Anonymity in BFT protocols using zero knowledge proofs

Varun Madathil
North Carolina State University
vrmadath@ncsu.edu

Vaibhav Singh
North Carolina State University
vsingh7@ncsu.edu

ChuWen Chen
North Carolina State University
cchen31@ncsu.edu

ABSTRACT

Reaching consensus among an anonymous set of systems have a wide range of applications in the modern age. From online voting, to blockchains, anonymity and privacy are important properties that are required by distributed systems. However consensus with anonymity is not very trivial, nor is it efficient. We use zero-knowledge proofs with every message to anonymously reach consensus. zk-SNARKs are a succinct way to implement zero knowledge proofs. We assume one-third fraction of byzantine nodes in our system and these static byzantine nodes behave arbitrarily in the protocol, witout conforming to the rules of the protocol. Our main result is that even though we may achieve an anonymous consensus protocol using zero knowledge proofs, the overall protocol is not very efficient.

KEYWORDS

zero knowledge, consensus, distributed systems

1 INTRODUCTION

Consensus in the presence of Byzantine faults is one of the most fundamental problems in distributed systems. In this paper we try to reach consensus in the presence of Byzantine faults for an anonymous set of nodes. We accomplish this by using a cryptographic primitive called zero-knowledge proofs[18]. We then show how our protocol can be used in a real-world use case.

1.1 Consensus protocols

Reaching consensus in the presence of faults was first studied by Shostak, Pease and Lamport [27]. Byzantine faults occur when nodes choose to behave maliciously and send arbitrary messages instead of adhering to the protocol. The Byzantine nodes try and disrupt the protocol with the motivation of the honest nodes not reaching an agreement. Now the number of Byzantine faults allowed in a network depends on the assumptions made in the protocol. Work like PBFT [9], [14], [24] can tolerate upto 1/3 of the nodes to be faulty. Papers like [28] and [23] tolerate f byzantine nodes when the total number of nodes is $5f + 1$. There are other protocols [25], [21] and [1] that can tolerate upto 1/2 the nodes to be byzantine, but these schemes are too cryptographically heavy or are impractical to implement in the real world.

Unpublished, Feb 04, 2019, Raleigh, NC

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Moreover the number of steps that the protocol terminates also varies with these assumptions. Probabilistically terminating (or randomized) protocols are those that do not have a fixed number of steps for the parties to terminate, and deterministic protocols are those that are guaranteed to terminate in a fixed number of steps. In this work we use a probabilistically terminating protocol, with less than $\frac{1}{3}$ of the nodes being malicious. Moreover, the nodes authenticate themselves to each other, when sending messages. We assume digital signatures for this case, which we replace with zero knowledge proofs.

1.2 Anonymity

What does anonymity in a consensus protocol mean? In a consensus protocol, where the byzantine nodes can send arbitrary messages, the parties authenticate the messages received. This is done by having each party digitally sign the message it sends to the other parties. The parties then can verify that the message was not tampered with and originated from the sender. But, to verify the digital signature of the party, the receiver needs to know the public key or the identity of the party it receives the message from. In this work, when we anonymize the protocol, we are trying to hide this identity of the party, but at the same time have verifiability of the message sent. The receiver would still know the set of public keys in the system, but it cannot tell which message was received from which party.

1.3 Real-world use cases of anonymous consensus

A very natural use-case of our protocol is one where the parties would like to agree on a value but not reveal what they picked their value to be. An example may be that of government voting on a bill that has to be passed. The officials might not want to reveal their choice of vote and would like their vote to be private. There exists many online voting schemes in the literature [5], but we show how we can implement an efficient anonymous online voting scheme using zero knowledge proofs.

Another use case might be in the field of auctions where the nodes need to come to a consensus on the original price of items. In this problem, a byzantine node might try to give arbitrary prices to the item, in an anonymous way. We therefore would require an anonymous agreement among the honest parties over each item.

Last but not the least anonymous consensus protocols get a lot of traction in blockchain protocols. Note that in Proof-of-stake blockchains where the miner is selected depending on his stake, does not wish to reveal his stake, but still must prove that he is selected to propose the block. Another approach of implementing Proof-of-stake protocols is by selecting a committee in every round (depending on the stake of the parties) and have this committee agree on a block for the rest of the parties in the network (for e.g.

in [17]). Our anonymous BFT protocol can definitely be used in such blockchain protocols by the committee members to achieve consensus on the next block.

1.4 Contributions

- We construct an anonymous consensus protocol where parties receive messages from nodes without knowing their identity. We construct this protocol using a cryptographic primitive called zero-knowledge proofs.
- We implement a prototype of the anonymous consensus protocol in Java, and use the snarkJs library for implementing the zero knowledge proofs.

1.5 Organization

The rest of the paper is organized as follows. Section 2 describes the necessary background on zero knowledge proofs and consensus that is required. We then describe the construction of our protocol in Section 3. In Section 4, we prove the correctness and argue the security properties of our protocol. In section 5, we describe our experimental setup that we use to implement the anonymous consensus protocol, and finally in section 6, we present results that we analyze from the experiments. We mention the related work in section 7 and finally conclude with some future work and conclusions in Section 8 and Section 9 respectively.

