Not a COINcidence: Sub-Quadratic Asynchronous Byzantine Agreement WHP

Shir Cohen
Technion – Israel Institute of Technology, Haifa, Israel
shirco@campus.technion.ac.il

Idit Keidar
Technion – Israel Institute of Technology, Haifa, Israel
idish@ee.technion.ac.il

Alexander Spiegelman
VMware Research, Herzliya, Israel
sasha.spiegelman@gmail.com

Abstract

King and Saia were the first to break the quadratic word complexity bound for Byzantine Agreement in synchronous systems against an adaptive adversary, and Algorand broke this bound with near-optimal resilience (first in the synchronous model and then with eventual-synchrony). Yet the question of asynchronous sub-quadratic Byzantine Agreement remained open. To the best of our knowledge, we are the first to answer this question in the affirmative. A key component of our solution is a shared coin algorithm based on a VRF. A second essential ingredient is VRF-based committee sampling, which we formalize and utilize in the asynchronous model for the first time. Our algorithms work against a delayed-adaptive adversary, which cannot perform after-the-fact removals but has full control of Byzantine processes and full information about communication in earlier rounds. Using committee sampling and our shared coin, we solve Byzantine Agreement with high probability, with a word complexity of $\widetilde{O}(n)$ and $O(1)$ expected time, breaking the $O(n^2)$ bit barrier for asynchronous Byzantine Agreement.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Cryptographic primitives; Mathematics of computing → Probabilistic algorithms

Keywords and phrases shared coin, Byzantine Agreement, VRF, sub-quadratic consensus protocol

Digital Object Identifier 10.4230/LIPIcs.DISC.2020.25

Related Version A full version of the paper is available at [16], `https://arxiv.org/abs/2002.06545`.

Acknowledgements We thank Ittai Abraham, Dahlia Malkhi, Kartik Nayak and Ling Ren for insightful initial discussions.

1 Introduction

Byzantine Agreement (*BA*) [28] has been studied for four decades by now, but until recently, has been considered at a fairly small scale. In recent years, however, we begin to see practical use-cases of BA in large-scale systems, which motivates a push for reduced communication complexity. In deterministic algorithms, Dolev and Reischuk's renown lower bound stipulates that $\Omega(n^2)$ communication is needed [18], and until fairly recently, almost all randomized solutions have also had (expected) quadratic word complexity. Recent work has broken this barrier [23, 21, 32], but not in asynchronous settings. We present here the first sub-quadratic asynchronous Byzantine Agreement algorithm. Our algorithm is randomized and solves binary BA *with high probability (whp)*, i.e., with probability that tends to 1 as n goes to infinity.

We consider a system with a static set of n processes, in the so-called "permissioned" setting, where the ids of all processes are well-known. Our algorithm tolerates f failures for $n \approx 4.5f$ (asymptotically). In addition, we assume a trusted *public key infrastructure* (PKI) that allows us to use *verifiable random functions* (VRFs) [30].

We assume a strong adversary that can adaptively take over processes, whereupon it has full access to their private data. It further sees all messages in the system. But we do limit the adversary in two ways. First, we assume that it is computationally bounded so that we may use the PKI. Second, as proven in [1] for the synchronous model, achieving sub-quadratic complexity is impossible when the adversary can perform after-the-fact removal, meaning that it can delete messages that were sent by correct processes before corrupting these processes. Here, we adapt the no after-the-fact removal assumption to the asynchronous model, and define a *delayed-adaptive adversary* based on causality [27].

We formalize the concept of VRF-based committee sampling as used in Algorand [21, 15], and adapt it to the asynchronous model. In a nutshell, the idea is to use a VRF seeded with each process's private key in order to sample uniformly at random $O(\log n)$ processes for a *committee*, and to have different committees execute different parts of the BA protocol. Each committee is used for sending exactly one protocol message and messages are sent only by committee members, thus reducing the communication cost. Whereas in Algorand's synchronous model a process can be sure it receives messages from all correct committee members by a timeout, in the asynchronous model this is not the case. Rather, processes make progress by waiting for some threshold number of messages. Without committees, this threshold is normally $n-f$ (waiting for more than $n-f$ processes might violate termination). But since committees are randomly sampled, we do not know the committee's exact size or the number of Byzantine processes in it. Thus, adapting committees to this model is somewhat subtle and requires ensuring certain conditions regarding the intersection of subsets of committees. In this paper we identify sufficient conditions on sampling, which ensure safety and liveness with high probability.

