VM-DPoSW, the Consensus Algorithm of BFDChain: The Design Principle and Quantitative Analysis

Befund Foundation Ltd.

Abstract— In this paper, we discuss the design detail of VM-DPoSW, the consensus algorithm that supports a robust BFDChain under the DAOS ecosystem. The workflow, design principle, implementation and its controllability analysis are discussed in detail.

I. Introduction

As one of the most important aspects of any blockchain systems, the consensus algorithm design is crucial to construct a robust and health blockchain ecosystem. In BFDChain, we designed a new consensus algorithm named VM-DPoSW, which is a virtual machine based hybrid system with both Proof of Work (PoW) and Delegated Proof of Stake (DPoS) support.

The rest of this paper is organized as follows: In Section II, we present the state of art review of popular consensus algorithms. We then provide detailed technical discussion of VM-DPoSW and its controllability analysis in Section III and IV. Section IV includes concluding remarks of our design.

II. State of art of Consensus Algorithms

A safe, orderly and healthy blockchain requires us to solve two fundamental problems: double spending and byzantine generals problem [8]. Double spending problem means to reuse the currency in two transactions simultaneously. Byzantine generals problem means during the peer to peer communication of the distributed system, some maliciously users may tamper the communication contents, thus lead to security breach or communication inconsistency.

In order to make the whole blockchain safe and consistent, the generation of block needs to reach a certain consensus, thus the consensus algorithm is one of the keys for any blockchain technologies. The common consensus algorithms are PoW, PoS, DPoS, PBFT, and RAFT.

PoW (Proof of Work): The workload proof mechanism, through a large number of HASH operations, calculates a suitable random number and produces a new block. And this is the safest way of security, but at the same time, it is also very energy consuming. Bitcoin [1] is the most typical PoW implementation.

PoS (Proof of Stake): The ownership proof mechanism, through the holding amount and holding time of the token, reduces the difficulty of the block production. This method solves the problem of energy consumption comparing with PoW, but there are certain bottlenecks in security, and system bifurcation is easy to appear. PPCoin [6] is one typical PoW implementation.

DPoS (Delegated Proof of Stake): The agent's equity proof mechanism, by which a certain number of agents are elected by the ballot papers, and the blocks are produced in a certain order between the agents. DPoS greatly reduces the number of verification node and improves transaction confirmation speed under the premise of security protection. However, the corresponding centralization degree is reduced. BitShares is an example of DPoS [2].

PBFT (Practical Byzantine Fault Tolerance): It is a practical Byzantine fault tolerance, and this kind of consensus cannot require the issuance of tokens, which is more suitable for the operation of the alliance chain. In 1999, the PBFT system [7][8] was proposed and the algorithm complexity was reduced to a polynomial level, which greatly improved efficiency. PBFT have 5 steps in its workflow, namely, 1)request, 2)pre-prepare, 3)prepare, 4)commit and 5)reply.

RAFT: To solve the consistency problem in PBFT, Lamport etc. proposed a new algorithm named Paxos, which is the initial prototype of RAFT. It was not until 2013 for RAFT algorithm to be formally proposed by Ongaro etc. in [9]. RAFT achieves the same effect as Paxos and is more convenient in engineering implementation and understanding.

For a specific business scenario, the consensus algorithm has a great impact on the participants' decisions. For the alliance chain with certain trust basis, most of them take PBFT as the first choice, and the PBFT consensus mechanism performs better when nodes are fixed and the number of nodes is less. In the low dependence environment, the robustness of the blockchain system is generally guaranteed by PoW, PoS, and DPoS.

III. BFDT PROXY VIRTUAL MACHINE (BPVM) CONSENSUS ALGORITHM

BFDChain serves the main chain for Befund's decentralized fund service platform for operating activities that are far more complex than that of Bitcoin and Ethereum. Thus, our goal is to design an efficient and robust consensus algorithm to support a sustainable and healthy eco-system.

We use virtual machine based hybrid DPoS and PoS (VM-DPoSW) consensus algorithm to achieve our design goal. Here is the implementation detail of VM-DPoSW:

3.1 Virtual Machine

Virtual Machines (VMs) are the abstract entities that perform mining work on BFDChain. VMs serve two purposes: first, they are the mining worker to solve the hash computing for the proof of work (PoW). Secondly, they are the delegates that represent the share stake of the shareholders of BFDT in the BFDChain eco-system (DPoS). To achieve this, VMs are created by smart contacts to have different computing powers, and the total number of the VMs are upper bounded in a given period of time based on supply and demands. BFDT Shareholders such as side chain owner, decentralized application developer, investors acquire the VMs with different computing power via bidding with BDFT. As VMs are the only eligible miners on the BFDChain ecosystem, and higher computing power represent higher voting right, the BFDT shareholders are incentive to invest on VMs and have BFDT locked in the BFDChain eco-system to achieve stable and healthy long-term growth.

3.2 Why VM-DPoSW?

In the original design of PoW, it is the hope of the designer that all mining workers can use the CPU to perform the mining work such that each node, even with different computing power (thus different hashing power), still has the equal opportunity to participate in the decision-making of the blockchain. However, with the development of the hardware such as GPUs and ASICs, and the aggregation of individual computing power into mining pools, the ordinary miners rarely have the opportunity to create a block. Furthermore, there are more and more criticize of PoW not being environment friendly and slowing down transaction speed on blockchain.

On the other side, the DPoS mechanism such as BitShares tries to tackle the problem of PoW by allowing each node to select the delegates based on its share stake. The top N delegates that have got the most votes have the accounting right. The sufficient decentralization is achieved as long as 50% of the voting shareholders believe their delegates are part of the delegates group that can perform the block creation and validation work [3]. Generally, the blockchain using DPoS is more efficient and power-saving than PoW because all of the blocks creation and validation occurred only on a group of delegates. Yet, there are more and more criticize from the community that pure DPoS only represents the interest of the large shareholders, and the small and medium shareholders rarely have the rights in the block chain decision making process.

The drawbacks of PoW and DPoS motivate us to come up VM-DPoSW, to balance the pros and cons of PoW and DPoS, for a stable, robust, and efficient consensus design.

3.3 How Does VM-DPoSW work?

We will use the Fig.1 to illustrate how VM-DPoSW works.

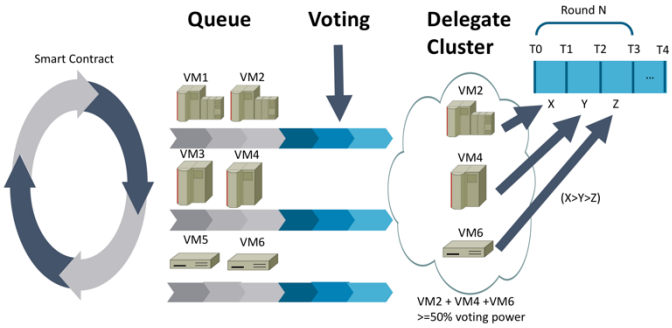

Fig.1 The workflow of VM-DPoSW

a. Create Virtual Machines

First, after a successful biding, the smart contracts on BFDChain are triggered to create virtual machines (VMs) that fall into different categories of computing power. For better illustration, we simplify the model to assume there are only three types of VMs: gold (large computing power), silver (medium computing power), and bronze (small computing power). The computing power of each type is designed such that gold > silver > bronze, i.e.,

$$VM_2^{gold} > VM_4^{silver} > VM_6^{bronze} \quad (1)$$

b. Queue Pool

Let's further assume the newly created virtual machines VM1/2, VM3/4, and VM5/6 belong to gold, silver, and bronze respectively. Right after VMs are created, their role is initially set as witness role, and are put into the queue pool as the delegate candidate.

