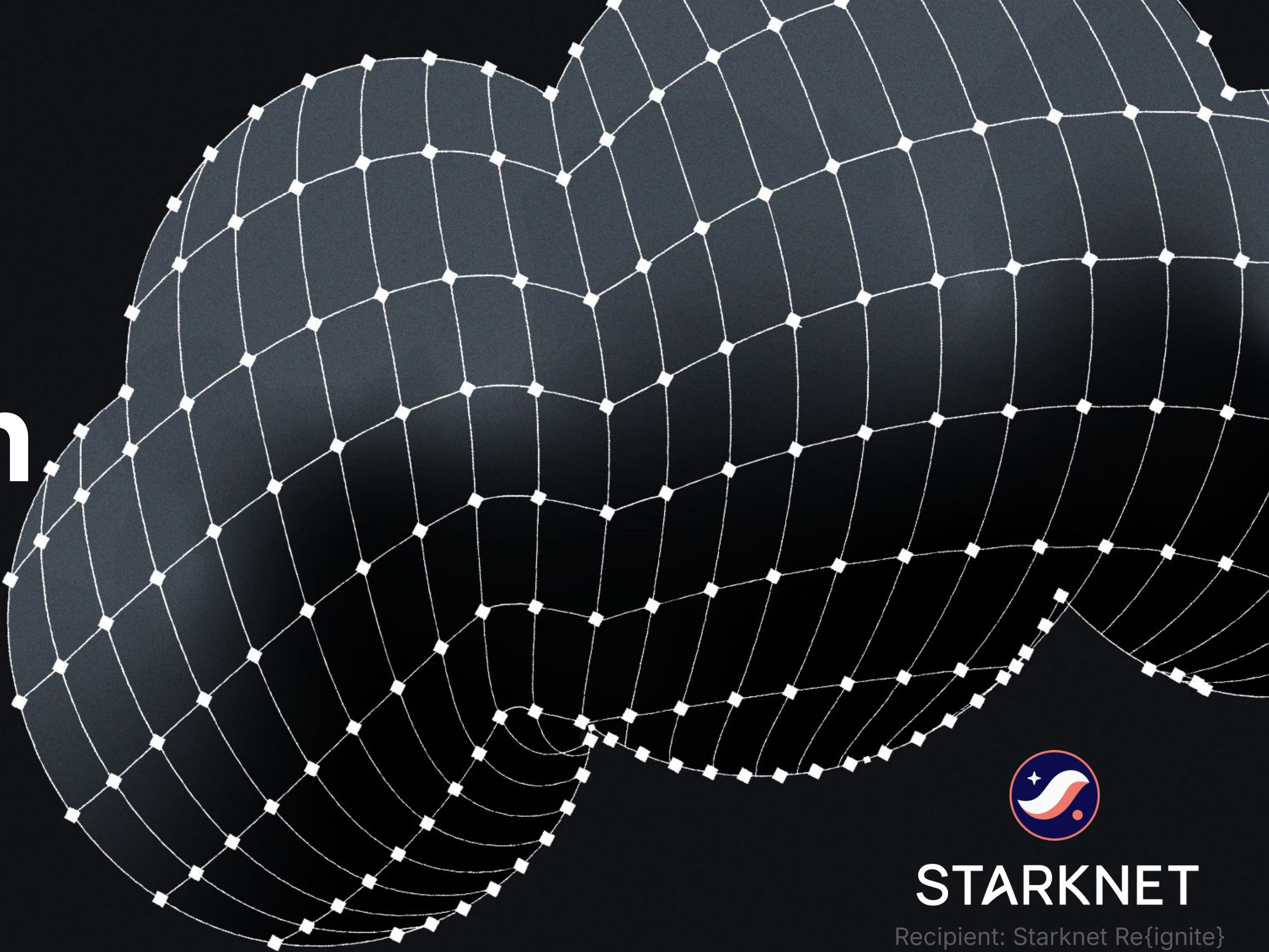
Moosh

Starknet - Re{ignite} Hackathon Submission

26 May 2025



STARKNET

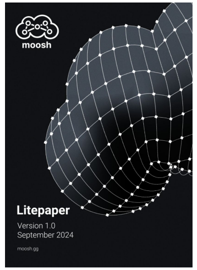
Recipient: Starknet Re{ignite}

About Us

We are a Team Building Web3 Cloud

What’s the Big Idea?