2 BACKGROUND AND PRELIMINARIES

2.1 Consensus

Let n and t be positive integers, such that $n = 3t + 1$. A protocol $\mathcal{P}$ is a binary consensus protocol if for every execution e of $\mathcal{P}$ with n players, in which each player starts with an initial value $v_i \in \{0, 1\}$ and every honest player j halts with probability 1, outputting a value $out_j \in \{0, 1\}$ such that the following two preprties are satisfied:

(1) *Agreement*: $out_i = out_j$ for all honest players i and j.
(2) *Consistency*: if for some v, $v_i = v$ for all honest players i, then $out_j = v$ for all honest players j.

The protocol ends when each honest player halts. The output of the protocol is v if for some honest player i, the $out_i = v$.

In this work we do not explicitly use t to denote the number of malicious parties, we use the equivalent $\frac{n}{3}$.

2.2 Fault tolerance in consensus protocols

In consensus protocols there are generally three classes of faults that we consider:

- Fail-stop/crash faults: Nodes may fail by just stopping the protocol and not sending any messages.
- Byzantine faults: Nodes may fail without stopping. Generally the nodes show some kind of arbitrary erratic, unexpected behavior. These nodes may be malicious and disruptive.
- Unfaithful faults: Nodes consider self-interest and correspondingly send messages in the protocol.

In this work we consider Byzantine faults, where the parties send arbitrary messages. We consider a weaker model of Byzantine faults which we clearly explain in Section 3.1.

2.3 Network Models

Broadly there are two network models when considering consensus protocols

- Synchronous network: A synchronous network assumes that there is a known upper-bound on message delay. That is all messages must be delivered in some pre-determined amount of time, and all participants in the network know this time they need to wait to receive all messages.
- Asynchronous network: In the asynchronous setting, there is no assumption on the network delay and messages may be received after arbitrary amounts of time. Messages may not be received also in this setting.

There are also other network models like a partially synchronous and semi-synchronous setting that lies in between these two models. In this work we assume the synchronous setting, that is all messages are received within a delay.

2.4 Zero Knowledge proofs

A Zero-knowledge protocol [18] is a method by which one party (the prover) can prove to another party (the verifier) that they know a value x, without conveying any information apart from the fact that they know the value x. Zero-knowledge protocols are also useful to avoid metadata leaks from side channels, or when an encrypted communication channel is not guaranteed.

Zero-knowledge proofs can be interactive (that is, require interaction between the prover and receiver for multiple rounds throughout the protocol lifecycle), or non-interactive [7](a single message needs to be sent from the prover to the verifier). For the project, we will be looking at non-interactive zero knowledge proofs only.

zk-SNARKS[3] are succinct non-interactive arguments of knowledge. They are succinct because the zero knowledge proof can be verified in milli-seconds, with a proof length of only a few bytes even if the program size is large. A lot of research in the constructing efficient SNARKs has been done in [2], [16] and [20]

To create zk-SNARKs the first step is to create an arithmetic circuit that models the function that we are trying to compute. The circuit takes has input gates and output gates. The input to the function is the value that goes on the input gate, and the value on the output gate should be the output of the function. To get some non-determinism, the circuit takes an *input* and an auxiliary input which is called the *witness*. The circuits have only bilinear gates, therefore one has to model the circuits carefully. The zk-SNARK for a arithmetic circuit satisfiability is defined by a triple of polynomial-time algorithms (KeyGen, Prove, Verify):

- $\mathsf{KeyGen}(C) \rightarrow (\mathsf{pk}, \mathsf{vk})$: On input a circuit, the key generator algorithm KeyGen probabilistically samples a proving key pk and a verification key vk. These keys are published as public parameters and can be used multiple times.
- $\mathsf{Prove}(\mathsf{pk}, x, a) \rightarrow \pi$. On input a proving key pk and any (x, a) the prover Prove outputs a non-interactive proof π for the statement x
- $\mathsf{Verify}(\mathsf{vk}, x, \pi) \rightarrow b$. On input a verification key vk, an input x and a proof π, the verifier Verify outputs $b = 1$ if he is "convinced" that the statement is valid.

Informally, a zk-SNARK satisfies the following properties.

Completeness. An honest prover can convince a verifier if he actually knows a valid (x, a).

Succinctness. The size of the proof should be constant with respect to the security parameter.

Proof of knowledge. If the verifier accepts a proof output by a prover, then the prover "knows" a witness for the given instance.

Perfect Zero Knowledge. Informally this means that the verifier should gain no knowledge about the secret the prover knows except the validity of the witness.

2.5 Digital Signatures

Digital signatures are a cryptographic primitive that was put forward by Diffie and Hellman in 1976 [13]. Signatures are basically used for source authentication - that is a receiving party is ensured that the message came from the correct source, source non-repudiation - a party having signed a message cannot later claim that this message was signed by him, and integrity - a signature on a message when verified, implies that the message was not tampered with.