c. Voting and Delegate Cluster

Sequentially, when the new transaction requests come, new smart contract is triggered to evaluate whether the delegate cluster pool has sufficient delegates to complete the transaction requests. If not, voting process is triggered to select additional witness from the queue pool to the delegate cluster pool. In our scenario, let's assume VM2 of gold type, VM4 of silver type, and VM6 of bronze type are selected as delegates and put into the delegate cluster pool, and they fulfill the requirements that

$$VM_2^{gold} + VM_4^{silver} + VM_6^{bronze} \geq 50\% VotingPower \quad (2)$$

d. Transaction Process

In VM-DPoSW, the transaction requests are processed in different "rounds" in the time spectrum, and in each "round", the hash difficulty is the same for all delegates. In our case, as illustrated in Fig.1., the round N starts at T0, and is expected to end at T3. Our algorithm is designed in the way that the total time in round N, $\Delta T = T3 - T0$, is equally divided into K slides, where K is the total number of delegates in the delegates cluster, i.e.,

$$T1 - T0 = T2 - T1 = T3 - T2 = \Delta T / 3 \quad (3)$$

Let's assume VM2 in gold type starts to serve the transaction request at T0 and stopped at T1 (the order may be different, and we will address the ordering in section 3.4). Within $\Delta T/3$ time frame, VM2 processed X number of transaction requests. Same scenario applies to VM4 and VM6 at T1 and T2, and each processed Y and Z number of transaction requests within $\Delta T/3$ time frame. Recall in terms of computing power, we have (1), and the hash difficulty is the same for all delegates in round N, thus we will have

$$X > Y > Z \quad (4)$$

We can see from eq.(3) that VM-DPoSW gives each delegate an equal opportunity to participate the mining process regardless the computing power of the delegates. On the other hand, eq. (4) shows that the delegate with higher computing power will process more transaction requests (thus more blocks) and thus generate more rewards for the shareholder with higher share stake, even it was only given same process time comparing with other delegates with lower computing power. In reality, we will put more constraints to ensure a sophisticated delegates system. For example, we may set the upper bound for the percentage of VMs in each category.

3.4 The signature and ordering of VM-DPoSW

As pointed out in [4], in PoW, the expected time to calculate a correct "nonce" is proportional the hash difficulty. i.e., the nonce H_n must satisfy the relations:

$$n \leq \frac{2^{256}}{H_d} \wedge m = H_m \quad (5)$$

With $(n,m) = PoW(H_{\backslash n}, H_n, d)$.

Where n is the mix-hash and m is the pseudo-random number cryptographically depend on H and d. $H_{\backslash n}$ is the new block's header H without the nonce and mix-hash components, and d is the current data set. PoW is the proof of work function. Eq.5 is the foundation of the security of the blockchain and is the fundamental reason why a malicious node cannot

propagate newly create blocks that would otherwise overwrite history.

In VM-DPoSW, however, we may choose to set the hash difficulty lower so that even the VM with lowest computing power can finish the hash computing quickly and can generate new block and process transactions in its given time window. While this design significantly increases the efficiency of the eco-system, it may increase the security vulnerability as malicious users may take advantage of the lower hash difficulty. This requires us to add additional security mechanism, namely, signature and random ordering, to safeguard the BFDChain ecosystem.

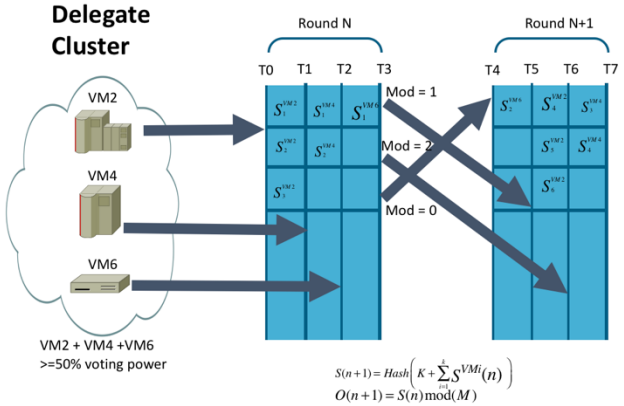

Fig.2 The signature and ordering of VM-DPoS

The design goal of the signature and random ordering is to ensure a given delegate VM in the delegate cluster can only process transaction request in the assigned "round" as well as the assigned time window. As illustrated in Fig.2., assume in round *N*, VM2 starts to perform the mining and create a new block at T0, we add a private key into the optional filed of the block and got a signature S_1^{VM2} by performing

$$S_1^{VM2} = Hash(K_1^{VM2}) \quad (6)$$

Where K_1^{VM2} is the private key value for the 1st block created by VM2 in the round *N*. Similarly, when VM2 creates the 2nd and 3rd block, the signature S_2^{VM3} and S_3^{VM3} are calculated by

$$S_2^{VM3} = Hash(K_2^{VM2} + S_1^{VM3}) \quad (7)$$

$$S_3^{VM3} = Hash(K_3^{VM2} + S_2^{VM3}) \quad (8)$$

respectively. Once VM2 creates its 3rd block, VM2 determines this is the last block it can process, it then broadcast the signature S_3^{VM3} to all other VM delegates, before T2.

Sequentially, VM4 and VM6 will be the second and third VMs to follow similar procedure to create their blocks and perform the signature broadcast at the end of their last block mining. In our case, S_1^{VM6} between T2 and T3, is the last signature in round *N*, and is the proof needed by each VM delegate to participate mining in the next round *N+1*. Image a malicious delegate tries to cheat the system by performing mining before round *N* finish and try to work on round *N+1*. Its mining of new block(s) will be rejected because it will not have the final signature S_1^{VM6} to sign the newly created block.

In round N+1, we can generalize (6)-(8) as below:

$$S(n+1) = Hash\left(K(n) + \sum_{i=1}^{k} S^{VMi}(n) \right) \quad (9)$$

Where *K(n)* is the key value at round N+1, and $\sum_{i=1}^{k} S^{VMi}(n)$ is the sum of the hash value of K number of VM delegate in the previous round (for example, we have K=3 in round N).

In addition, we use the following mechanism to determine the order of VM delegate in round N+1:

$$O(n+1) = S(n) \bmod (M) \quad (10)$$

Where *S(n)* is the signature of the last block in round *N* (i.e., S_1^{VM6} in our example) and M is the number of VM delegates. The mod operation will determine the serving order of each VM delegate in round *N+1*. In our case, in *N+1*, the mod result for VM2, VM4, and VM6 are 1, 0, and 2. Thus VM6 is scheduled to start to create block first at time stamp T4, followed by VM2 (3 blocks starting at T5), and VM4 (2 blocks starting at T6).

If there is conflict during the mod operation, we point the VM delegate to the next available slot. In case a particular VM delegate is not able to generate block in its given time windows, we will use the signature in the previous broadcast.

IV. The Controllability Analysis of VM-DPoSW

In any system design, the controllability is the most import aspect to inspect. The consensus algorithm design is not an exception.

By "controllable", we mean to evaluate whether

VM-DPoSW consensus algorithm can be properly managed even with heavy transaction requests from the main chain and side chain, and whether our algorithm can steer the resource efficiency over BFDChain eco-system from any initial value to the optimum state within a limited time window. This kind of controllability property is a crucial to achieve queue stabilization, delay bounds, and optimal resource control.

