SIGHTSTEEPLE:
Agreeing to Disagree with Functional Blockchain Consensus

Aditya Ahuja
Indian Institute of Technology Delhi
New Delhi, India
aditya.ahuja@iitdalumni.com

Abstract—Classical and contemporary distributed consensus protocols, may they be for binary agreement, state machine replication, or blockchain consensus, require all protocol participants in a peer-to-peer system to output (agree on) exactly the same information as part of the consensus payload. Although this model of consensus is extensively studied, and is useful for most consensus based decentralized applications, it falls short of defining correct distributed systems which mandate participant credential based privileged visibility into the consensus payload, through the consensus protocol itself.

We introduce a new paradigm for permissioned blockchain consensus, called functional blockchain consensus. Functional blockchain consensus allows each blockchain protocol participant to output some distinct sub-information of the list of transactions, as a function of the credentials of the participant in the blockchain system, instead of outputting the entire list of transactions. We motivate a novel rational-fault adversary to compromise functional blockchain consensus, and present a blockchain protocol called SIGHTSTEEPLE, that achieves functional blockchain consensus in the said fault model. SIGHTSTEEPLE relies on a novel combination of standard blockchain consensus and functional encryption, among other primitives, to achieve its goal of correctness. Further, we outline the realization of novel decentralized applications through functional blockchain consensus.

I. INTRODUCTION

Distributed consensus, which can manifest in the form of binary agreement [1], [2], state machine replication [3], [4], or blockchain consensus [5], [6], [7], requires a set of networked processes to agree on some information. In each manifestation, the notion of consensus is to agree on an identical snapshot of the information as part of the consensus payload, symmetrically, by each of the processes involved. Although this notion of consensus may be useful for symmetric information based decentralized applications (dApps), it precludes decentralized applications requiring consensus on sensitive information, where there is a need for privileged visibility into the consensus payload for each of the participant processes.

From a pedagogical perspective, there is a lack of consensus paradigms and protocols where visibility into the consensus payload is predicated on the credentials of the consensus protocol participants. Presently, distributed consensus is in general defined for a peer-to-peer system, and to intentionally preclude the credentials that the consensus protocol participants may possess: those credentials, which may define the privilege of their visibility into the consensus payload. Consequently, as at least an academic exercise, there is a need for defining paradigms for asymmetric consensus: the consensus protocol participants may agree on some sub-information, which is any information that may be inferred from the complete consensus payload, as a function of their credentials in the distributed system, once those credentials are established and agreed to in a decentralized setting.

One way to achieve asymmetric consensus is to ensure that the information contained in the consensus payload that is being considered by all processes is identical, however the agreed view1 or summary of the payload, and the consequential distributed ledger, is allowed to be different for different processes, as long as there exists a hierarchy of inference across the views of each of the processes. The hierarchy of inference should necessitate that some views are implied by other views, and some information is concealed across views, thereby ensuring an asymmetric consistency across all processes. Such credential based consensus definitions and protocols for secure consensus payload views for each of the involved processes (similar to secure information flow [8]), resulting in continuously growing logs which are the output of the consensus protocol, do not exist yet to the best of our knowledge.

There are also practical motivations for asymmetric consensus based decentralized applications. For instance, cryptocurrencies [9] (with permissioned equivalents such as Ripple [10]) with sensitive transactions may require asymmetric distributed ledgers, which allow different processes to see different summaries of the list of transactions, or allow processes to learn the list of transactions only when certain preconditions are met. Furthermore, existing privacy preserving permissionless cryptocurrencies, either achieve regulation post consensus, once the transaction is on-chain (for instance in ZCash [11]), or are extremely difficult to regulate (in the case of Monero [12]). Permissioned cryptocurrencies like Ripple do not support regulatory compliance. Thus, there is a case for achieving regulatory compliance as part of blockchain consensus, and for

1We use ‘view’ to denote any sub-information that can be implied by the complete information contained in the consensus payload, and will formally define a view later.
defining a consensus paradigm in that direction.

Apart from cryptocurrencies, there has been an explosion of blockchain-based decentralized applications in recent times [13]. As such, there is a motivation for blockchain-based (privacy-preserving) information flow hierarchies in decentralized applications and organizations as a whole, perhaps through separate yet hierarchical blockchains across the blockchain protocol participants. Consequently, it is befitting and opportune to consider, both as an academic exercise and a practical curiosity, asymmetric blockchain consensus models and protocols, for defining hierarchical blockchains: models that generalize standard blockchains by accommodating credential-based asymmetric agreement on the list of transactions.

**Our Contributions**

In this paper, we make the following contributions.

*Introducing Functional Blockchain Consensus (Section II).* We present a player model for permissioned blockchain consensus, where blockchain protocol participants (or players) have different credentials towards their visibility into the blockchain payload. We formally define a block payload view, which is any information that can be inferred from the complete list of transactions. We then introduce our new paradigm of consensus, called *functional blockchain consensus*, which, given the credentials of all players in the blockchain system, allows (i) each honest player to output a distinct block payload view, as a function of its credentials in the system, and (ii) allows each honest player to know that its honest counterparts output a correct block payload view. Functional blockchain consensus may result in different blockchains for different players (with some blockchains being implied by other blockchains), and so we formally show that functional blockchain consensus is a generalization of traditional blockchain consensus.