The standard security of the digital signature scheme is that a malicious party cannot forge a valid signature on a new message, even if it has access to a signing oracle, that can sign messages.

In this work we use the simple RSA signature scheme, that we describe below:

RSA Signature Scheme

KeyGen: (p, q)
Compute $n = p \cdot q$
Compute $\phi(n) = (p-1) \cdot (q-1)$
Choose an integer e such that $1 < e < \phi(n)$
Compute $d = e^{-1} \mod \phi(n)$
Signing Key: (d, n)
Verification Key: (e, n)
Sign(msg, d, n): Compute $\sigma = H(msg)^d \mod n$. Output (σ, msg)
Verify(σ, msg, e, n): Check $\sigma^e \mod n \stackrel{?}{=} H(msg)$

3 THE ANONYMOUS CONSENSUS PROTOCOL

In this section we describe our anonymous consensus protocol, that allows parties to come to consensus on a bit in the presence of malicious nodes. We first describe the adversarial model and the assumptions that we make.

3.1 Adversarial Model

We consider a weakened adversary, that can be classified as byzantine, but does not fully conform to the byzantine properties. More specifically, adversary's capabilities are the following:

- In every step of the protocol, a malicious node sends a random bit, that does not follow the specification of the protocol. If according to the protocol an honest node were to send the bit b, the malicious node completely disregards the computation of the bit b, and instead randomly computes $b^* \xleftarrow{r} \{0, 1\}$.
- The malicious node can send arbitrary bits to different parties, i.e., it may send b to a node A and $1 - b$ to node B.
- A malicious node will send a message in every step. It cannot choose to not send out a bit. This bit that the malicious node sends may or may not be consistent with the protocol.
- The malicious nodes **do not** collude. That is each malicious node sends a bit to other nodes, without taking into account what other malicious nodes send. This implies there is no single adversary that controls all malicious nodes. Each malicious node independently tries to disrupt the consensus protocol.
- A malicious node cannot change the message sent by an honest node.

3.2 Assumptions

We make certain assumptions that are required for the protocol:

- We assume that the network is fully connected and is synchronous. That is, messages sent out in a step is guaranteed to be delivered before the next step.
- The adversarial influence or the number of malicious nodes in the system is strictly less than $\frac{1}{3}$ of the total number of nodes in the network.

3.3 The protocol

Each party P_i is initialized with a bit b_i, and the end goal of the protocol is for all honest parties to come to consensus on a bit b.

Setup: We assume a trusted setup that generates a common reference string crs, that all parties can use to create their zero knowledge proofs. Each party also has a broadcast list, $\mathcal{L}$ which consists of all the parties who will receive the message that the party sends.
Key Generation: Each party creates a signing key, SIG.sk and a verification key SIG.vk. Using the crs as input, parties also generate a proving key and a verification key for the zero knowledge proofs.

The protocol is a three-step loop, where honest parties compute the bit for the next step depending on the bits they receive from the previous step. Along with the bit, the party also sends a zero knowledge proof that proves that the created bit is from the party. This is done by using the proving key and a "witness" that is basically the signature secret key and the bit b. We denote the proof created by party P_i as π_i.

The protocol also maintains a counter R, that specifies the number of times the loop was run. Each party updates the round number each time the the third step of the loop is complete.

Protocol AnonymousConsensus

Step 1

Party P_i sets toSend to its bit b_i creates a zero knowledge proof π_i. P_i then sends (toSend, π_i) to all parties in $\mathcal{L}$.

P_i waits for a time period λ, until it receives messages from all parties. For each message received, P_i using its verification key, verifies that the zero knowledge proof received with each message is valid. Out of the valid messages:

(1) If num(0) $\geq \frac{2}{3}n$, then P_i sets toSend $= 0*$, outputs $out_i = 0$ and $HALTS$.
(2) If num(1) $\geq \frac{2}{3}n$, then P_i sets toSend $= 1$

(3) Else set toSend = 0

P_i then computes a zero knowledge proof to show that proves that the message toSend is correctly signed by P_i.

Step 2

P_i sends (toSend, π_i) to all parties in $\mathcal{L}$.

P_i waits for a time period λ, until it receives messages from all parties. For each message received, P_i using its verification key, verifies that the zero knowledge proof received with each message is valid. Out of the valid messages:

(1) If num(1) $\geq \frac{2}{3}n$, then P_i sets toSend = 1∗, outputs $out_i = 1$ and *HALTS*.
(2) If num(0) $\geq \frac{2}{3}n$, then P_i sets toSend = 0
(3) Else set toSend = 1

P_i then computes a zero knowledge proof to show that proves that the message toSend is correctly signed by P_i.

Step 3

P_i sends (toSend, π_i) to all parties in $\mathcal{L}$.