Assume

$$f(K,S(n)) = Hash\left(K + \sum_{i=1}^{k} S^{VMi}(n) \right) \quad (11)$$

$$g(S(n),M) = S(n) \bmod (M) \quad (12)$$

Eq.(9) and (10) yields

$$S(n+1) = f(K,S(n)) \quad (13)$$

$$O(n+1) = g(S(n),M) \quad (14)$$

Eq. (13) and (14) describe a non-linear discrete system, where the state vector $x(n) = [s(n), o(n)]^T$ represent the array of signature hash value and the ordering of the VM delegate. The input vector $u(n) = [K, M]^T$ represent the array of the key value and the number of VM delegates. The linearization is necessary to analyze the controllability [5]. Assume the equilibrium point is $(s(n)_0, o(n)_0, K_0, M_0)$, all of which are positive real numbers; linearizing Eqs. (13), (14) the equilibrium point, we obtain the following linearized system in state space:

$$\begin{pmatrix} \delta\dot{S}(n+1) \\ \delta\dot{O}(n+1) \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial S} & \frac{\partial f}{\partial O} \\ \frac{\partial g}{\partial R} & \frac{\partial g}{\partial O} \end{pmatrix} \begin{pmatrix} \delta S(n) \\ \delta O(n) \end{pmatrix} + \begin{pmatrix} \frac{\partial f}{\partial K} & \frac{\partial f}{\partial M} \\ \frac{\partial g}{\partial K} & \frac{\partial g}{\partial M} \end{pmatrix} \begin{pmatrix} \delta K \\ \delta M \end{pmatrix} \quad (15)$$

Let $A = \begin{pmatrix} \frac{\partial f}{\partial S} & \frac{\partial f}{\partial O} \\ \frac{\partial g}{\partial R} & \frac{\partial g}{\partial O} \end{pmatrix}, B = \begin{pmatrix} \frac{\partial f}{\partial K} & \frac{\partial f}{\partial M} \\ \frac{\partial g}{\partial K} & \frac{\partial g}{\partial M} \end{pmatrix}$, we have

$$\delta\dot{x}(n+1) = A\delta x(n) + B\delta u(n) \quad (16)$$

By modern control theory [5], the system is controllable iff $U = [B \; AB]$ is full row rank. As $(s(n)_0, o(n)_0, K_0, M_0)$ are all positive real numbers, we can conclude the $U = [B \; AB]$ is full row rank, and thus the VM-DPoSW is controllable.

V. Conclusions

In this paper, we have discussed the design detail of VM-DPoSW, the consensus algorithm that support a robust BFDChain under the DAOS ecosystem. We discussed our motivation, the work flow and the additional security mechanism such as signature and random ordering. At last, we use the modern control theory to prove the VM-DPoSW is controllable under the state space analysis.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", 2009.

[2] "https://bitshares.org/,".

[3] "https://bitshares.org/technology/delegated-proof-of-stake-consensus/,".

[4] Ethereum: A secure decentralized generaliaed transaction ledger EIP-150 revision", Gavin Wood

[5] Z. Bubnicki, Modern control theory, Spring Berlin Heidelberg New York 2005.

[6] S. King and S. Nadal, "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake,", 2012.

[7] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in Symposium on Operating Systems Design and Implementation, 1999, pp. 173--186.

[8] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem," Acm Transactions on Programming Languages & Systems, vol. 4, pp. 382-401, 1982.

[9] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," Draft of October, 2013.

The Micoservices Based Mainchain/Sidechain (MMS) Architecture: BFDChain Design Principles

Befund Foundation Ltd.

Abstract— In this paper, we discuss the design principles and architecture of BFDChain. We propose a new Micoservices based Mainchain/Sidechain (MMS) architecture, which is a four-layer architecture design including mainchian layer, microservices core layer, sidechain layer and Dapps layer. We further discuss the different side chains such as DAOS Management Sidechain, System Service Sidechain, and Ecosystem Sidechain. Finally, the cross-chain communication is also discussed in detail.

I. Introduction

As a decentralized monetary fund service platform, BFDChain is dedicated on providing fast, robust, secure and easy-to-use fund management service. However, current blockchain technology has issues yet to be solved. Miscellaneous data, the performance of single chain structure, system upgrading[1], all of these issues are blockers to approach BFDChain's ultimate goal by applying existing blockchain infrastructure.

This paper introduces a refined sidechain design which helps BFDChain solve common issues named above without losing the advantages of blockchain architecture.

The rest of this paper is organized as follows: In Section II, we present the analysis why do we need to design a mainchain and sidechain. We then provide detailed technical discussion of the architect of BFDChain in terms of different components in Section III and some basic scenarios and workflow in IV. Section IV includes concluding remarks of our design.

II. Why Does BFDChain Use Mainchain and Sidechain Design?

BFDChain is designed to provide various of services including monetary fund management, issuing data currency, trading transaction certification, legal consulting [1]. These services have completely different service interfaces, data format, contract format. Also, each of the services will have its own metadata, management targets, transaction principle. If all of these data are mounted on the a single chain, like the traditional Ethereum[2] does, the potential issues are multi-folded:

a. Chain Length explosion.
Because PoW in BFDChain will not be as hard as Ethereum[2] and BitCoin[3], the number of transactions in each block is likely to be small. With all transactions recorded on the mainchain, BFDChain will run out of total asset in a short time.

b. Impossible access control.
Different services might have different users and group permissions. This will also be applicable to the granularity of the data access and user authentication.

c. Hard chain upgrade and new service registration.
BFDChain will not be completed in the first version. Instead, new features and services will keep coming up in new releases. Without sidechain, new features and services cannot be rolled out incrementally. Any unforeseen issue will impact all services at the same time.

With sidechain, data isolation is the nature. This will not only mitigate the pressure on the mainchain, but also make each service more focus on its own business logic instead of sacrificing computation power on data segregation.

III. BFDCHAIN ARCHITECT

The BFDChain's architect is illustrated in Fig.1 below.

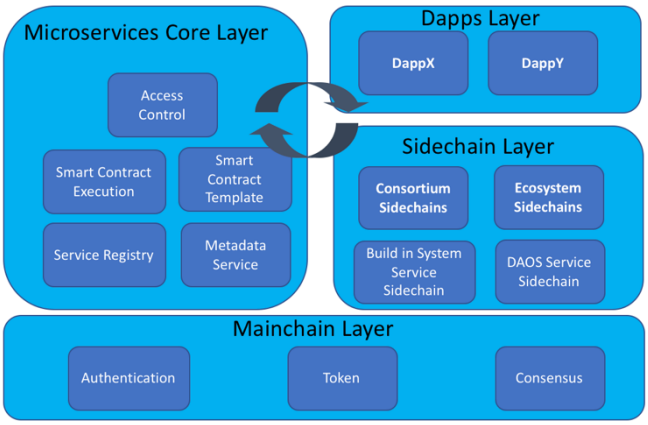

Fig.1 The overall architect of BFDChain

From Fig.1, we can see the BFDChain has four-layer architecture:
Layer 1: Mainchain layer
Layer 2: Microservices core layer
Layer 3: Sidechains layer
Layer 4: Dapps layer
Where the mainchain layer is the backbone layer for BFDChain. Dapps layer is built on top of sidechain layer, and both layers are in parallel to the microservices core layer and are both able to communicate with microservices core layer directly via REST APIs.

3.1 Mainchain Layer

To well decoupled the functionality of services, all services provided by BFDChain will be on sidechains. As a result, the major usages of the mainchain is to record all BFDT transactions plus providing basic infrastructure of BFDChain, such as Authentication and Consensus mechanism. We will illustrate them as below:

Authentication Mechanism Module

In the mainchain, we will use general ECC for cryptophytic purpose. For the organization that has the regulation requirements, we will use Unlikable Secret Handshake (USH) [11] to achieve the sidechain storage, anonymous and unlinkable coin spending for BFDChain.

