Formal Security Analysis of ss2DNS

Ali Sadeghi Jahromi(✉), AbdelRahman Abdou, and Paul C. van Oorschot

Carleton University, Ottawa, Canada
{alisadeghijahromi@cmail, abodu@scs, paulv@scs}.carleton.ca

Abstract. Motivated by limitations in DNSSEC, ss2DNS has been recently proposed as an alternative scheme that augments recursive resolver and nameserver interactions with stronger real-time security and privacy properties. In this paper, using a symbolic model of ss2DNS in the Tamarin protocol verifier, and the assumption that none of the relevant cryptographic keys are compromised, we formally verify the security and privacy properties of the DNS resolution process with the root zone when using ss2DNS. The validity of these proofs is then extended to the other subordinate zones. We also explore the impact of key and entity compromises on ss2DNS security and privacy properties.

Keywords: DNS Security · Formal Analysis · Tamarin

1 Introduction

Essentially all interactions over the Internet are preceded by a DNS resolution, which translates human-readable domain names into corresponding IP addresses [16,17]. DNS serves as a critical component of the Internet, facilitating the resolution of hundreds of billions of queries everyday.[1] Therefore, the security and privacy of the DNS resolution process impacts all entities and services that depend on its functionality [6]. The DNS resolution process typically involves two stages: the communication between a stub resolver and a recursive resolver (Stage 1), and the interactions between a recursive resolver and authoritative nameservers (ANSes) (Stage 2) [18]. Attackers have exploited the absence of security in Stage 2 to compromise security and privacy [10,12,13].

We recently proposed the ss2DNS protocol, designed to provide real-time security and privacy properties in Stage 2 [18]. Use of ss2DNS in Stage 2 effectively mitigates vulnerabilities within this stage, including cache poisoning [6,12,13], replay attacks using stale responses [1,4,11], eavesdropping [2], and large-scale surveillance [10]. ss2DNS utilizes a delegation mechanism to mitigate the risk of

The official version of this paper appears in the proceedings of the 30th European Symposium on Research in Computer Security (ESORICS 2025), at https://dx.doi.org/10.1007/978-3-032-07894-0_23. The present version is the authors' copy for personal use, and may differ in minor respects. In some preliminary versions of this work, the protocol was named DNSSEC+. It was renamed ss2DNS to avoid suggesting it is a direct extension of DNSSEC.

[1] https://vercara.com/resources/2023-dns-traffic-and-trends-analysis

exposure of a zone's long-term key to attacks by avoiding the need to replicate this key throughout the ANSes of the zone. Additionally, it secures transmission of DNS messages by employing symmetric authenticated encryption, with encryption keys generated using Diffie-Hellman (DH) key agreement.

The original paper [18], in which ss2DNS was proposed, introduced its design, implemented a prototype, and conducted a performance evaluation. It defines nine properties, including several that hold by design (*e.g.,* properties such as achieving single round-trip DNS resolution are inherently satisfied). As a more robust scheme in terms of security and privacy properties, ss2DNS (if standardized and adopted) could be used to resolve millions (or more) of queries daily. To increase confidence in the security and privacy properties of ss2DNS, a formal analysis is provided herein. To do so, we develop a symbolic model of the protocol using the Tamarin Prover [15] tool. The security and privacy properties of ss2DNS are then formally proven using this model. Finally, the model is used to assess the impact of key and entity compromises, demonstrating the consequences of such compromises on these properties. Our analysis finds that all of the originally defined properties of ss2DNS hold as expected when none of the keys are compromised. Moreover, compromise of any single long-term private key within the protocol does not, in isolation, compromise the validity of the security and privacy properties of responses.

2 Background

In this section, we present an overview of symbolic modeling and property verification of security protocols in Tamarin, followed by background on the components and the theory of operation of ss2DNS. Readers familiar with these may wish to skip over this review section.

2.1 The Tamarin Prover (review)

In symbolic models of security protocols, cryptographic messages are expressed as *terms* (instead of bitstrings), which are used to model data and operations in a protocol. These primitives are used in *rules* that deterministically model protocol behavior, from which properties of interest are defined and verified. Tamarin is a tool used for symbolic modeling and formal verification of security protocols [15]. It offers pre-defined symbolic cryptographic primitives, such as DH key agreement, symmetric encryption, and digital signatures. Tamarin has been employed for modeling and analysis of security protocols such as TLS 1.3 [5], 5G authentication [3], RHINE [9], and other security protocols [19]. Thus, Tamarin is an appropriate tool for a formal security analysis of ss2DNS.

The default attacker model in Tamarin is the Dolev-Yao [8] adversary model, which assumes complete control over the network: the adversary can read, modify, drop, or fabricate and inject network messages. A Tamarin user defines a symbolic model of a protocol in Tamarin syntax and specifies the properties to be proven. Tamarin is capable of automatically generating proofs to establish

the validity of the specified properties, without requiring manual interaction. Tamarin also provides an interactive mode with a graphical interface that offers a step-by-step manual proof development process, with visualization of the constraint solver and any discovered attacks.

Protocol Specification. In Tamarin, the allowed protocol interactions and steps are specified using multiset rewriting rules. These rules form a labeled transition system, with a global state composed of *facts* that represent the state information. The transition system begins with an empty multiset of facts, and this multiset evolves as rules are executed. The following example demonstrates a Tamarin rule, named `Example_DH`. Each rule has a left-hand-side (LHS/premise) and a right-hand-side (RHS/conclusion) that specify the multisets before and after the rule execution. If all the LHS facts exist in the current state, the rule can be executed, resulting in the removal of the LHS facts from the current global state and the addition of the RHS facts to the state.

```
1 rule Example_DH:
2     [Fr(a)] // premise (LHS)
3     --[Secret(a)]→ // action/event facts
4     [!Ltk($A, a), Out(g^a)] // conclusion (RHS)
```

In Tamarin, facts are typically represented as `F(t1,...,tn)`, where `F` represents the name of the fact, and `t1` to `tn` are the terms that represent variables, messages, functions, and other symbolized elements. On the LHS (line 2), the fact `Fr()` is a built-in Tamarin fact, used for modeling fresh random values such as keys, and a is a freshly generated term, which will be used as a DH private key in this example. On the RHS, `Out()` is another built-in fact in Tamarin that models an entity sending out a message to the adversary-controlled network. Here (g^a) models the public component of a DH key in the built-in DH theory of Tamarin. The fact `!Ltk()` stores identity A with its fresh private key a. The facts within the LHS and RHS are state facts that, upon execution of the rule, are respectively removed and added to the global state. The '!' symbol indicates that a fact is persistent: it will not be removed from the system's state when it is consumed by rules and can be used as often as needed. There is another type of fact, such as `Secret()` in `Example_DH`, which is not present in the LHS and RHS and is categorized as an event/action fact. Action facts are not added to or consumed from the system's state. Instead, they serve to label transitions by recording the execution of rules, and they are added to the protocol execution trace. The different possible executions of the rules generate a set of potential execution traces made up from action facts. These traces serve as the basis for defining and verifying properties.