P_i waits for a time period λ, until it receives messages from all parties. For each message received, P_i using its verification key, verifies that the zero knowledge proof received with each message is valid. Out of the valid messages:

(1) If num(0) $\geq \frac{2}{3}n$, then P_i sets toSend = 0.
(2) If num(1) $\geq \frac{2}{3}n$, then P_i sets toSend = 1.
(3) Else set toSend = $\mathsf{lsb}(\mathsf{min_j}(\mathsf{H}(\mathsf{SIG}_{\mathsf{sk_j}}(\mathsf{R}, \mathsf{r}))))$.

P_i then computes a zero knowledge proof to show that proves that the message toSend is correctly signed by P_i.

4 PROOFS

We prove the security and correctness of our protocol in two parts. We first show that the protocol will terminate, and that when the protocol terminates all the honest parties will have agreed on the same bit. We then show how the zero knowledge proof hides the identity of the party but at the same time convinces others that the messages received are from the correct party.

4.1 Proof of Correctness

We prove that the protocol is correct with each of the lemmas:

Lemma 4.1. *In any step of the protocol, no party can receive* $> \frac{2n}{3}$ *messages for both bit 0 and for bit 1.*

Proof. Assume a contradiction, that a party P_i receives $> \frac{2n}{3}$ for 0 and $> \frac{2n}{3}$ for 1. This implies P_i receives $\frac{4n}{3}$ messages,

By our model, each party sends only one message in each step. This implies the total number of messages a party receives is n.

Thus we have a contradiction since P_i is receiving $\frac{4n}{3} > n$ messages. Hence a party cannot receive $> \frac{2n}{3}$ messages for both bit 0 and for bit 1. □

Lemma 4.2. *If an honest party HALTS in a step of round R, then all honest parties will have halted by the end of round R+1.*

Proof. Assume that an honest party P_i has halted in Step 1 of a round R. This implies that P_i received greater than $\frac{2}{3}n$ of the messages for bit 0 with valid proofs.

We note that if P_i received $\frac{2}{3}n$ of the messages, then at least $\frac{n}{3}$ of those messages came from honest nodes. (According to our model $< \frac{n}{3}$ of the nodes can be malicious).

Moreover, according to our model, since all messages sent from the previous step is received in the next step, all other parties also receive at least $\frac{n}{3}$ honest messages for bit 0.

Now two cases can arise:

- **CASE 1**: num(0) for all the other parties is $> \frac{2n}{3}$
- **CASE 2**: num(0) for all the other parties is in the range $(\frac{n}{3}, \frac{2n}{3}]$.

If for other parties, we have CASE 1, then the parties will HALT, since it is in step 1. And therefore, those parties will have terminated in round R.

If for other parties we have CASE 2, then they all set their toSend bit to be 0, since none of these parties are going to see $\frac{2n}{3}$ messages for bit 1. Thus all honest parties continue on to the next step with an agreement on 0. They cannot terminate in step 2 or 3 of that round (R). But in step 1 of round $R + 1$, all honest parties send messages for bit 0, and thus all of them HALT.

Thus, if a party terminates in round R, all honest parties will terminate in round $R + 1$. □

Lemma 4.3. *The protocol* AnonymousConsensus *will terminate in an expected 9 steps.*

Proof. Assume that by Step 3 of round 1, no party has HALTED. This implies no party saw greater than $\frac{2n}{3}$ messages for a bit in step 1 and 2. Now in Step 3 if a single party sees $> \frac{2n}{3}$ messages for 0(resp. 1), then it terminates in Step 1(resp. 2) of the next round. Moreover from Lemma 4.2 all honest parties will terminate in the next round in the worst case.

Now assume that in step 3 no party received greater than $\frac{2n}{3}$ for either 0 or 1. Then each party computes sets their toSend bit to be the least significant bit of the smallest of the hashes of the signatures of all the parties. Now two cases arise :

- **CASE 1**: The smallest hash belongs to an honest party. If this is the case then all honest parties agree on the same bit and they are in agreement and the protocol terminates in the next round. This will occur with probability 2/3.
- **CASE 2**: The smallest hash belongs to a malicious party. If this is the case, then different parties can see different smallest hashes, since the malicious party may choose to send the signature on a bit 0 to some parties and a signature on bit 1 to other parties. This can occur with probability 1/3

Now for the protocol to terminate, we require that an honest party have the smallest hash value. This happens with probability 2/3. Now consider the case that in step 3 if there are some parties that did see greater than $\frac{2n}{3}$ messages for 0(or 1, but not both by Lemma 4.1).

With probability $\frac{1}{2}$ the least significant bit of the hash of the honest party is going to be 0 (resp. 1).

This implies with probability $\frac{2}{3} \times \frac{1}{2} = \frac{1}{3}$ parties will be in agreement at the end of step 3.

Now since each round is a three step loop, the expected number of step to terminate is $3 \times \frac{1}{1/3} = 9$. Thus in an expected 9 steps the protocol terminates.

□

From the above three lemmas we have proven that the protocol terminates in an expected 9 steps and that if even one honest party halts with a bit b then all honest parties will halt with the same bit b. This satisfies both conditions of a consensus protocol - termination and consistency.

4.2 Anonymity Proof

We use the EdDSA signature algorithm[6] that attests that each message sent is correct and from the sender. The edDSA protocol used is as follows.