*Presenting SIGHTSPEEL under an economically incentivized, payload view compromise adversary (Section IV).* After presenting some preliminary constructions (Section III), we motivate a new adversary model under functional blockchain consensus, termed a *rational* adversary. A rational adversary, apart from maximizing its revenue through the consensus protocol (which may include any combination of block rewards, transaction fees, or invalid double spending transactions), would simultaneously want to maximize its block payload view and try to learn the complete list of transactions instead of some summary of it. To that end, the adversary would be willing to mislead the honest players towards learning incorrect payload views. Under a rational adversary controlling less than one-third of the players in the system, over a partially synchronous network, we present our protocol called SIGHTSPEEL. SIGHTSPEEL is constructed by amending the Byzantine-fault tolerant version of STREAMLET [7], and by using verifiable functional encryption schemes [15].

**Our goals, and open problems:** In this work, we intend to initiate the study of hierarchical visibility into the blockchain payload, through a new functional blockchain consensus protocol. We discuss the impossibility of Byzantine-fault tolerant SIGHTSPEEL (Section IV-A). We will not give exact construction of any functional encryption scheme, but point out their existence and viability for various privacy-preserving consensus based regulated decentralized application use cases (Section V-B).

**Related Work**

*Asymmetric trust, and relaxing consensus.* There have been proposals to model asymmetric Byzantine quorum systems over an asynchronous network, where each consensus protocol participant is free to choose which participants it considers faulty, and which it considers honest (non-faulty) [16], and consequential consensus protocols have been proposed [17]. There have been proposals to relax the definition of consensus (more specifically, relaxing the definition of termination within consensus) in blockchains, over an asynchronous network [18]. None of these contributions permit an asymmetric visibility of the consensus payload, nor advocate for asymmetry on the agreed information for the participants in the protocol.

*Hybrid blockchains.* Hybrid blockchains, which have a public chain and multiple private subchains to realize the decentralized application [19], [20], are different from SIGHTSPEEL where blockchain payload visibility can change for each player on the same chain.

*Solutions at the intersection of blockchains and functional encryption.* There have been proposals to outsource decryption under a functional scheme, with incentivization, to blockchains [21]. Privacy-preserving energy trading in blockchain empowered smart grids has been proposed by leveraging functional encryption [22]. Secure distributed smart meters have been defined using a combination of blockchains and functional encryption [23]. A power efficient elliptic curve pairing crypto-processor has been proposed for blockchains and functional encryption [24]. None of these contributions define a consensus model that can be realized using a combination of standard blockchains and functional encryption, which is central to our contribution.

II. FUNCTIONAL BLOCKCHAIN CONSENSUS

In this section, we introduce functional blockchain consensus.

A. The Player Model

We refer to the blockchain protocol participants, which are (polynomial-time) interactive Turing machines, as *players*. The set of players is given by \( \{ n \} := \{ 1, 2, ..., n \} \), where some players are honest (non-faulty) and others are faulty. Further, each player \( i \in \{ n \} \) has some credentials \( \kappa_i \in \{ 0, 1 \}^* \), with the unique highest credential denoted by \( \kappa^* \). Let \( \mathcal{C} = (\kappa_i)_{i \in \{ n \}} \) denote the list of credentials for all players. Further, there exists a third party for trusted setup, called *init-party*, that does

\begin{itemize}
  \item Our contributions are inspired from and are a refinement to a patent on functional blockchain consensus [14], with appropriate permissions from the invention owners.
\end{itemize}
not participate in consensus, but distributes the credentials to each player.

B. Block Payload View

We first introduce a block payload view, which has a special connotation in functional blockchain consensus (not to be confused with view change in state machine replication, or a real-time snapshot of the blockchain state in standard blockchains [7]). A block payload view for a specific player in functional blockchain consensus, is the sub-information of the list of transactions that the said player outputs, and includes in its blockchain. We formalize this through the following definition.

**Definition 1 (Block Payload View).** A set of functions \( F \) is a set of block payload view functions iff \( \forall txs \in \{0,1\}^*, \forall f \in F, f(txs) \) is implied by \( txs \). Further, there exists an identity function \( f^* \in F \), such that \( \forall txs \in \{0,1\}^*, f^*(txs) = txs \), and a null function \( f_\bot \in F \), such that \( \forall txs \in \{0,1\}^*, f_\bot(txs) = \bot \). Further, \( \forall txs \in \{0,1\}^*, \forall f \in F \), we call \( f(txs) \) a block payload view of \( txs \) under view function \( f \).

Examples of block payload views. Instances of block payload views include view functions that provide the smallest transaction in the list of transactions, or provide the sub-list of the transactions by a specifically designated transacting party, or provide the sum of the tokens exchanged in all the transactions in the transaction list.

Mapping players’ credentials to their permissible payload view. Given a player with certain credentials, there needs to be a correspondence between the player’s credentials and the view function (s)he is eligible for. Let \( \Psi : \{0,1\}^* \rightarrow F \) be the function, determined by the init-party, that provides this mapping. Also, it is true that \( \Psi(c^*) = f^* \).

C. Defining Functional Blockchain Consensus

Having presented the player model and introduced block payload views, we now formally define functional blockchain consensus.

**Definition 2 (Functional Blockchain Consensus).** Assume there exist \( n \) players with credentials \( C \), and each player is eligible to learn a block payload view set \( F \), through \( \Psi \). A blockchain protocol achieves functional blockchain consensus, if it attains the following consensus goals (with all but negligible probability in the security parameter), for each epoch \( e \) of the blockchain system when the block payload \( txs^e \) is added consistently to the blockchain:

1. Functional Hierarchy Consistency: For each honest player \( i \in [n] \), player \( i \) outputs \( (\Psi(c_i) = f_i \in F)_{i \in [n]} \).
2. Block Payload View Integrity: For each honest player \( i \in [n] \), player \( i \) outputs \( f_i(txs^e) \), and \( i \) knows that each honest player \( j \in [n], j \neq i \) outputs \( f_j(txs^e) \). Further, if for some honest player \( i \in [n] \), \( f_i(txs^e) = f^*(txs^e) = txs^e \), then \( i \) verifies that \( txs^e \) is valid (according to the rules of the dApp).
3. Liveness: If some honest player with highest credentials receives a valid block payload \( txs \) in some round, that payload will eventually be summarized and finalized in each honest player’s blockchain.