‘Decentralized + Cloud Marketplace’ = moosh.gg

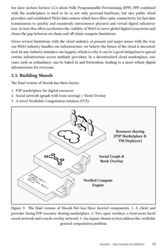


moosh

Litepaper

Version 1.0
September 2024

moosh.gg



See later section Section 3.2.4 about Fully Programmable Provisioning (FPP). FPP combined with the marketplace is used to tie in not only personal hardware, but also public cloud providers and established Web2 data centers which have fiber optic connectivity for fast data transmission to quickly and seamlessly interconnect physical and virtual digital infrastructure. In turn this effort accelerates the viability of Web3 to serve global digital ecosystems and closes the gap between on-chain and off-chain compute limitations.

Given several limitations with the cloud industry at present and major issues with the way our Web3 industry handles our infrastructure, we believe the future of the cloud is decentralized. In any industry mistakes can happen, which is why it can be a good mitigation to spread certain infrastructure across multiple providers. In a decentralized cloud marketplace, use-cases such as redundancy can be baked in and frictionless, leading to a more robust digital infrastructure for everyone.

2.3. Building Moosh

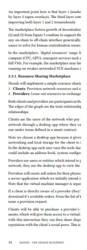
The final version of Moosh has three layers:

1. P2P marketplace for digital resources
2. Social network (graph with trust scoring) + Mesh Overlay
3. A novel Verifiable Computation solution (VCE)

Resource sharing (P2P Marketplace & VM Deployer)
Social Graph & Mesh Overlay
Verified Compute Engine

Figure 3: The final version of Moosh Net has three layered components. 1. A client and provider facing P2P resource sharing marketplace. 2. Two open overlays: a trust-score laced social network and a mesh overlay network. 3. An engine chosen to best address the verifiable general computation problem.

MOOSH - THE DESIGN BLUEPRINT 13



An important point here is that layer 1 (marke
by layer 2 (open overlays). The third layer com
improving both layer 1 and 2 tremendously.

The marketplace fosters growth of decentralize
(2) and (3) from Figure 3 combine to support th
any on-chain to off-chain interface generically.
nance to solve for human-centralization issues.

In the marketplace, ‘digital resources’ range f
compute (CPU, GPU), emergent services such a
full VMs. For example, the marketplace may lat
running on weaker networked VCE machines i

2.3.1. Resource Sharing Marketplace

Moosh will implement a simple resource shari
1. **Clients**: Provision network resources and a
2. **Providers**: Lease out resources in exchange

Both clients and providers are participants in th
The edges of the graph are the trust relationshi
relationships.

Clients are the users of the network who pay
network through a desktop app where they ca
one under terms defined in a smart contract.

Note we choose a desktop app because it gives
networking and local storage for the client to i
In the desktop app each user runs the node dae
could include an address-book, system configu

Providers are users or entities which intend to
network, they use the desktop app to view the

Providers will create sell orders for their physic
a server application which we initially intend o
Note that the virtual machine manager is separ

If a client is directly aware of a provider (they'
download it's available orders. From the list of
issue a provision request.

Clients will be able to purchase a provider's
ments, which will give them access to a virtual
with this interaction they can then share disp
reputation with the client's social peers. This is

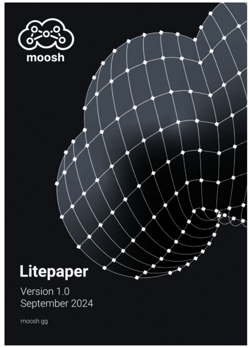
14 MOOSH - THE DESIGN BLUEPRINT

Litepaper <www.moosh.gg>

What’s the Big Idea?