Consensus Mechanism Module

BFDChain uses an innovate consensus mechanism named VM-DPoSW to process the consensus. Some high-level principles are listed below:
a. The Virtual machines (VMs) are created by the smart contract for mining and delegation purpose.
b. VMs are put into different categories based on different computing power, and put into the candidate pool.
c. VMs from each category are selected as the delegates and put into the delegates cluster pool, and ensure the sum of sharehold represented by the VMs are >50% of the total sharehold.
d. Divide the time sprearum into N slots. In each time slot, allocate same number of time windows for VMs in each category to perform the mining of the new block.
e. Essentially, VM-DPoSW gives each delegate an equal opportunity to participate the mining process regardless the computing power of the delegate. Furthermore, the delegate with higher computing power will process more transitions request (thus more blocks) and generate more rewards to the shareholder with higher share stake, even it was given equal process time comparing with other delegates with lower computing power.

Token Module

Similar to the concept of "Gas" in the Ethereum, BFDChain use "BFDT" to fuel the transaction and provide rewards as incentive for miners. Token module is used to manage and record the BFDT transactions.

3.2 Microservices Core Layer

Mircroservices, as point out in [9][10], is "an architecture style, in which large complex software applications are composed of one or more services." The main characteristics of Microservices are 1) highly Modulated. Service A does not need to know the implementation detail of another service B before them can communicate; 2) services are independently deployed with loosely coupled. 3) The communication among services are usually through REST API.

Here in BFDChain, as sidechains share a lot of common characteristics and require a set of common services, we design the microservcies core layer to provide shared services such as service registry, metadata service, smart contract templates and

execution, and access control service to sidechains and their DApps.

The following services are included in the BFDChain Microservcies Core:

a. Service Registry
This Service is used to manage the metadata of management services. Service Registry Service will be acted as the Manager of the Manager (MoM) for BFDChain platform and will be responsible for services (including mainchain and sidechains) upgrade, rollout, triage and rollback. Also, any new services provided by BFDChain will be registered here with its metadata before the actual use. Services metadata will typically include service uuid, service symbol, service smart contract address, commission fee type (percentage or fix rate or dynamic lambda function), DApp DNS address and so on.

b. DS Metadata Service
This Service will be used to manage the metadata of each DS, such as fund raise, issuance, liquidation, authentication, real time exchange rate to BFDT and services using history.

c. Smart Contract Template and Execution Service
This service is two folded. First, smart contract template will store various templates of smart contract used for fund transaction or metadata modification.

Secondly, the Smart Contract Execution is used as decentralized data store for Smart Contract Execution DApp, such as DSL definition, execution container prerequisites and execution container package location.

d. Access Control Service
Used as decentralized data store for Access Control DApp.

We need to point out that the services modules above can also be treated as special sidechains so that each module can be further extended to add sub-sidechain functionalities.

By implementing the microservices core layer, we achieved the following benefits:

- Technology Heterogeneity
- Resilience
- Scaling
- Ease of Deployment
- Organizational Alignment
- Composability
- Optimizing for Replaceability.

For example, when a sidechain A wants to perform an upgrade, the microservices layer helps A to achieve the upgrade goal by just upgrading the metadata information registered in the microservices, instead of whole sidechain A upgrade, hence greatly increases the efficiency.

3.3 Sidechain Layer
The sidechain layer is built on top of mainchain layer. It takes all of innovating features in the mainchain such as authentication and consensus mechanism, and makes use of microservice core to provide support for service DApps as well as customized token managements.

We define four different types of sidechains, namely, the built-in system service sidechain, the DAOS sidechain, ecosystem sidechain and consortium sidechain. We will address the detail of each kind of sidechain in the next section.

3.4 DApps Layer
The DApps layer is built on top of sidechain layer. It provides various application services to investors such as media and legal services. Each DApps is decoupled, single-featured and has a corresponding sidechain as its data support.

IV SIDECHAIN DESIGN

The overview of side chain design and its relationship to the mainchain and microservices core is illustrated in Fig.2.

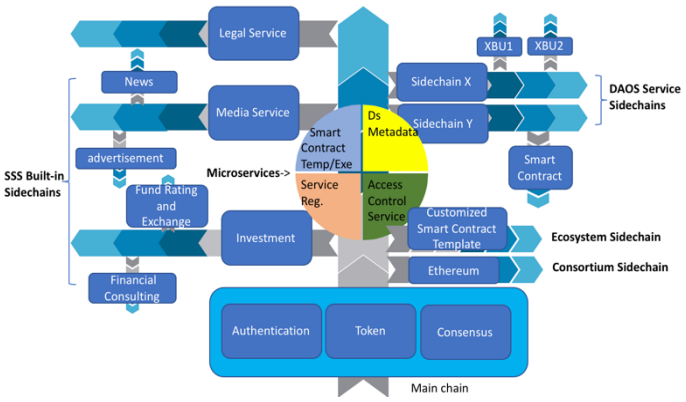

Fig. 2 The side chain design and its relationship to the mainchain and microservices

4.1 System Service Sidechain

System Service Sidechain (SSS) is a type of build-in sidechains in BFDChain platform to provide management service to support DAOS services. Each SSS has its corresponding DApp and used as a decentralized data store and lock store for its DApp. Currently, we have defined following SSSes, as illustrated in Fig.2.

a. **Fund Exchange Sidechain.**
Used as decentralized data store for Fund Exchange Service DApp. Fund Exchange Service is used for fund exchange with BFDC.

b. **Fund Rating Sidechain**.
Used as decentralized data store for Fund Rating DApp.

c. Legal Service Sidechain
Used as decentralized data store for Legal Service DApp.

d. Media Service Sidechain
Used as decentralized data store for Media Service DApp.

e. Financial Consulting Sidechain
Used as decentralized data store for Financial Consulting DApp.

4.2 DAOS Sidechain

DAOS Sidechain (DS) is a type of sidechains that client created on BFDChain platform based on a smart contract. With a DS, client is able to use all of the services provided by BFDChain platform. A DS can be created with or without a new token. The exchange rate of which can be fixed or float to the token on BFDChain's mainchain, namely BFDT. Alternatively, client can also create a DS using BFDT as an exchanging currency. Eligible investors of each DS will exchange currency on DS itself. All of transactions will only be recorded in DS but not on the mainchain. As a result, DS will be a denotation of the organization willing to use BFDChain services as well as a physical blockchain used by Cryptocurrencies or Cryptocurrency derivatives created on BFDChain platform.

4.3 Ecosystem Sidechains

Ecosystem Sidechain (ES) is a sidechain that support DApps platform ecosystem. Currently, we have defined following ES.

a. Smart Contract Sidechain.
Different from Smart Contract Template Sidechain, this sidechain stores smart contract written by clients.

4.4 Consortium Sidechains

All of sidechains described above are mounted to the mainchain. They are able to use various services and refined chain infrastructure provided by the mainchain. However, other public chain such as Ethereum can choose to form a consortium relationship with BFDChain so that members within the consortium can share common service such as creating exchanging currency. In this case, only service usages will be recorded on BFDChain as a separated DS.

V BFDCHAIN OPERATIONS

5.1 Cross-chain communication

Currently, Two-Way-Pegging[4] technique is widely used in blockchain community for cross-chain communication. However, with various fine-grained management services, BFDChain cross-chain communication is not necessarily be implemented in the chain level. Instead, DApps level with rich API choices can be a good place for some use cases. According to this, BFDChain will implement both chain level and DApps level communication with two types of cross-chain communication.

a. Metadata Modification Communication

Metadata modification communication is defined as DS or SSS chain metadata changes such as organization registration, fund raise, issuance, liquidation, service registration, service upgrade and so on. These operations are critical, less frequent and less efficiency requirement. For these operations, chain level cross-chain communication will be enforced by DS Metadata Service (DS metadata change) or Service Registry Service (SSS metadata change). BFDChain chain level cross-chain communication will use Two-Way-Pegging technique.