It is instructive to give an explanation of Definition 2. In the first requirement for achieving functional blockchain consensus, each honest player must output that each player in the system is eligible for a block payload view congruent to its credential in the system. In the second requirement, it is ensured that each honest player knows that each honest player did indeed learn a block payload view in accordance with its view function. In the final requirement, it is just ascertained that every valid block payload eventually goes on-chain. Kindly note that in the most general case, the credentials of each player can be a function of time (which means that the correct payload view function of the players can be a function of time).

D. Hierarchical Player Blockchains

We introduce some terminology first. We say a payload view is notarized\(^3\) (similar terminology in STREAMLET), once it receives a threshold of votes from some of the players and is eligible to be eventually confirmed in the player’s blockchain. We say that a notarized payload view is finalized once is is confirmed as a part of the player’s blockchain.

For each player \( i \in [n] \), and an arbitrary epoch \( e \), the player’s blockchain under functional blockchain consensus, is given by \( \text{chain}^i_{t, e} := (\text{chain}^i_{t,e-1}, H^*(f_i^e(txs^e)), f_i^e(txs^e)) \), with \( e' < e \), notarized \( f_i^e(txs^e) \) linked to notarized \( f_i^{e'}(txs^{e'}) \), and \( \text{chain}^i_0 \) is the genesis block. The standard blockchain, which is ideal (corresponding to the payload view function \( f^* \)), is given by \( \text{chain}^{*, e} := (\text{chain}^{*, e-1}, H^*(txs^{e'}), txs^e) \), similarly. Note that each player’s notarized blockchain might be a block-tree in general, with the finalized blockchain being a sub-chain of the notarized block-tree. We will denote each player \( i \)’s finalized blockchain by \( \text{chain}^i \), and the ideal finalized blockchain by \( \text{chain}^{*} \) (dropping the epoch superscript).

**View Functions’ Hierarchy.** We first define the binary relation \( \preceq \) over the set of credentials. \( \forall i_1, i_2 \in [n], \kappa_{i_1} \preceq \kappa_{i_2} \) implies that player \( i_2 \) has no lesser credentials than player \( i_1 \), and consequently for each epoch \( e \), payload view \( txs^e_{i_1} = f_{i_1}(txs^e) \) should be implied by payload view \( txs^e_{i_2} = f_{i_2}(txs^e) \). This is denoted equivalently with \( f_{i_1} \preceq f_{i_2} \), or even \( \text{chain}^i_{t, e} \preceq \text{chain}^i_{t, e'} \). From Definition 1, it is evident that \( \forall f \in F, f_{\bot} \preceq f \preceq f^* \).

It is easy to see that \( (F, \preceq) \) is a partial order, as the binary relation \( \preceq \) over \( F \) is reflexive, anti-symmetric and transitive. \( \forall f_1, f_2 \in F, \exists \text{dist}_{\preceq}(f_1, f_2) \) to be the number of functions on the path between \( f_1 \) and \( f_2 \) in the partial order \( (F, \preceq) \). From Definition 1, it is evident that \( \forall f_1, f_2 \in F, \text{dist}_{\preceq}(f_1, f_2) \leq \text{dist}_{\preceq}(f_{\bot}, f^*) \).

An example of player blockchains under functional blockchain consensus (Fig. 1). We give a toy example of

---

\(^3\) An equivalent notion of a notarized block, is a mined block in Nakamoto consensus blockchains [9].
Hierarchical player blockchains resulting from functional blockchain consensus, in a $n = 4$ player network, in Figure 1. For each player $i \in \{1, 2, 3, 4\}$, let $\text{txs}_i := f_i(\text{txs})$, for some $f_i \in \mathcal{F}$. Per epoch of the blockchain, the first player learns the 10 highest transactions in value, the second player learns the highest transaction, the third player learns the sum of the transactions, and the fourth player learns the entire list of transactions. The ordering of each player blockchain under $\preceq$ is given in Fig. 1(c).

Hierarchical player blockchains generalize standard blockchains. $\forall i \in [n], \forall e$, it is the case that $f_i(e) = f^*$, then it is true that each honest player’s payload view is identical and contains all the transactions for each block in each epoch: $\forall e, i \in [n], \text{chain}_i = \text{chain}^*$. In this instance, each player’s blockchain under functional blockchain consensus is no different than a standard blockchain.

### III. Preliminaries

We first present the preliminary assumptions and constructions required by the SIGHTSTEPEL protocol.

#### A. The Execution Model

The Player Model. We assume that the players $[n]$ are ordered with non-increasing static credentials, by the init-party: $\forall i_1, i_2 \in [n], i_1 \leq i_2$, either $\kappa_{i_2} \leq \kappa_{i_1}$ or $\kappa_{i_1} = \kappa^*$ and $\kappa_{i_2}$ are unrelated. We denote the subset of players that can participate in block proposal$^4$ (outlined in Section III-C) by $[m]$, where $m \leq n$. $\forall i \in [m], \kappa_i = \kappa^*$, and $\forall j \in \{m + 1, m + 2, \ldots, n\}, \kappa_j < \kappa^*$ ($j$ has lower than highest credentials). We refer to all the players in $[m]$ as head players.

