WAVE: A Peer-to-Peer Radiant Blockchain Name System

A peer-to-peer naming service would enable direct networking without going through DNS or any centrally managed name lookup service. The base Bitcoin[1] protocol provides the essential components required for a naming service that allows payments to be sent using human-friendly readable names, resolve hostnames and generally all the functions of DNS. Using a software fork of Bitcoin, called Radiant[2], we design a UTXO-based system that provides a self-evident, tamper resistant chain to track names using a prefix tree in an easy to verify manner for clients and servers. The system is secured by Radiant miners at the base layer and therefore has the identical security guarantees as any transaction with all the benefits of SPV (Simplified Payment Verification). Each name can provably only have one owner, and is represented as a Non-Fungible Token (NFT) that can be updated and transferred without any intermediaries. Radiant uniquely enables each name token to behave as a polymorphic singleton smart contract and be used and composed dynamically with other smart contracts.

1. Introduction

Internet communications relies extensively on the use of DNS and related technologies such as DNSSEC[3]. While the system works well enough under normal circumstances it has serious known weaknesses that are easy to exploit. Even in the absence of an attack, it still suffers from the inherent weaknesses of the trust based model from the need to trust root name servers. There have been many large scale attacks on DNS for the purposes of stealing cryptocurrency and other sensitive information, indeed these attacks continue to this day and are fundamentally unpreventable due to the inherent security flaws of the systems themselves.

What is needed is an electronic naming system based on cryptographic proof instead of trust, allowing any two willing parties to communicate directly with each other without the need for a trusted third party. By using proof-of-work blockchain transactions, it is computationally impractical to reverse and forge and therefore protects parties from the various attacks that DNS is susceptible to. In this paper we propose a solution to this problem by using the native scripting abilities of Radiant[2], a SHA512/256 proof-of-work UTXO-based blockchain, to generate a computational proof of ownership and authenticity. This system is secure as long as the underlying blockchain itself is secure.

Our solution offers unprecedented speed, efficiency and flexibility owing to the use of Radiant's unique reference system to create compact proofs of size `O(1)` – a single unspent transaction output. With some trade-offs the system can be augmented to operate on Bitcoin and other UTXO-based blockchains, with the loss of compact proofs, instead requiring O(n) size in the name history, and would also lack dynamic resolution in other smart contracts. Radiant has everything that is needed to make a truly global permissionless peer-to-peer name system to replace DNS and out perform every other blockchain name service in existence.

2. Name Prefix Tree

We define a name as a chain of transactions representing a branch of a prefix tree. Each letter is represented as a unique output of a transaction which stems from a root transaction. For the purposes of the design we shall stipulate there are a total of 37 supported characters (ie: The ascii characters: - (hyphen), a *to* z, and 0 *to* 9). Each transaction "fans out" and has a total of at least 37 outputs representing each character. This can be generalized to base-n and can even be adapted for spatial and multi-dimensional namespace allocations, the discussion of which is left for future avenues of research.

An extension transaction can be created for any letter by spending the associated letter's unspent output (UTXO) and reproducing a 38-output fan-out extension which acts as the next level in the prefix tree. Each unspent output is spent in exactly the same way as it's parent by repeating the 38 output fan-out pattern. To be exact, a special 0th output must be created at each level called the "Claim Token" in addition to the 37 outputs representing each character.

