A Practical Dynamic Enhanced BFT Protocol

Feng Shen, Yu Long(✉), Zhen Liu(✉), Zhiqiang Liu(✉), Hongyang Liu, Dawu Gu(✉), and Ning Liu(✉)

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
{shenfeng2017,longyu,liuzhen,ilu_zq,LiuHongyang,dwgu,ningliu}@sjtu.edu.cn

Abstract. Emerging as a distributed system maintaining a public ledger via consensus protocol, blockchain technology is showing its great potential in various scenarios such as supply chain, financial industry, internet of things (IoT), etc. Among kinds of consensus protocols, Byzantine Fault Tolerance (BFT) protocols are playing an important part in the design of the blockchain system. Most BFT protocols, however, are static with no support for a dynamic property (i.e. nodes can join/leave a working system) and lack mechanisms to punish faulty nodes, which highly limit their wider adoption in the practical settings. This paper presents a dynamic enhanced BFT (DEBFT) protocol that is designed to support dynamic property and faulty nodes punishment. Based on HoneyBadger BFT, DEBFT employs Dynamic Threshold Identity-based Encryption and Distributed Key Generation to enable changes of the consensus group without reconfiguring the whole system, besides, evaluation metrics are also introduced to evaluate consensus nodes and clear faulty ones out of the system.

Keywords: Blockchain · Consensus · Dynamic property · BFT protocols

1 Introduction

The emergence of Bitcoin in 2008 [18] opened up the era of blockchain technology, which is an ingenious innovation that realizes consensus among distributed nodes by combining P2P network, distributed database, consensus protocol, cryptography, game theory and so on. Serving as a distributed ledger maintained by multiple participants, blockchain has shown its great potential in cryptocurrencies, financial industry, internet of things (IoT) and many other scenarios.

Blockchain can be classified into permissionless and permissioned ones according to entry limitation for network nodes. In a permissionless blockchain such as Bitcoin and Ethereum [21], anyone can join the system without a specific identity, whereas in a permissioned setting, every node maintaining the public ledger has an identity, and this identity is known to any other nodes in the system

J. K. Liu and X. Huang (Eds.): NSS 2019, LNCS 11928, pp. 288–304, 2019.
https://doi.org/10.1007/978-3-030-36938-5_17

even though they may not trust each other. With the development of blockchain technology, permissioned blockchain is gaining more and more attraction within business and financial fields for its efficiency and controllability.

In the design of blockchain, the consensus mechanism is a core part to reach an agreement on the global ledger. In traditional distributed systems like database and file system, Byzantine Fault Tolerance (BFT) protocols have been intensively studied. Normally, BFT protocols run among a fixed set of consensus nodes, and can finally reach deterministic consensus and high efficiency with the tolerance of less than 1/3 malicious nodes. Due to these merits, BFT protocols especially PBFT [8] and its derivatives [4,19] are widely deployed to construct consensus mechanisms in permissioned blockchain systems, e.g. Hyperledger Fabric [1], Tendermint [13] et al.

1.1 Research on BFT Protocols

Classical BFT protocols need a leader to lead the consensus process, and it will be replaced if it is found faulty. PBFT [8] proposed in 1999 is a typical BFT protocol of such a case, and it is the first practical BFT protocol that works under weak synchronous network assumption. Later works extend PBFT to simplify the design and reduce the cost, e.g. Zyzzyva [12] proposed in 2007 allows clients to adopt proposals from the leader optimistically and solve the inconsistency if needed. However, the main problem for BFT protocols with a leader is its vulnerability when suffering DoS attacks [16].

Some leaderless BFT protocols have been constructed in the face of DoS attacks. HoneyBadger BFT [16] is the first practical asynchronous BFT protocol without a leader. It combines threshold encryption with an efficient realization of Asynchronous Common Subset (ACS), together with a random selection method for proposals. With these in hand, the communication cost of HoneyBadger BFT is greatly reduced. In 2018, Duan et al. [9] extended HoneyBadger BFT to BEAT by replacing its cryptography components. And BEAT provides different versions for various scenarios. Hashgraph [3] is another leaderless BFT protocol that takes advantage of a kind of gossip about gossip design to realize virtue voting, as a result, it saves much communication overhead.

Besides the single consensus scheme, BFT protocols are also introduced together with other consensus protocols to achieve better performance. Typical blockchain schemes including Byzcoin [11], RapidChain [22], Elastico [15] et al. combine proof-of-work (PoW) with BFT to achieve both security and efficiency. Apart from them, Cosmos [14] combines delegated proof-of-stack (DPoS) with BFT under the same consideration.

All BFT protocols listed above, however, can only support static setting in which network nodes are fixed, and they can not support dynamically changes of the consensus group without reconfiguring the whole system. Indeed, the change of consensus group for a permissioned blockchain system is of necessity in business and financial fields, and without support for dynamic property, the reconfiguration process may be burdensome and costly. On the other hand, how to detect inactive or even malicious nodes is also of great importance, but it is not referred in most BFT protocols. Based on these observations, we carry out this work to build an enhanced BFT protocol atop HoneyBadger BFT, and a comparison among related BFT protocols is shown in Table 1.