Credentials’ Initialization. We assume the trusted benevolent init-party is the permissioned system administrator, that initializes the system by distributing the correct credentials, does not participate in consensus, and cannot flag adversarial players. During setup, the init-party makes $\Psi$ public. Each player $i \in [n]$ only knows its $\kappa_i$ through the init-party, unless $\kappa_i = \kappa^*$, in which case $i$ knows $C$.

The Network Model. We assume that there exists a permissioned, authenticated blockchain network of $n$ players. We assume, as in STREAMLET, that the clocks of all players are synchronized, and block proposal occurs in epochs. We assume that the network obeys partial synchrony $[25]$, where, there exists a known finite number of rounds $\Delta$, and an unknown Global Stabilization Time $GST$, such that for any message sent by any honest player at round $r_0$, the said message is received by all honest players in $[n]$ by round $\max(r_0, GST) + \Delta$.

We ignore the impact of computation times of cryptographic routines on our message delays (as in STREAMLET).

The Fault Model. We assume there exists an unknown, static partition of $[n]$, of honest and faulty players ($\mathcal{H}, A$). The honest players in $\mathcal{H}$ follow the protocol specification as is, and the faulty players in $A$ deviate from the specified protocol under a novel rational-fault: briefly, a rational adversary would try to maximize its revenue from participation in the consensus protocol, and simultaneously try to maximize its visibility in the blockchain payload.

We assume that given the static adversary, there is at least one head player that is not compromised by it: at least one

---

Fig. 1. An example of player blockchains (upto a height $h$), for 4 players, under functional blockchain consensus. (a) [Left] The original blockchain. (b) [Middle] The payload view per player, resulting in a separate blockchain for each player. (c) [Right] The hierarchy of inference across the players’ blockchains.
player in \([m]\) is honest, to eliminate the possibility of invalid transaction proposal by the adversary.

B. Streamlet: The Base Protocol

SIGHTSTEEPLE will be an amendment to the streamlined blockchain protocol Streamlet [7]. Streamlet will be considered over a partially synchronous network, with a Byzantine-fault\(^5\) adversary. For each block, consensus in Streamlet takes place in four stages: block proposal, block vote, block notarization (when the block receives a threshold of votes), and block finalization (when the block is confirmed). These four stages will be revised and re-interpreted in SIGHTSTEEPLE. For details on Streamlet, please see Appendix A-A.

C. Metablocks, Metachain and Player Blockchains

The Metablock. In SIGHTSTEEPLE, we introduce a ‘metablock’ as a super block containing encrypted information about the block payload (the list of transactions \(\times S\)). Each player can selectively read part of the information contained in the metablock, as per its privileges towards the block payload. Since only head players have the highest credentials in the SIGHTSTEEPLE system, metablocks can solely be proposed by them. We will denote, for each epoch \(e\), the metablock using \(M\). The Metachain. The ‘metachain’ would simply be the blockchain of metablocks. We would denote, for each epoch \(e\), the presently notarized metachain by \(\text{mchain}^e\) (which may be a tree of metablocks), and the final metachain at any epoch \(mchain\).

Player Blockchains are implied by the SIGHTSTEEPLE Metachain. Since each metablock in the metachain contains information that can be selectively inferred by each player, based on the encrypted information on the list of transactions as part of the metablock, each honest player \(i \in [n]\) can deduce \(\text{chain}^e_i\) from \(\text{mchain}^e\), for each epoch \(e\).

D. Basics of Functional Encryption

Functional encryption will be employed in SIGHTSTEEPLE to preferentially reveal information to each player as part of each metablock. Under a functional encryption scheme [26], given the encryption of a message \(\text{msg} \in \{0, 1\}^*\), the decryptor can recover \(f(\text{msg})\) if provided with the secret key \(sk_f\) under the scheme by the encryptor for a particular function \(f\). Under a verifiable functional encryption scheme [15], the decryptor can validate a verifiable functional encryption scheme [15], the decryptor wants to fool the decryptor by supplying a key \(sk\) scheme by the encryptor for a particular function\(^6\) the encryption of a message \(\text{msg} \in \{0, 1\}^*\), define signed message under scheme \(\Gamma_{\text{sig}}\) by player \(i\) as \(\text{msg}||\Gamma_{\text{sig},i}\) and encrypted message under scheme \(\Gamma_{E}\) for player \(i\) as \((\text{msg})||\Gamma_{E,i-1}\).

Byzantine-fault tolerant Streamlet will be denoted by \(\Pi_{\text{bft}}\). The rational-fault tolerant SIGHTSTEEPLE protocol will be denoted by \(\Pi_{\text{rft}}\). We will use \(M \leftrightarrow \text{msg}\) to append a message \(\text{msg} \in \{0, 1\}^*\) to metablock \(M\).

IV. SIGHTSTEEPLE UNDER RATIONAL FAULT TOLERANCE

We now present SIGHTSTEEPLE for functional blockchain consensus under a novel rational-fault model. Before that, we argue that Byzantine-fault tolerant SIGHTSTEEPLE is impossible.

A. Impossibility of (Secret Key based) BFT SIGHTSTEEPLE

Asymmetric block payload visibility based on encrypted on-chain information as part of the metablock, and a secret key per player, can never be Byzantine fault tolerant. This is because an adversarial player can just broadcast its secret key after the metablock finalization, thereby violating the payload view integrity on any lower credential honest player. Due to this payload view malleability post payload finalization, Byzantine-fault tolerant SIGHTSTEEPLE is impossible, as is formalized by the following attack.