Randomized BA algorithms can be seen as if processes toss a random coin at some point during the protocol. While some protocols toss a local coin [9, 12] and require exponential expected time to reach agreement, others use the abstraction of a *shared coin*, which involves communication among processes and results in the same coin toss with some well defined *success rate* [34, 14, 13, 21, 24]. In this work we present an asynchronous shared coin algorithm that uses a VRF and provides a constant success rate with an equal probability for tossing 0 and 1. Unlike previous shared coin implementations, our solution does not require a priori knowledge of the set of participants, which makes it useful in committee-based constructions. We then adapt our coin to work with committees and use it to devise a sub-quadratic BA algorithm.

In summary, this paper presents the first formalization of randomly sampled committees using cryptography in asynchronous settings. Based on this technique, it presents the first sub-quadratic asynchronous shared coin and BA whp algorithms. Our algorithms have expected $\widetilde{O}(n)$ word complexity and $O(1)$ expected time.

Roadmap. The rest of this paper is organized as follows. Section 2 describes the model; Section 3 reviews related work. In Section 4, we present our shared coin algorithm and in Section 5, we formalize committee sampling. Then, in Section 6, we use the coin and the committee sampling to construct a BA whp algorithm. We end with some concluding remarks in Section 7.

2 Model and Preliminaries

We consider a distributed system consisting of a well-known static set Π of n processes and a *delayed-adaptive adversary* (see definition below). The adversary may adaptively corrupt up to $f = (\frac{1}{3} - \epsilon)n$ processes in the course of a run, where $\max\{\frac{3}{8 \ln n}, 0.109\} + \frac{1}{8 \ln n} < \epsilon < \frac{1}{3}$. A corrupted process is *Byzantine*; it may deviate arbitrarily from the protocol. In particular, it may crash, fail to send or receive messages, and send arbitrary messages. As long as a process is not corrupted by the adversary, it is *correct* and follows the protocol.

Delayed-adaptive adversary. In the synchronous model, one defines a *late* adversary [35, 25, 7, 4], which at the beginning of round r, can observe the state of the system at the beginning of round $r-1$. This assumption prevents "after-the-fact" removals of messages sent by processes before being taken over by the adversary [1, 21], as required for achieving a sub-quadratic communication cost. We adapt this assumption to the asynchronous model. Since in asynchronous models the natural order between messages is Lamport's happens-before relation [27], we use the notion of causality instead of "rounds" to define what messages the adversary may observe when scheduling other messages. We denote by $m \rightarrow m'$ the fact that m causally precedes m'. The adversary is formally defined as follows:

▶ **Definition 1** (delayed-adaptive adversary). *The delayed-adaptive adversary may adaptively corrupt up to f processes over the course of a run and schedules all messages. The adversary has full access to corrupted processes' private information and can observe all communication, but it can use the contents of a message m sent by a correct process for scheduling a message m' only if $m \rightarrow m'$.*

In addition, we assume that once the adversary takes over a process, it cannot "front run" messages that that process had already sent when it was correct, causing the correct messages to be supplanted. Blum et al. [10] achieve this property by using a separate key to encrypt each message, and deleting the secret key immediately thereafter.

Cryptographic tools. We assume a trusted PKI, where private and public keys for the processes are generated before the protocol begins and processes cannot manipulate their public keys. In addition, we assume that the adversary is computationally bounded, meaning that it cannot obtain the private keys of processes unless it corrupts them. Furthermore, we assume that the PKI is in place from the outset. (Recall that we assume a permissioned setting, so the public keys of the n processes are well-known). These assumptions allow us to use verifiable random functions, as we now define.

A *verifiable random function (*VRF*)* is a pseudorandom function that provides a proof of its correct computation [30]. Given a secret key sk, one can evaluate the VRF on any input x and obtain a pseudorandom output y together with a proof π, i.e., $\langle y, \pi \rangle = \text{VRF}_{sk}(x)$. From π and the corresponding public key pk, one can verify that y is correctly computed from x and sk using the function $\text{VRF-Ver}_{pk}(x, \langle y, \pi \rangle)$. Additionally, a VRF needs to satisfy *uniqueness*. More formally, a VRF guarantees the following properties:

- Pseudorandomness: for any x, it is infeasible to distinguish $y = \text{VRF}_{sk}(x)$ from a uniformly random value without access to sk.
- Verifiability: $\text{VRF-Ver}_{pk}(x, \text{VRF}_{sk}(x)) = true$.
- Uniqueness: it is infeasible to find $x, y_1, y_2, \pi_1, \pi_2$ such that $y_1 \neq y_2$ but $\text{VRF-Ver}_{pk}(x,\langle y_1, \pi_1 \rangle) = \text{VRF-Ver}_{pk}(x, \langle y_2, \pi_2 \rangle) = true$.

Efficient constructions for VRFs have been described in the literature [17, 20].