Table 1. Comparation among typical BFT protocols

BFT protocols	Optimal Comm. Compl	Worst Comm. Compl	Dynamic property	DoS resilience
PBFT	$O(N^2)$	∞	No	No
HoneyBadger BFT	$O(N)$	$O(N)$	No	Yes
BEAT	$O(N)$	$O(N)$	No	Yes
This work	$O(N)$	$O(N)$	Yes	Yes

Comm. Compl means Communication Complexity

1.2 Our Contribution

- We propose a new dynamic enhanced BFT (DEBFT) protocol under leaderless setting with $O(N)$ communication complexity, which allows consensus nodes to dynamically join and leave the network.
- We design Join and Quit protocol, providing detailed protocol procedures as well as data structures of relevant messages types. By Join and Quit protocol, DEBFT allows a consensus node to join or leave the consensus group without reconfiguring the whole system.
- We describe the metrics to assess the behavior of a consensus node, and design Clear protocol to clear malicious or inactive consensus nodes out of the system to maintain its long-term benign work.
- We analyze key properties of the DEBFT, including *dynamic property, fairness, agreement, and total order.*

1.3 Paper Organization

The rest of this paper is organized as follows. We start by introducing the system module (Sect. 2). After that, the overview of HoneyBadger BFT is described (Sect. 3), as well as the building blocks of this work (Sect. 4). Next, we give the detailed protocol design (Sect. 5) and its security analysis (Sect. 6), then we conclude the paper (Sect. 7).

2 System Module

Participants Definition. There are three kinds of participants in this system: client, dealer, and consensus node (noted as node in the following context for simplification). Here we depict their roles in our protocol.

- Clients generate transaction requests and broadcast them to all nodes. Clients will also gather feedback from nodes. Enough signatures from different nodes on one transaction denote it has been output as a consensus result.
- Dealer represents a trusted party providing validity check for nodes. Hence it is responsible for the initialization of the system and leads the dynamic joining process of nodes.
- Nodes are validated by the dealer before joining the system. They will receive transactions from clients, then execute the protocol together to get the consensus result that decides which transactions should be output.

Timing Assumption. The system works under a partially synchronous network assumption [10]. For all nodes computation proceeds in synchronized rounds and messages are received by their recipients within some specified time bound. We assume that the nodes are equipped with synchronized clocks to guarantee this round synchronization. In our system, there exists an adversary. In every round of communication, the adversary can wait for the messages of the uncorrupted nodes to arrive, then decide on his computation and communication strategy for that round. The adversary can still ensure that his messages delivered to the honest nodes on time. Therefore we should always assume the worst case that the adversary speaks last in every communication round.

Security Module. Assuming that there exists an adversary willing to prevent the system from making consensus or to subvert the system, here gives the definition of its ability.

The adversary can completely control up to t corrupted nodes, and t satisfies $3t+1 \leq n$ where n is the total number of consensus nodes. The controlled nodes are called faulty nodes standing on the opposite of honest nodes which totally obey the protocol. Faulty nodes can choose arbitrary malicious actions, including not responding, sending conflicting messages to different nodes, corrupting other nodes and so on.

3 Reviewing HoneyBadger BFT

HoneyBadger BFT is the first practical asynchronous BFT protocol. Clients in the network will send their transaction requests to all nodes, and nodes execute the BFT protocol in consecutive rounds. At the beginning of a round every node will raise its proposal and at the end of this round a common subset of proposals will be output as the consensus result. Figure 1 shows the basic working procedures. Next we will depict main components applied in HoneyBadger BFT.

3.1 Protocol Components

HoneyBadger BFT mainly consists of two components: threshold encryption and Asynchronous Common Subset (ACS). In every round, each node chooses a set of transactions as its proposal and encrypts it through threshold encryption scheme, then submits the ciphertext as input to ACS module and a common subset of these ciphertexts will be output. At last the subset will be decrypted still by threshold encryption scheme to get final consensus results.

HoneyBadger BFT uses the threshold encryption scheme TPKE from Baek and Zheng [2]. In the design of TKPE, to decrypt the ciphertext, at least $t+1$ nodes need to get their decryption shares separately and combine them together. This design ensures that the adversary controlling less than t faulty nodes can not get the plaintext without the help of honest nodes. As TPKE is secure under the adaptive chosen ciphertext attack, its application in HoneyBadger BFT helps to realize censorship resilience property which prevents the adversary from delaying honest client requests on purpose. ACS is the main module to reach consensus among nodes. As Fig. 1 shows, it consists of Reliable Broadcast (RBC) module and Binary Agreement (BA) module. In HoneyBadger BFT, RBC module from Cachin and Tessaro [7] is used to transmit the proposal of each node to all other nodes. BA module from Mostéfaoui [17] is used to decide on a bit vector indicating which proposals should be output as the consensus result.