Decentralized Cloud Marketplace

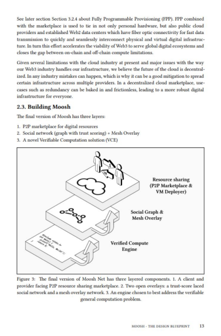
Moosh is a decentralized peer-to-peer (P2P) social network for joining trustless & trust-dependent systems and sharing digital resources.


moosh

Litepaper

Version 1.0
September 2024

moosh.gg


See later section Section 3.2.4 about Fully Programmable Provisioning (FPP). FPP combined with the marketplace is used to tie in not only personal hardware, but also public cloud providers and established Web2 data centers which have fiber optic connectivity for fast data transmission to quickly and seamlessly interconnect physical and virtual digital infrastructure. In turn this effort accelerates the viability of Web3 to serve global digital ecosystems and closes the gap between on-chain and off-chain compute limitations.

Given several limitations with the cloud industry at present and major issues with the way our Web3 industry handles our infrastructure, we believe the future of the cloud is decentralized. In any industry mistakes can happen, which is why it can be a good mitigation to spread certain infrastructure across multiple providers. In a decentralized cloud marketplace, use-cases such as redundancy can be baked in and frictionless, leading to a more robust digital infrastructure for everyone.

2.3. Building Moosh

The final version of Moosh has three layers:

1. P2P marketplace for digital resources
2. Social network (graph with trust scoring) + Mesh Overlay
3. A novel Verifiable Computation solution (VCE)

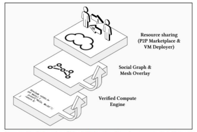
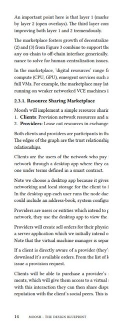

Resource sharing (P2P Marketplace & VM Deployer)
Social Graph & Mesh Overlay
Verified Compute Engine

Figure 3: The final version of Moosh Net has three layered components. 1. A client and provider facing P2P resource sharing marketplace. 2. Two open overlays: a trust-score laced social network and a mesh overlay network. 3. An engine chosen to best address the verifiable general computation problem.

MOOSH - THE DESIGN BLUEPRINT 13


An important point here is that layer 1 (marke
by layer 2 (open overlays). The third layer com
improving both layer 1 and 2 tremendously.

The marketplace fosters growth of decentralize
(2) and (3) from Figure 3 combine to support th
any on-chain to off-chain interface generically.
nance to solve for human-centralization issues.

In the marketplace, 'digital resources' range f
compute (CPU, GPU), emergent services such a
full VMs. For example, the marketplace may lat
running on weaker networked VCE machines i

2.3.1. Resource Sharing Marketplace

Moosh will implement a simple resource shari
1. **Clients**: Provision network resources and a
2. **Providers**: Lease out resources in exchange

Both clients and providers are participants in th
The edges of the graph are the trust relationshi
relationships.

Clients are the users of the network who pay
network through a desktop app where they ca
one under terms defined in a smart contract.

Note we choose a desktop app because it gives
networking and local storage for the client to i
In the desktop app each user runs the node dae
could include an address-book, system configu

Providers are users or entities which intend to
network, they use the desktop app to view the

Providers will create sell orders for their physic
a server application which we initially intend o
Note that the virtual machine manager is sepa

If a client is directly aware of a provider (they'
download it's available orders. From the list of
issue a provision request.

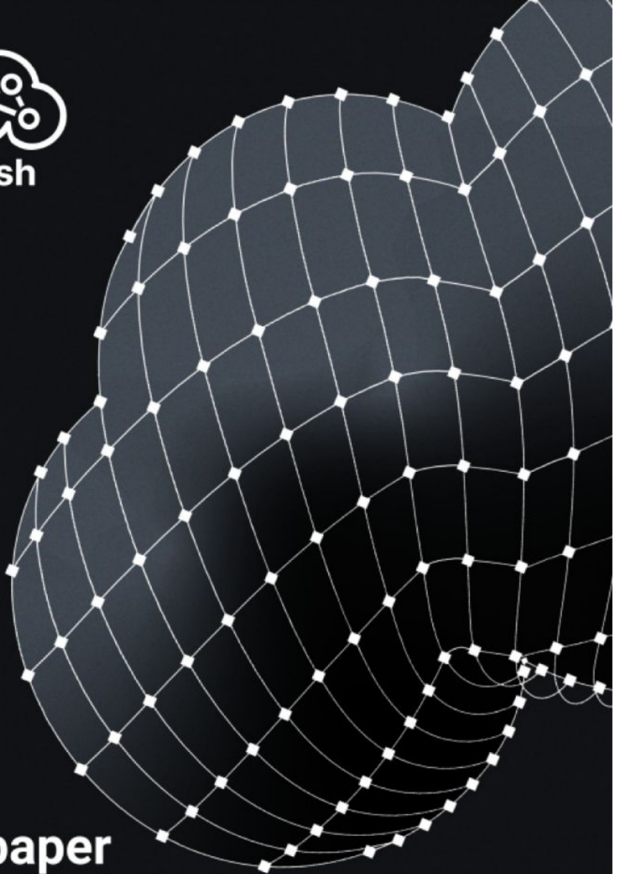
Clients will be able to purchase a provider's
ments, which will give them access to a virtual
with this interaction they can then share disp
reputation with the client's social peers. This is