Communication. We assume that every pair of processes is connected via a reliable link. Messages are authenticated in the sense that if a correct process p_i receives a message m indicating that m was sent by a correct process p_j, then m was indeed generated by p_j and sent to p_i. The network is asynchronous, i.e., there is no bound on message delays.

Complexity. We use the following standard complexity notions [3, 31]. While measuring complexity, we allow a *word* to contain a signature, a VRF output, or a value from a finite domain. We define the *duration* of an execution as the longest sequence of messages that are causally related in this execution until all correct processes decide. We measure the expected *word communication complexity* of our protocols as the maximum of the expected total number of words sent by correct processes and the expected *running time* of our protocol as the maximum of the expected duration. In both cases the maximum is computed over all inputs and applicable adversaries and expectation is taken over the random VRF outputs.

3 Related Work

Lower bounds. Our assumptions conform with a number of known bounds. Deterministic consensus is impossible in an asynchronous system if even one process may crash (by FLP [19]) and requires $\Omega(n^2)$ communication even in synchronous systems [18]. As for randomized Byzantine Agreement, Abraham et al. [1] state that disallowing after-the-fact removal is necessary even in synchronous settings for achieving sub-quadratic communication.

Asynchronous BA and shared coin algorithms. The algorithms we present in this paper belong to the family of asynchronous BA algorithms, which sacrifice determinism in order to circumvent FLP. We compare our solutions to existing ones in Table 1.

Ben-Or [9] suggested a protocol with resilience $n > 5f$. This protocol uses a local coin (namely, a local source of randomness) and its expected time complexity is exponential (or constant if $f = O(\sqrt{n})$). Bracha [11] improved the resilience to $n > 3f$ with the same complexity. The complexity can be greatly reduced by replacing the local coin with a shared one with a guaranteed success rate.

Later works presented the shared coin abstraction and used it to solve BA with $O(n^2)$ communication. Rabin [34] was the first to do so, suggesting a protocol with resilience $n > 10f$ and a constant expected number of rounds. Cachin et al. [13] were the first to use a shared coin to solve BA with $O(n^2)$ communication and optimal resilience. Mostefaoui et al. [31] then presented a signature-free BA algorithm with optimal resilience and $O(n^2)$ messages that uses a shared coin abstraction as a black box; the shared coin algorithm we provide in Section 4 can be used to instantiate this protocol. All of the aforementioned algorithms solve binary BA, where the processes' initial values are 0 and 1; a recent work solved multi-valued BA with the same $O(n^2)$ word complexity [3].

BA algorithms also differ in the cryptographic assumptions they make and the cryptographic tools they use. Rabin's coin [34] is based on cryptographic secret sharing [36]. Some later works followed suit, and used cryptographic abstractions such as threshold signatures [3, 13]. Other works forgo cryptography altogether and instead consider a full information model, where there are no restrictions on the adversary's computational power [14, 24]. In this model, the problem is harder, and existing works achieve very low resilience [24] ($n > 400f$) or high communication complexity [14]. In this paper we do use cryptographic primitives. We assume a computationally bounded adversary and rely on the abstraction of a VRF [30]. VRFs were previously used in blockchain protocols [21, 22, 5] and were also used by Micali [29] to construct a shared coin in the synchronous model.

Table 1 Asynchronous Byzantine Agreement algorithms.

Protocol	n >	Adversary	Word complexity	Termination	Safety
Ben-Or [9]	$5f$	adaptive	$O(2^n)$	w.p. 1	✓
Rabin [34]	$10f$	adaptive	$O(n^2)$	w.p. 1	✓
Bracha [11]	$3f$	adaptive	$O(2^n)$	w.p. 1	✓
Cachin et al. [13]	$3f$	adaptive	$O(n^2)$	w.p. 1	✓
King-Saia [24]	$400f$	adaptive	polynomial	whp	✓
MMR [31]	$3f$	adaptive	$O(n^2)$	w.p. 1	✓
Our protocol	$\approx 4.5f$	delayed-adaptive	$\tilde{O}(n)$	whp	whp

Several works [2, 8, 26, 33, 37] solve BA with subquadratic complexity in the so-called optimistic case (or "happy path"), when communication is timely and a correct process is chosen as a "leader". In contrast, we focus on the worst-case asynchronous case.

Committees. We use committees in order to reduce the word complexity and allow each step of the protocol to be executed by only a fraction of the processes. King and Saia used a similar concept and presented the first sub-quadratic BA protocol in the synchronous model [23]. Algorand proposed a synchronous algorithm [21] (and later extended it to eventual synchrony [15]) where committees are sampled randomly using a VRF. Each process executes a local computation to sample itself to a committee, and hence the selection of processes does not require interaction among them. We follow this approach in this paper and adapt the technique to the asynchronous model.