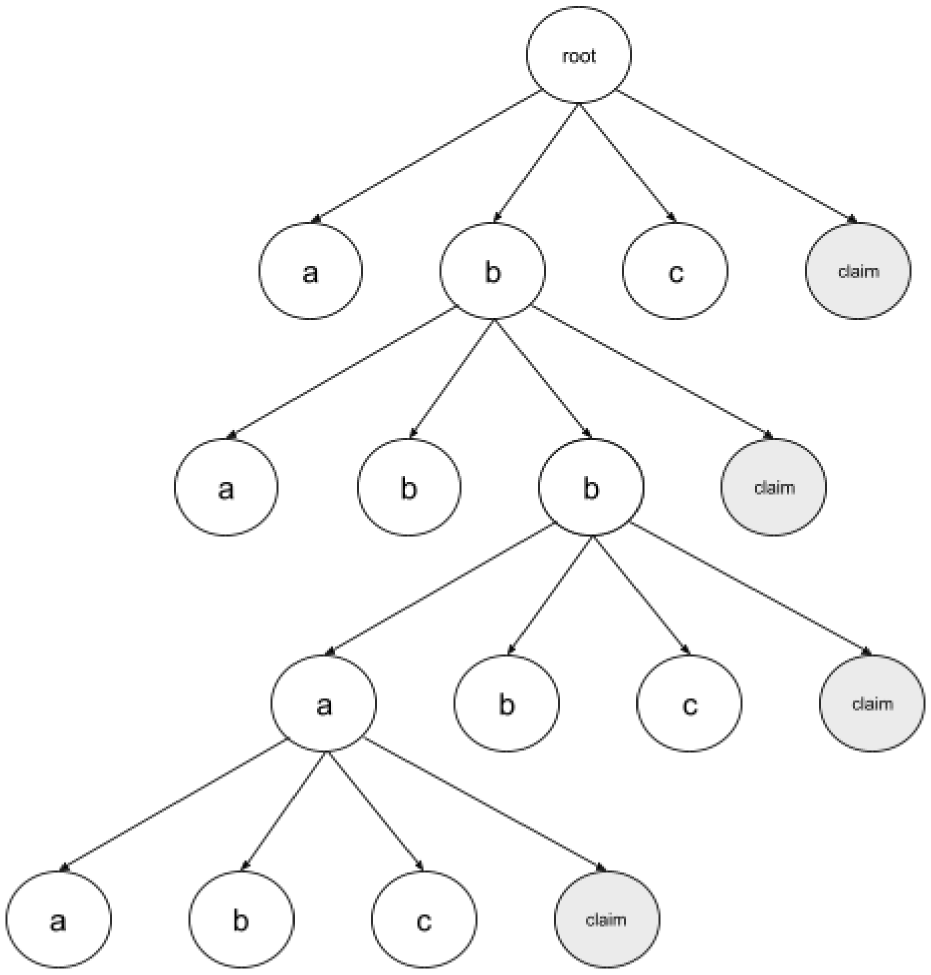

Figure 1. Name Prefix Tree. Each node is represented by UTXO. All nodes are extension nodes and must replicate the "fan-out" character pattern. The claim UTXOs (shaded) represents name ownership. Only characters a, b, and c are illustrated for simplicity.

Users can extend any node and supply funding inputs to incentivize miners to process transactions. Each level contains exactly one Claim Token. In order for a user to claim a name, they must supply adequate funding inputs to lock in a non-zero amount of Satoshis into the Claim Token for the lifetime of the name which helps prevent users from buying up the entire namespace by imposing a cost.

Resolve names: follow 'root' to each letter down to the claim output. **Only letters 'a', 'b', 'c' are shown for brevity.**