When DS Metadata Service or Service Registry Service receives these queries, it will initiate the transaction and do Two-Way-Pegging between the mainchain and the target sidechain. Once the transaction is confirmed in both chains, Fund Metadata Service or Service Registry Service will commit to its SSS to finished the whole transaction.

b.Service Oriented Communication

Service Oriented Communication is defined as data interactions involving DApps and microservice core such as fund rating query and legal service query. For these types of communication, DApps level communication would be a better choice than mainchain level communication.

First, SSS and ES typically do not have huge amount of hot data. This means their DApps can easily be adopt with cache. Depends on the requirements of DApps, Consistent Hashing [5], Memcached[6], Redis cluster[7] are all handy hashing techniques to have faster data access.

On the other hand, even though Two-Way-Pegging only works on chain level, it requires mainchain as a third-party communicator and chains need to wait for the creation of couple of new blocks to make sure the transaction is acknowledged. Moreover, sidechains are locked at this time until the whole transaction is completed. Finally, BFDChain as a completed platform, all of these data exchanges information will eventually be synced up with DApps. However, if transaction happens in the DApps level, syncing with DApps will be simplified; most of queries will be hit by cache; queries which needed to be committed on both side can commit individually with application level retry.

In BFDChain, cross-chain communication between DSs are discouraged due to the fund isolation. This means at least one side of transactions are cache manageable. Thus DApps level communication should be consider first.

5.2 Fund Management

As mentioned above, fund in BFDChain is an abstraction of DS plus its metadata (including smart contract).

a. Fund Creation and Exchange Rate

Fund creation will be initiated by clients on BFDChain platform. Once DS Metadata Service receive the request, it will generate smart contract based on clients' fund raise application or directly use clients' customized smart contract. Then DS Metadata Service will create a sidechain using Two-Way-Pegging and execute the smart contract for creation.

Note that BFDChain's smart contract for fund will also include exchange rate. Whether a new fund using fixed exchange rate to BFDT or not is also configurable by user.

For floating exchange rate fund, same amount of BFDT will be transferred from mainchain to the new fund chain upon creation. This is like ICO on Ethereum. When liquidating the fund, any BFDT left should be transferred back to mainchain.

Fixed exchange rate fund is almost the same, but new token is just a notation with exchange rate to BFDT recorded in the smart contract. All of actual transactions will be the same as exchange BFDT on the mainchain.

b. System Service Request

Fund user will be able to request system service at any time. The communication will through DApps level. There are following steps:

1) Service Registry get the request and find the actual System Service and its smart contract.
2) System Service will query DS Metadata Service to get fund metadata.

3) With fund metadata and smart contract, System Service will perform local service if necessary and pack required data and send to Smart Contract Execution Service to execute the smart contract.
4) Once smart contract is executed and acknowledged. Smart Contract Execution Service will inform System Service and DS Metadata Service to commit the change separately.

Note that each step above will check DApps cache first and skip costly operation in case of cache hit.

c. Commission Fee

There will be two kinds of commission fee, one written in smart contract and used to pay System Services. This kind of commission fee will be defined either by the percentage of transaction amount or by a fixed rate. All of the SSS commission fee will be enforced when service request creating a new block on SSS.

The other commission fee is defined as fund trading commission fee. Each fund trade transaction will need to pay commission fee to BFDChain platform as well as the fund management organization. The amount of the commission fee is written in the smart contract as a fixed rate along with a split ratio to pay BFDChain platform. Whenever a new block created in DS, commission fee will be enforced as part of consensus algorithm. Commission fee will originally be generated as the sidechain token. And will pay to the fund management organization according to the split ratio right away. The rest part of commission fee will be handled by Fund Exchange Service in a batch job and return to BFDChain platform as BFDT.

d. Smart Contract Execution

To provide a secure and robust platform, BFDChain will use container to perform most of smart contract and service executions. Smart Contract Execution service have container template for each service and have an interface to dynamically insert customized smart contract. Each container execution environment will only be readable by users in access control list. Users will not have write permissions to the container, all of the user interactions should through the smart contract.

e. Access Control

In BFDChain platform, access control is more on system service availability. Most of fund are managed by a group of users and Access Control Service will make BFDChain platform be able to provide different levels of service for different users within one fund group.

VI Conclusion

In this paper, we have discussed BFDChain sidechain management architecture in high level. We discussed the reason of choosing sidechain architecture as well as various sidechain services be provided in BFDChain platform. We also describe the communication protocol BFDChain used between chains. At last, we discuss how those sidechains are management and served as a completed system.

References

[1]http://www.BFDChain.io/assets/upload/BFDChain_Whitepaper_EN.pdf?v=1.0.0.61
[2] Ethereum: A secure decentralized generaliaed transaction ledger EIP-150 revision", Gavin Wood
[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", 2009.
[4] Adam Back, Matt Corallo et, al "Enabling Blockchain Innovations with Pegged Sidechains" 2014
[5] I. Stoica, R. Morris, D. et, al. "Chord: A scalable peer-to-peer lookup protocol for internet applications". IEEE/ACM Transactions on Networking, 11(1):17–32, 2003.
[6] http://memcached.org/
[7] https://redis.io/topics/cluster-spec
[8] Sompolinsky Y., Zohar A. (2015) Secure High-Rate Transaction Processing in Bitcoin. In: Böhme R., Okamoto T. (eds) Financial Cryptography and Data Security. FC 2015. Lecture Notes in Computer Science, vol 8975. Springer, Berlin, Heidelberg
[9] Shahir Daya et, al. "Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach", Aug. 2015, https://www.redbooks.ibm.com/redbooks/pdfs/sg248275.pdf.
[10] Sam Newman , "Building Microservices: Designing Fine-Grained Systems, O'Reilly Media Inc., Feb.2015

[11] P. Duan, “USH for Anonymous and Unlinkable Coin Spending for BFDChain”, http://www.befund.io/assets/upload/Befund_USH_EN.pdf?v=1.0.0.63
[12] “VM-DPoSW, the Consensus Algorithm of BFDChain: The Design Principle and Quantitative Analysis, http://www.befund.io/assets/upload/Befund_Consensus_Algorithm_EN.pdf?v=1.0.0.63

USH for Anonymous and Unlinkable Coin Spending for BFDChain

Pu, Duan
Befund Foundation Ltd.

Abstract— In this paper, we propose Unlinkable Secret Handshake (USH), which is based on Tate Paring cryptography, as a new protocol to provide anonymous and privacy-preserving coin spending for BFDChain.

I. Introduction

As one of the most important security aspects, how to achieve unlinkability and anonymity is critical to the success of any blockchain systems because of its decentralized nature. We designed an unlinkable secret handshake (USH) protocol for BFDChain, which provides more anonymous and privacy-preserving coin spending.

The rest of this paper is organized as follows: In Section II, we present the motivation to design and use USH for BFDChain. Then we provide a summary of threats to privacy and anonymity to on-chain currencies. We then provide detailed technical discussion of USH on BFDChain and its security analysis in Section IV and Section V, respectively. Section VI concludes this paper.