ATTACK 1 (SIGHTSTEEPLE-BFT). Assume there exists a Byzantine player \(i' \in A\), and an honest player \(i \in \mathcal{H}\), with \(\kappa_i \geq \kappa_{i'}\) and \(\neg \kappa_i \leq \kappa_{i'}\). Assume at some epoch \(\bar{e} > e\), the metablock \(M\) is finalized, then player \(i'\) can violate the block payload view integrity of player \(i\) for epoch \(e\), by broadcasting \(\Gamma_{\text{rft}, sk_{i'}}\) over the network at epoch \(\bar{e}\).

Consequently, SIGHTSTEEPLE may only be proposed for a weaker adversary.

B. Rational-fault Adversary: Motivation and Definition

We consider rational players which wish to (i) maximize their revenue from the block payload, in terms of block reward (if the protocol is incentivized, as in Bitcoin [9]), transaction fees, and through invalid transactions (for instance double spending in a cryptocurrency); and (ii) maximize their payload view (under \(\preceq\)). Further, rational players may want to mislead honest players by supplying them a secret key (under the functional encryption scheme) for an incorrect view function, thereby forcing them to output an incorrect view of the payload, and violating the block payload view integrity for honest players, even when the metachain is consistent. An example to illustrate such an attack on head players in a cryptocurrency.

---

\(^5\)A Byzantine faulty player can deviate from the protocol arbitrarily with the sole purpose of compromising consensus.

\(^6\)Let \(\Gamma \in \{0, 1\}^{*}\) be a collision resistant hash function, which is ideal under the random oracle model (its image is uniformly distributed). Let \(\Gamma_{\text{sig}}\) denote a signature scheme, \(\Gamma_{E}\) denote a public key encryption scheme, and \(\Gamma_{\text{vfe}}\) denote a verifiable functional encryption scheme [15]. Given a message \(\text{msg} \in \{0, 1\}^*\), define signed message under scheme \(\Gamma_{\text{sig}}\) by player \(i\) as \((\text{msg})||\Gamma_{\text{sig},i}\) and encrypted message under scheme \(\Gamma_{E}\) for player \(i\) as \((\text{msg})||\Gamma_{E,i-1}\).
is given below. Consequence for honest head players under such an attack is that they cannot propose payloads after the attack (as payloads may not be verifiable), inducing an effective denial-of-service (different from conventional DoS attacks as in [28]). Thus it is imperative to design a protocol with verifiable view function keys for resilience to a rational adversary.

**ATTACK 2 (SightSteeple without Γ_{FE}).** Let \( f'(\text{txs}) := \text{txs} \) with reduced value of each tx by 1 unit. Consider, for some epoch \( e \), a rational leader \( L_e = i^* \in A \) supplies \( sk_f \) instead of \( sk_{f^*} \) to an honest \( i \in [m] \). Now, for the smallest \( e' > e \), with \( L_{e'} = i \), if \( i \) proposes a metablock containing payload \( \text{txs}^e \), the said metablock will not be voted up by any honest head player (due to the impression of proposal of invalid double spending transactions).

**Rational Players’ Utility Function.** We present the utility of the rational adversary \( A \), which is a function of the metablock proposed and notarized in the current epoch \( e \). Briefly, the utility function is a convex combination of the revenue \( \tau_A \) for the adversary resulting from the potential confirmation of the payload \( \text{txs}^e \) (which could be any combination of block reward, if the consensus protocol is incentivized, transaction fees, or transactions by the adversary in the payload), and the visibility into the payload given by the payload view function \( v_{FE}^{f_1} \) for each faulty player \( i^* \). We give the normalized utility function \( v_{FE}^{f_1} \) next, where \( \beta_1, \beta_2 \in (0, 1) \), with \( \beta_1 + \beta_2 = 1 \): \[
\begin{align*}
    v_{FE}^{f_1}(M^e) &= \beta_1 \cdot \tau_A(\text{txs}^e) + \beta_2 \cdot \sum_{i \in A} \frac{dist(i, f_1)}{\sum_{i \in A} dist(i, f_1) + 1}
\end{align*}
\] (1) We assume that rational players wish to maximize their utility under \( v_{FE}^{f_1} \) from participation in rational-fault tolerant SightSteeple, and so would choose metablock proposal strategies to that end.

### C. Metablock Structure

**The genesis block.** The players in \([n]\) initialize the system by agreeing on the genesis block \( \text{gen} := (0, [n], \mathcal{C}, \mathcal{F}, \Psi, \Gamma, \Gamma_{FE}, H^*) \). The genesis block is notarized when at least \( \frac{2m}{3} \) players vote on it (a vote by a player is just a signed hash of the genesis block by that player). We will modify the vote and notarization rule for the metablock.

**The metablock (by honest leaders).** The metablock for SightSteeple by honest leaders is presented next. The metablock contains the current epoch number \( e \), hash of the previous metablock \( M^{e'} \) to which the current metablock is linked, public parameters \( pp^e \) under the scheme \( \Gamma_{FE} \), encryption of the list of transactions \( \text{txs}^e \) under \( \Gamma_{FE} \), and, for each player \( i \), hash of the current player chain \( \text{chain}_i^{e-1} \), payload view function \( f_i^e \) for \( i \), and the encryption of the secret key \( sk_{f_i^e} \) under \( \Gamma_{FE} \), recoverable by \( i \).