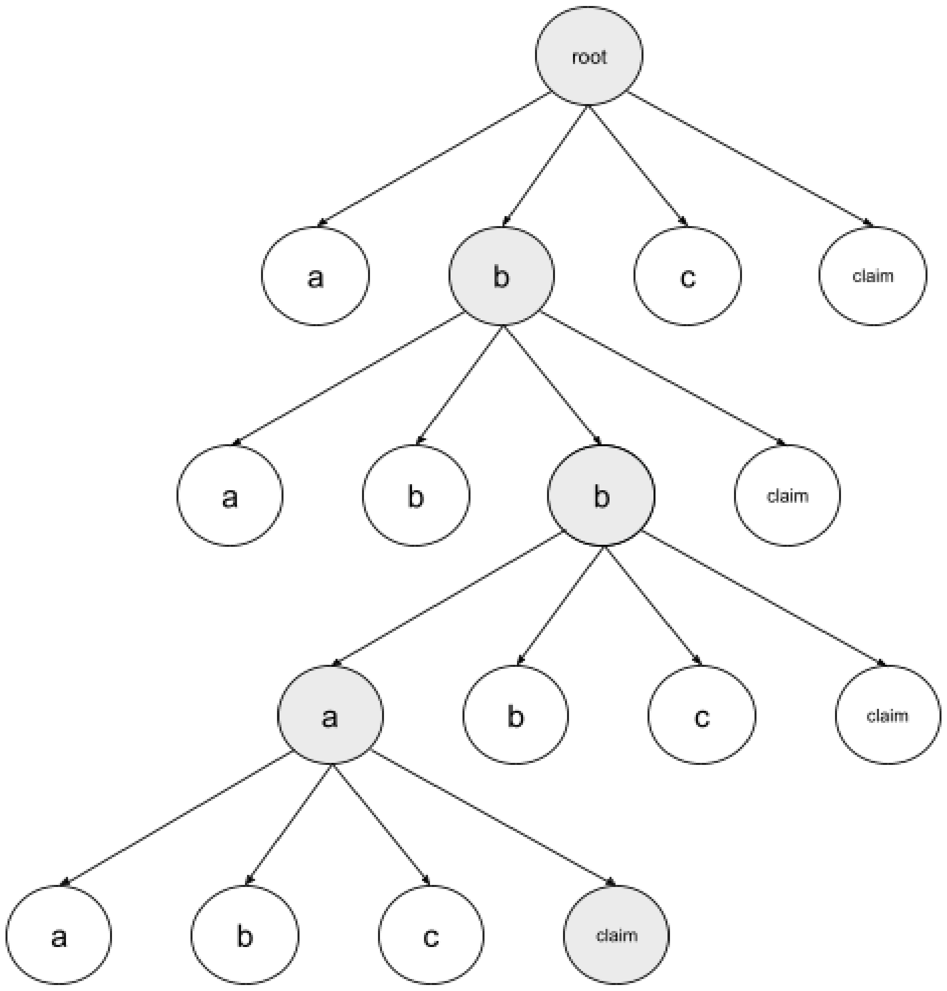

Figure 2. Determining the correct claim UTXO requires tracing the spends of the root node to the desired name lookup. The shaded nodes above show the path taken for the name "bba" and the corresponding claim UTXO.

The Claim UTXO is a Layer-1 Non-Fungible token that must maintain the contractID (minting outpoint) and pass along the address of the current owner. Arbitrary data can be attached with subsequent spends that act as metadata and Zone Records (Identical to DNS). Note that the format of the `OP_RETURN` can be changed and expanded as needs arise, since the format is not locked into the smart contract, but instead interpreted by clients when resolving the chain of UTXO spends.

3. Transaction Contract

The format of the transaction outputs are as follows:

1) Determine upfront the claimAddress that corresponds to a known private key used to claim a name via the Claim Token.
2) Each spend of the contract must create at least 38 outputs where the 0th output is the Claim Token which is a layer-1 NFT, and outputs 1 to 38 reproduce the name transaction contract exactly but each representing the characters (-, a to z, 0 to 9). This ensures that the prefix data structure is repeated at each level and thus easy to verify with indexers and wallets.
3) Each Claim Token must be a deterministic layer-1 NFT that can only have a single linear history and one owner at any moment.

By convention the deployed output of the contract forms the name tree root and has a globally identity (also called rootID) and can also can be identified by the unique outpoint (36 bytes)

Users can rely on the chain of transactions as authoritative because the contract is designed in such a way that the rules must be followed via covenants. To be safe, the user must request a Merkle Proof (also called "SPV" - Simplified Payment Verification) for the last node (representing the last letter) in the chain of transactions to provide proof that the branch of the prefix tree is authoritative.

4. Name Ownership

The ownership of a name is associated with the holder of the private key that corresponds to the address that is associated in the UTXO representing the Claim Token. The operational requirements of the Claim Token are as follows:

1) The deployment of Claim Token output will have a push data field of the 36 byte outpoint of the previous output being spent. In the case of Radiant, this corresponds to a unique singleton reference.
2) After the initial deployment, all subsequent spends of the UTXO must supply the 36 bytes of the outpoint (transaction hash and output index) of the deployment (ie: "minting") output in the first field to maintain the singleton identity.
3) Ownership transfers are simply updating the address of the owner of the Claim Token
4) Subsequent transfers and data updates simply trace the Claim Token through and follow the first-in first-out rule in that a Claim Token spent at an input will be traced through to the first output, and the second spent Claim Token will be traced to the second output, and so on.