14 MOOSH - THE DESIGN BLUEPRINT

moosh.gg

Litepaper <www.moosh.gg>

moosh



Litepaper

Version 1.0
September 2024

moosh.gg

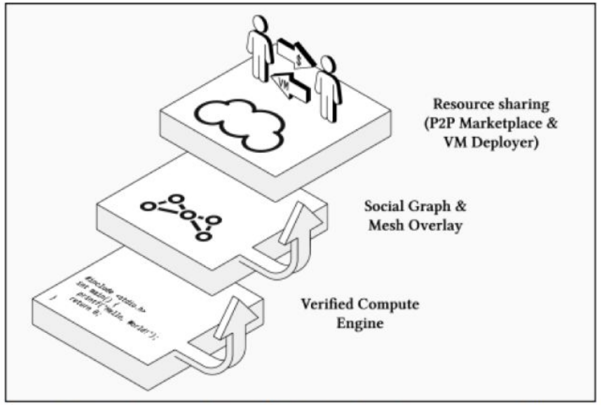
See later section Section 3.2.4 about Fully Programmable Provisioning (FPP). FPP combined with the marketplace is used to tie in not only personal hardware, but also public cloud providers and established Web2 data centers which have fiber optic connectivity for fast data transmission to quickly and seamlessly interconnect physical and virtual digital infrastructure. In turn this effort accelerates the viability of Web3 to serve global digital ecosystems and closes the gap between on-chain and off-chain compute limitations.

Given several limitations with the cloud industry at present and major issues with the way our Web3 industry handles our infrastructure, we believe the future of the cloud is decentralized. In any industry mistakes can happen, which is why it can be a good mitigation to spread certain infrastructure across multiple providers. In a decentralized cloud marketplace, use-cases such as redundancy can be baked in and frictionless, leading to a more robust digital infrastructure for everyone.

2.3. Building Moosh

The final version of Moosh has three layers:

1. P2P marketplace for digital resources
2. Social network (graph with trust scoring) + Mesh Overlay
3. A novel Verifiable Computation solution (VCE)


Resource sharing (P2P Marketplace & VM Deployer)
Social Graph & Mesh Overlay
Verified Compute Engine

```
#include <stdio.h>
int main() {
  printf("Hello, World!");
  return 0;
}
```

Figure 3: The final version of Moosh Net has three layered components. 1. A client and provider facing P2P resource sharing marketplace. 2. Two open overlays: a trust-score laced social network and a mesh overlay network. 3. An engine chosen to best address the verifiable general computation problem.

An important point here is that layer 1 (marke
by layer 2 (open overlays). The third layer com
improving both layer 1 and 2 tremendously.

The marketplace fosters growth of decentralize
(2) and (3) from Figure 3 combine to support the
any on-chain to off-chain interface generically.
nance to solve for human-centralization issues.

In the marketplace, 'digital resources' range f
compute (CPU, GPU), emergent services such a
full VMs. For example, the marketplace may lat
running on weaker networked VCE machines i

2.3.1. Resource Sharing Marketplace

Moosh will implement a simple resource sharin

1. **Clients**: Provision network resources and a
2. **Providers**: Lease out resources in exchange

Both clients and providers are participants in th
The edges of the graph are the trust relationship
relationships.