**The metablock (by adversarial leaders).** The metablock for SightSteeple by rational leaders is also presented next. The metablock is the same as that from the honest leaders, except that for each \( i \in A \), the secret key \( sk_{f_i^e} \) under \( \Gamma_{FE} \) is replaced by \( sk_{f_i^e} \).

### SS-RFT Metablock:

**The Contents of \( \mathcal{M} \) by Leaders in \( \mathcal{H} \)**

- Initialize \( \mathcal{M}_H^e := \phi \)
- \( \mathcal{M}_H^e \leftarrow (e, H^*(M^e), \Gamma_{FE}, pp^e, \Gamma_{FE}, \text{Enc}_{pp^e}(\text{txs}^e))_{\Gamma_{syg}, L_e} \)
- \( \forall i \in [n]: \)
  - \( \mathcal{M}^e_{H} \leftarrow (i, H^*(\text{chain}_i^{e-1}), f_i^e, (\Gamma_{FE}, \text{sk}_{f_i^e})_{\Gamma_{syg, L_e}} \}

**The Contents of \( \mathcal{M} \) by Leaders in \( \mathcal{A} \)**

- Initialize \( \mathcal{M}_A^e := \phi \)
- \( \mathcal{M}_A^e \leftarrow (e, H^*(M^e), \Gamma_{FE}, pp^e, \Gamma_{FE}, \text{Enc}_{pp^e}(\text{txs}^e))_{\Gamma_{syg}, L_e} \)
- \( \forall i \in [n] \setminus \mathcal{A}: \)
  - \( \mathcal{M}^e_{\mathcal{A}} \leftarrow (i, H^*(\text{chain}_i^{e-1}), f_i^e, (\Gamma_{FE}, \text{sk}_{f_i^e})_{\Gamma_{syg, L_e}} \}

### D. The SightSteeple Protocol

The SightSteeple Protocol \( \Pi_{ss}^{\text{rft}} \) is presented in Algorithm 1. For this protocol, it is assumed that for the rational adversary \( \mathcal{A} \), \( |A| < \frac{2m}{3} \).

#### Algorithm 1: SightSteeple (\( \Pi_{ss}^{\text{rft}} \))

**Leader Election:**
\[
\forall e, L_e := H^*(e) \mod m
\]

**Metablock Proposal:**
If \( L_e \in \mathcal{H}, M^e = \mathcal{M}_H^e \). If \( L_e \in \mathcal{A}, M^e = \mathcal{M}_A^e \)

**Metablock Validation and Vote (first \( M^e \) from \( L_e \)):**
Each honest \( i \in [n] \) asserts \( f_i^e = \Psi(\kappa_i) \) and \( sk_{f_i^e} = \Gamma_{syg} \text{sk}_{f_i^e}(\kappa_i) \). Each honest \( i \in [m] \) also asserts \( \text{txs}^e \) is valid. If assertions succeed for \( i \), broadcast \( V_i^e = (i, e, H^*(M^e), yes)_{\Gamma_{syg, i}} \), otherwise broadcast \( V_i^e = (i, e, H^*(M^e), no)_{\Gamma_{syg, i}} \).

**Metablock Notarization:**
\( M^e \) is notarized when at least \( \frac{2m}{3} \) players vote ‘yes’, and no player votes ‘no’.

**Metablock Finalization (from Streamlet \( \Pi_{ss}^{\text{sh}} \)):**
If in any notarized metachain, there exist three hash-linked metablocks with consecutive epoch numbers, the prefix of the metachain up to the second of the three metablocks is considered final. Further, when a metablock is finalized, its parent chain is also finalized.

**Protocol Outline:** For each epoch, the metablock proposing leader is elected as a random member of \([m]\), as a function of \( e \). If the leader is honest, it proposes \( \mathcal{M}_H^e \) to the network. Otherwise, the rational leader proposes \( \mathcal{M}_A^e \). On receiving the first metablock from the leader, each honest player \( i \) in \([n]\) validates its contents to ensure that the secret key it received
is that for $\Psi(\kappa_i)$. The honest head players also validate that $txs^e$ is valid. Post validation, the honest players in $[n]$ reply by broadcasting their vote (denoted by $V^e_i, \forall i \in [n]$) to the network. Each vote is either a ‘yes’ vote if the validation succeeds, or a ‘no’ vote if the validation fails. The metablock is notarized once it achieves a ‘yes’ vote from at least all the honest players, and receives no ‘no’ votes. The metablock is finalized according to the finalization rule of the Byzantine-fault tolerant version of STREAMLET $\Pi^r_{bf}$ (Sec. 3 in [7]).

**Rational Player Voting Policy:** We now show that it is not necessary for rational players to vote in order to maximize their utility under $v^e_A$, for any epoch $e$.

It is in the interest of rational players that, for the maximization of the utility function $v^e_A$, $\forall e, M^e$ is notarized: if $M^e$ is not notarized, $v^e_A = 0$, but if $M^e$ is notarized, there is a possibility that $M^e$ would be finalized, and consequently $v^e_A > 0$ (since $\text{dist}(\langle f, \Gamma^e \rangle) > 0, \forall f \in A$). This implies that for metablocks $M^e_{H}$ and $M^e_{A}$, no rational player will ever vote no. Further, since honest players will always vote ‘yes’ for $M^e_{H}$ and $M^e_{A}$, consequently both these metablocks will be notarized, the rational players need not vote ‘yes’.

1) **Correctness:** We first show that the best metablock response by rational head players is $M^e_{A}$.