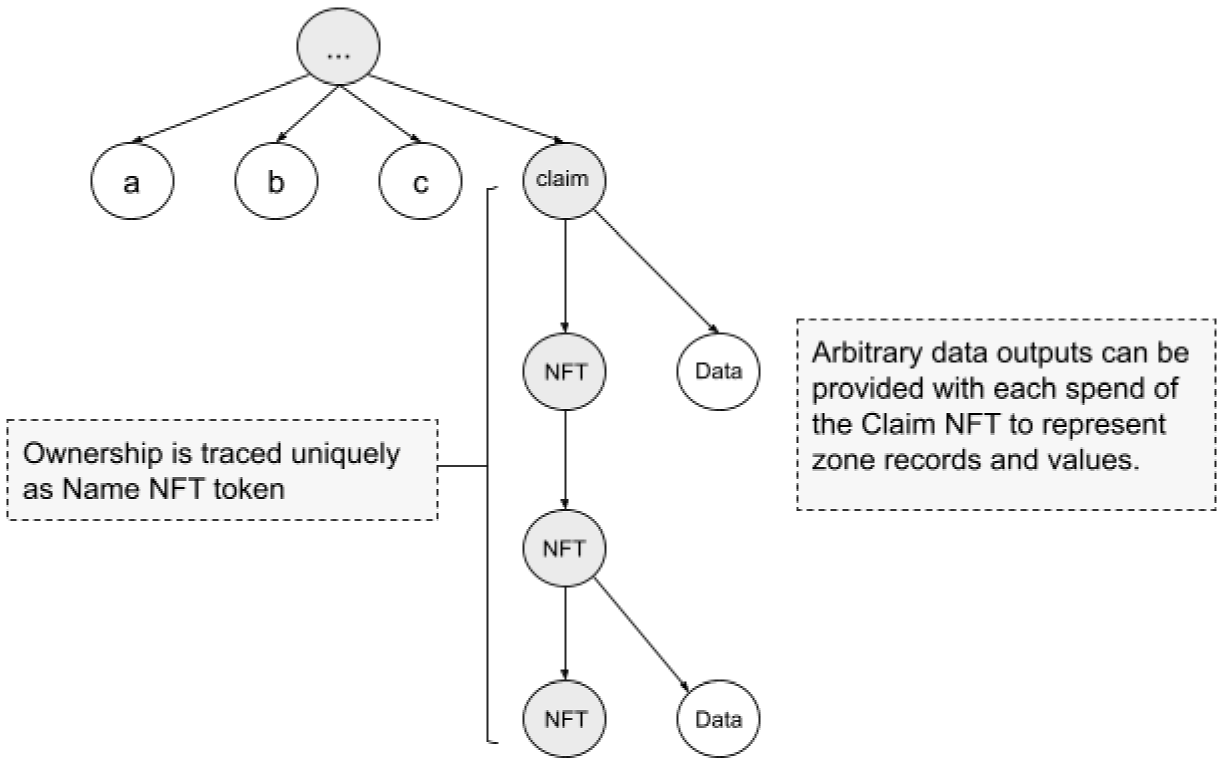
...
a
b
c
claim
NFT
Data
NFT
NFT
Data
Ownership is traced uniquely as Name NFT token
Arbitrary data outputs can be provided with each spend of the Claim NFT to represent zone records and values.

Figure 4. Name ownership is determined by the most recent holder of the unspent output (UTXO) of the Non-Fungible Token anchored at the name in the name tree.

We have in essence created a polymorphic NFT token that can take on any conceivable logic while still maintaining a clear unambiguous identity and historical ownership title and history. Of course care must be taken by users and wallets so that no malicious code is loaded, but it is always up to them to choose to change the contract code or simply use the base NFT.

5. Name Resolution Service

To resolve a name to the latest owner and state, a *resolver* service may traverse the prefix tree branch starting at the root transaction and the nodes representing each letter of the name. Once the name is resolved to the Claim Token, then the resolver service must provide all subsequent spends of the Claim Token to the latest unspent output for Bitcoin and other UTXO-based blockchains. With Radiant however, only the latest UTXO needs to be provided because it will contain the unique reference identifying the Claim Token which is the compact proof.

For example, to resolve the name “atlas” the following transactions must be returned to the client’s *resolve* request:

Prefix Tree transactions:

- `root → “a” → “t” → “l” → “a” → “s” → Claim Token`

For a total of 6 transactions to get the initial owner and location of the Claim Token. To resolve the latest state, then the Claim Token’s spends must also be followed until the unspent output

(UTXO) is reached. If a name was updated 1,000 times, then it would require an additional 1,000 transactions to be returned. In practice this is a manageable size and more detail is provided below in *Section 9. Calculations*. As mentioned before, it is only a requirement with Bitcoin and other UTXO-based blockchains to retrieve the entire transaction history because with the Radiant blockchain we have compact proofs and only need the latest transaction.