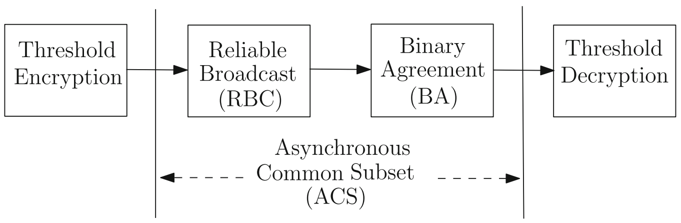

Fig. 1. HoneyBadger BFT

3.2 Protocol Security Properties

On a high level HoneyBadger BFT satisfies below properties [16]:

- (Agreement) If any correct node includes transaction tx into its consensus result, then every correct node includes tx into its consensus result.
- (Total Order) If one correct node has output the sequence of transactions $\langle tx_0, tx_1, \dots tx_j \rangle$ and another has output $\langle tx'_0, tx'_1, \dots tx'_{j} \rangle$, then $tx_i = tx'_i$ for $i \leq min(j, j')$.
- (Censorship Resilience) If a transaction tx is input to $n - t$ correct nodes, then it is eventually output by every correct node. Intuitively this means the adversary cannot prevent a transaction from being output as a consensus result.

3.3 Limitations of HoneyBadger BFT

As the first practical asynchronous BFT protocol which guarantees both liveness and safety, HoneyBadger BFT has abundant application scenarios like banks and financial institutions. It is especially suitable for permissioned blockchain applications, where a fixed number of nodes are authorized to enter the system.

However, HoneyBadger BFT is a traditional static BFT protocol which means that it could not support a consensus node to join or leave the consensus group without reconfiguring the whole system. On the other hand, there may exist malicious or inactive nodes in the system, and the current protocol lacks the function to detect these faulty nodes to exclude them from the system, this may hamper the long-term benign working of the system.

Considering above limitations, this paper focuses on the dynamic property and clear function of BFT protocol. Through the application of cryptography components introduced in Sect. 4, together with rational protocol procedure, we construct the new protocol DEBFT which is based on HoneyBader BFT.

4 Building Blocks

This section describes several cryptography components applied in our work. We will separately show their functions and details.

Dynamic Threshold Identity-Based Encryption (DTIBE). The threshold encryption scheme TPKE [2] in HoneyBadger BFT can only set the threshold parameter and define the consensus group during the configuration phase. We utilize the dynamic threshold identity-based (DTIBE) scheme from Susilo et al. [20] for its dynamic property, where after the initialization of the system, a sender can dynamically select the set of recipients as well as dynamically set the threshold t upon the creation of ciphertext. This character meets the need of joining and quitting of members in a working consensus system. Figure 2 shows the details.

DTIBE Scheme

A DTIBE scheme is comprised of the following five steps.

– $\boldsymbol{DTIBE.Setup(1^\lambda, N) \rightarrow (MPK, SK)}$

• Takes the security parameter 1^λ and the maximum network node number N as input, the setup algorithm outputs (MPK, SK), where MK is the master public key and SK is the master secret key.

– $\boldsymbol{DTIBE.Gen(ID, MPK, SK) \rightarrow d_{ID}}$

• Takes as input an identity ID and the master key pair (MPK, SK), the key generation algorithm outputs a private key of ID, which is denoted by d_{ID}.

– $\boldsymbol{DTIBE.Enc(ID_1, ID_2, \ldots, ID_n, t, M, MPK) \rightarrow CT}$

• Takes as input a set of identities $ID_1, ID_2, \ldots, ID_n$, a threshold number t, a message M and the master public key MPK, where $n \leq N$ and $\lfloor (n-1)/3 \rfloor \leq t \leq n$, the encryption algorithm outputs a ciphertext, which is denoted by CT.

– $\boldsymbol{DTIBE.ShareDec(CT, d_{ID}, MPK) \rightarrow S_{ID}}$ ***or*** $\boldsymbol{\perp}$

• Takes as input a ciphertext CT for $(ID_1, ID_2, \ldots, ID_n, t)$, a private key d_{ID} of identity ID and the master public key MPK, the decryption algorithm outputs a decryption share denoted by S_{ID} or $\perp$.

– $\boldsymbol{DTIBE.Combine(CT, S_{ID_1'}, S_{ID_2'}, \ldots, S_{ID_n'}, MPK) \rightarrow M}$ ***or*** $\boldsymbol{\perp}$

• Takes as input a ciphertext CT for $(ID_1, ID_2, \ldots, ID_n, t)$, at least t decryption shares $S_{ID_1'}, S_{ID_2'}, \ldots, S_{ID_n'}$ and the master public key MPK, the combination algorithm outputs the message M or $\perp$.