II. MOTIVATION

One significant advantage to use on-chain currency is that user's real-world identity is not disclosed when sending or receiving payment, i.e. cryptocurrency payment is sent to or received by user's blockchain address that is a hashed value of a single-time used elliptic curve point. However, as payment is usually for real-world offline transaction, payer may need to disclose his email address or shipping address to complete the offline transaction or prove his eligibility for the transaction. For example, Alice wants to purchase alcohol from Bob. She needs to disclose her real-world identity document to prove that she is over 21 years old before getting legally qualified for the alcohol purchase. This causes two potential risks: (1) Bob links Alice's real-world identity to her cryptocurrency identity/address (2) if Alice wants to purchase alcohol in the future, she will need to disclose the same identity again to others which increase the risk to link up all her previous transactions.

To solve this issue, we propose USH to achieve anonymity and unlinkability for user's credential like the identity mentioned in the above example. We assume that before the coin spending, both payer and payee have already obtained credentials from a central authority that cooperates with BFDChain (for exmaple, DMV already built a sidechain on BFDChain). Credential assignment can be built as a service or sub-chain attached to BFDchain. The credential is to prove that the owner of the credential is a member of a group, i.e., eligible for certain purchase. USH protocol guarantees that (1) Bob cannot link Alice's real-world identity to her BFDchain ID/address after proving that she is member of the group (2) Alice's credential is unlinkable, i.e., future same kind of transactions won't disclose her previous transaction. Furthermore, even the central authority cannot know that it was Alice who made the purchase from the public ledger of BFDchain.

It is also well known that on-chain cryptocurrency like BFDchain is built on elliptic curve cryptograph (ECC), i.e., payer/payee's addresses in coin wallet are hashed values of elliptic curve point for signing and verifying by elliptic curve digital signature algorithm (ECDSA). USH is also designed on the same ECC based cryptosystem, so the new unlinkability and anonymity for coin spending can be easily implemented on BFDchain.

III. PRIVACY ISSUES IN CRYPTOCURRENCY

There are many security concern and attacks for on-

chain currencies, e.g., doubling spending [1] and mining pool attacks [2] for Bitcoin. This paper focuses on the privacy and anonymity issues for cryptocurrencies. They achieve anonymity by keeping public keys anonymous, i.e., the public can see payment is sent from payer to payee, but cannot link the transaction to anyone's real identity. The reason is that both payer and payee use their hashed elliptic curve point as the address to send and receive payment. To further achieve the unlinkability, it is advised to use a new key pair for each transaction so that no multiple transactions can be linked to the same user. However, the anonymity and unlinkability provided by the current on-chain currency are vulnerable through different attacks like address reuse, blockchain and public address analysis [3][4], etc. Also, adversary can make use of public web crawlers the correlates social networks with cryptocurrency address like Bitcoin [5]. In [6] the authors mention that an adversary can associate the offline data like emails and shipping addresses with the online information, then eventually get the private information about the user. In [7] the authors show that cryptocurrency transactions can be linked to the user cookies and then to the user's real identity and the user's purchase history is revealed. Research [8][9] also shows that the IP address in cryptocurrency P2P network might be the vulnerability that compromises unlinkability.

To defend again these attacks, in [10] the authors propose ZeroCoin, which applies zero-knowledge authentication protocol to Bitcoin to provide anonymity. An extension of ZeroCoin is proposed by [11] as ZeroCash, which uses an improved version of zero-knowledge proof. Also, the second generation of cryptocurrencies like Ethereum [12] are designed with better consideration for security and privacy issues.

IV. USH FOR UNLINKABLE AND ANONYMOUS COIN SPENDING ON BFDCHAIN

The motivation to design and use of USH for BFDchain is to provide privacy-preserving and anonymous coin spending for both online and offline interactions for participants. We design USH based on Tate Paring cryptography that is also built on top on elliptic curve so that our protocol can be implemented on BFDChain easily.

4.1 Mathematical Background

Definition 1: Bilinear Pairing

A pairing is a bilinear map e: $G_1 \times G_1 \rightarrow G_2$ if, for any P, Q $\in G_1$ and any a, b $\in Z^*_q$ we have $e(a \cdot P, b \cdot Q) = e(a \cdot P, Q)^b = e(P, b \cdot Q)^a = e(P, Q)^{a \cdot b}$ and $e(P,Q) = e(Q, P)$

To provide an efficient computation of the bilinear map, we choose G_1, G_2 and e as a set of points on an elliptic curve, a multiplicative cyclic group over integers and Tate pairing, respectively. ECC has a special discrete logarithm problem, Elliptic Curve Discrete Logarithm Problem (ECDLP), defined as the basis of elliptic curve cryptography (ECC). Based on the parameters chosen above, another assumption we need is the *BDH Assumption* described as follows:

Definition 2: Bilinear Diffie-Hellman (BDH) Assumption

Given P, $a \cdot P$, $b \cdot P$, $c \cdot P$ for random a, b, c $\in Z^*_q$ and P $\in G_1$, it is not possible to compute $e(P, P)^{a \cdot b \cdot c}$ with a non-negligible probability, i.e., it is hard to compute $e(P, P)^{a \cdot b \cdot c}$

In the follows we present the detail of the use of the homomorphic randomization in USH.

Definition 3: Homomorphic Randomization Function in SH

Suppose user U joins a group g_i by obtaining a credential $g_i \cdot H_1(ID_U)$, where $g_i \in G_2$ represents the group secret and $H1(ID_U) \in G1$ represents U's assigned pseudonym. U selects a large random integer $s_U \in G_2$, and uses the addition operation defined in G_1 to compute $s_U \cdot g_i \cdot H_1(ID_U)$ (s_U times of addition of $g_i \cdot H_1(ID_U)$).

Formal Definition of Secret Handshake: Alice has a pair consisting of a pseudonym and a credential, *ID_A* and *C_A*, assigned from a central authority. Bob has his *ID_B* and *C_B* assigned from a central authority. After the execution of an SH protocol, Alice and Bob know that *C_A* and *C_B* indicate common same group membership if SH succeeds; otherwise, Alice (Bob) cannot know anything about *C_B*(*C_A*). In addition, Alice (Bob) cannot know who she (he) interacted with.

4.2 USH for BFDChain

We will use the Fig.1 to illustrate how USH works on top of BFDChain.

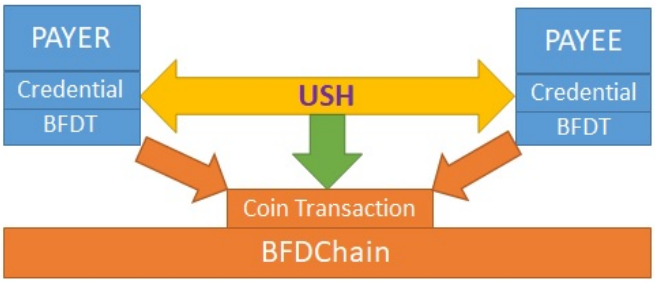

Fig.1 USH for BFDChain

In this section, we present the detail of USH on BFDChain. We assume that payer and payee have already registered in a central authority (CA) and obtained credential that can be used to authenticate their membership to a secret group. CA collaborates with BFDChain so that the credential assignment to payer and payee is a service on the chain, a sub-chain attached to the main chain. Then the payer and payee apply the unlinkable and anonymous secret handshake protocol, USH, to authenticate that the payer is eligible for the transaction/purchase. The payer can also verify the payee's eligibility if required. If authentication succeeds, the payer and payee continue for their transaction with BFDT. Otherwise, either party can cancel the transaction.