Defining Properties. In Tamarin, the properties are formulated as lemmas, which are guarded first-order logic formulas defined over action facts and timepoints. To verify a security property, its lemma must hold across all possible execution traces with no counterexamples. Tamarin either provides a proof if the property holds in all behaviors or returns a counterexample showing an attack. In some cases, the automatic proof construction process in Tamarin may fail to produce a proof or a counterexample with the allocated time and resources, and the verification process fails to terminate. In this case, the property is neither proved nor disproved. In such cases, the interactive mode of Tamarin can be used to identify the underlying problem and guide the tool by writing special intermediate lemmas and refining the protocol model to help the automatic proof generation process to reach a conclusive termination.

The following example presents a simple secrecy lemma, named `Example_Secrecy`. Here, `K()` (line 4) is a built-in action fact that represents the adversary's knowledge of a specific term, such as a 'key' in this example. The `#i` and `#j` are temporal variables, representing timepoints. This lemma asserts that for all possible protocol behaviors and values of the variable 'key' (line 2), if the 'key' is captured in the execution trace by the action fact `secret()` at timepoint `#i` (line 3), there exists no timepoint `#j` such that (denoted by '.') the adversary knows the 'key' (line 4). If Tamarin identifies a counterexample at which the value 'key' is known to the adversary, it stops the proof process and presents the attack in the output.

```
1 lemma Example_Secrecy:
2     "All key #i .
3     Secret(key) @i
4     ==> not (Ex #j . K(key) @j)"
```

In addition to lemmas used to demonstrate security and privacy properties of a modeled protocol, another type of lemma (executability lemma) is used to ensure that the model can execute to completion properly. These lemmas ensure that the properties are verified in a properly modeled and executable model of the protocol.

2.2 ss2DNS (review)

ss2DNS [18] is a recently proposed protocol aimed at enhancing the security and privacy properties of the DNS resolution process in Stage 2. A security and privacy comparison of ss2DNS with other secure DNS schemes has been conducted in the referenced paper [18]. Additionally, ss2DNS's performance was evaluated in comparison to other secure DNS schemes by measuring resolution time, server-side processing time, CPU utilization, and the impact of network fragmentation on performance.

It provides real-time security and privacy properties while avoiding the replication of long-term private keys (associated with zones) across nameserver instances. This is achieved through a novel delegation mechanism. As demonstrated in Figure 1, ss2DNS involves the use of three primary components: (1) delegation of authorization within each zone to ANSes, (2) a reverse-tree trust model within the DNS hierarchy, and (3) a secure DNS resolution process between resolvers and ANSes.

Long-term Keys in ss2DNS: In ss2DNS, a key server integrated with each zone is responsible for managing the zone's long-term keys: the Long-Term Signing Key (`LTSK`) and the Long-Term Agreement Key (`LTAK`). As illustrated in Figure 1, the root zone keys, denoted $\underline{\text{LTSK}}_0$ and $\underline{\text{LTAK}}_0$, reside in the root zone's key server (KS_0); the suffix refers to the level of the zone within the DNS hierarchy, with '0' being the root zone, '1' for Top-Level Domain (TLD), and so on. `LTSK` is a signing key, which is used to authorize nameserver instances within a zone, enabling response authenticity. `LTAK` is a DH agreement key used by resolvers to derive query encryption keys, to protect the confidentiality of queries. Underlined keys denote private components of asymmetric key pairs.

Delegation Within Each Zone: The delegation mechanism in ss2DNS involves a zone's key server authorizing ANSes of the zone to securely respond to queries by signing the public part of the Short-Term Signing Key (`STSK`) of ANSes, using the zone's `LTSK`. These STSKs are signature keys with short lifetimes (*e.g.,* hours to days), used for providing response authenticity. ANSes must renew their public part of STSKs (sending them to the zone's key server to be signed) before they expire.

As shown in Figure 1 (teal dashed boxes, labeled 1), ANSes of a zone send their STSKs to the key server within their zone; the key server signs the STKSs using the zone's LTSK and returns a signature on the public part of STSK (sig(STSK)) through a secure Out-Of-Band (OOB) channel to the ANSes of the zone. This approach minimizes the exposure of the private long-term key (LTSK) by eliminating the need for its replication, and also mitigates the risk of compromise of the private STSK, as STSKs automatically expire after a predefined short validity period.

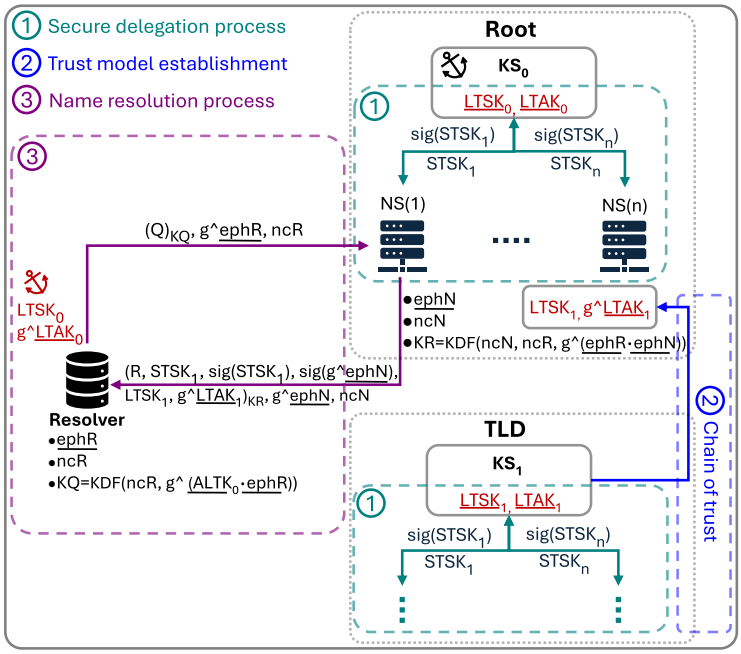

Fig. 1. Workflow of ss2DNS (Underlined keys are private; others are public)

Forming a Trust Model: ss2DNS employs a reverse-tree trust model within the DNS hierarchy, analogous to DNSSEC. To establish this trust model, each zone below the root transmits its long-term public keys (LTSK and LTAK) to its parent zone. For instance, as shown in Figure 1 (blue dashed box, labeled 2), the TLD zone sends the public components of its $LTSK_1$ and $g\hat{}$LTAK$_1$ keys to the root, forming a chain of trust to the root zone (the trust anchor). Similarly Second-Level Domains (SLDs) send their $LTSK_2$ and $g\hat{}$LTAK$_2$ to their parent TLDs.

The root zone's long-term public keys ($LTSK_0$ and $g\hat{}$LTAK$_0$) are embedded in resolver software as ss2DNS trust anchors. As the resolver traverses the DNS hierarchy, it obtains a child zone's long-term public keys from its parent, enabling secure query encryption and response authentication. This process relies on the chain of trust established through the transmission of public keys from each zone to its parent zone, extending ultimately to the root zone (the trust anchor).