EdDSA specified as ed25519:

KeyGen : $(\mathbb{E}, q, d, G, l, h; a, A)$
$\mathbb{E}$: an Edward elliptic curve $-x^2 + y^2 = 1 + dx^2y^2$ over $\mathbb{F}_q$
q : a prime $2^{255} - 19$
$d : -\frac{121665}{121666} \in \mathbb{F}_q$
G, l: a random basse point in $\mathbb{E}$ with prime order l
h : a hash, instanmtiated with SHA512
Signing key: $a \xleftarrow{\$} [1, l-1]$
Verification Key: $A = aB \in \mathbb{E}$

Sign$(a; m) : \langle R = rB, s = r + h \cdot h(A\|R\|m)\rangle \in \mathbb{E} \times \mathbb{Z}_l$
where $r \xleftarrow{\$} [1, l-1]$.

Verify$(A; R, s)$: $sB \stackrel{?}{=} R + h(A\|R\|m)A$

Parties prove in zero knowledge that they know the secret key a as shown above when they sign their bit b, in each step. We used an existing implementation of zero knowledge proofs for the EdDSA scheme from the circom library [19]. In this work we do not show the proof that the implementation and the scheme is zero knowledge since it is out of the scope of the work.

5 EXPERIMENTS

We implemented a Byzantine Agreement prototype based on [24] in Java, consisting of approximately 2,000 lines of code. We worked on an existing Gossip Protocol implementation present at [29], and added or modified 1,000 lines of code to the existing implementation. We evaluated signing messages using Zero Knowledge Protocols for which we used ZK-Snarks, a JavaScript library. We used ECDSA, the current state of the art message signing algorithm for Zero Knowledge message signing, and compared its performance with plain message signing using 4096-bit RSA. Plain message signing was implemented in Java using the Java Cryptographic Extensions (JCE) library.

Our implementation supports setting up random (configurable) seeds for each node, and random (configurable) list of honest and malicious parties. We ran the following experiments, first for four nodes and then for 10 nodes:
1.) Seeds all 0, no malicious users to verify consistency,
2.) Random seeds, one malicious user for both four-node and ten-node system,
3.) Random seeds, n/3 malicious users (where n is the total number of nodes)

In our implementation each user connects to all other peers, accepts incoming connections from other peers, and gossips messages to all of them. We currently provide each user with an "address book" file listing the IP address and port number for every user's public key. In the real world we imagine the nodes could gossip this information, signed by their respective private keys, and share the messages in a public bulletin board. Our implementation is susceptible to Sybil attack, in which a potential adversary could create multiple copies of itself and attach them to the network. This has not been addressed here, but might be addressed in a future work.

6 RESULTS

Our experiments quantitatively answers the following:

- What is the number of steps the protocol AnonymousConsensus takes to reach agreement and how does it scale as the number of nodes grows?
- How does the protocol perform when some of the nodes misbehave? And how does the protocol perform when the number of malicious nodes increases?
- How much time does it take to create and verify a zero knowledge proof, and how does it compare to digital signature creation and verification time?
- How many steps does the protocol take to terminate when the honest nodes start with arbitrary bits vs all honest nodes starting with the same bit.

To answer these questions, we deploy our prototype of the AnonymousConsensus protocol on the Google Cloud Platform. We set up ten Ubuntu 18.04 virtual machines (dual core, 8 GB memory) and broadcast the commands to simultaneously run the build in all machines.

6.1 Number of steps to complete the protocol

Figure 1 represents the relationship between the number of nodes (both honest and malicious) present in the system and the average number of steps seen in our experiments. The average number of steps remains constant at about 3 (with the maximum number of steps observed being 5). In our experiments we assumed that the number of malicious users was less than $\frac{1}{3}$ of the total number of users. All the experiments ended in either 1,2,4 or 5 steps. Even though the expected number of steps was 9, our protocol terminated in a max of 5 steps (when all parties agreed on 1). This is expected since the malicious nodes do not behave in an intelligent way and send only arbitrary messages. Moreover the fact that we experimented with only 10 nodes might have affected the number of steps to reach consensus. An interesting thing to note is that on increasing the node size from 4 to 10 the average number of steps did not change. Also note that different experiments with the same starting bits and same number of malicious nodes gave different results. This is due to the probabilistic terminating property of the protocol. Just based on how the malicious parties behaved, we have two different terminating steps.