Clients are the users of the network who pay
network through a desktop app where they ca
one under terms defined in a smart contract.

Note we choose a desktop app because it gives
networking and local storage for the client to
In the desktop app each user runs the node dae
could include an address-book, system configu

Providers are users or entities which intend to p
network, they use the desktop app to view the

Providers will create sell orders for their physic
a server application which we initially intend o
Note that the virtual machine manager is separ

If a client is directly aware of a provider (they'
download it's available orders. From the list of k
issue a provision request.

Clients will be able to purchase a provider's
ments, which will give them access to a virtual
with this interaction they can then share dispu
reputation with the client's social peers. This is

Our Submission

MooshID: Efficient Post-Quantum On-Chain Identity System

Overview of Submission - (1/2)

We are entering for the **'Best use of Starknet'** track of the 2025 Re{ignite} hackathon.

We've made a breakthrough, introducing a new primitive in quantum resistant signature verification using the Falcon cryptography scheme.

- Future downstream contracts that build on our submission by anyone don't need to worry about storage, they just use a small 252 bit hash.
- Call data is 28x smaller for small Falcon 512 key size, and 56x smaller for big Falcon 1024 key size.

Our effort aligns with Bitcoin also wanting to use PQC Falcon in BIP360 …

Quantum proof signatures and public keys are orders of magnitude larger than predecessors, directly increasing the amount of data stored per transaction on a blockchain and bloating Smart Contract (SC) storage. We have made a **significant reduction** in size of a Quantum Resistant pubkey [We reduce it from 897 or 1793 bytes down to 252 bits]. We store it once on chain (this costs gas but this addresses spam protection), then once stored you can verify gas-free in the future (computation is done on RPC node instead of via gas payments). This reduces state bloat, increases block throughput and lowers SC storage gas cost!

We have applied the solution to the use case of an on-chain marketplace, where a *client* pays a *provider* for services/resources.

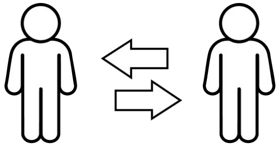
Overview: 3 Steps to use the Solution - (2/2)

Contract 1: **Key Registry**

First actor deploys key registry contract that lets other people store their key material in a storage map. Maps keys to key-material on chain.

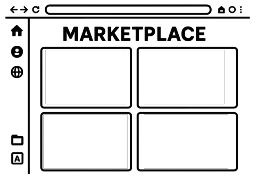
Contract 2: **Address-Based Verifier**

Read contract: Lets anyone verify a signature using the hash of anyone's pubkeys that originally signed it.



Contract 3: **Marketplace Escrow Contract**

Use case - Escrow contract for two actors. A *client* pays for services from an ID'd *provider*. A dispute mechanism is built in for intermediaries to resolve conflicts on chain.



Contract 1: Key Registry (1/1)

To bootstrap the system, an initial person deploys this smart contract named ‘key registry.’ This contract lets other people store their key material in a storage map. It maps key-hash to key-material on chain. In our use-case *providers* of services are these *people*.

- We use Falcon for Post Quantum Crypto signatures
- Poseidon felt252 for hashing Falcon pubkeys

The read functions are:

get_key_owner() - Takes a hash and returns the wallet that registered it to the contract

get_public_key() - Get’s a pubkey from submitted hash.

The key-registry owner is what is important for scaling past the storage capacity of a single Key Registry smart contract.

*’Key-material’ in our case is a Falcon public key

Contract 2: Address-Based Verifier (1/2)

This is a read contract that lets anyone verify a signature using the hash of the pubkey that originally signed it.

- This is experimental as we use a new Starkware exploration team library called S2morrow (1 month old)
- We borrow their verify-uncompressed function

One public key signs any kind of message, with no length limit on input, the output is a felt 252 size.

Contract 2: Address-Based Verifier (2/2)

- Take three inputs: key_hash, signature, message
- It queries the Key Registry to get the Falcon public key from the key_hash
- Then it verifies the signature for the message for that public key
- Contract returns True or False i.e. determines Valid or Invalid signature

In our demo use case the marketplace *provider* has to sign a predetermined message that is the listingID of their services listed on the marketplace.