Fig. 2. The algorithm procedure of DTIBE

Distributed Key Generation (DKG). HoneyBadger BFT uses BA to decide on a bit vector representing the proposals to be output. Its BA scheme uses the Boldyreva's pairing-based threshold signature scheme [5] as a randomizer to build the common coin which can not support changes of consensus group flexibly. In this work we use the Distributed Key Generation (DKG) scheme from Gennaro et al. [10] to generate randomness and the updated BA is represented as $\widetilde{BA}$ in our work. To distinguish from the primitive ACS in HoneyBadger BFT, we define the combination of RBC and $\widetilde{BA}$ as Round Common Subset (RCS) in our scheme. Figure 3 shows the details to generate randomness by the DKG scheme.

the Common Coin Based on DKG

DKG allows a set of n parties to jointly generate a pair of public and private keys according to the distribution defined by the underlying cryptosystem.
For discrete log based schemes, DKG amounts to generating a secret sharing of a random, uniformly distributed value x and making public the value $y = g^x$. Hence we use this as a randomizer to build the common coin in $\widetilde{BA}$ protocol.
Assuming there are n nodes labeled from P_1 to P_n, each node has identities of all other nodes and each node can not be fabricated. The public parameter is node number n and faulty node number t which works as a threshold and satisfies $\lfloor (n-1)/3 \rfloor \leq t \leq n$.

– ***Generating x :***
• Each player P_i performs a $Pedersen - VSS$ of a random value z_i as a dealer.
• Each player verifies the values, then builds the set of non-disqualified players $QUAL$.
• The distributed secret value x is not explicitly computed by any party, but it equals $x = \sum_{i \in QUAL} z_i mod\ q$.
– ***Extracting $y = g^x$ mod p***
• Each player $i \in QUAL$ exposes $y_i = g^{z_i}\ mod\ p$ via Feldman VSS.
• After verification, each player computes $y = \prod_{i \in QUAL} y_i$.
– ***Generating randomness***
• Get $Value = h(y)$ via a hash function $h()$.

Fig. 3. The common coin based on DKG

Boneh-Boyen Short Signature Scheme. In Sect. 2, we assume the identity of a node can not be fabricated. For this purpose, a node needs to transmit kinds of messages to other nodes with its signature. In this work this is ensured by using the Boneh-Boyen short signature scheme [6], and its main procedures are shown in Fig. 4.

Boneh-Boyen Signature Scheme

– ***Key Generation*** **:**

• $KeyGen(k) \rightarrow (SIG_{pk_i}, SIG_{sk_i})$. k stands for the security parameter. SIG_{sk_i} is used for a node to generate signature and SIG_{pk_i} is used by other nodes to check the validity of its signature.

– ***Signature Generation*** **:**

• $SigGen(m, SIG_{sk_i}) \rightarrow \delta$. m is the message to be signed and δ stands for the output signature.

– ***Signature Verification*** **:**

• $SigVerify(\delta, SIG_{pk_i}) \rightarrow 1\ or\ 0$. Other nodes verify whether δ and SIG_{pk_i} matches. If it is a valid signature output 1 otherwise output 0.

Fig. 4. Boneh-Boyen signature scheme

5 Protocol Design

This chapter provides the detailed description of our dynamic enhanced BFT protocol DEBFT, including the initialization phase, regular consensus procedure and Join/Quit/Clear protocols for specific functions.

5.1 System Initialization

As mention in Sect. 3, there exist clients, a dealer, and consensus nodes in the system. In our system, D_{trust} represents the trusted dealer. N is the possible maximal number of nodes. n is the current number of nodes. Let P_i represents the ith node in the system, where $i \in \{1, 2, \cdots, n\}$. t is the maximal number of faulty nodes, which satisfies $t = \lfloor (n-1)/3 \rfloor$.

Before the system begins to work, each node P_i with identity ID_i will generate its signature key pair (SIG_{pk_i}, SIG_{sk_i}) by the Boneh-Boyen signature scheme. SIG_{pk_i} is the public key of node P_i, and SIG_{sk_i} is its secret key to generate signature. D_{trust} verifies whether each node P_i should join the consensus group, then executes the $DTIBE.Setup$ algorithm and broadcasts master public key MPK to all valid nodes in network. D_{trust} will generate and send the secret key d_{ID_i} for node P_i via the $DTIBE.Gen$ algorithm. The public information of node P_i can be represented by a pair (ID_i, SIG_{pk_i}) which is signed and bound by the D_{trust}.

5.2 Regular Consensus Procedure

This part depicts the whole process for transactions to be confirmed. The consensus protocol proceeds in consecutive rounds. Assuming the current round is numbered as r, each node generates its proposal then the consensus protocol executes among all nodes. Finally the consensus result of round r is a common subset of the proposals. Figure 5 gives an intuitive representation of the whole procedure, detail steps are as follows.

– ***step* 1: *Batch Selection***

- Clients send requests to all nodes, and each node P_i stores enough requests in a local FIFO queue buf_i. There is a properly set parameter B representing the batch size. P_i randomly selects B/n elements from the first B elements of buf_i.

– ***step* 2: *Threshold Encryption***

- P_i chooses the decryption set and uses the $DTIBE.Enc$ algorithm to encrypt the elements selected in step 1, and gets ciphertext v_i as output.