6.2 Varying the initial distribution of bits

In Figure 2 we try to compare the number of steps the protocol took to terminate when the parties started with the same bits when they

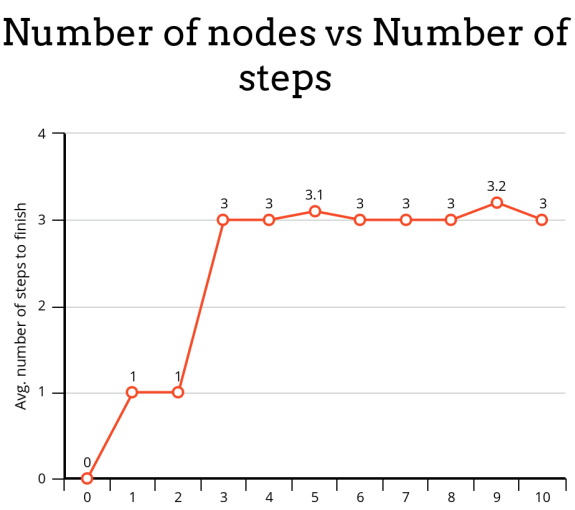

Figure 1: Comparing number of nodes with the number of steps, where the number of nodes increased from 0 to 10 and the number of malicious nodes were $< \frac{1}{3}$ of the total number of nodes. The values are an average and each experiment was done 10 times.

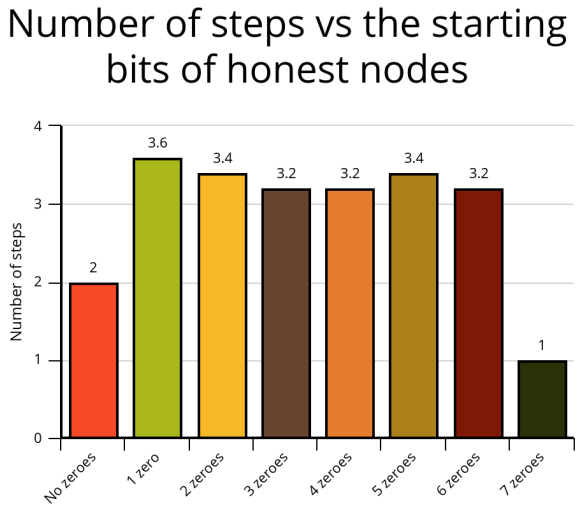

Figure 2: Analysis of number of steps to complete the protocol when the honest parties were already in agreement before the start of the protocol vs when the parties started with arbitrary bits. The experiments were done with 10 nodes, where 3 nodes were malicious

start with arbitrary bits. This is an interesting experiment since we are trying to test the consistency property of the Byzantine Agreement protocol. When parties start with the same bit, they terminate in the very first terminating step, which is step 1 if all parties agree on bit 0 and step 2 if all parties agree on bit 1. For bit 1, the terminating steps are $2, 5, \ldots$ When the malicious parties were arbitrarily sending messages, we saw cases of the protocol terminating in 5 steps and all parties agreed on 1. We again note that the overall average of all the steps comes to around 3.2 and this is expected as the malicious nodes are only sending arbitrary messages.

6.3 Varying the number of malicious nodes

In figure 3 we compare how the protocol behaves on increasing the number of malicious nodes. As expected the average number of steps increases with the number of malicious nodes, but overall they are very close to each other (approximately 3). We ran each experiment 10 times and show the average of the number of steps it takes to terminate.

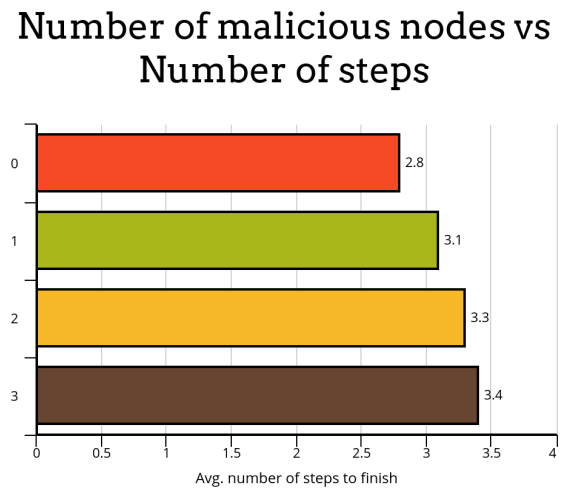

Figure 3: Comparing the number of steps taken to terminate the protocol when varying the number of malicious nodes from 0 to 3 when the total number of nodes is 10

6.4 Signatures and SNARKs

In figure 4 we compare the time taken for each comparable operation of SNARKs and signatures. The SNARK evaluated was for the EdDSA signature scheme which is supposed to be efficient in terms of size and verification time, and the signature scheme evaluated was the RSA signature scheme. Our protoype implements the RSA signature scheme. We compare the key generation: which is the proving and verification key in the case of SNARKs and signing and verification key in the case of signatures. We also compare the proof creation vs signature creation and the verification times for both primitives. As expected the signature scheme took very little time to finish all operation, and the SNARK operations took almost a 500× time to be evaluated. This is expected since SNARKs require some heavy cryptography.

