dnsprovejs Documentation

Makoto Inoue

Dec 10, 2018

Contents:

CHAPTER 1

Introduction

dnsprovejs is a tool to convince an Ethereum DNSSEC oracle of the contents of DNS records

1.1 Functionalities

- Fetches DNS information of given domain name and type
- Validates DNS reponses and constructs proofs
- Submits the proofs into DNSSEC(Domain Name System Security Extensions) Oracle smart contract
- Works from both browser and from node.js

CHAPTER 2

Quick start

2.1 Installing

```
npm install '@ensdomains/dnsprovejs' --save
```

2.2 Usage

```
var provider  = web3.currentProvider;
var DnsProve  = require('dnsprove');
var dnsprove  = new DnsProve(provider);
var textDomain = '_ens.matoken.xyz';
var dnsResult = await dnsprove.lookup('TXT', textDomain);
var oracle    = await dnsprove.getOracle('0x123...');
var proofs = dnsResult.proofs;

if(dnsResult.found){
    await oracle.submitAll(dnsResult, {from:nonOwner});
}else if (dnsResult.nsec){
    await oracle.deleteProof(
        'TXT', textDomain,
        proofs[proofs.length -1],
        proofs[proofs.length -2],
        {from:nonOwner}
    );
}else{
    throw("DNSSEC is not supported")
}
```

Alternatively, if you want to submit the proof not only to Oracle contract but also to claim via *dnsregistrar*, then you can call *getAllProofs* and pass the result into the *proveAndClaim* function.

```
let dnsResult = await dnsprove.lookup('TXT', '_ens.matoken.xyz', address);
let oracle    = await dnsprove.getOracle(address);
let data = await oracle.getAllProofs(dnsResult, params);
await registrar.methods
    .proveAndClaim(encodedName, data[0], data[1])
    .send(params)
```

2.3 DnsRegistrar

The example above demonstrated the case to call *proveAndClaim* smart contract function directly but we have a wrapper libray at DNSregistrar which calls DNSSEC Oracle and DnsRegistrar in one function call.

```
var DNSRegistrarJs = require('@ensdomains/dnsregistrar');
dnsregistrar = new DNSRegistrarJs(provider, dnsregistraraddress);
dnsregistrar.claim('foo.test').then((claim)=>{
    claim.submit({from:account});
})
```