**Lemma 1 (Rational Leader Metablock).** Assuming that rational players wish to maximize their utility under $v^e_A$, the dominant strategy on metablock proposal for each rational head player $i' \in [m]$ is $\sigma^e_{M^e_{A}}$, for each epoch $e$ when $L_e = i'$.

**Proof.** The payoff for rational leaders as part of $v^e_A$ is on (i) the revenue from the block payload confirmation; and (ii) the visibility into the list of transactions. For (i), note that the rational leader may attempt to fork the metachain to orphan some metablocks, if it results in a higher revenue for it. The rational leader may also consider announcing two metablocks in quick succession for the same epoch in which it is a leader if it receives a second payload in the same epoch which has a higher revenue possible. For (ii), the rational leaders’ payoff is maximized when all faulty players learn $txs^e, \forall e$. This can only happen when each faulty player receives the secret key $\Gamma_{\text{VE, sk}}_{f_i}$ for each epoch in which a rational player is elected leader.

Finally, it is easy to see that $v^e_A = 0$ if the rational leader’s block is notarized, and $v^e_A > 0$ if the rational leader’s block is notarized (even if the payload related revenue is zero, the payload view payoff is positive). Consequently, both (i) and (ii) are achievable only when a rational leader’s metablock is notarized, which is only possible when each honest player $i$ receives $\Gamma_{\text{VE, sk}}_{f_i}$.

These arguments imply that the best choice of a metablock from rational leaders $i' \in [m]$ is $M^e_{A}$, denoted by the strategy $\sigma^e_{M^e_{A}}$.

We now show that the SIGHTSTEEPE protocol is correct.

**Theorem 2 (SS-RFT Correctness).** The SIGHTSTEEPE protocol $\Pi^r_{bf}$ achieves functional blockchain consensus, in the presence of a rational-fault adversary $A$, with $|A| < \frac{2}{3}$.

**Proof.** Since the notarization and finalization rules in $\Pi^r_{bf}$ are equivalent to those in $\Pi^r_{bf}$, the $\Pi^r_{bf}$ metachain will be consistent across all players (Theorem 3 in [7]). We will now show that $\Pi^r_{bf}$ achieves the three goals of functional blockchain consensus (Definition 2), considering a consistent metachain $M^e_{H}$ from an arbitrary epoch $e$, and remembering the metablock response from honest leaders is $M^e_{H}$ and from rational leaders is $M^e_{A}$ (Lemma 1):

(i) **Functional Hierarchy Consistency:** Since all honest players vote on the genesis block which contains $\langle \{n\}, C, F, \Psi \rangle$, and vote ‘yes’ on the metablock $M^e$ which contains $\langle \{n\} \rangle$, it is implied that all honest players output $\langle \Psi(\kappa_i) = f^e_i \in B \rangle, i \in [n]$. 

(ii) **Block Payload View Integrity:** Since each honest player voted ‘yes’ on the metablock (which is one of $M^e_{H}$ or $M^e_{A}$), and no player voted ‘no’, it is implied that the verification of $f^e_i$ under $\Gamma_{\text{VE}}$ succeeds for each honest player $i \in [n]$, and so it true that each honest player knows that each honest player $i \in [n]$ outputs $f^e_i(txs^e)$. Further, since each honest head player voted ‘yes’, it is true that $txs^e$ is valid.

(iii) **Liveness:** The $\Pi^r_{bf}$ metablock finalization rule is identical to the $\Pi^r_{bf}$ block finalization rule. Thus, the liveness of $\Pi^r_{bf}$ is implied by Theorem 6 in [7].

The $\Pi^r_{bf}$ metachain implies each player chain: Consider, for any epoch $e$, the metachain $\text{mchain}^e$ and the most recent metablock $M^e_{\text{H}}$ in it. Also consider, for each honest player $i \in [n]$, the sub-metachain $M^e_{i}$ of $M^e_{\text{H}}$. $M^e_{i}$ contains:

1. $(e, H^*(M^e_{i}), \Gamma_{\text{VE, PP}}$, $\Gamma_{\text{VE, Enc}}(txs^e)), \Pi_{gs, L_e}$
2. $(i, H^*(\text{chain}^e_{i-1}), f^e_i, (\Gamma_{\text{VE, sk}}_{f^e_i})_{i \in [n]}), \Pi_{gs, L_e}$

From both these messages, it is easy for player $i$ to imply $\text{chain}^e_i = (\text{chain}^e_{i-1}, H^*(f^e_i(txs^e)), f^e_i(txs^e))$, by recovering the encrypted secret key $\Gamma_{\text{VE, sk}}_{f^e_i}$ under $\Gamma_{\text{VE}}$.

V. **DISCUSSION**

A. **SIGHTSTEEPE achieves Block Payload Privacy**

Function privacy [29] is not achieved in the present version of SIGHTSTEEPE, as the view functions are public in the metachain, in order to ensure functional hierarchy consistency. We now give a brief discussion on how for each epoch $e$ with payload $txs^e$, each player $i \in [n]$ learns nothing more about $txs^e$ than $f^e_i(txs^e)$ due to the security properties of the underlying functional encryption scheme. Broadly speaking, message privacy under a functional encryption scheme is achieved as long as the encryption of any two messages under the said scheme are computationally indistinguishable. In the SIGHTSTEEPE protocol, the adversary sees 1 payload and less than $\frac{2}{3}$ functions applied on the payload, in each epoch (which has a separate instantiation of the verifiable functional encryption scheme parameters). Thus SIGHTSTEEPE requires at least $\frac{2}{3}$-selective-function message privacy [29], which can
be achieved through the use of the scheme [30] as part of the $\Gamma_{\text{FIE}}$ scheme, as long as $n$ is a polynomial in the security parameter of the $\Gamma_{\text{FIE}}$ scheme (for rigorous arguments on the security of functional encryption schemes please see [29]).