DNS Resolution in ss2DNS: Upon completing the establishment of the chain of trust and the delegation process within the zones, a resolver can use ss2DNS to resolve DNS records in Stage 2, as illustrated in Figure 1 (depicted within the dashed purple box labeled 3). The left-hand side purple box demonstrates a DNS resolution instance with nameserver 1 from the root zone. A resolver, with access to the root zone's long-term agreement public key ($g \hat{} \underline{\text{LTAK}_0}$), embedded as a trust anchor, generates a private ephemeral agreement key ($\underline{\text{ephR}}$) and a nonce (ncR). It then creates a query (Q) and encrypts it using a symmetric key (KQ), which is derived from the master DH key ($g \hat{} (\underline{\text{LTAK}_0} \cdot \underline{\text{ephR}})$) and ncR using a Key Derivation Function (KDF), and sends the encrypted query along with g$\hat{}\underline{\text{ephR}}$ and ncR to the root zone's nameserver 1 (demonstrated by the top purple arrow).

Upon receiving the query, Nameserver 1 derives the same KQ, decrypts the query, and prepares a response (R). The symmetric key for response encryption differs from the key for encrypting the query (KQ). The nameserver then generates an ephemeral private agreement key ($\underline{\text{ephN}}$) and a nonce (ncN), and subsequently derives a response encryption key (KR) by applying a KDF to ($g \hat{} (\underline{\text{ephR}} \cdot \underline{\text{ephN}})$) and ncN with the resolver's nonce (ncR). As illustrated by the purple bottom arrow, the encrypted part of the response includes the response (R) and the nameserver's short-term signature key $STSK_1$ and its signature ($sig(STSK_1)$) obtained in the delegation process. Additionally, a signature of public ephN ($sig(g \hat{} \underline{\text{ephN}})$ generated by $\underline{STSK_1}$ is included by which the resolver can trust $g \hat{} \underline{\text{ephN}}$. The long-term keys of the TLD child zone ($LTSK_1$ and $g \hat{} \underline{\text{LTAK}_1}$) are also included, which will be used by the resolver to securely interact with the child zone in subsequent queries. After encrypting the response, along with the aforementioned signatures and keys, using KR, the nameserver appends g$\hat{}\underline{\text{ephN}}$ and ncN, which will be used for response decryption by the resolver.

Upon receiving the response, the resolver uses g$\hat{}\underline{\text{ephN}}$ and ncN to generate KR to decrypt the response. It verifies the $sig(STSK_1)$ using $LTSK_0$ (the trust anchor) and thereby authenticates $STSK_1$. Next, it verifies the signature of the ephemeral agreement key ($sig(g \hat{} \underline{\text{ephN}})$) using $STSK_1$, verifying the authenticity of the public agreement key of the nameserver ($g \hat{} \underline{\text{ephN}}$), used for encrypting the response. The resolver also extracts $LTSK_1$ and $g \hat{} \underline{\text{LTAK}_1}$ from the response, which will be used to securely query the TLD child zone. This resolution process continues with the subordinate zones until the resolver queries the authoritative zone for the queried record and obtains the final response, using ss2DNS in Stage 2.

3 Modeling ss2DNS in Tamarin

This section begins by listing the design properties of ss2DNS and specifying the security- and privacy-related subset formally verified by Tamarin. Next, it defines the underlying assumptions and threat model used to develop the symbolic model of ss2DNS and to prove its properties. Finally, it presents an overview of the model used for delegation and DNS resolution processes within ss2DNS.

3.1 Stated Properties of ss2DNS

The original paper [18] defines nine properties to be fulfilled by ss2DNS. Table 1 summarizes these and highlights the subset of these that are security and privacy properties, formally proven in the present paper using Tamarin. ss2DNS encrypts both queries and

responses to achieve *message confidentiality*. Additionally, the response encryption keys are freshly generated, with the objective of providing *forward secrecy*. Another security property that ss2DNS provides is *entity authentication* for nameservers, guaranteeing that transmitted responses are both authenticated and *resilient to replay attacks*. The defined entity authentication in ss2DNS is unilateral, as only the nameservers authenticate themselves to the resolvers; the resolvers do not authenticate themselves to the nameservers they query.

For the remaining properties, the original paper [18] demonstrated how the design of ss2DNS satisfies the objectives of *avoiding duplication of long-term secrets* through a secure delegation process, achieving *single round-trip* DNS resolution, and employing an *established trust model*. The property of *minimum amplification* is satisfied by the design of messages that maintain a constant amplification factor compared to DNSSEC [18]. Moreover, the *fail-closed* objective was described as a procedural design choice to mitigate within-protocol downgrade attacks in ss2DNS [18].

Table 1. Properties of ss2DNS: The first four properties are supported by Tamarin proofs herein, while the remaining five were established by design [18].

ss2DNS Property	Target	notes
Message confidentiality	DNS queries/responses	Tamarin proof
Forward secrecy	DNS response keys	Tamarin proof
Entity authentication	Nameservers	Tamarin proof
Mitigate replay	DNS responses	Tamarin proof
Avoid duplicating LTK	DNS zones	by design
Single round-trip	DNS resolution	by design
Established trust model	DNS hierarchy	by design
Minimum amplification	DNS responses	by design
Fail-closed	DNS resolution process	by design

3.2 Assumptions and Threat Model

As outlined in Section 2.2, there are three main components in ss2DNS: the delegation of authorization to the nameservers within each zone, a trust model, and the DNS resolution process. The establishment of the trust model, in which the child zones below the root send their long-term public keys to their parent zone, takes place through a secure OOB channel. Thus, it is assumed that this process is securely completed prior to any name resolution. The delegation process involves signing the short-term keys within a zone and sending them to the nameservers through a confidential and authenticated OOB channel established between the zone's key server and its nameservers. The delegation process must also be completed prior to the initiation of any name resolution. The security and privacy properties defined in Table 1 (*i.e.,* the first four properties) are for the DNS resolution process in Stage 2, and the preceding processes of trust model establishment and delegation to nameservers are assumed to be completed via their secure OOB channels [18].

We model the communication between the nameservers and key servers for the delegation process within zones as secure OOB channels whose messages the adversary (in

Tamarin) cannot access. Additionally, we assume that there is a trust model established between the zones within the DNS hierarchy through an authenticated OOB channel between the zones. In practice, as seen in DNSSEC, this can be done via a registrar's web interface, where the domain owner adds the Delegation Signer (DS) record to its parent zone. Using these two assumptions, we formally verify the properties of the DNS resolution process in ss2DNS using the Tamarin prover.

In our model, we limit the name resolution to the root zone and do not continue the resolution down the DNS hierarchy. By proving the security and privacy properties for DNS resolution with a root's nameserver, it is assumed that, in ss2DNS, the resolver obtains the child (*i.e.,* TLD) zone's authentic long-term public keys from the root. The long-term public keys of the child zone are assumed to be transmitted authentically to the root during the trust model establishment phase. Thus, in turn, the formal verification of the name resolution for the TLD zone would be similar to the root, but this time with the TLD's long-term keys. In this manner, proving the properties of DNS resolution with the root zone under the defined assumptions ensures that the same properties hold for subsequent resolutions with subordinate zones in the DNS hierarchy, unless the assumptions differ.