To ensure that the branch of the prefix tree and subsequent spends of the Claim Token are authoritative, a Merkle Proof (SPV) must be requested for the last spend of the Claim Token only and compared against the block headers.

6. Updating Zone Records

The design of the system is agnostic to the structure and semantics of data storage and zone records. The Claim Token NFT is designed in a way that additional data carrier outputs can be attached without affecting functionality. For example, when spending a Claim Token, a JSON payload can be attached as the second output that maps keys to values.

Sample zone records for name “aabbcc”:

```
{
   “xpub”: “xpub123…”,
   “address”: “18ac9L…hKTt”,
   “avatar”: “https://someimagesite.com/myavatar.png”,
   “name”: “Perseas”,
   “sub”: {
       “acme”:  {
          type: “arecord”,
          value: “173.9.4.193”
       },
       “support”:  {
              type: “hash”,
             value: “320320452604526”,
      “sub”: {
             “next”:  {
                Type: “cname”,
                value: “exampleco.com”
             }
       }
          }
       }
}
```

The example above can be interpreted as follows:

The “xpub” key can be used to derive payment or custom addresses for such purposes as messaging or e-commerce orders. The “address” key is a public address to receive donations. Basic user profile information is indicated by “avatar” and “name”.

Furthermore we can simulate the subdomain system by mapping keys to hostnames, transaction hashes or IPv4 or IPv6 addresses. For example, if we have the name “atlas” and using the above configuration, there are defined the following 3 subdomains:

- `aabbcc.acme → 173.9.4.193`
- `aabbcc.support → 320320452604526`
- `aabbcc.support.next → exampleco.com`

Updating or deleting keys is as simple as spending the Claim Token and providing the new JSON key values or the value of *null* in the case of deletes. Any user-space convention is possible and it is not just restricted to JSON but any file and protocol type. This provides a convenient way to bridge or overlay with different systems including DNS itself.

7. Permissions

The basic method of using a JSON convention for defining and updating zone records will be adequate for many users, however businesses and large organizations will require flexible controls and custom permissions to manage their zone records.

In order to handle a scenario where there are many thousands of subdomains, each potentially controlled by a different department, we can create hierarchies of name prefix trees and assign the Claim Tokens to the respective departments.

Building on the example above in *Section 6. Updating Zone Records,* we introduce the concept of *mounting* name sub-trees. Consider the following example JSON definition at the top-level name “atlas”:

```
{
   “sub”: {
       “finance”:  {
          type: “subtree”,
          value: “g1…143af00000000”
       },
       “support”:  {
          type: “subtree”,
          value: “a4…953aa00000000”
       }
   }
}
```

The definition above is defining the subdomains “finance.atlas” and “support.atlas” to have their own respective name trees rooted at their own `contract_id`. The organization merely has to deploy their own root name contract, using their own *private* claimAddress to start a new tree for being able to spawn sub names under their subdomain.

For instance, the Finance department could have names for each employee or service for the top level name “aabbcc”::

```
“aabbcc.finance.atlas”
“aabbcc.finance.jen”
“aabbcc.support.region-a.contact”
```

As we can see this is a powerful and flexible technique that allows the full range of use-cases required by enterprises and business use-cases while being able to maintain top level control and delegate management of names to specific departments or users.

8. Localization

To save space, prevent ambiguity due to capitalization, and also provide predictable prefix indexing - only a one byte field is provided and it is limited to a strict subset of ascii of 37 characters (ie: The ascii characters: - (hyphen), a *to* z, and 0 *to* 9).

In order to represent the wider range of characters provided by Unicode, we use Punycode[4] as the representation of Unicode.

> *Using Punycode, host names containing Unicode characters are transcoded to a subset of ASCII consisting of letters, digits, and hyphens, which is called the Letter-Digit-Hyphen (LDH) subset. For example, München (German name for Munich) is encoded as Mnchen-3ya.*
> *Source*: https://en.wikipedia.org/wiki/Punycode

Punycode is the preferred encoding method for UTF8 character representations in the Domain Name System and name adopts this convention as well.

9. Calculations

We consider the costs associated with resolving a name and it’s updates, since the resolver returns *all* records from the root of the prefix tree to the latest unspent output of the Claim Token, it is important to understand the bandwidth and storage requirements involved.