### B. Functional Blockchain Consensus based dApps

We discuss possible applications distributed ledgers resulting from functional blockchain consensus for permissioned dApps. Consensus based Regulatory Compliance in Cryptocurrencies [9]. Permissioned cryptocurrencies with regulated transactions, based on sub-types of functional encryption, can be constructed using functional blockchain consensus. Such cryptocurrencies can be initialized by retaining the original consensus protocol participants as highest credential members of the network, and introducing regulators as lower credential members of the network. In this setting, a functional blockchain consensus protocol would ensure that the consensus would not be achieved unless regulatory compliance is met. The first sub-type of functional encryption we consider is attribute based encryption (ABE) [31], which allows recovery of the plaintext if the decryptor satisfies certain attributes. Using ABE, SIGHTSTEEPLE can be defined to allow players in specific federal jurisdictions only to learn the complete list of transactions. The next sub-type of functional encryption we consider is predicate encryption (PE) [26], which allows recovery of the plaintext if some predicate on the plaintext is true. SIGHTSTEEPLE can be defined with PE to allow a subset of players to learn the list of transactions only if a specific regulated transaction has a transaction in it. Finally, a functional encryption scheme with the inner-product functionality (IP) [32] can be used to learn the sum of a sub-sequence of the plaintext. SIGHTSTEEPLE with IP can be used to allow regulators to only learn the sum value of all crypto-tokens exchanged in the list of transactions.

**Other dApps** [13]. Functional blockchain consensus can facilitate the need for asymmetric records for privacy preserving agreement on classified information in governance [33], [34] (for instance on citizenship and voting records), healthcare [35], [36], [37] (on patient healthcare records), and decentralized IoT network management [13].

### C. Future Directions

**Performance Analysis.** We intend to profile the throughput of the SIGHTSTEEPLE protocol, as a function of $n$, while factoring in and highlighting the delay induced by the computation of the associated verifiable functional encryption constructions.

**Off-chain metablock creation for privacy preservation.** Presently, the block payload view decryption is a part of SIGHTSTEEPLE, for the validation of the metablock. In future, in order to ensure $\forall e$, each player $i \in [n]$ provably learns nothing more than $f^{\text{tx}}(\text{tx}^e)$, metablock proposal may be made an off-chain activity. Options to outsource metablock creation include payload view function key generation through decentralized blockchain-based multi-party computation [38], or through dynamic decentralized functional encryption [39], or through an alternate, oracle blockchain system [21].

**Functional blockchain consensus in alternate fault models.** SIGHTSTEEPLE has been constructed to be resilient to rational-faults. In future, (secret) key-less protocols for functional blockchain consensus may be proposed for tolerance to a combination of Byzantine and rational players in the presence of altruistic/honest players (the BAR model [40]), or functional blockchain consensus may be attained in the absence of honest players altogether (as in TENDERSTAKE [4]).

We so believe that through this contribution, SIGHTSTEEPLE would be a stepping stone towards defining new consensus paradigms and protocols for asymmetric agreement on privileged information.

### APPENDIX A

**BACKGROUND**

### A. STREAMLET: The Basics

STREAMLET [7] is a simple blockchain protocol where consensus evolves in four streamlined stages to achieve consistency and liveness: (i) a block is proposed by a random leader on the set of all players; (ii) the first correct block seen by honest players is voted on; (iii) a block is considered ‘notarized’ once at least $\frac{2}{3}$ players vote on it; and lastly (iv) if a player sees three adjacent notarized blocks, with consecutive epoch numbers, then the second of the three blocks, along with its parent chain, is finalized. For our contribution, we would only consider STREAMLET over a partially synchronous network [25], with a Byzantine-fault adversary of size $\frac{2}{3}$ in the network, denoted by $\Pi_{\text{BM}}^*$. For $\Pi_{\text{BM}}^*$, please see (Thm.3, [7]) for consistency, and (Thm.6, [7]) for liveness.

### B. Fundamentals of Functional Encryption

Functional encryption allows the decryptor to recover any function of the message from the encryption of the message, instead of allowing the decryptor to recover the entire message from its encryption. A functional encryption scheme [26], given a set of functions $\mathbf{F}$ over some message space $M$, is a tuple of four probabilistic polynomial time algorithms $(\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$ where, $\forall m \in M$:

- $(pp, msk) \leftarrow \text{Setup}(\lambda)$
- $sk_f \leftarrow \text{KeyGen}(msk, f)$ for some $f \in \mathbf{F}$
- $ctx \leftarrow \text{Enc}_{pp}(m)$
- $f(m) \leftarrow \text{Dec}(sk_f, ctx)$

where the decryption succeeds with at least an overwhelming probability in the security parameter $\lambda$. The parameters $pp$ are public, whereas the key $msk$ to generate the function secret key(s) is private.

A verifiable functional encryption scheme $\Gamma_{\text{FIE}}$ [15] supports, in addition to the base algorithms $(\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$, two additional algorithms $(\text{VerCT}, \text{VerKey})$, such that

- $0/1 \leftarrow \text{VerCT}(pp, ctx)$ (output true iff the ciphertext $ctx$ was generated using the correct public parameters $pp$)
- $0/1 \leftarrow \text{VerKey}(pp, f, sk_f)$ (output true iff the secret function key $sk_f$ indeed corresponds to the function $f$).