The described name resolution process in Section 2.2 (Figure 1) is the *privacy-enforcing* mode of ss2DNS in which the queries are encrypted. ss2DNS also offers a *no-privacy* mode in which the queries are not encrypted but the responses are authenticated and encrypted. Herein, we model and prove the security and privacy properties of ss2DNS in the *privacy-enforcing* mode, for which we can also prove the secrecy of queries. Moreover, in Section 5, where we show the implications of key compromises, we demonstrate that the response properties remain verifiable if query secrecy is compromised (*e.g.,* when `LTAK` is compromised), which is an analogous scenario to the no-privacy mode, wherein the queries are in plaintext.

In addition to the above protocol-related assumptions, in symbolic analysis we exclude computational attacks and primarily focus on the interactions and abstract operation of the protocol. Consequently, symbolic analysis inherently incorporates certain assumptions as part of its abstraction. The first assumption is that the cryptographic primitives within the model are perfect. For example, an adversary can generate a valid signature if and only if it knows the signing private key. Freshly generated random terms, such as nonces and cryptographic keys, are assumed to be unpredictable and unique. Since the information in the symbolic model is abstracted to terms, an entity, including the adversary, either has complete knowledge of a term or no knowledge at all, with no possibility of having partial knowledge of the term.

Common threats in Stage 2 of the DNS resolution include inline adversaries (between a recursive resolver and ANSes). These can be located within distinct Autonomous Systems (ASes), each maintained by different administrative policies. Such entities have access to DNS messages and can read, modify, delete, and inject fabricated messages. Additionally, off-path adversaries can send queries to a recursive resolver and inject false responses, thereby poisoning the cache of the resolver[12,13]. In Section 4, we focus on proving the properties of ss2DNS against both in-line and off-path network-based attacks, and we use the default Dolev-Yao [8] attacker in the Tamarin prover. In Section 5, in order to analyze the impact of host and key compromises on the properties, we model host-based attacks by adding the private keys of ss2DNS to the adversary's knowledge and analyzing how it affects the verified properties.

3.3 Formal Modeling of ss2DNS in Tamarin

To model the ss2DNS delegation process in Tamarin, we use four Tamarin rules:

D1. The initialization of a key server within the root zone, responsible for generating and securely storing the long-term keys of the zone (*i.e.,* LTSK and LTAK).
D2. The initialization of the zone's ANS instances and their respective short-term signing keys (STSKs), followed by the transmission of these STSKs to the key server via an OOB channel modeled in Tamarin, which assumes that the messages are transferred over a channel not accessible to the adversary.
D3. Receiving the nameserver's STSKs by the initialized key server in Step 1, which then uses the zone's LTSK to sign the STSKs. The signed STSKs are subsequently returned to the nameservers through a secure OOB channel.
D4. Finally, the nameservers receive their signed STSK, enabling their use in the DNS resolution process for responding to queries.

Subsequently, the DNS resolution process between resolvers and the nameserver instances of the zone is modeled using three Tamarin rules.

R1. For a resolver generating and sending an encrypted query to a root zone nameserver: As in Figure 1, the resolver generates a fresh ephemeral key `ephR` and `ncR`, using them with the root zone's public agreement key `LTAK` (installed as a trust anchor) to derive the query encryption key. Additionally, the resolver generates a fresh query, which is not known to the adversary. The resolver then encrypts the query and transmits it to the network, along with `g^ephR` and `NCR`. This message will be received by both the adversary and a root zone nameserver. The resolver also stores the current query state, including the query itself, the ephemeral agreement key `ephR`, and the nonce `NCR`. These terms will be used later in Rule 3 for response decryption.

R2. For a nameserver of the root zone that receives an encrypted query and returns an authenticated, encrypted response to the resolver: the nameserver, having access to the private `LTAK`, decrypts the query using the received `g^ephR` and `ncR` terms from the query. The nameserver then generates a fresh ephemeral key `ephN` and a nonce `ncN`, which are used alongside the resolver's nonce and ephemeral agreement key to derive the response encryption key. As depicted in the purple box in Figure 1, the nameserver appends its STSK and the corresponding signature (sig(`STSK`)) to the response. Additionally, the nameserver signs `g^ephN` using its STSK and includes the signature sig(`g^ephN`) in the response before encryption. Finally, the nameserver sends `g^ephN`, `ncN`, and the encrypted portion of the response to the network, which will be received by the resolver and adversary.

R3. The resolver that initiated the query in the first rule receives, decrypts, and validates the authenticity of the response. In the Tamarin model, we stored the transmitted query state, including its ephemeral key (`ephR`), nonce (`ncR`), and the query itself, in the first rule. The resolver receives the response and obtains the ephemeral agreement key of the nameserver (`g^ephN`) and the nameserver's nonce (`ncN`) from the response and uses them along with its ephemeral key and nonce to derive the response decryption key. Upon decrypting the response, the resolver obtains the STSK and sig(`STSK`) from the response. The resolver then verifies sig(`STSK`), using the root zone's public `LTSK`, which is included in its software as a trust anchor. After verifying the authenticity of the STSK, the resolver verifies the signature of the ephemeral key of the nameserver (sig(`ephN`)) using the public STSK from the response. Once both signatures are verified, the resolver can trust the `STSK` and `ephN` used for encryption.

The Tamarin rules D1-D4 are presented in Appendix A.1, and the rules R1-R3 are provided in Appendix A.2.

4 Modeling the Security Properties

In this section, we model the specified security properties of ss2DNS as lemmas using Tamarin's syntax and subsequently verify that these properties hold.

4.1 Secrecy of DNS Messages

Message secrecy in formal verification is generally defined as follows:

$$msg\_secrecy \triangleq \forall\ msg\ \ i.\ \ Secret(msg)\ @i\ \ \rightarrow not\ \exists\ j.\ K(msg)@j$$

This means: the definition of *msg_secrecy* is for all instances of the term *msg*, where *msg* is captured by the *Secret()* action fact at timepoint i, there does not exist a timepoint j at which the adversary knows the *msg*.

Secrecy of Query Data: We formally define the secrecy property of queries generated and sent by a legitimate resolver by the lemma specified in Listing 1.1. This lemma asserts that whenever an encrypted query is sent by a legitimate resolver at timepoint i (captured by the instance of `QDataSecret(R, QData)` action), and none of the private agreement or signing keys of ss2DNS are compromised (lines 4-8), then there is no timepoint j at which the adversary knows the plaintext of the query message. In the verification of this property in Tamarin, it is assumed that all keys within the protocol remain unknown to the adversary.

```
1 lemma query_secrecy:
2  "All R QData #i .
3   QDataSecret(R, QData) @i & Role('R') @i &
4   not (Ex Z #t1 . RevLTAK(Z) @t1) &
5   not (Ex R #t2 . RevEphR(R) @t2) &
6   not (Ex N #t3 . RevEphN(N) @t3) &
7   not (Ex Z #t4 . RevLTSK(Z) @t4) &
8   not (Ex N #t5 . RevSTSK(N) @t5)
9   ==> not (Ex #j. K(QData) @j)"
```

Listing 1.1. Lemma for query secrecy (verified by Tamarin)

We used the `query_secrecy` lemma and verified the secrecy of the query data in Tamarin. Additionally, we independently verified the secrecy of the query encryption key using another lemma. In ss2DNS, we assert that the `query_secrecy` property also implicitly proves the confidentiality of the query session key (*i.e.,* the derived key used for query encryption). If the query encryption key is compromised, this property will be violated, as the adversary can decrypt and access the query data in plaintext.