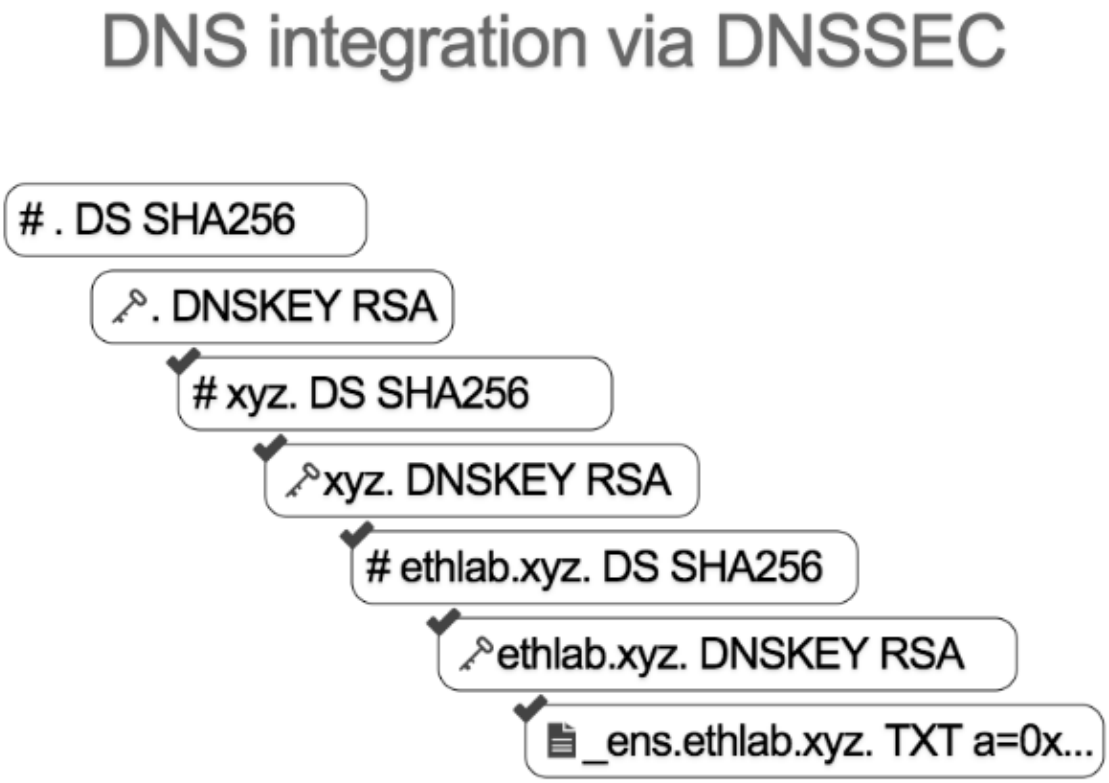
CHAPTER 3

Under the hood

3.1 How it works

DNSSEC (The Domain Name System Security Extensions) establishes a chain of trust from the root key which signed by ICANN (.) and down through each key. We start off knowing the hash of the root key of DNS (this is hard coded in the smart contract oracle). Given the hashes of that key, we can pass in the actual key, we can verify that it matches the hash and we can add it to the set of the trusted records.

Given that key, we can now verify any record that is signed with that key, so in this case, it's the hash of the root of the xyz top level domain. Given that, we can recognize the key, and so on and so forth.



So next, you can regonize the key for the ethlab.xyz. Given that, we can recognize the hash for the key, and then key itself,and finally you can verify a signed text record that contains the Ethereum address.

3.2 Constructing a proof

To construct a proof to submit to Oracle smart contract, you first need to provide name, rrsig, and rrset.

name name of the dns record

```
let name = '.'
```

rrsig Resource record digital signature

```
let sig = { name: '.',
    type: 'RRSIG',
    ttl: 0,
    class: 'IN',
    flush: false,
    data:
    { typeCovered: 'DNSKEY',
        algorithm: 253,
        labels: 0,
        originalTTL: 3600,
        expiration: 2528174800,
        inception: 1526834834,
```

(continues on next page)

(continued from previous page)

```
        keyTag: 5647,
        signersName: '.',
        signature: new Buffer([]) }
    }
```

rrset DNS resource record sets

```
let rrs = [
        {
            name: '.',
            type: 'DNSKEY',
            ttl: 3600,
            class: 'IN',
            flush: false,
            data: { flags: 257, algorithm: 253, key: Buffer.from("1111", "HEX")}
        },
        {
            name: '.',
            type: 'DNSKEY',
            ttl: 3600,
            class: 'IN',
            flush: false,
            data: { flags: 257, algorithm: 253, key: Buffer.from("1111", "HEX")}
        },
        {
            name: '.',
            type: 'DNSKEY',
            ttl: 3600,
            class: 'IN',
            flush: false,
            data: { flags: 257, algorithm: 253, key: Buffer.from("1112", "HEX")}
        }
    ]
```

Once the data is constructed, Instantiate *Result* object with the data which builds the proofs. *toSubmit* will generate the input data.

```
      const Result = require('@ensdomains/dnsprovejs/dist/dns/result')
      let result = new Result([{name,sig,rrs}])
      result.proofs[0].toSubmit()
      // [
↪'0x0030fd0000000e1096b0e2d05b01a692160f0000030000100000e100006010103fd1111000030000100000e1000060
↪',
'0x' ]
```

CHAPTER 4

Libraries

4.1 DnsProver

DnsProver allows you to lookup a domainname for a given DNS type and returns the DNS record and proofs which you can submit into DNSSEC Oracle.

`DnsProver.`**`lookup`**(*type*, *query*)

lookup takes DNS record type and name and returns *DnsResult* object.