– ***step* 3: *Agreement on Ciphertexts***

- $RCS[r]$ is responsible for generating the common subset of proposals for round r. In this phase n nodes run $RCS[r]$ together, and each node P_i passes v_i as input to it. $RCS[r]$ consists of n reliable broadcast instances $\{RBC_i\}_n$ to disseminate the n proposals and n binary agreement instances $\{\widetilde{BA}_i\}_n$ to collect votes for the n proposals. Here node P_i is the sender of RBC_i and $\widetilde{BA}_i$ decides whether its proposal will be accepted. Detailed process is listed as follows.

– v_i is the input of RBC_i, and RBC_i will disseminate it to all other nodes.
– For node P_i, upon delivery of v_j from RBC_j, if input has not yet been provided to $\widetilde{BA}_j$, then provide input 1 to $\widetilde{BA}_j$. as referred previously DKG scheme is used to generate randomness during the process of $\widetilde{BA}_j$.
– Upon delivery of value 1 from at least $n-t$ instances of $\widetilde{BA}$, provide input 0 to each instance of $\widetilde{BA}$ that has not yet been provided input.
– Once all instances of $\widetilde{BA}$ have completed, get output from $RCS[r]$: an agreement on a common subset $\{v_j\}_{j\in S}$, where $S \subset [1\ldots n]$ containing all indexes of each $\widetilde{BA}$ that delivered 1.

– ***step* 4: *Threshold Decryption***

- Each node P_i uses $DTIBE.ShareDec$ to calculate its decryption shares for $v_j, j \in S$.
- Each node P_i multicasts the shares to all other nodes.
- For each element in $\{v_j\}_{j\in S}$, node P_i waits until receiving at least $t+1$ valid shares, then uses $DTIBE.Combine$ to get the plaintext $\{y_j\}_{j\in S}$.
- Let $\{block\}_r = sorted(\cup y_j), j \in S$, such that $\{block\}_r$ is sorted in a canonical order (e.g., lexicographically).
- P_i updates buf_i and gets ready for round $r+1$.

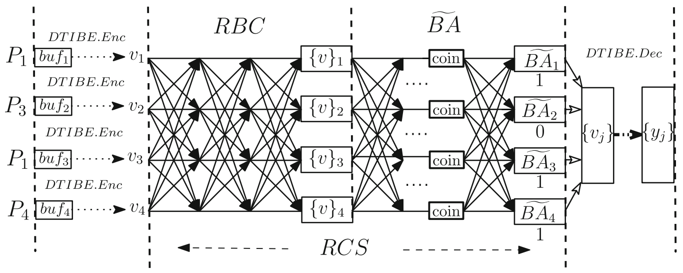

Fig. 5. Regular consensus procedure

5.3 Join Protocol

Join protocol is designed to allow a new node to join the network without stopping and reconfiguring the whole system.

Assuming there already exist n nodes in the system, a new node numbered as $n+1$ is willing to join the system, which satisfies $n+1 \leq N$. The protocol process is shown in Fig. 6 and we show the details as follows.

– ***step* 1: *Join Request Phase***

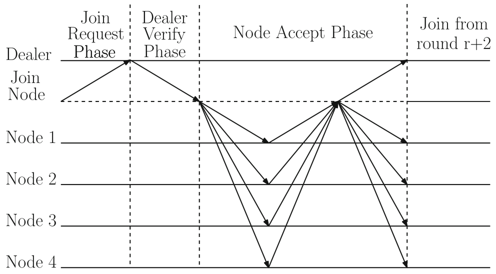

Fig. 6. Join protocol

- Node P_{n+1} invokes $KeyGen$ of Boneh-Boyen signature scheme to generate its key pairs $(SIG_{pk_{n+1}}, SIG_{sk_{n+1}})$.
- Node P_{n+1} sends $JoinReq$ to D_{trust}.

$$JoinReq = ((Join, SUB_{n+1}, SIG_{pk_{n+1}}, ID_{n+1}), SigGen(h(m), SIG_{sk_{n+1}})) \quad (1)$$

$Join$ is a message type. SUB_{n+1} is the materials submitted by P_{n+1} to be verified by D_{trust}. $SIG_{pk_{n+1}}$ is the public key of node P_{n+1}. ID_{n+1} is the ID of P_{n+1}. $h(m)$ is the hash of the concatenation of the information above, and this applies to all $h(m)$ in the following description. $SigGen(h(m), SIG_{sk_{n+1}})$ is the signature of P_{n+1} on $h(m)$.

– ***step* 2: *Dealer Verify Phase***

- After receiving $JoinReq$ from node P_{n+1}, D_{trust} firstly checks SUB_{n+1} to verify whether node P_{n+1} should join the system, then checks the correctness of $SigGen(h(m), SIG_{sk_{n+1}})$.
- If node P_{n+1} passes above verification, D_{trust} uses $DTIBE.Gen$ to generate $d_{ID_{n+1}}$ as the private key of node P_{n+1} in DTIBE scheme. Next D_{trust} sends $JoinAcc$ with $d_{ID_{n+1}}$ to node P_{n+1}.

$$JoinAcc = ((Accept, JoinReq), SigGen(h(m), SIG_{sk_{D_{trust}}})) \tag{2}$$

$Accept$ is a message type. $SigGen(h(m), SIG_{sk_{D_{trust}}})$ is the signature of D_{trust} on $h(m)$.

