DNSSEC Oracle Audit

This review was performed upon the commit hash b211bd2.

This code depends on the specifications written up in various RFCs including but not limited to:

- RFC4034
- RFC4035
- RFC4509
- RFC5155
- RFC5702

Complexity

The complexity of this Oracle code should not be underestimated, therefore I suggest deployment on a testnet and attempting to run the code with as many records as possible.

Graphs

Inheritance Tree

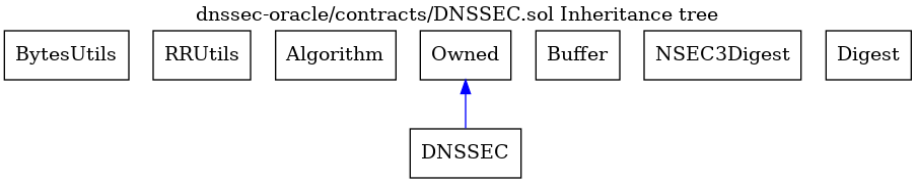

Figure 1: DNSSEC Inheritance Tree

State Graph

Displays which functions directly modify which state variables.

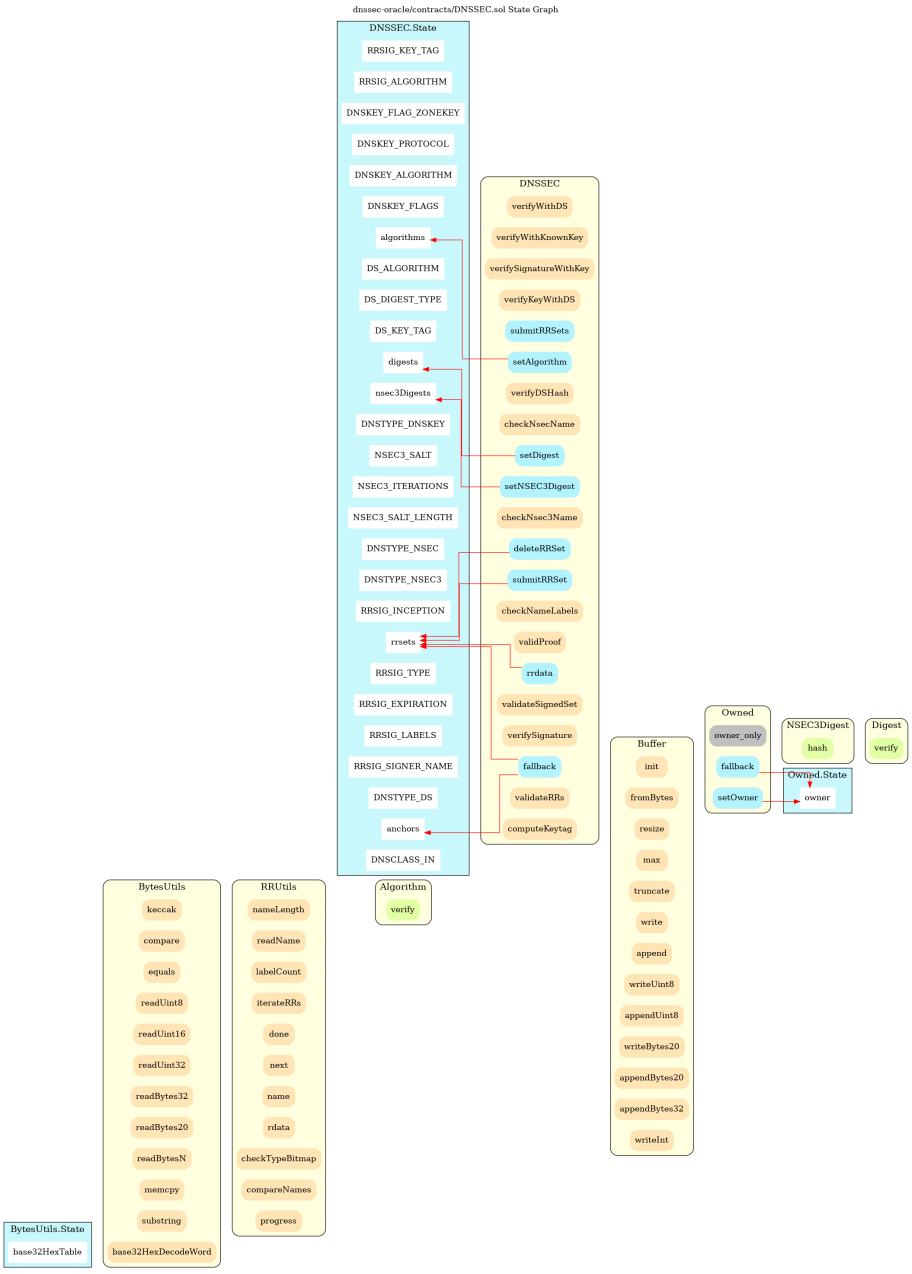

Figure 2: DNSSEC State Graph

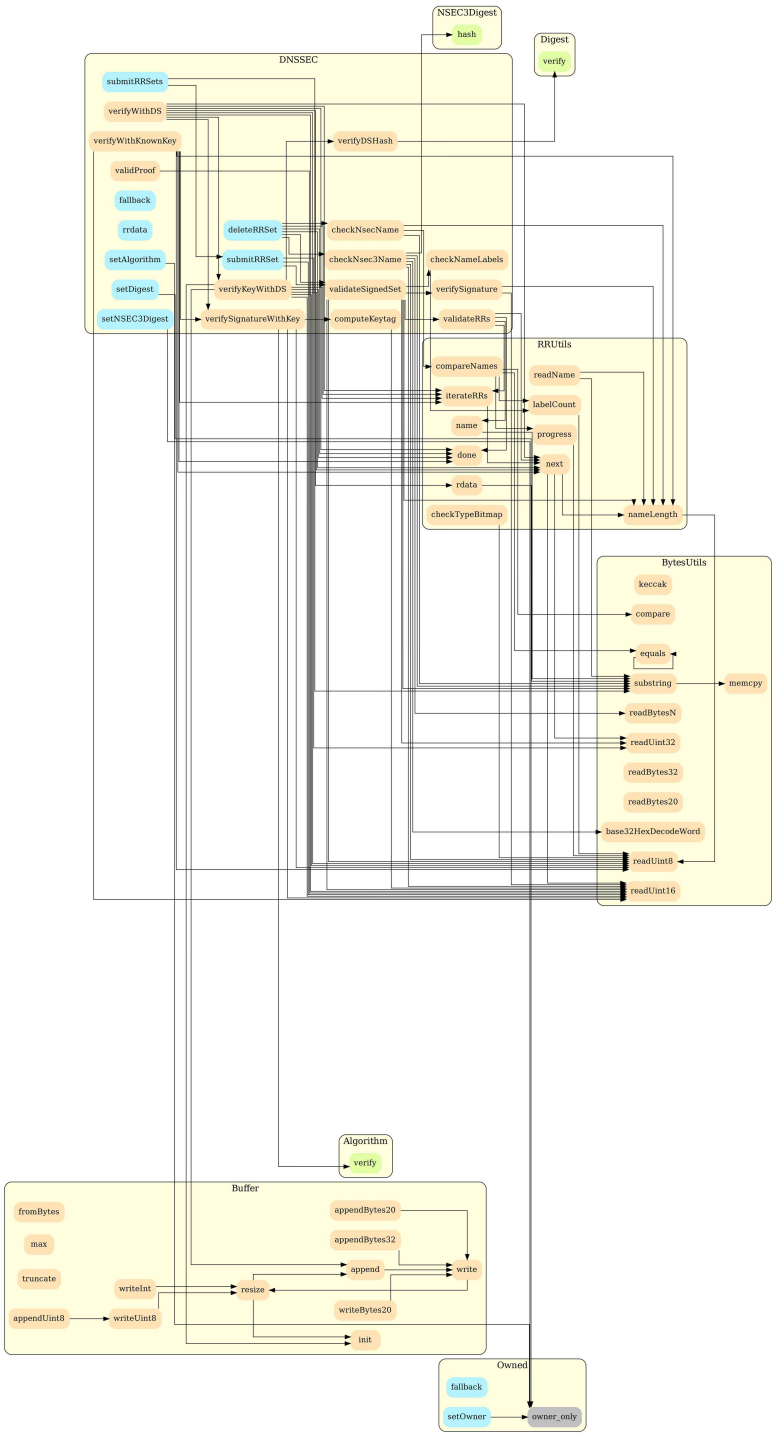

Figure 3: DNSSEC Call Graph

Call Graph

DNSSEC.sol

Constants

- `DNSCLASS_IN` is set to `1` as defined in RFC1035
- `ALGORITHM_RSASHA256` is set to `8` as defined in RFC5702
- `DIGEST_ALGORITHM_SHA256` is set to `2` as defined in RFC4509

DNS Types

- `DNSTYPE_DS` is set to `43`
- `DNSTYPE_RRSIG` is set to `46`
- `DNSTYPE_NSEC` is set to `47`
- `DNSTYPE_DNSKEY` is set to `48`

