random numbers where needed

6.5. Downgrade Prevention

Applications supporting multiple protocols MUST NOT automatically downgrade from secure to less secure protocols. For example:

* Never downgrade from DoH to plain DNS
* Never ignore certificate validation failures
* Never bypass authentication requirements

If a secure connection fails, the implementation SHOULD report the error rather than attempting insecure alternatives.

7. Implementation Considerations

7.1. Platform Integration

DNS stamp support can be integrated at various levels:

7.1.1. Operating System Level

* System resolver configuration
* Network configuration tools
* VPN client integration

7.1.2. Application Level

* Web browsers
* DNS proxy software
* Network diagnostic tools

7.1.3. Library Level

* DNS client libraries
* HTTP client libraries
* Security frameworks

7.2. Performance Optimization

7.2.1. Caching

Implementations SHOULD cache: - Decoded stamp data structures - Certificate validation results - Bootstrap resolver results - Connection state for persistent protocols

7.2.2. Connection Management

* Reuse connections for multiple queries
* Implement appropriate timeout strategies
* Handle connection failures gracefully
* Support connection pooling for concurrent queries

7.3. User Interface Considerations

Applications SHOULD:

* Display decoded stamp contents clearly
* Allow copying stamps to clipboard
* Support QR code generation/scanning
* Provide stamp validation feedback
* Show security properties prominently

7.4. Debugging Support

Implementations SHOULD provide:

* Detailed logging of stamp parsing
* Connection attempt diagnostics
* Certificate validation details
* Performance metrics
* Error context for troubleshooting

8. IANA Considerations

8.1. DNS Stamps URI Scheme Registration

IANA is requested to register the “sdns” URI scheme in the “Uniform Resource Identifier (URI) Schemes” registry:

* *Scheme name*: sdns
* *Status*: Permanent
* *Applications/protocols*: DNS client applications using DNS Stamps
* *References*: This document |

9. References

9.1. Normative References

[I-D.draft-denis-dprive-dnscrypt]
Denis, F., "The DNSCrypt Protocol", Work in Progress, Internet-Draft, draft-denis-dprive-dnscrypt-06, 15 April 2025, <https://datatracker.ietf.org/doc/html/draft-denis-dprive-dnscrypt-06>.

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

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

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

[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/rfc/rfc5280>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/rfc/rfc6125>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <https://www.rfc-editor.org/rfc/rfc7858>.

[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>.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/rfc/rfc8484>.

[RFC9250] Huitema, C., Dickinson, S., and A. Mankin, "DNS over Dedicated QUIC Connections", RFC 9250, DOI 10.17487/RFC9250, May 2022, <https://www.rfc-editor.org/rfc/rfc9250>.

9.2. Informative References

[ODOH] "Oblivious DNS over HTTPS", 2023, <https://datatracker.ietf.org/doc/draft-pauly-dprive-oblivious-doh/>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/rfc/rfc3552>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/rfc/rfc5116>.

[RFC8310] Dickinson, S., Gillmor, D., and T. Reddy, "Usage Profiles for DNS over TLS and DNS over DTLS", RFC 8310, DOI 10.17487/RFC8310, March 2018, <https://www.rfc-editor.org/rfc/rfc8310>.

[RFC9230] Kinnear, E., McManus, P., Pauly, T., Verma, T., and C.A. Wood, "Oblivious DNS over HTTPS", RFC 9230, DOI 10.17487/RFC9230, June 2022, <https://www.rfc-editor.org/rfc/rfc9230>.

Appendix A. Complete Examples

This appendix provides complete examples of DNS stamp encoding with step-by-step explanations.

A.1. Example 1: Plain DNS

Configuration:

* Server: 192.0.2.53
* Port: 53 (default)
* Properties: DNSSEC (bit 0 set)

Step-by-step encoding:

1. Protocol identifier: 0x00
2. Properties: 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 (bit 0 set, little-endian)
3. LP(“192.0.2.53”): 0x0A ‖ “192.0.2.53” = 0x0A 0x31 0x39 0x32 0x2E 0x30 0x2E 0x32 0x2E 0x35 0x33
4. Concatenate: 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0A 0x31 0x39 0x32 0x2E 0x30 0x2E 0x32 0x2E 0x35 0x33
5. Base64url encode: AAEAAAAAAAAACjE5Mi4wLjIuNTM

6. Final stamp: sdns://AAEAAAAAAAAACjE5Mi4wLjIuNTM

A.2. Example 2: DNSCrypt

Configuration:

* Server: 198.51.100.1
* Port: 5553
* Provider public key: e801...bf82 (32 bytes)
* Provider name: 2.dnscrypt-cert.example.com
* Properties: DNSSEC, No logs, No filter (bits 0, 1, 2 set)

Step-by-step encoding:

1. Protocol identifier: 0x01
2. Properties: 0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00
3. LP(“198.51.100.1:5553”): 0x11 ‖ address
4. LP(public key): 0x20 ‖ 32 bytes of key
5. LP(“2.dnscrypt-cert.example.com”): 0x1B ‖ provider name
6. Concatenate all components
7. Base64url encode
8. Final stamp: sdns://AQcAAAAAAAAAETE5OC41MS4xMDAuMTo1NTUzIOgBsd...

A.3. Example 3: DNS-over-HTTPS

Configuration:

* Hostname: dns.example.com
* Path: /dns-query
* No specific IP address
* Certificate hash: 3b7f...b663 (32 bytes)
* Properties: No logs (bit 1 set)

Step-by-step encoding:

1. Protocol identifier: 0x02

2. Properties: 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00

3. LP(“”): 0x00 (empty address)

4. VLP(cert hash): Since it’s the only hash, same as LP: 0x20 ‖ 32 bytes

5. LP(“dns.example.com”): 0x0F ‖ hostname

6. LP(“/dns-query”): 0x0A ‖ path

7. No bootstrap IPs

8. Concatenate, base64url encode

9. Final stamp: sdns://AgIAAAAAAAAAAAAD2Rucy5leGFtcGxlLmNvbQovZG5zLXF1ZXJ5

Appendix B. Test Vectors

This appendix provides test vectors for validating DNS stamp implementations.

B.1. Test Vector 1: Plain DNS with IPv6

Input:
  Protocol: Plain DNS
  Address: [2001:db8::1]:53
  Properties: DNSSEC

Encoded stamp:
  sdns://AAEAAAAAAAAADlsyMDAxOmRiODo6MV0

Decoded:
  Protocol ID: 0x00
  Properties: 0x0100000000000000
  Address: "[2001:db8::1]"

B.2. Test Vector 2: DoH with Multiple Certificate Hashes

```
Input:
  Protocol: DNS-over-HTTPS
  Hostname: dns.example.com
  Path: /dns-query
  Cert Hash 1: 1111111111111111111111111111111111111111111111111111111111111111
  Cert Hash 2: 2222222222222222222222222222222222222222222222222222222222222222
  Properties: None

Encoded stamp:
  sdns://AgAAAAAAAAAAACCRERERERERERERERERERERERERERERERERERERESAiIiIiIiIiIiIiIiIiIi
IiIiIiIiIiIiIiIiIiIiD2Rucy5leGFtcGxlLmNvbQovZG5zLXF1ZXJ5

Decoded:
  Protocol ID: 0x02
  Properties: 0x0000000000000000
  Address: ""
  Hash count: 2
  Hash 1: 1111111111111111111111111111111111111111111111111111111111111111
  Hash 2: 2222222222222222222222222222222222222222222222222222222222222222
  Hostname: "dns.example.com"
  Path: "/dns-query"
```

B.3. Test Vector 3: DoT with Bootstrap

```
Input:
  Protocol: DNS-over-TLS
  Hostname: dot.example.com:853
  Address: 192.0.2.1
  Bootstrap: 198.51.100.1, 203.0.113.1
  Properties: No logs, No filter

Encoded stamp:
  sdns://AwYAAAAAAAAACTE5Mi4wLjIuMQAPZG90LmV4YW1wbGUuY29tCwwxOTguNTEuMTAwLjELMjAzLjAuMTEz
LjE

Decoded:
  Protocol ID: 0x03
  Properties: 0x0600000000000000
  Address: "192.0.2.1"
  No certificate hashes
  Hostname: "dot.example.com:853"
  Bootstrap count: 2
  Bootstrap 1: "198.51.100.1"
  Bootstrap 2: "203.0.113.1"
```

Appendix C. Acknowledgments

The author would like to thank the DNSCrypt community for their extensive feedback and implementation experience. Special recognition goes to the developers of the various DNS stamp implementations who helped refine the format through practical deployment.

Thanks also to the teams behind secure DNS protocols - DNSCrypt, Anonymized DNSCrypt, DoH, DoT, and DoQ - whose work made DNS stamps both necessary and useful. Their efforts to improve DNS privacy and security provided the foundation for this specification.

Author’s Address

Frank Denis
Individual Contributor
Email: fde@00f.net