– ***step* 3: *Node Accept Phase***

- After receiving $JoinAcc$, Node P_{n+1} broadcasts $JoinAcc$ to all nodes.
- For each node $P_i, i \in (1, \ldots, n)$, after receiving $JoinAcc$ from node P_{n+1}, if the signature of D_{trust} is valid then replies $NodeJoinAcc$ to node P_{n+1}.

$$NodeJoinAcc = ((JoinConf, ID_{n+1}, ID_i, r), SigGen(h(m), SIG_{sk_i})) \tag{3}$$

$JoinConf$ is a message type. ID_i is the ID of node P_i. r is the current round number. $SigGen(h(m), SIG_{sk_i})$ is signature of P_i on $h(m)$.

- After receiving $2t{+}1$ $NodeJoinAcc$ messages, node P_{n+1} packages and broadcasts them to all nodes and D_{trust}. P_{n+1} will join the network from round $r+2$. This design is to ensure that a node will only join the system from the beginning of a round, and ensures there exists at least the time of a round for messages to be delivered.
- After receiving $2t+1$ valid $NodeJoinAcc$, node P_i will include node P_{n+1} into the system from round $r+2$.

5.4 Quit Protocol

Quit protocol is designed to allow a node to quit from the network without stopping and reconfiguring the whole system. When a node $P_k, k \in (1, \ldots n)$ wants to leave the network, The protocol process is shown in Fig. 7 and below gives the details.

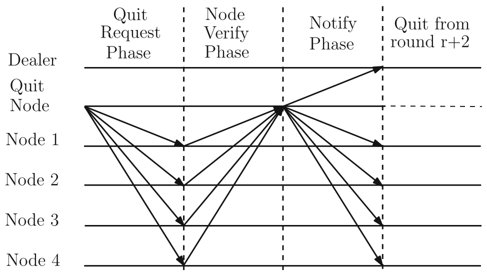

Fig. 7. Quit protocol

– ***step* 1: *Quit Request Phase***

- Node P_k broadcasts $QuitReq$ to all nodes in the system.

$$QuitReq = ((Quit, ID_k, SIG_{pk_k}), SigGen(h(m), SIG_{sk_k})) \tag{4}$$

$Quit$ is a message type. ID_k is the identity of P_k and SIG_{pk_k} is its public key. $SigGen(h(m), SIG_{sk_k})$ is signature of node P_k on $h(m)$.

– ***step* 2: *Node Verify Phase***

- For each node $P_i, i \in (1, \ldots, n)$ in the system, after receiving $QuitReq$ from node P_k, if the signature is valid replies $NodeQuitAcc$ to node P_k.

$$NodeQuitAcc = ((QuitConf, ID_k, ID_i, r), SigGen(h(m), SIG_{sk_i})) \tag{5}$$

$QuitConf$ is a message type. ID_i is the identity of node P_i. r is the current round number. $SigGen(h(m), SIG_{sk_i})$ is the signature of P_i on $h(m)$.

– ***step* 3: *Notify Phase***

- After receiving $2t + 1$ $NodeQuitAcc$ messages, node P_k packages and broadcasts them to all nodes and D_{trust}. P_k will quit from the network from round $r + 2$.
- After receiving the $2t + 1$ valid $NodeQuitAcc$ message, node P_i will exclude node P_k from the network from round $r + 2$.

5.5 Clear Protocol

Clear protocol is design to exclude malicious or inactive nodes from consensus group. Before depicting details of Clear protocol, we describe how to evaluate actions of nodes to find faulty ones.

Firstly consider malicious behavior, if node P_i detects malicious behavior of node P_j, it will invoke Clear protocol at once. Generally, malicious behavior includes below items:

– The proposal of node P_j is finally output in the common subset, while it contains invalid transactions.
– Node P_j is found sending conflicting messages to different nodes.

Another kind of faulty behavior is being inactive, which needs to be measured properly. Assuming w is number of rounds properly chosen, if during the most recent w rounds, node P_j output less than p proposals totally, then other honest nodes can invoke Clear protocol for it. Assume there already exist n nodes in the system and a node $P_s, s \in (1, \ldots n)$ is detected as malicious or inactive. To keep the liveness and robust of the system, P_s needs to be cleared from the network. The protocol process is shown in Fig. 8 and the details is given as follows.

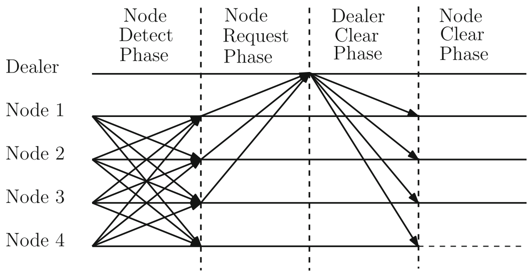

Fig. 8. Clear protocol

– ***step* 1: *Node Detect Phase***

- Each node P_i evaluates actions of all nodes during consensus processes, and will note some nodes as inactive or malicious.