Secrecy of Response Data: The ss2DNS response encryption key is distinct from the encryption key used for transmitting its corresponding query. In addition to the secrecy of the query data, we also verify the secrecy of the response data received by a legitimate resolver as an answer to a query sent by the same resolver. We define the

lemma for response data secrecy, as demonstrated in Listing 1.2, and use Tamarin to prove this lemma. This lemma asserts that for all protocol behaviors, if an encrypted response to a query of a legitimate resolver is received by the resolver at timepoint i (captured by the instance of `RDataSecret(RData)` action), and none of the private agreement or signing keys are compromised, then there exists no timepoint j at which the adversary knows the plaintext response. In this lemma, similar to query secrecy, we assume that none of the protocol private keys are known to the adversary.

```
lemma response_secrecy:
 "All RData #i .
  RDataSecret(RData) @i & Role('R') @i &
  not (Ex Z #t1 . RevLTAK(Z) @t1) &
  not (Ex R #t2 . RevEphR(R) @t2) &
  not (Ex N #t3 . RevEphN(N) @t3) &
  not (Ex Z #t4 . RevLTSK(Z) @t4) &
  not (Ex N #t5 . RevSTSK(N) @t5)
  ==> not (Ex #j. K(RData) @j)"
```

Listing 1.2. Lemma for response secrecy (verified by Tamarin)

The proof of the `response_secrecy` lemma in Tamarin verifies that DNS responses of queries received by a legitimate resolver remain unknown to the adversary. Additionally, we independently verified the secrecy of the response encryption key using a separate lemma. Furthermore, in the context of ss2DNS, the `response_secrecy` lemma implicitly ensures the secrecy of the response encryption keys; if the response encryption key becomes known to the adversary, this property would not hold, as the adversary would then be able to decrypt and learn the response. Thus, *response encryption key secrecy* is a necessary condition for *response message secrecy* in ss2DNS.

In ss2DNS, if an adversary accesses the plaintext response, query secrecy is also compromised since the query data (*e.g.,* question and ID fields) is included in the response, per the DNS standard [17]. On the other hand, the compromise of the plaintext query does not directly lead to the compromise of the response. This is because the keys used to encrypt queries and responses are distinct, and the response section of data is not included in the query messages. However, given that DNS data is generally public, an adversary with knowledge of the query's question might independently resolve the same query to determine the corresponding response. Thus, while the adversary cannot directly extract the response from ss2DNS queries, access to plaintext queries enables inference of the response through independent resolution. This inherent characteristic of DNS highlights the importance of the confidentiality of both queries and responses, as the lack of secrecy in one may render the secrecy of the other futile.

4.2 Forward Secrecy

In ss2DNS, response session keys are derived from ephemeral keys of both resolvers and nameservers. As a result, these session keys satisfy forward secrecy, meaning that even if the long-term zone and nameserver signing keys (`LTSK` and `STSK`) are compromised, the session keys from past DNS resolutions remain unknown to the adversary. As shown in Listing 1.3, the forward secrecy lemma is formally defined and proved in our Tamarin

model. For all protocol behaviors, when a resolver receives a response encrypted with a session key at timepoint i (modeled by `RKSecret(Rkey) action`) and none of the private agreement keys are compromised at any time, and also none of the long-term signing keys are compromised before the response is received (lines 4 and 5), then there is no timepoint j at which the adversary knows the session key.

As stated in lines 4 and 5, the long-term keys LTSK and STSK must not be compromised before the response is received and captured by `RKSecret()` fact at timepoint i. Therefore, by adding (`t1 < i` and `t2 < i`) to the conditions, we are allowing the possibility that the adversary can know LTSK and STSK after the response is received at i. If the signing keys LTSK and STSK are compromised before the response is received, and the adversary has access to a query, the adversary could use these keys to impersonate the zone or nameserver and inject a false response; the encryption key would then be known to the adversary, violating both secrecy and forward secrecy of the response encryption keys. Thus, compromising query secrecy and one of the signing keys undermines the forward secrecy of response keys and secrecy of responses.

```
1 lemma response_sesskey_FwdSecrecy:
2  "All Rkey #i .
3   RKSecret(Rkey) @i & Role('R') @i &
4   not (Ex Z #t1 . RevLTSK(Z) @t1 & t1 < i) &
5   not (Ex N #t2 . RevSTSK(N) @t2 & t2 < i) &
6   not (Ex Z #t3 . RevLTAK(Z) @t3) &
7   not (Ex R #t4 . RevEphR(R) @t4) &
8   not (Ex N #t5 . RevEphN(N) @t5)
9   ==> not (Ex #k. K(RKey) @k)"
```

Listing 1.3. Lemma for response key forward secrecy (verified by Tamarin)

4.3 Unilateral Authentication

Nameserver unilateral authentication is formally defined as a unilateral injective agreement [14,20], as illustrated in the lemma in Listing 1.4. The unilateral authentication lemma is defined as an agreement on a specified set of values, including keys and identities, in the matching runs between resolvers and nameservers [14,7], and then proved using Tamarin. It asserts that for all protocol behaviors, each `Commit` action by the resolver upon receiving a response with the specified data implies that, if none of the private agreement and signing keys within the protocol are known to the adversary, there exists a unique `Running` action (lines 10–12) performed by the nameserver with the same data. As there is a unique `Running` action preceding each `Commit` action, ss2DNS effectively prevents replay attacks by utilizing a fresh nonce for each response, which is included in the agreed upon `data` in this lemma.

The proof of the authentication property in Listing 1.4 guarantees that the run between the resolver and nameserver is unique (not replayable) and they have the same view of the data exchanged (one-way injective agreement). However, this lemma does not aim to validate that `RES2` is equal to `RES`, as unilateral authentication implies that only the resolver authenticates the nameserver (*i.e.,* `NS` in lines 3 and 10 are equal), and the nameserver does not explicitly check the identity of the resolver. This

```
lemma unilateral_injective_agreement:
 "All RES NS data #i.
  Commit(RES, NS, 'Resolver', data) @i &
  not (Ex Z #t1 . RevLTAK(Z) @t1) &
  not (Ex R #t2 . RevEphR(R) @t2) &
  not (Ex N #t3 . RevEphN(N) @t3) &
  not (Ex Z #t4 . RevLTSK(Z) @t4) &
  not (Ex N #t5 . RevSTSK(N) @t5)
 ==> (Ex RES2 #j. Running(NS, RES2, 'NS', data) @j & j < i &
 not (Ex RES3 NS2 #i2. Commit(RES3, NS2, 'Res', data) @i2
  & not (i2 = i)))"
```

Listing 1.4. Lemma for unilateral nameserver authentication (verified by Tamarin)

is because in ss2DNS, only resolvers are responsible for authenticating nameservers, whereas the nameservers do not authenticate resolvers.