Our main idea is to use the homomorphic randomization function in the Protocol Randomization phase to randomize user's assigned pseudonyms to provide reusability of credentials. More specifically, we let a user generate a secret random number in execution of the secret handshake. The user multiplies the random number to an elliptic curve point that represents the user's pseudonym. The random number minimizes the correlation among authentication messages even they are produced by reuse of the same credential. First we present the parameters that are required in USH as (q, G_1, G_2, e, H_1, H_2). Here q is a large prime number, G_1 denotes an additive cyclic group of prime order q, G_2 denotes a multiplicative cyclic group of the same order q, H_1 is a collision-free hash function that maps a string with arbitrary length to an element in G_1, H_2 is a collision-free hash function that maps a string with arbitrary length to a string with fixed length, and e denotes a bilinear map. G_1 and G_2 are selected in such a way that *Discrete Logarithm Problem (DLP)* is assumed to be hard in both of them.

Our proposed USH has three phases: *Protocol Initialization*, *Group Secret Mapping and Authentication Computing*. To be consistent with other cryptographic protocol, we use Alice to represent payer and Bob for payee. In the follows we present the detail of the three phases.

Phase 1: Protocol Initialization

The CA determines the pairing parameters $(q, G_1, G_2, e, H_1, H_2)$ and group secrets $[g_1, ..., g_n]$, where $g_i \in G_2$. The CA publishes the pairing parameters while keeping the group secrets in private. Alice requests the CA to join the group with group secret $g_A \in [g_1, ..., g_n]$. The CA verifies Alice's qualification to decide whether Alice can join the group. If yes, the CA grants the group membership to Alice by issuing her a credential $g_A \cdot H_1(ID_A) \in G_1$, where $g_A \in G_2$. The credential is a secret of Alice to prove her membership in group g_A to another user in the same group. Alice cannot deduce g_A from $g_A \cdot H_1(ID_A)$ and $H_1(ID_A)$ by the assumption that DLP is hard in G_1. It is important for preventing forgery of credentials. Users Alice and Bob use their credentials, $K_A = g_A \cdot H_1(ID_A)$ and $K_B = g_B \cdot H_1(ID_B)$, to generate authentication messages to one another. Alice randomly generates two large random numbers, n_{A1} and s_{A1}. Alice computes $W_{A1} = s_{A1} \cdot H_1(ID_A)$, as described in the homomorphic randomization function above. That said, s_{A1} is used to minimize the correlations of authentication messages produced by the same credential. Since using a credential multiple times will not create messages that can link to the user identity, our credential is reusable. n_{A1} prevents replay attacks. Bob also randomly generates two large numbers, n_{B1} and s_{B1}, for the same purpose. Detailed interactions of group secret mapping phase and authentication computing phase are described as follows:

Phase 2 and Phase 3: Group Secret Mapping and Authentication Computing

(a) *Alice* $\rightarrow$ *Bob:* n_{A1}, $W_{A1} = s_{A1} \cdot H_1(ID_A)$

(b) *Bob: Compute* $V_{B,A} = H_2(U_{B,A} \,||\, n_{A1} \,||\, n_{B1} \,||\, 0)$, $U_{B,A} = e(W_{A1}, s_{B1} \cdot K_B)$. *Here Bob implements the secret mapping function to map his own credential* $K_B = g_B \cdot H_1(ID_B)$ *from an element in* G_1 *to an element in* G_2 *through bilinear map e.*

(c) *Bob* $\rightarrow$ *Alice:* n_{B1}, $W_{B1} = s_{B1} \cdot H_1(ID_B)$, $V_{B,A}$

(d) *Alice: Compute* $V'_{B,A} = H_2(U'_{B,A} \| n_{A1} \| n_{B1} \| 0)$, $U'_{B,A} = e(W_{B1}, s_{A1} \cdot K_A)$. *If* $V_{B,A} = V'_{B,A}$, *then Alice knows Bob belongs to the same group, i.e.,* $g_A = g_B$. *Otherwise, Bob belongs to a different group, i.e.,* $g_A \neq g_B$ *or Bob belongs to no group. Here Alice implements the result computing function to find out whether* $H_2(U'_{B,A} \| n_{A1} \| n_{B1} \| 0) = H_2(U_{B,A} \| n_{A1} \| n_{B1} \| 0)$.

(e) *Alice* $\rightarrow$*Bob:* $V_{A,B} = H_2(U'_{B,A} \| n_{A1} \| n_{B1} \| 1)$

(f) *Bob: Compute* $V'_{A,B} = H_2(U_{B,A} \| n_{A1} \| n_{B1} \| 1)$. *If* $V_{A,B} = V'_{A,B}$, *Bob knows that Alice belongs to the same group. Otherwise, Alice belongs to a different group, i.e.,* $g_A \neq g_B$ *or Alice belongs to no group.*

The protocol succeeds when $V_{B,A} = V'_{B,A}$ and $V_{A,B} = V'_{A,B}$ in steps (d) and (f). Based on the BDH assumption, it succeeds if, and only if, $g_A = g_B$. Otherwise, if it fails, Alice and Bob only know $g_A \neq g_B$. Users other than Alice and Bob cannot know whether $g_A = g_B$ or not, because they cannot compute $V'_{B,A}$ and $V'_{A,B}$ without K_A and K_B. A sketch of proof for $V_{B,A} = V'_{B,A}$ is shown in (1). The rest of proof for $V_{A,B} = V'_{A,B}$ can be derived similarly.

$$
\begin{aligned}
V_{B,A} &= H_2(U_{B,A} \| n_{A1} \| n_{B1} \| 0) = H_2(e(W_{A1}, s_{B1} \cdot K_B) \| n_{A1} \| n_{B1} \| 0) \\
&= H_2(e(s_{A1} \cdot H_1(ID_A), s_{B1} \cdot g_B \cdot H_1(ID_B)) \| n_{A1} \| n_{B1} \| 0) \\
&= H_2(e(s_{A1} \cdot g_B \cdot H_1(ID_A), s_{B1} \cdot H_1(ID_B)) \| n_{A1} \| n_{B1} \| 0) \\
&= H_2(e(s_{B1} \cdot H_1(ID_B), s_{A1} \cdot g_B \cdot H_1(ID_A)) \| n_{A1} \| n_{B1} \| 0) \\
&= H_2(e(W_{B1}, s_{A1} \cdot K_A) \| n_{A1} \| n_{B1} \| 0) \quad \text{//iff, } g_A = g_B \\
&= H_2(U'_{B,A} \| n_{A1} \| n_{B1} \| 0) = V'_{B,A} \qquad (1)
\end{aligned}
$$

The secret handshake protocol guarantees that 1) Bob cannot link Alice's real-world identity to her BFDChain address after authenticating her credential. 2) Alice's credential is unlinkable. In future the same kind of transactions won't be linked to her previous transaction. Furthermore, even the CA cannot know that it was Alice who made the purchase with Bob given the public ledger of BFDChain.

V. SECURITY ANALYSIS

In the proposed USH protocol we consider any passive attack that aims at compromising confidentiality of sensitive information by analyzing transmitted messages, e.g., dictionary attack to transmitted messages. We consider both outside adversaries and inside adversaries as follows.

Outside adversary: a malicious party that is an outsider of an authentication process. An outside adversary does not participate in the authentication process and knows nothing about the transmitted messages.

Inside adversary: a malicious party that participates in an authentication process. The adversarial insider may have some matched group secrets with the targeted victim and try to discover other unmatched group secrets the victim has.

Based on the adversary model and attack model introduced above, we claim that our privacy-preserving correlation technique provides the following main security properties: unlinkability with reusable credential and group membership authenticity. In Theorem 1 we first prove that USH that uses homomorphic randomization function provides unlinkability with reusable credential.