The DNS Type IDs all match those defined in RFC4034.

- `DNSTYPE_NSEC3` is set to `50` as defined in RFC5155

DS Wire Format

Various constants are set to help with the parsing of RDATA for DS Resource Records.

- `DS_KEY_TAG` is set to `0`, representing the index of the 2 octet Key Tag field.
- `DS_ALGORITHM` is set to `2`, representing the index of the 1 octet Algorithm field.
- `DS_DIGEST_TYPE` is set to `3`, representing the index of the 1 octet Digest Type field.
- `DS_DIGEST` is set to `4`, representing the index of the Digest field.

The Wire Format is defined in RFC4034.

RRSIG RDATA Wire Format

Various constants are set to help with the parsing of RDATA for RRSIG Resource Records.

- `RRSIG_TYPE` is set to `0`, representing the index of the 2 octet Type Covered field.
- `RRSIG_ALGORITHM` is set to `2`, representing the index of the 1 octet Algorithm field.
- `RRSIG_LABELS` is set to `3`, representing the index of the 1 octet Labels field.
- `RRSIG_TTL` is set to `4`, representing the index of the 4 octet Original TTL field.

- `RRSIG_EXPIRATION` is set to `8`, representing the index of the 4 octet Signature Expiration field.
- `RRSIG_INCEPTION` is set to `12`, representing the index of the 4 octet Signature Inception field.
- `RRSIG_KEY_TAG` is set to `16`, representing the index of the 2 octet Key tag.
- `RRSIG_SIGNER_NAME` is set to `18`, representing the index of the Signer's Name field.

Note

- A constant for the index of the Signature field is not defined.

The Wire Format is defined in RFC4034.

DNSKEY Wire Format

Various constants are set to help with the parsing of RDATA for DNSKEY Resource Records.

- `DNSKEY_FLAGS` is set to `0`, representing the index of the 2 octet Flags Field.
- `DNSKEY_PROTOCOL` is set to `2`, representing the index of the 1 octet Protocol Field.
- `DNSKEY_ALGORITHM` is set to `3`, representing the index of the 1 octet Algorithm Field.
- `DNSKEY_PUBKEY` is set to `4`, representing the index of the Public Key Field.

The Wire Format is defined in RFC4034.

NSEC3 Wire Format

Various constants are set to help with the parsing of RDATA for NSEC3 Resource Records.