Summary. In this section, we formally modeled the security and privacy properties of ss2DNS by defining their lemmas. Subsequently, we used Tamarin to automatically construct proofs for each property in the presence of the Dolev-Yao adversary. The analysis did not yield any counterexamples indicating potential attacks, and the proof for each lemma was generated in Tamarin.

5 Implications of Key and Entity Compromises

In Section 4, we defined and proved the properties of ss2DNS under the assumption that if none of the agreement or signing keys within the protocol are compromised, then the defined properties for the protocol are valid and verifiable in Tamarin. In this section, we investigate the implications of key compromises in ss2DNS using Tamarin. To this end, we allow each agreement or signing private key within the protocol to be compromised (become known to the adversary) and then determine whether the properties from Section 4 still hold. Subsequently, we extend our analysis for multiple protocol keys being compromised, modeling the compromise of different entities within ss2DNS. As an example, we find that if resolvers are compromised, only the ephemeral private agreement key of the resolvers (`ephR`) will be compromised.

5.1 Single Key Compromises

Long-term Agreement Key (`LTAK`): The long-term agreement key of a zone (`LTAK`) is used by resolvers to derive query encryption keys. We allow this key to become known to the adversary and retry to generate the proofs for the lemmas that were proved in Section 4. As summarized in Table 2, if the `LTAK` is compromised, only the *query secrecy* property can no longer be proved, and Tamarin identifies a corresponding attack. Using the `LTAK`, with the resolver's nonce and public agreement key, an adversary can derive the query encryption key and thereby decrypt the query. However, since the remaining properties are related to responses and the response encryption keys are different from those for queries, the other properties remain valid.

Long-term Signing Key (`LTSK`): The long-term signing key (`LTSK`) of a zone is used by the key server to sign the public signing keys of the nameservers as part of

the zone-side delegation process, thereby enabling the nameservers to respond to client queries with verifiable authenticity. We allow this key to become known to the adversary and attempt to recreate the proofs from Section 4. The compromise of LTSK alone does not undermine the *query secrecy* property, as it does not impact the confidentiality of queries; thus Tamarin successfully proves it. Similarly, *response secrecy* remains intact and can be proved by Tamarin, since an adversary possessing the LTSK cannot decrypt responses, nor can they generate or inject false responses without access to the corresponding query, which must be included within the response. Furthermore, *response key forward secrecy* remains verifiable in Tamarin, since compromising LTSK does not affect the private agreement keys used for response encryption. An adversary must access the query to impersonate a nameserver and inject a false response.

Finally, when the LTSK is compromised, the resolver can still authenticate the nameserver, preserving the validity of the *entity authentication* property in Tamarin. Since the adversary cannot access the query information and response encryption keys, the adversary cannot compromise the authenticity of the nameserver responses received by legitimate resolvers (as proved by Tamarin).

Short-term Signing Key (STSK): Similar to the long-term signing key (LTSK), as proved by Tamarin, the compromise of private (STSK) does not impact the *query secrecy* property, as this key is not utilized for query encryption. Furthermore, the compromise of a nameserver's STSK alone does not undermine the proof of *response secrecy* in Tamarin. This is because the STSK is not directly used for encrypting responses, and without access to the query data embedded in responses, an adversary cannot generate and inject false responses that are acceptable and known to the adversary.

Regarding the *response forward secrecy* property, possession of the STSK alone does not allow the adversary to compromise the forward secrecy of response keys. Additionally, the adversary cannot impersonate the nameserver, as the query data is still required to generate responses, and this data cannot be accessed by the attacker solely by having access to STSK. Regarding *entity authentication*, it remains verifiable as the adversary does not have access to the query to generate false responses, and also does not have access to response encryption keys to compromise the authenticity of the legitimate nameserver responses directly.

Ephemeral Nameserver Agreement Key (ephN): ephN is the ephemeral private agreement key used on the nameserver-side to derive the response encryption key. Compromise of this key directly undermines the *response secrecy* property in Tamarin, enabling the adversary to derive the response encryption key. Additionally, as the query data is embedded within responses, this compromise would also undermine the *query secrecy* as the adversary obtains query information from its corresponding response.

Regarding the *response key forward secrecy*, the compromise of the ephemeral agreement keys renders the response encryption key accessible to the adversary, thereby violating forward secrecy regardless of the secrecy of the long-term keys. Furthermore, possession of the response encryption key allows the adversary to directly violate the *nameserver authentication* property (*e.g.,* by modifying responses).

Ephemeral Resolver Agreement key (ephR): ephR is the ephemeral private agreement key used for deriving both query and response encryption keys. Thus, if the ephR of resolvers are known to the adversary, then both the *query secrecy* and *response secrecy* would be directly compromised. Additionally, as the response encryption key becomes known to the adversary, the *response key forward secrecy* property does not hold regardless of the secrecy of the long-term keys. Finally, the *nameserver authentication* would also be compromised if ephR is compromised, as an adversary knowing this key can directly modify the nameserver-generated responses.

Table 2. Properties retained under compromise. Columns represent the defined protocol properties, and rows show the keys and entities within ss2DNS. Each table cell indicates whether a property still holds if the corresponding key or entity is compromised. (Keys compromised due to entity compromises in ss2DNS—*Nameserver*: LTAK, STSK, and ephN; *Key server*: LTAK and LTSK; *Resolver*: ephR)

Compromised key/entity	Query Secrecy	Response Secrecy	Response Key Fwd Secrecy	Nameserver Authentication
LTAK	✗	✓	✓	✓
LTSK	✓	✓	✓	✓
STSK	✓	✓	✓	✓
ephN	✗	✗	✗	✗
ephR	✗	✗	✗	✗
Nameserver	✗	✗	✗	✗
Key server	✗	✗	✗	✗
Resolver	✗	✗	✗	✗

5.2 Entity Compromises

We now explain the implication if standalone DNNSEC+ entities, namely the resolver, nameserver, and key server, become compromised or untrusted.

Resolver: In ss2DNS, the only private key that resolvers have access to is the ephemeral private agreement keys (`ephR`) they generate. Consequently, if resolvers are compromised, the adversary would gain access to `ephR`. The impact of a resolver compromise on the defined security properties is therefore equivalent to the compromise of `ephR` itself. Thus, all of the defined query and response properties would be violated by the attacker if resolvers are compromised, as shown in Table 2.

Nameserver: In ss2DNS, nameservers have access to the zone's long-term private agreement key (`LTAK`), their short-term signing key (`STSK`), and ephemeral agreement keys (`ephN`). Consequently, if nameservers are compromised, all these keys become known to the adversary. Possession of the ephemeral agreement key (`ephN`) alone enables the adversary to compromise all defined security properties (as shown in Table 2), regardless of the secrecy of the other keys. Therefore, the compromise of a nameserver renders all the properties ineffective.