– ***step* 2: *Node Request Phase***

- If node P_i considers node P_s as malicious or inactive, it will send message $ClearReq$ to D_{trust}.

$$ClearReq = ((Clear, ID_s, ID_i), SigGen(h(m), SIG_{sk_i})) \tag{6}$$

$Clear$ is a message type. ID_s is the identity of the node to be cleared. ID_i is the identity of P_i. $SigGen(h(m), SIG_{sk_i})$ is signature of P_i on $h(m)$.

– ***step* 3: *Dealer Clear Phase***

- If D_{trust} has received $ClearReq$ about ID_s from more than t different nodes, it will broadcast $ClearAcc$ to the whole network and add P_s into its blacklist.

$$ClearAcc = ((Clr, ID_s), SigGen(h(m), SIG_{sk_{D_{trust}}})) \tag{7}$$

Clr is a message type. $SigGen(h(m), SIG_{sk_{D_{trust}}})$ is signature of D_{trust} on $h(m)$.

– ***step* 4: *Node Clear Phase***

- After receiving *ClearAcc* about ID_s from D_{trust}, node P_i will immediately clear P_s from its view of the network, and add it into its local blacklist.

6 Security and Performance

In this section, we will analyze several properties of the new scheme, including dynamic property, censorship resilience, agreement, and total order. We also show the performance and further consideration of DEBFT protocol.

6.1 Dynamic Property

Theorem 1. *In an already working system, requests from nodes to join or leave the consensus group will finally take effect if and only if they are valid.*

Proof. When a new node wants to join the system, the dealer will verify its request and sign if it is valid. Since the dealer is assumed to be reliable, nodes can trust the messages sent by the dealer with valid signatures. Next current consensus nodes will send $NodeJoinAcc$ messages representing their acceptance. Later when every honest node receives $NodeJoinAcc$ from $2t+1$ nodes it can ensure at least $t+1$ honest nodes has accepted this new node. This stands for the acceptance of the consensus group and every honest node will finally include the new node into the consensus group.

When a node currently in the consensus group wants to leave the system, it will broadcast *QuitReq* to all other nodes with its signature which cannot be fabricated. Other nodes will send *NodeQuitAcc* and finally receive at least $2t+1$ *NodeQuitAcc* from other nodes. This quorum contains at least $t+1$ honest nodes representing the acceptance of the consensus group and finally all honest nodes will exclude this node from the consensus group.

Since the dealer provides validation for new nodes, the sybil attack cannot take effect. And an honest node will only quit from the network by sending *QuitReq* by itself because its signature cannot be fabricated. In a word, only valid requests will finally take effect.

Theorem 2. *The system could clear malicious and inactive nodes timely and correctly.*

Proof. Through communication among nodes during the consensus process, malicious actions of faulty nodes could be detected by honest nodes, and inactive nodes could be detected through statistical results collected in recent rounds. All honest nodes will sponsor *Clear* protocol to exclude the malicious or inactive nodes from the system, and the dealer will receive enough signatures for a faulty node and process to clear it. On the other hand, the adversary could not attack an honest node to exclude it from the system, since the threshold is $t+1$ and there are at most t malicious nodes.

6.2 Normal Consensus Properties

Theorem 3. *The system satisfies censorship resilience property.*

Proof. Due to DTIBE scheme applied in the protocol, at any time a node specifies the decryption set and a threshold t according to its view of all nodes. DTIBE scheme ensures the input to RCS protocol is secure under the adaptive chosen ciphertext attack, which can avoid the adversary from early getting the content of proposals and deliberately delaying a specific request. DKG scheme is adopted to generate randomness in $\widetilde{BA}$ and it can effectively avoid any bias on randomness. These properties together guarantee that the final consensus result will not be manipulated by the adversary. Now let T be the size of requests previously input to any correct node before request m, since in every round, the number of committed requests has the same order with batch size B, then m will be committed within $O(T/B + \lambda)$ epochs except with negligible probability, where λ is a security parameter.

Theorem 4. *The system satisfies agreement property.*

Proof. For the property of DKG all honest nodes in the system will generate the same unbiased randomness, which is resistant to the attack of $t+1$ faulty nodes. Hence RCS protocol guarantees that in each round r a common subset will be output. If any honest nodes output request m, then m must be the plaintext of one component in the common subset of $RCS[r]$. According to the robustness of DTIBE scheme, every honest node will give their shares to decrypt the ciphertext to get request m, and m will finally be the output of all honest nodes.

Theorem 5. *The system satisfies the total order property.*

Proof. Firstly, since the system works in consecutive rounds, the consensus results of different rounds have an order. When considering the output requests in the same round, there exists an public ordering method(e.g., lexicographically), as a result all honest nodes will output requests in the same order.

The main consensus procedure of this new protocol inherits from HoneyBadger BFT, as a result DEBFT inherits the merits of its communication complexity which is $O(N)$, detail proof is shown in [16].