- `NSEC3_HASH_ALGORITHM` is set to `0`, representing the index of the 1 octet Hash algorithm Field.
- `NSEC3_FLAGS` is set to `1`, representing the index of the 1 octet Flags Field.
- `NSEC3_ITERATIONS` is set to `2`, representing the index of the 2 octet Iterations Field.
- `NSEC3_SALT_LENGTH` is set to `4`, representing the index of the 1 octet Salt Length Field.
- `NSEC3_SALT` is set to `5`, representing the index of the 1 Salt Field.

The Wire Format is defined in RFC5155.

Constructor

This function sets the root DS resource record to the DNSSEC trust anchor. The `RRSetUpdated` event is then emitted.

setAlgorithm

Adds an `algorithm` to the algorithms mapping.

setDigest

Adds a `digest` to the digests mapping.

setNSEC3Digest

Adds a `digest` to the NSEC3 digests mapping.

submitRRSets

This function allows for multiple RRSets to be submitted, each chunk must be submitted in the format `<uint16 length><bytes input><uint16 length><bytes sig>`.

A while loop is executed, where each seperate chunk is parsed. The result is then used as input variables for a call to the `submitRRSet` function.

Finally the last RRSet submitted is returned, **is this necessary?**

submitRRSet

This function allows the submission of a is RRSet as described in RFC4035.

Initially this function calls the `validateSignedSet` function, which in turn returns the `name` and `rrs`.

Next the `inception` field is read, this is a `uint32` found at the `RRSIG_INCEPTION` offset. After this, the `typecovered` field is read, this is a `uint16` found at the `RRSIG_TYPE` offset.

The `RRSet` is loaded from storage, if it has already been inserted, it is ensured that the current `inception` is greater than or equal to the one which has been previously stored.

If the previously stored hash is equal to the hash of the `rrs`, the function returns as the set has already been inserted.

The new `RRSet` is then inserted, and the `RRSetUpdated` event is emitted.

deleteRRSet

Initially this function calls the `validateSignedSet` function, which in turn returns the `name` and `rrs`.

It is then ensured that the `inception` of the set to delete, is smaller than or equal to that present in the passed `nsec` set.

If the dns type of the `RRSet` is `NSEC`, then the `checkNsecName` function is called. If it is `NSEC3` then the `checkNsec3Name` function is called. Otherwise a `revert` is raised.

Finally the `RRSet` is deleted.

Suggestions

- Do not use an iterator, as it makes the code more confusing. Only one iteration is possible anyway.
- `revert` on `L235` can be removed, as it will never be reached.

checkNsecName

This function first ensures that the name length of the `RDATA` is equal to the data length. It then compares the `deleteName` to the `nsecName`. If the `deleteName` is the same as the `nsecName`, it is ensured that the delete type exists in the type bitmap. Otherwise, the next name is read from the `RDATA`, and it is ensured that the `deleteName` comes after the `nsecName`. Finally if the `compareNames` function returns a value smaller than `0`, it is required that `deleteName` comes before the next name.

Relevant sections: - RFC4034 Section 4.1 - RFC4034 Section 4.3 - RFC4034 Section 6.1

checkNsec3Name

This function first gets the calculates the `deleteNameHash`, using the `hash` function from a specified `digest`. It then reads the `Hash length` field and ensures the value is smaller than 32. The `nsecNameHash` is then read. If the `nsecNameHash` is identical to the `deleteNameHash`, it is ensured that the `deleteType` does not exist in the type bitmap. Otherwise, it is ensured that the `deleteNameHash` comes after the `nsecNameHash`. If the `nextNameHash` comes after the `nsecNameHash`, it is also ensured that the `nextNameHash` comes after the `deleteNameHash`.

Relevant sections: - RFC5155 Section 3

Suggestions

- Store `NSEC3_SALT + saltLength` in a variable, instead of recalculating it.

validateSignedSet

This function first ensures that the submitted proof is valid by calling the `validProof` function. Then the `validateRRs` function is called, the return value of this call is then used when calling the `checkNameLabels` function. Next it is ensured that the `expiration` date is later than now, and the `inception` date is before now. Finally the `verifySignature` function is called, and the `name` and `rrs` (RR Data) is returned.

validProof

Ensures that the `hash` of an `RRSet` is equal to the 20 bytes of the hashed `proof`.

validateRRs

This function validates a set of `RR`s.