The base name output contract is about 500 bytes and has a fan-out factor of 38 for a total of about 18 kb bytes for each letter. We will assume the average name is 10 characters long which takes us to 180 kb to return the branch of the prefix tree corresponding to the name up to the initial Claim Token deployment.

With the location of the Claim Token NFT determined in 180 kb, now the resolver must return *all* of the spends of the Claim Token NFT. The layer-1 NFT is approximately 60 bytes per output. Assume that each name is updated many times in it’s life and we can calculate the total bandwidth and data required to resolve a name to the latest state on the client side. Let’s also assume that each update consists of another 1kb payload on the generous side, which brings us to 1 kb per Claim Token update:

`100 updates → 180 kb (base) +` *`100`* `* 1kb =` **`550 kb`**

```
1,000 updates → 180 kb (base) + 1,000 * 1kb = 1,180 kb ~ 1 MB
10,000 updates → 180 kb (base) + 10,000 * 1kb = 10,180 kb ~ 10 MB
100,000 updates → 180 kb (base) + 100,000 * 1kb = 100,180 kb  ~100 MB
```

As we can see from the calculations, this protocol offers a good balance between bandwidth requirements, quantity and size of updates. It is worth mentioning again that the verification and therefore calculation of state is performed on the client side and therefore provides the strongest guarantees of security while keeping bandwidth costs low. With Radiant, the entire history is not required but the latest single UTXO (less than a few hundred bytes) due to the compact proof system offered with the unique reference system.

How does this impact the UTXO set and blockchain size if it replaces all DNS names as of 2024?

As of 2024, there were just over 350 million domain names registered. If we go with the above estimate of assuming there are 10 character names, and assume each domain name is updated 1,000 times each during the lifetime and with each average update size estimated at being 1 kb.

```
Total # of UTXOs that will exist: 350,000,000 * 38 = 13,300,000,000
Total size of UTXOs = 13,300,000,000 * 500 = ~ 6.7 TB
Total size of update history = 350,000,000 * 1000 * 1kb = ~ 350 TB
```

The total UTXO set can fit into a single 8 TB Samsung SSDs for a cost of about $120 USD each (as of 2024). We can also store the entire history updates in 44 * 8 TB drives for a total cost of about $5,300 USD.
As we can see from the above calculations, storing the entire DNS system using this system is within reach and relatively affordable for a miner or business who wants to start an all-in-one hosting and name resolution service with the capacity of DNS, but with superior security.

11. Future Improvements

Improvement areas include the reduction of storage requirements, alternate encodings, resiliency against denial of service and name registration front run prevention. For example, an immediate improvement is to use a base-7 encoding for a fan-out factor of 7, with 2 transactions representing a single character for example. Other more compact encodings of the prefix tree have not yet been explored, but seem like a promising area of future research.

There can be a number of ways to limit the claiming of names by locking up more Satoshis in the Claim Tokens, requiring extra proof-of-work or even paying an additional mining fee above and beyond the normal required amount.

12. Conclusion

We have proposed an electronic naming system based on cryptographic proof instead of trust that works for Bitcoin and any UTXO-based blockchain. In particular the Radiant blockchain is optimally positioned due to it's unique reference system which offers compact O(1) proofs of name ownership. By leveraging blockchain transactions as a tamper resistant chain of ownership and history, we show how any willing parties can resolve names directly with each other without the need for a trust third party name service. To solve this, we proposed introducing a transaction contract that enforces the rules that creates a prefix tree of names and ownable tokens that are globally unique for each name. The system is robust and provides at least as good security as DNSSEC, but in practice will actually be much stronger because there is no central party to trust and all clients validate transactions themselves. We demonstrated how to implement flexible permissions and subdomains to meet all the capabilities of DNS and more. Finally we showed that the system is efficient and can scale to handle and replace the entire totality of the world's 350 million DNS names and records for about $5,300 USD in storage costs.

13. References

[1] Satoshi Nakamoto "Bitcoin: A Peer-to-Peer Electronic Cash System", 2008, https://bitcoin.org/bitcoin.pdf

[2] "Radiant: A Peer-to-Peer Digital Asset System", 2022, https://radiantblockchain.org/radiant.pdf

[3] "DNSSEC: DNS Security Extensions", 1999-, https://www.dnssec.net/rfc

[4] "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", 2006, http://www.nicemice.net/idn/punycode-spec.gz