Theorem 1. USH holds Unlinkability on Homomorphic Randomization

A privacy-preserving authentication protocol that uses homomorphic randomization function in protocol initialization phase guarantees unlinkability with reusable credential.

Proof: Suppose user k requests to join a group with group secret $g_k \in G_2$. The CA grants the group membership to k by issuing a credential $g_k \cdot H_1(ID_K) \in G_1$, where ID_k represents k's assigned pseudonym associated with g_k. k cannot deduce g_k from $g_k \cdot H_1(ID_k)$. There are two adversaries, t_1 and t_2, who want to know whether they are interacting with same party and which group this party belongs to. Assume that t_1 is not a member in group g_k and t_2 is. t_2 may communicate with other legitimate owners of group secret g_k, corrupt some valid parties and obtain their secrets. Here we use U^k to denote the set of users who own the group secret g_k. Now we define a *Linkability Detection Game* as follows.

Step 1: The adversaries t_1 *and* t_2 *communicate with k based on their own choices. From* t_1*'s viewpoint, he is interacting with party* k_1*; from* t_2*'s viewpoint, he is interacting with party* k_2.

Step 2: T_2 *selects other parties* U^c *and corrupt them.*

Step 3: After the executions of USH protocol t_1 *and* t_2 *want to find out whether* $k_1 = k_2$. *If yes, they know that they are interacting with a same party and* t_2 *knows that* k_2 *has group secret* g_k.

We say that t_1 and t_2 win the Linkability Detection Game if they find out that $k_1 = k_2$ and k_2 has group secret g_k.
Now we define the following probabilities:

$G_L = Pr[t_1 \text{ and } t_2 \text{ win Linkability Detection Game}] - 0.5$

When t_2 does not compromise any valid owner of U^k, the above probability becomes:

$G_{L|(U^c \cap U^k)=\varnothing} = Pr[t_1 \text{ and } t_2 \text{ win Linkability Detection Game} \mid (U^c \cap U^k)=\varnothing] - 0.5$

Here we say that our protocol holds unlinkability with reusable credential if $G_{A|(U^c \cap U^k)=\varnothing}$ is negligible for any adversaries t_1 *and* t_2. In our protocol k generates random numbers s_{k1} and s_{k2} to manipulate his assigned pseudonym ID_k and get elliptic curve points $W_{k1} = s_{k1} \cdot H_1(ID_k)$ and $W_{k2} = s_{k2} \cdot H_1(ID_k)$ as his randomized identities for t_1 and t_2, respectively. Here W_{k1} and W_{k2} are generated to map k's pseudonyms to random ECC points through the homomorphic randomization function as random oracles. By the assumptions that DLP is hard in G_1, no polynomial-time adversary cannot deduce s_{k1} and s_{k2} from $W_{k1} = s_{k1} \cdot H_1(ID_k)$ and $W_{k1} = s_{k1} \cdot H_1(ID_k)$ and thus cannot know W_{k1} and W_{k2} are generated based on the same pseudonym ID_k. That said, their guess about whether $k_1 = k_2$ is no better than a random guess. Our protocol holds unlinkability with reusable credential.

Theorem 2. USH holds Group Membership Authenticity

In our protocol, the group membership authenticity means that an adversary cannot convince another party that it owns the group memberships as this party has by interacting with him. In other words, the adversary cannot impersonate owners of some targeted group secrets. Because of the hardness of ECDLP and BDH assumption, our protocol provides group membership authenticity that any polynomial-time adversary only has negligible probability of cheating as a valid owner of some group membership without corrupting another valid owner of the targeted group secret. Now we define the *Group Membership Owner Impersonation game*.

Proof: suppose there is an adversary A who aims at impersonating the owner of a group secret s. A may communicate with legitimate owners of the targeted s in network G, corrupt some valid users and obtain their secrets. Here we use U^T to denote the set of users who own the targeted s. A picks a target user $u^T \in U^T$, and wants to convince u^T that A is also an owner of the targeted s, i.e. $A \in U^T$. We define the *Group Membership Owner Impersonation Game* for a randomized, polynomial-time adversary A as follows.
Step 1: The adversary A communicates with owners of the targeted s based on its own choice. A may compromise certain user $U^C \subseteq U$ and obtain their secrets, where U^C denotes the set of compromised users and U denotes the whole user set in the network G.
Step 2: A selects a target user $u^T \notin U^C$ and $u^T \in U^T$, where users in U^T own the targeted s.
Step 3: A wants to convince u^T that A owns the s, i.e. $A \in U^T$.

We say that A wins the Group Membership Owner Impersonation Game if it convinces u^T that it is an owner of s.
Now we define the following probabilities:

$G_A = Pr[A \text{ wins Group Membership Owner Impersonation Game}]$

When A does not compromise any valid owner of s, the above probability becomes:

$G'_{A|(U^C \cap U^T)=\varnothing} = Pr[A \text{ wins Group Membership Owner Impersonation Game} \mid (U^C \cap U^T)=\varnothing]$

Here we say that our protocol holds *Group Membership Authenticity* if $G'_{A|(U^C \cap U^T)=\varnothing}$ is negligible for any adversary A. Similar to Theorem 1, we can further prove that USH holds group membership authenticity. Detail of the proof is omitted here since it is out of the scope of this paper.

Theorem 1 proves that payer only needs to obtain the credential from the central authority once, then he can process this credential with homomorphic randomization and use it any times for authentication without being linked to his real identity or his original credential. Theorem 2 proves that no one can impersonate the owner of the assigned credential.

VI. CONCLUSION

In this paper, we proposed USH to achieve privacy-preserving and anonymous coin spending for BFDChain. We discussed the motivation and provided the formal security analysis of the proposed protocol. It will be critical for the success of any on-chain currency since it allows participants to bring

their real-world membership to the cryptocurrency for their coin spending with unlinkability and anonymity.

REFERENCES

[1] G. O. Karame, E. Androulaki, and S. Capkun, "Double-spending fast payments in bitcoin," in Proceedings of the 2012 ACM Conference on Computer and Communications Security, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 906–917.

[2] N. T. Courtois and L. Bahack, "On subversive miner strategies and block withholding attack in bitcoin digital currency," CoRR, vol. abs/1402.1718, 2014.

[3] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in Financial Cryptography and Data Security: 17th International Conference, FC 2013. Springer Berlin Heidelberg, 2013, pp. 6–24.

[4] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun,"Evaluating user privacy in bitcoin," in Financial Cryptography and Data Security: 17th International Conference, FC 2013. Springer Berlin Heidelberg, 2013, pp. 34–51.

[5] M. Fleder, M. S. Kester, and S. Pillai, "Bitcoin transaction graph analysis," CoRR, vol. abs/1502.01657, 2015.

[6] P. Koshy, D. Koshy, and P. McDaniel, "An analysis of anonymity in bitcoin using p2p network traffic," in Financial Cryptography and Data Security: 18th International Conference, FC 2014,. Springer Berlin Heidelberg, 2014, pp. 469–485.

[7] S. Goldfeder, H. A. Kalodner, D. Reisman, and A. Narayanan, "When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies," CoRR, 2017.

[8] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin p2p network," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ser.CCS '14. ACM, 2014, pp. 15–29.

[9] A. Biryukov and I. Pustogarov, "Bitcoin over tor isn't a good idea," in 2015 IEEE Symposium on Security and Privacy, May 2015, pp. 122–134.

[10] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in 2013 IEEE Symposium on Security and Privacy, May 2013, pp. 397–411.

[11] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in 2014 IEEE Symposium on Security and Privacy, May 2014, pp. 459–474.

[12] Ethereum: A secure decentralized generaliaed transaction ledger EIP-150 revision", Gavin Wood