Key Server: In ss2DNS, key servers are responsible for generating and managing the long-term agreement key of a zone (`LTAK`), which is used for deriving query encryption keys. Furthermore, key servers also manage the long-term signing key of the zone (`LTSK`), which facilitates the delegation of authorization to the nameservers within a zone. Compromise of the `LTAK` enables an adversary to decrypt queries, thereby undermining the *query secrecy* property. Access to plaintext queries, combined with the compromised `LTSK`, allows the adversary to impersonate the zone or associated nameservers and inject false responses. The compromise of these two long-term keys also enables an adversary to impersonate a zone and to inject false responses using arbitrary response encryption keys, violating both *response secrecy* and *response key forward secrecy*. Additionally, the ability to impersonate nameservers and to inject false responses violates the *nameserver authentication* property.

As shown in Table 2's last three rows, compromising any entity in the ss2DNS resolution process defeats all four properties proved in Section 4. Aside from securing

such entities from attacks that expose private keys, confidentiality and authenticity of communications between all ss2DNS entities are required (otherwise, all four properties could analogously be defeated). This includes the delegation-related message exchanged between each zone's ANSes and key server, and data exchanged (*e.g.,* public keys for long-term key updates; see Section 2.2) between parent and child zones.

6 Discussion

Query Compromise. The compromise of a query in ss2DNS alone does not result in the adversary gaining knowledge of its corresponding response. However, since DNS records are not typically secret, an adversary who knows the query can resolve it to obtain its corresponding response. This is not a flaw in ss2DNS, but an inherent property of DNS. While the lack of query confidentiality in DNS allows adversaries to independently resolve the same queries, DNS responses to a single query can vary in specific scenarios. For example, responses may differ when Content Delivery Networks (CDNs) return responses based on the geographic/network location of the query's source IP address, or scenarios where different load-balancing mechanisms are employed.

Additionally, some nameservers are configured to provide different responses based on query metadata and querent profile, including the transport layer protocol, source address, and other metadata parameters. For example, a nameserver might whitelist "`ANY`" queries to specific network locations or addresses. Such configurations may result in different DNS responses for identical queries. Consequently, even when queries are not encrypted, an adversary resolving the same query may not receive a response identical to the one provided to the original querent.

Secrecy in Tamarin. In Tamarin, secrecy is defined as the adversary's lack of knowledge of a specific term, such as a message or a key. In our model, an adversary can gain such knowledge in two main ways: by accessing the plaintext of encrypted messages from a legitimate protocol execution or by impersonating a protocol entity and injecting a forged response, thereby knowing its plaintext. The latter does not constitute a direct secrecy attack but rather an impersonation attack; consequently, the encrypted legitimate protocol interactions are not necessarily compromised. Nevertheless, the adversary's ability to inject an arbitrary message that it is aware of represents a violation of the message secrecy. In this paper, we prove message secrecy in Tamarin's conventional manner and have considered both scenarios as violations of secrecy.

Model Details. Tamarin has a pre-computation phase to determine the sources of protocol and intruder facts to reuse them in later analysis. However, for certain protocols, Tamarin may fail to correctly identify the sources of specific facts, resulting in an incomplete pre-computation phase. To address this issue, users must manually define a special type of lemma, referred to as a "source lemma", which is considered an invariant, to explicitly clarify the origins of values for which Tamarin lacks sufficient information. Alternatively, users can use `auto-sources` to automatically resolve partial deconstructions. In our work, we defined and used a source lemma to enable Tamarin to successfully complete the pre-computation phase. This source lemma and other details defined in the model are included in the publicly available source code of our theory.[2]

[2] `https://github.com/Ali-Jahromi/FormalAnalysis-ss2DNS`

7 Conclusion

In this paper, using a symbolic model for the secure delegation mechanism and the name resolution process of ss2DNS in the Tamarin prover, we formally verified four security and privacy properties of ss2DNS in its privacy-enforcing mode. The results demonstrate that under the specified assumptions, the name resolution process in ss2DNS satisfies the defined security and privacy properties. We then analyzed the impact of key and entity compromises on these properties. The analysis demonstrated that the compromise of any single signing key within zones does not affect any of the four properties. However, as expected, the compromise of core entities of ss2DNS (*e.g.,* malware on ANSes, key servers, or resolvers) within the protocol can undermine all four security and privacy properties.

Acknowledgments. The last two authors acknowledge support from Natural Sciences and Engineering Research Council of Canada (NSERC) through Discovery Grants.

A Modeling ss2DNS Using Tamarin Rules

A.1 The Secure Delegation Rules

The four rules from Listing 1.5 to 1.8 demonstrate the process of secure delegation by the key server of the root zone to the ANSes of the zone, by signing their short-term signing key (`STSK`) using the zone's long-term signing key (`LTSK`) as described in Section 3.3.

```
rule Key_Server_Init: \\Initializing the key server
    [ Fr(~LTSK) \\Long-term signing key
    , Fr(~LTAK) \\Long-term agreement key
    ]
    --[OnlyOnce('Key_Server_Init') \\At most one key server
    ,  KeyServerInit($K, ~LTSK, pk(~LTSK)) \\Excutability
    ]->
    [ !ZKS_Sk($K, ~LTSK) \\Private LTSK
    , !ZKS_Pk($K, pk(~LTSK)) \\Public LTSK
    , !ZKA_Sk($K, ~LTAK) \\Private LTAK
    , !ZKA_Pk($K, 'g' ^ ~LTAK) \\Public LTAK
    , Out(<pk(~LTSK), 'g' ^ ~LTAK>) \\Publish public keys
    ]
```

Listing 1.5. Key server initialization.

A.2 DNS Resolution

Listings 1.9 to 1.11 demonstrate the three steps of the DNS resolution process between a resolver and a nameserver of the root zone in the privacy-enforcing mode, as described in Section 3.3.

```
rule NS_Sign_Request:
    [ Fr(~STSK) \\Short-term signing key
    , Fr(~nid) \\nameserver ID
    ]
    --[ NS_Sign_Req($N, ~nid, ~STSK) \\Excutability ]->
    [ NS_Wait_Sig(< $N, ~nid, ~STSK, pk(~STSK)>)
    , NS_Sends_to_Sign(<$N, ~nid, pk(~STSK)>)  \\Secure OOB channel
    ]
```

Listing 1.6. Nameserver requests the signing of its STSK.

```
rule Key_Server_Signs_NS:
    [ !ZKS_Sk($K, LTSK) \\Private LTSK
    , NS_Sends_to_Sign(< $N, nid, stpk >)\\Sign request from OOB
    ]
    --[ KS_Signs($N, nid, stpk, sign(<$N, nid, stpk>, ltkKS)) ]->
    [ Signed_NS_key(sign(<$N, nid, stpk>, ltkKS), <$N, nid, stpk> )
    ]
```