Contract 3: Marketplace Escrow (1/2)

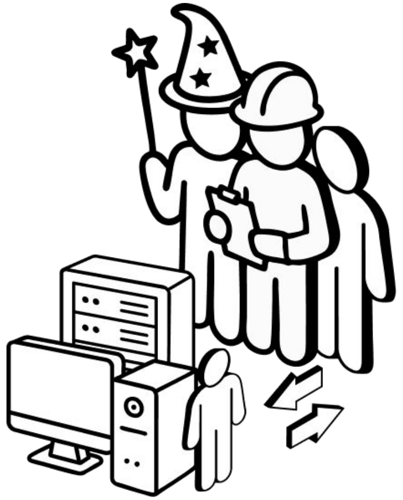
This marketplace contract is dependent on both the Key Registry SC, and the Address-based Verify SC.

It uses the Key Registry to resolve the pubkey from a pubkey hash for free (no gas).

The good thing is that any Starknet app can use this, the Key Registry and Verify SC's are generic. Only the Escrow SC is use-specific.

Ideally we would have a single marketplace-deployed escrow SC to handle multiple agreements instead of currently making the client deploy the Escrow SC for one agreement. However, the *provider* can verify the Starknet class hash of the escrow SC to check the *client* didn't deploy a malicious SC.

Note: Only the provider needs to be ID'd on MooshID!



Contract 3: Marketplace Escrow (2/2)

Step 1: A *provider* creates listing of their services in a Web3 marketplace app (this is outside of the scope for this hackathon). The listing would have an associated on-chain listingID attested by a blockchain node network.

Step 2: A *Client* (1) deploys escrow contract, (2) supplies the key-hash of the *provider* they want to make an agreement with, and (3) deposits funds into the contract

Step 3: If everything goes well, at the end of the agreement-period the *provider* can execute a claim function in the escrow contract and extract the funds. Note, the claim function only works if the agreement-period is over.

- A *provider* has a Falcon pubkey and privkey. The pubkey is hashed in the Key Registry contract, the privkey only they know.
- To claim the funds the *provider* signs the listingID with their falcon private key on their local machine and gives it to the escrow contract. In turn the escrow contract interacts with the Address-verifier to check if the signature is valid or not.
- If signature is valid, funds are released, if-not then no funds are released.

Optional: Step 3.5: If there's a dispute over the services provided before the agreement's tenure is completed (e.g. services are not up to the desired standards), the provider gets partial payment up to the time served, the remainder is returned to the client. Note only the client is able to initiate the dispute.

USER INTERFACE

Demo built in
To showcase

LINKS & RESOURCES

Github: https://github.com/moosh-network/starknet-reignite
Video:
Smart Contract Classes deployed on mainnet:

- Key Registry:
- Address-based-verifier:
- Escrow:

Project Extensions

What to do next?

- From client side we can call the contract recursively to do message chain verification. To handle complex messaging for advanced escrow disputes requiring a history of agreement terms or other information
- We would want to allow agreements to be extendable. Modification of terms.
- The initial single Key-registry contract will deploy new Key Registry whenever needed, i.e. if one SC is full (max is 100 keys today: 100 keys in 3000 Starknet SC storage slots) the Key Registry Factory can spawn further contracts once the first is full.
- In today's example it is important to trust who deploys the escrow contract class. Need to remove this trust.
  - Root of trust improvements. In future the Key Registry contract owner can be extended be a single router wallet that will deploy all the keys based on how much capacity is needed. I.e. as one SC runs out of space, another is created. The Router maps hash to the registry and so on. This is similar to how how uniswap charges for creating new liquidity pools.
  - Factory & Router Cairo contracts for Key Registry to scale over 100 falcon keys per SC
- Extend solution to use a Falcon certificate instead of just a normal key, so that a cert can also have a shortened address on Starknet.
  - Further extend to open an on-chain Falcon certificate in a CLI or browser UI for an Web3 app.
- Extend logic to verify two parties consent to a single agreement.
  - Would go over 3 milllion steps if we have to verify two addresses using current methodology - requires optimization.
- Explore extending this idea for a Peer-to-Peer marketplace for Virtual Machines.