It does so by iterating over the set, firstly the `RR` class is ensured to be equal to `IN (Internet)`. If the `name` variable has not been initialized, it sets it to the current name. Otherwise it is ensured that the `name` length is equal to the `name` length at the current iterator offset, and that the current `name` is equal to the `name` of the current `RR`. Finally it is ensured that the `DNS Type` is equal to the value of the passed `typecovered` variable.

Suggestions

- I personally feel as though an explicit `return` would make this function more legible.
- The length check on `L369` can probably be removed.

checkNameLabels

This function checks that the name labels count is either equal to the passed variable, or that the count is equal to `label` plus one and that the first character is equal to `*`.

verifySignature

This function runs `RRSIG` verifications. It is first ensured that the Signer's name has the same length as the passed name. It is then ensured that the Signer's name is identical to the passed name.

The `type covered` field is then read, if the `type` is equal to `DS`, then the `verifyWithDS` function is called. Otherwise, if the type is equal to `DNSKEY`, then the `verifyWithKnownKey` function is called. If it is neither, the call gets reverted.

Relevant sections: - RFC4034 Section 3

Suggestion

- The length check could potentially be eliminated, the second check would also fail if the length doesn't match up.
- Remove the magic number `18`, replace it with the `RRSIG_SIGNER_NAME` constant.

verifyWithKnownKey

This function attempts to verify an `RRSet` against an already known public key.

It does so, by iterating the `RRSet`. It ensures that the name length of the proof is equal to the signer name length in the original data. It then ensures that the names are identical. Finally it calls `verifySignatureWithKey`, and if this call returns true, the function also returns true. If all `RR`s have been iterated and `verifySignatureWithKey` has never returned true, false is returned.

Suggestion

- The length check could potentially be eliminated, the second check would also fail if the length doesn't match up.

verifyWithDS

This function attempts to verify an `RRSet` against an already known public key.

It does so, by iterating the `RRSet`. If the DNS Type is not equal to `DNSKEY`, the function returns false. The `RData` is then loaded, the `verifySignatureWithKey` function is called, if the return value is called then the `RRSet` is self-signed, and the `verifyKeyWithDS` function is called and the return value is returned. Finally, if none of the iterations return a false is returned at the end of the function.

verifySignatureWithKey

This function first ensures that there is a verifier for the specified algorithm. It then ensures that the `DNSKEY_PROTOCOL` is equal to `3`. Next it is ensured that the `DNSKEY_ALGORITHM` field in the `RDATA` is equal to the passed `algorithm` variable. Next it is ensured that the passed `keytag`, equals to the `keytag` generated by the `computeKeytag` function. It is then ensured that the Zone Flag bit exists. Finally `verify` is called on the algorithm, the result is returned.

Suggestions

- Remove the `3` on `L487` and replace it with a constant, or reference RFC4034 Section 2.1.2.

verifyKeyWithDS

This function attempts to verify a key using DS records.

It does so, by iterating the `RRSet`. If the key tag of the current data does not match the passed key card, the set is skipped. If the algorithm of the current data is not equal to the passed algorithm, the set is skipped. The `DS_DIGEST_TYPE` is then read from the data, a new buffer is created containing the key name and data. The `verifyDSHash` function is called with the newly created buffer and the digest type. If the return value is true, true is returned. Otherwise iterations continue. If true was never returned, false is returned at the end of the function.

verifyDSHash

Calls the `verify` function on a specific `digest` using the passed input parameters. If the `digest` does not exist, `false` is returned.

computeKeytag

This implementation matches the example implementation in RFC4034.

No issues were detected.