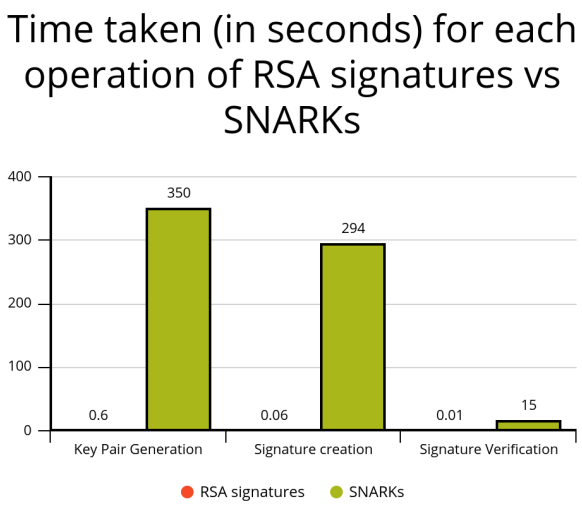

Figure 4: Comparing the time taken to create and verify SNARKs and signatures

In figure 5 we analyze the time taken for each operation for the SNARKs creation. A lot of time goes into the setup (approximately 11 minutes), which includes the circuit creation and key pair generation. Note that this is a one-time setup, where all parties can use the same keys over and over again. SNARKs are generally optimized for verification time, and takes only a few seconds. To create each proof, the parties need to create a witness and then create a proof by giving this witness as input to the circuit. This takes approximately 2 minutes to run.

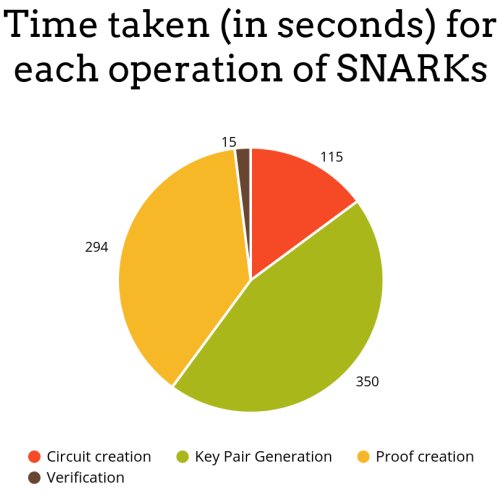

Figure 5: Evaluating operations of circuits that are required by each party in each step.

Finally in figure 6 we evaluate each of the operations for the signature scheme. Here note that unlike SNARKs we only have a key generation phase in the setup (as opposed to creating circuits). Again, this takes up most of the time in comparison with the signing and verification of signatures. But the overall time taken to do any of the operations is much less than time taken to do any operation with SNARKs.

Figure 6

7 RELATED WORK

Anonymous Agreement. Research in the field of anonymous agreement was done in [26]. The paper assumes that the network has a channel between every pair of processors, the processors have unique names that are globally known and for every received message the identifier of the sending processor is assumed to be known to the receiver. But these assumptions only imply that each processor has a mapping between the number of communication channels (links) and the identifiers of the processors on the other side of the channels. The Janus algorithm [8] also proposes an anonymous agreement technique for a distributed system that can tolerate only crash-faults. They assume a shared memory model where processes communicate with each other by writing and reading to atomic shared registers. We note that this protocol is only crash-tolerant and not byzantine-fault tolerant like our protocol.

Privacy-preserving Blockchains Another related area is that of privacy-preserving blockchains. Ouroboros Crypsinous [22] a privacy-preserving proof-of-stake consensus protocol ensures that the miner, who proposes the block, is anonymous. But all parties in the network can verify that the proposed block is correct, and that the miner had the authority to create the block. Crypsinous uses zero knowledge proofs and even have an implementation using zk-SNARKs to realize their zero knowledge proofs. Ganesh et al. [15] describe an anonymous verifiable random function that can be used to anonymously elect a leader or a committee that decides the block for the rest of the parties, in a proof-of-stake protocol.

Cryptographic techniques for anonymity There have been certain attempts to achieve anonymity other than zero knoweldge proofs or SNARKs. Special kind of signatures have been used to achieve this anonymity. Blind signatures [10], ring signatures [4], group signatures [12], and DC-nets [11] are some of the cryptographic primitives that can be considered to achieve anonymity.

8 LIMITATIONS AND FUTURE WORK

In this work we consider a very weak notion of Byzantine faults where the malicious nodes do not behave intelligently, but send random bits to different parties in each step. Ideally, the Byzantine node should decide on the bits to send to other parties depending on the bits received from them. The motivation for the adversarial nodes is to disrupt the protocol and keep the parties from not terminating as long as possible. The other motivation for adversarial nodes is to convince two parties to terminate with different values. In our modeling of the adversary we only considered the malicious nodes to send arbitrary messages and thus hope that it disrupts the protocol. This is definitely future work for us, where we try to model the adversary such that they can collude and together decide on what messages to send to each party depending on the bits received from the previous step.

Another limitation of the implementation is that we had to use very inefficient SNARKs that took way more time than was expected. We would like to implement the protocol with SNARKs that take less time, and try the protocol with other primitives where the zero knowledge proofs are (lighter) message authentication codes and not (heavier) digital signatures.