Arguments

- **`type`** (*`string`*) – eg: TXT
- **`query`** (*`string`*) – eg: _ens.yourdomain.xyz

Returns **Object** – DnsResult - contains list of results retrieved from DNS record and proofs which are constructed from the record and used to submit into DNSSEC Oracle

`DnsProver.`**`getOracle`**(*address*)

getOracle returns Oracle object

Arguments

- **`address`** (*`string`*) – DNSSEC Oracle contract address

Returns **Object** – Oracle - allows you to call DNSSEC oracle functions

4.2 Oracle

Oracle interacts with DNSSEC Oracle smart contract.

`Oracle.`**`knownProof`**(*proof*)

kownProof

Arguments

- **`proof`** (*`Object`*) –

- **proof.name** (*string*) – eg 'ethlab.xyz'
- **proof.type** (*type*) – eg 'TXT'

Returns Object – oracle_proof - contains list of results retrieved from DNS record and proofs

Oracle.**submitAll**(*result*, *params*)

submitAll submits all required proofs into the DNSSEC oracle as one transaction in a batch.

Arguments

- **result** (*Object*) –
- **params** (*Object*) – from, gas, gasPrice, etc

Oracle.**getAllProofs**(*result*)

getAllProofs returns all the proofs needs to be submitted into DNSSEC Oracle. It traverses from the leaf of the chain of proof to check if proof in DNSSEC Oracle and the one from DNS record matches with valid inception value. This function is used so that it can pass the necessary proof to *dnsregistrar.proveAndClaim* function.

Arguments

- **result** (*Object*) –

Returns string – data

Returns Object – prevProof

Oracle.**submitProof**(*proof*, *prevProof*, *params*)

submitProof submits a proof to Oracle contract. If *prevProof* is *null*, the oracle contract uses hard-coded root anchor proof to validate the validity of the proof given. *params* is used to pass any params to be sent to transaction, such as *{from:address}*.

Arguments

- **proof** (*Object*) –
- **prevProof** (*Object*) –
- **params** (*Object*) – from, gas, gasPrice, etc

Returns boolean – success - returns true unless transaction fails

Oracle.**deleteProof**(*type*, *name*, *proof*, *prevProof*, *params*)

deleteProof deletes a proof

Arguments

- **type** (*string*) – eg: TXT
- **name** (*string*) – eg: _ens.matoken.xyz
- **proof** (*string*) –
- **prevProof** (*string*) –
- **params** (*Object*) – from, gas, gasPrice, etc

OracleProof(*proof*)

Arguments

- **proof** (*Object*) –
- **proof.inception** (*number*) – time the signature was generated
- **proof.inserted** (*number*) – time the record was inserted into DNSSEC oracle
- **proof.hash** (*string*) – hash of proof stored in DNSSEC oracle

- **proof.hashToProve** (*string*) – hash of proof constructed from DNS record
- **proof.validInception** (*boolean*) – true if inception in DNSSEC oracle is older than the one from DNS record.
- **proof.matched** (*boolean*) – true if inception is valid and hash is matched

4.3 Proof

Proof contains rrset and signature data which is submitted into DNSSEC Oracle.

Proof(*name*, *type*, *sig*, *inception*, *sigwire*, *rrdata*)

Arguments

- **name** (*string*) –
- **type** (*string*) –
- **sig** (*string*) –
- **inception** (*number*) –
- **sigwire** (*string*) –
- **rrdata** (*string*) –

Proof.**toSubmit**()

toSubmit returns an array consisting of hex string of sigwiredata (concatinatd string of sigwire and rrdata) and its signature

Returns **array** – data

4.4 DnsResult

DnsResult is an object returned by calling *lookup* function and contains information about the DNS record.

DnsResult(*dns_result*)

Arguments

- **dns_result** (*Object*) –
- **dns_result.found** (*boolean*) – true if the given record exists
- **dns_result.nsec** (*boolean*) – true if the given record does not exist and NSEC/NSEC3 is enabled
- **dns_result.results** (*Array*) – an array of SignedSet containing name, signature, and rrs
- **dns_result.proofs** (*Array*) – an array of proofs constructed using results
- **dns_result.lastProof** (*string*) – the last proof which you submit into Oracle contruct

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index