6.3 Further Consideration

As defined in Sect. 2, DEBFT protocol works under a partial synchronous network. Compared to the asynchronous network assumption, this limitation is due to the work environment of DKG scheme applied in $\widetilde{BA}$ [10]. It has been proved that the DKG scheme can safely work under a partially synchronous communication module. To extend DEBFT to asynchronous environment, we should find a way to generate randomness resistant to t malicious nodes under asynchronous assumption in a dynamic changing group. When still using DKG scheme, this can be solved by specifying the $QUAL$ set before the beginning of a round, and one solution is to generate $QUAL$ set from previous consensus result. This aspect could be further researched.

7 Conclusion

In this paper, we have proposed a practical dynamic enhanced BFT protocol. While reserving the merits of HoneyBadger BFT, through the combination of two appropriate cryptography components and rational protocol design, we realized dynamic joining and quitting functions for nodes and function to clear faulty nodes for the system. Analysis of several properties is given. It is believed that this new protocol has a wide range of application scenarios.

Acknowledgement. We would like to thank the anonymous reviewers for their helpful feedback. The authors are supported by the National Natural Science Foundation of China (Grant No. 61572318, 61932014, 61672347, 61672339), and the Shanghai Science and Technology Innovation Fund (Grant No. 19511101400).

References

1. Androulaki, E., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, p. 30. ACM (2018)
2. Baek, J., Zheng, Y.: Simple and efficient threshold cryptosystem from the gap Diffie-Hellman group. In: 2003 Proceedings of the Global Telecommunications Conference, GLOBECOM 2003, San Francisco, CA, USA, 1–5 December 2003, pp. 1491–1495 (2003). https://doi.org/10.1109/GLOCOM.2003.1258486
3. Baird, L.: The swirlds hashgraph consensus algorithm: fair, fast, byzantine fault tolerance. Swirlds Technical report SWIRLDS-TR-2016-01 (2016)
4. Bessani, A.N., Sousa, J., Alchieri, E.A.P.: State machine replication for the masses with BFT-SMART. In: 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, 23–26 June 2014, pp. 355–362 (2014). https://doi.org/10.1109/DSN.2014.43
5. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36288-6_3
6. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. J. Cryptol. **21**(2), 149–177 (2008). https://doi.org/10.1007/s00145-007-9005-7
7. Bracha, G.: Asynchronous Byzantine agreement protocols. Inf. Comput. **75**(2), 130–143 (1987)
8. Castro, M., Liskov, B., et al.: Practical Byzantine fault tolerance. In: OSDI 1999, pp. 173–186 (1999)
9. Duan, S., Reiter, M.K., Zhang, H.: BEAT: asynchronous BFT made practical. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 2028–2041. ACM (2018)
10. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptol. **20**(1), 51–83 (2007). https://doi.org/10.1007/s00145-006-0347-3

11. Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing bitcoin security and performance with strong consistency via collective signing. In: 25th USENIX Security Symposium, USENIX Security 2016, Austin, TX, USA, 10–12 August 2016, pp. 279–296 (2016). https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kogias
12. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.L.: Zyzzyva: speculative byzantine fault tolerance. ACM Trans. Comput. Syst. **27**(4), 7:1–7:39 (2009). https://doi.org/10.1145/1658357.1658358
13. Kwon, J.: Tendermint: consensus without mining (2014). http://tendermint.com/docs/tendermint.pdf
14. Kwon, J., Buchman, E.: Cosmos: a network of distributed ledgers (2016). https://cosmos.network/whitepaper
15. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharding protocol for open blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 17–30. ACM (2016)
16. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of BFT protocols. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016, pp. 31–42 (2016). https://doi.org/10.1145/2976749.2978399
17. Mostéfaoui, A., Hamouma, M., Raynal, M.: Signature-free asynchronous byzantine consensus with $t < n/3$ and $o(n^2)$ messages. In: ACM Symposium on Principles of Distributed Computing, PODC 2014, Paris, France, 15–18 July 2014, pp. 2–9 (2014). https://doi.org/10.1145/2611462.2611468
18. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). www.bitcoin.org
19. Sousa, J., Bessani, A.: Separating the WHEAT from the chaff: an empirical design for geo-replicated state machines. In: 34th IEEE Symposium on Reliable Distributed Systems, SRDS 2015, Montreal, QC, Canada, 28 September–1 October 2015, pp. 146–155 (2015). https://doi.org/10.1109/SRDS.2015.40
20. Susilo, W., Guo, F., Mu, Y.: Efficient dynamic threshold identity-based encryption with constant-size ciphertext. Theor. Comput. Sci. **609**, 49–59 (2016). https://doi.org/10.1016/j.tcs.2015.09.006
21. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, vol. 151, pp. 1–32 (2014)
22. Zamani, M., Movahedi, M., Raykova, M.: RapidChain: scaling blockchain via full sharding. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, 15–19 October 2018, pp. 931–948 (2018). https://doi.org/10.1145/3243734.3243853