9 CONCLUSIONS

In this work, we construct an anonymous consensus protocol. We use zero knowledge proofs to hide the identity of the parties, but at the same time have verifiable messages. We assume a weakened adversarial model where nodes may be Byzantine but act independently. We implement the protocol using a gossip protocol and the snarkjs library. We see that the practical implementation gave us results that took much lesser number of steps (approximately 3) as the expected theoretical number of steps (expected 9). This may be attributed to the fact that the number of nodes we tested was for a maximum of 10, and the fact that we greatly weakened the power of the malicious nodes. We also note that even though anonymous consensus is achievable, it does not come for free. One has to bear with the large overhead of time taken to prove and verify zero knowledge proofs in each step making the overall protocol very inefficient.

10 ACKNOWLEDGEMENTS

We thank Prof. Xiaohui Gu and Ting Dai for useful inputs on the project, without which we would not have been able to make certain decisions on the course of the project. We also acknowledge the structure of the CSC 724 course that allowed us to take up this very interesting project and thus learn much about distributed systems, in such a short span of time.

REFERENCES

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. 2019. *Synchronous byzantine agreement with expected O (1) rounds, expected O (n2) communication, and optimal resilience.* Technical Report. Cryptology ePrint Archive, Report 2018/1028, 2018. https://eprint. iacr. org âĂę.
[2] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046. https://eprint.iacr.org/2018/046.
[3] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture.. In *USENIX Security Symposium*. 781–796.
[4] Adam Bender, Jonathan Katz, and Ruggero Morselli. 2006. Ring signatures: Stronger definitions, and constructions without random oracles. In *Theory of Cryptography Conference*. Springer, 60–79.
[5] David Bernhard and Bogdan Warinschi. 2016. Cryptographic Voting âĂŤ A Gentle Introduction. Cryptology ePrint Archive, Report 2016/765. https://eprint.iacr.org/2016/765.
[6] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *Journal of Cryptographic Engineering* 2, 2 (2012), 77–89.
[7] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. 1990. *Non-interactive zero knowledge.* Technical Report. MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE.
[8] Zohir Bouzid, Pierre Sutra, and Corentin Travers. 2011. Anonymous agreement: The janus algorithm. In *International Conference On Principles Of Distributed Systems*. Springer, 175–190.
[9] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.
[10] David Chaum. 1983. Blind signatures for untraceable payments. In *Advances in cryptology*. Springer, 199–203.
[11] David Chaum. 1988. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology* 1, 1 (1988), 65–75.
[12] David Chaum and Eugène Van Heyst. 1991. Group signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 257–265.
[13] Whitfield Diffie and Martin Hellman. 1976. New directions in cryptography. *IEEE transactions on Information Theory* 22, 6 (1976), 644–654.
[14] Pesech Feldman and Silvio Micali. 1997. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM J. Comput.* 26, 4 (1997), 873–933.
[15] Chaya Ganesh, Claudio Orlandi, and Daniel Tschudi. 2018. Proof-of-Stake Protocols for Privacy-Aware Blockchains. Cryptology ePrint Archive, Report 2018/1105. https://eprint.iacr.org/2018/1105.
[16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. 2016. ZKBoo: Faster Zero-Knowledge for Boolean Circuits. Cryptology ePrint Archive, Report 2016/163. https://eprint.iacr.org/2016/163.
[17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 51–68.
[18] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM Journal on computing* 18, 1 (1989), 186–208.
[19] iden3. 2019. circomlib. https://github.com/iden3/circomlib.
[20] Pyrros Chaidos Jens Groth Christophe Petit Jonathan Bootle, Andrea Cerulli. 2016. Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting. Cryptology ePrint Archive, Report 2016/263. https://eprint.iacr.org/2016/263.
[21] Jonathan Katz and Chiu-Yuen Koo. 2006. On expected constant-round protocols for Byzantine agreement. In *Annual International Cryptology Conference*. Springer, 445–462.
[22] Thomas Kerber, Markulf Kohlweiss, Aggelos Kiayias, and Vassilis Zikas. 2018. Ouroboros Crypsinous: Privacy-Preserving Proof-of-Stake. Cryptology ePrint Archive, Report 2018/1132. https://eprint.iacr.org/2018/1132.
[23] J-P Martin and Lorenzo Alvisi. 2006. Fast byzantine consensus. *IEEE Transactions on Dependable and Secure Computing* 3, 3 (2006), 202–215.
[24] Silvio Micali. 2016. Byzantine agreement made trivial.
[25] Silvio Micali and Vinod Vaikuntanathan. 2017. Optimal and Player-Replaceable Consensus with an Honest Majority. (2017).
[26] Michael Okun and Amnon Barak. 2008. Efficient algorithms for anonymous Byzantine agreement. *Theory of Computing Systems* 42, 2 (2008), 222–238.
[27] Marshall Pease, Robert Shostak, and Leslie Lamport. 1980. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)* 27, 2 (1980), 228–234.
[28] Pavel Raykov, Nicolas Schiper, and Fernando Pedone. 2011. Byzantine fault-tolerance with commutative commands. In *International Conference On Principles Of Distributed Systems*. Springer, 329–342.
[29] TTyler. 2012. Gossip Protocol. (2012).