Listing 1.7. Key Server signs the STSK of the nameserver.

```
rule NS_Receives_Signed:
    [ NS_Wait_Sig(< $N, nid, stsk, stpk >) \\Waiting for delegation
    , Signed_NS_key(sig_stpk, <$N, nid, stpk>) \\Signature from OOB
    , !ZKS_Pk($K, pkLTSK) \\Public LTSK
    ]
    --[
      Eq(verify(sig_stpk, <$N, nid, stpk> , pkLTSK), true)
    , NS_Rcv_Signed(sig_stpk, <$N, nid, stpk>)
    ]->
    [ !NS_sstk_Signed_PK(sig_stpk, <$N, nid, stpk>)
    , !NS_sstk_Signed_SK(sig_stpk, <$N, nid, stpk>, stsk)
    , Out(< stpk, sig_stpk >) \\Public public STSK and signature
    ]
```

Listing 1.8. Nameserver receives its signed STSK.

```
rule Res_1:
        let
          queryKey = kdf(< ~NCR, ZKAP ^ ~ephR >)
          query_data = < ~query, 'Query' >
        in
        [ !ZKA_Pk($K, ZKAP) \\Public LTAK
        , Fr(~ephR) \\Fresh ephemeral agreement key
        , Fr(~query) \\Fresh query
        , Fr(~NCR) \\Fresh nonce
        ]
        --[
          ResolverSentQuery($R, ~query, ~ephR) \\Executability
        , Role('R') \\Resolver role
        , QKeySecret($R, ~ephR, queryKey) \\Query key secrecy
        , QDataSecret($R, query_data) \\Query data secrecy
        ]->
        [ Out( <~NCR, 'g' ^ ~ephR, senc(query_data, queryKey)> )
        , Res_State_1($R, ~ephR, ~query, ~NCR)
        , !EskR($R, ~ephR)
        ]
```

Listing 1.9. Resolver Sends Query

```
rule NS_1:
    let
      queryKey = kdf(<NCR, epkR ^ ZKA>)
      RespKey = kdf(< ~NCN, NCR, epkR ^ ~ephN >)
      query_data = < query, 'Query' >
      ephSig = sign('g' ^ ~ephN, STSK)
      response_data = < 'Response', ~response, query, $N, nid, stpk
    , ephSig, stk_sig >
    in
    [ !ZKA_Sk($K, ZKA) \\Private LTAK
    , In(<NCR, epkR, senc(query_data, queryKey)>) \\Receiving query
    , Fr(~ephN) \\Fresh ephemearl agreement key
    , Fr(~NCN) \\Fresh nonce
    , Fr(~response)
    , !NS_sstk_Signed_SK(stk_sig, <$N, nid, stpk>, STSK)]
    --[
      Role('N') \\Nameserver role
    , NS_Sends_Resp($R, $N, query, 'Response', nid, STSK, stk_sig)
    , Running($N, $R, 'NS', < query, RespKey >)
    ]->
    [ !EskN($N, ~ephN)
    , Out(< ~NCN, 'g' ^ ~ephN, senc( response_data, respKey)>)
    ]
```

Listing 1.10. Nameserver Responds a Query

```
rule Res_2:
    let
      ResponseKey = kdf(<Nnc, Rnc, epkN ^ ephR>)
      response_data = <'Response', response, query, $N, nid, stpk,
    sig_epkN, stk_sig>
    in
    [
      In(<Nnc, epkN, senc(response_data, ResponseKey)>) \\Response
    , Res_State_1($R, ephR, query, Rnc) \\Query information
    , !ZKS_Pk($K, ltkPK) \\LTSK public key
    ]
    --[
      Eq(verify(sig_epkN, epkN, stpk), true) \\Verify sig
    , Eq(verify(stk_sig, <$N, nid, stpk>, ltkPK), true)\\Verify sig
    , Role('R') \\Resolver role
    , SecretR(response_data)
    , ResolverReceivesResponse( response, query)
    , RKeySecret(ResponseKey)
    , Commit($R, $N, 'Res', <query, ResponseKey>)
    ]-> []
```

Listing 1.11. Resolver Receives Response

References

1. Ariyapperuma, S., Mitchell, C.J.: Security vulnerabilities in DNS and DNSSEC. IEEE Conference on Availability, Reliability and Security (IEEE ARES). (2007).
2. Atkins, D., Austein, R.: Threat analysis of the Domain Name System (DNS). Request for Comments (RFC3833). IETF RFC Editor (2004).
3. Basin, D., Dreier, J., Hirschi, L., Radomirovic, S., Sasse, R., Stettler, V.: A formal analysis of 5G authentication. ACM SIGSAC Conference on Computer and Communications Security. (2018).
4. Bernstein, D.J.: DNSCurve: Usable security for DNS. (2009). Available: `https://dnscurve.org`
5. Cremers, C., Horvat, M., Scott, S., van der Merwe, T.: Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. IEEE Symposium on Security and Privacy (S&P). (2016).
6. Dai, T., Jeitner, P., Shulman, H., Waidner, M.: From IP to transport and beyond: Cross-layer attacks against applications. In: ACM SIGCOMM Conference. (2021).
7. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. In: Designs, Codes and Cryptography. 2(2): 107-125 (1992).
8. Dolev, D., Yao, A.: On the security of public key protocols. In: IEEE Transactions on Information Theory. 29(2): 198-208 (1983).
9. Duan, H., Fischer, R., Lou, J., Liu, S., Basin, D., Perrig, A.: RHINE: Robust and high-performance Internet naming with E2E authenticity. USENIX NSDI. (2023).
10. Grothoff, C., Wachs, M., Ermert, M.: NSA's MORECOWBELL: Knell for DNS. (2017). Technical report.
11. Hao, S., Zhang, Y., Wang, H., Stavrou, A.: End-users get maneuvered: Empirical analysis of redirection hijacking in content delivery networks. In: USENIX Security. (2018).
12. Herzberg, A., Shulman, H.: Fragmentation considered poisonous, or: One-domain-to-rule-them-all.org. In: IEEE Conference on Communications and Network Security (CNS). (2013).
13. Kaminsky, D.: Black ops 2008: Its the end of the cache as we know it. In: Black Hat USA. (2008).
14. Lowe, G.: A hierarchy of authentication specifications. In: Proceedings of the 10th Computer Security Foundations Workshop. IEEE (1997).
15. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The Tamarin prover for the symbolic analysis of security protocols. Computer Aided Verification (CAV). (2013).
16. Mockapetris, P.: Domain names - Concepts and facilities. In: Request for Comments (RFC1034). IETF RFC Editor (1987).
17. Mockapetris, P.: Domain names - Implementation and specification. In: Internet Requests for Comments (RFC1035). IETF RFC Editor (1987).
18. Sadeghi Jahromi, A., Abdou, A., van Oorschot, P.C.: ss2DNS: A Secure DNS Scheme in Stage 2. 2025. Under submission; technical report available at: `https://arxiv.org/abs/2408.00968v2`.
19. Schmidt, B., Meier, S., Cremers, C., Basin, D.: Automated analysis of Diffie-Hellman protocols and advanced security properties. In: Computer Security Foundations Symposium (CSF). IEEE (2012).
20. Wilson, J., Asplund, M., Johansson, N.: Extending the authentication hierarchy with one-way agreement. In: Computer Security Foundations Symposium (CSF). IEEE (2023).