Following initial publication of our work, Blum et al. [10] have also achieved subquadratic BA WHP under an adaptive adversary. Their assumptions are incomparable to ours – while they strengthen the adversary to remove the delayed adaptivity requirement, they also strengthen the trusted setup. Specifically, they use a trusted dealer to a priori determine the committee members, flip the shared coin, and share it among the committee members. In contrast, we use a peer-to-peer protocol to generate randomness, and require delayed adaptivity in order to prevent the adversary from tampering with this randomness. As in our protocol, setup has to occur once and may be used for any number of BA instances.

4 Shared Coin

We describe here an asynchronous protocol for a shared coin with a constant success rate against the delayed-adaptive adversary. We assume that for every $r \in \mathbb{N}$, shared_coin(r) is invoked by all correct processes and that the invocation of shared_coin(r) by some process p is causally independent of its progress at other processes. The definition of a shared coin is given below.

▶ **Definition 2** (Shared Coin). *A shared coin with success rate ρ is a shared object that generates an infinite sequence of binary outputs. For each execution of the procedure shared_coin(r) with $r \in \mathbb{N}$, all correct processes output b with probability at least ρ, for any value $b \in \{0, 1\}$.*

The pseudo-code for our shared coin is presented in Algorithm 1. Our protocol is composed of two phases of messages passing. Each process first samples the VRF with its private key and the protocol's argument in order to generate a random initial value. For brevity, we denote by VRF_i the VRF with p_i's private key. Using a VRF to generate a random initial value effectively weakens the adversary as Byzantine processes can neither choose their initial values nor equivocate. If a Byzantine process would try to act maliciously, the VRF proof would easily expose it and its message would be ignored.

In each phase of the protocol, each process sends one value to every other process. The receiver validates the received values using the VRF proofs, which are sent along with the values. We omit the proof validation from the code for clarity. After two phases of communication, each process chooses the minimum value it received in the second phase and outputs its least significant bit. We follow the concept of a common core, as presented by Attiya and Welch for the crash failure model [6], and argue that if a core of $f+1$ correct processes hold the global minimum value at the end of phase 1, then by the end of the following phase all processes receive this value. We exploit the ϵ parameter in our resilience definition to bound the number of values held by $f+1$ correct processes. We show that this number is linear in n and hence with a constant positive probability, by the end of the second phase, all correct processes receive the global minimum among the VRF outputs and therefore produce the same output.

Algorithm 1 Protocol shared_coin(r): code for process p_i.

```
1: Initially first-set, second-set = $\emptyset$
2: $v_i \leftarrow VRF_i(r)$
3: send $\langle \text{FIRST}, v_i \rangle$ to all processes

4: upon receiving $\langle \text{FIRST}, v_j \rangle$ with valid $v_j$ from $p_j$ do
5:     if $v_j < v_i$ then $v_i \leftarrow v_j$
6:     first-set $\leftarrow$ first-set $\cup \{j\}$
7:     when $|$first-set$| = n - f$ for the first time
8:         send $\langle \text{SECOND}, v_i \rangle$ to all processes

9: upon receiving $\langle \text{SECOND}, v_j \rangle$ with valid $v_j$ from $p_j$ do
10:    if $v_j < v_i$ then $v_i \leftarrow v_j$
11:    second-set $\leftarrow$ second-set $\cup \{j\}$
12:    when $|$second-set$| = n - f$ for the first time
13:        return $LSB(v_i)$
```

We now prove that the shared coin has a constant success rate. We say that a value v is *common* if at least $f+1$ correct processes receive v by the end of phase 1. Denote by c be the number of different common values. The next two lemmas give a lower bound on c and on the probability that the global minimum among the VRF outputs is common.

▶ **Lemma 3.** *In Algorithm 1, $c \geq \frac{9\epsilon}{1+6\epsilon} n$.*

Proof. In a given run of the algorithm, define a table T with n rows and n columns, where for each correct process p_i and each $0 \leq j \leq n-1$, $T[i,j] = 1$ iff p_i receives $\langle \text{FIRST}, v \rangle$ from p_j before sending the second message in line 8. Each row of a correct process contains exactly $n-f$ ones since it waits for $n-f$ $\langle \text{FIRST}, v \rangle$ messages (line 7). Each row of a faulty process is arbitrarily filled with $n-f$ ones and f zeros. Thus, the total number of ones in the table is $n(n-f)$ and the total number of zeros is nf. Let k be the number of columns with at least $2f+1$ ones. Because each column represents a value and out of the $2f+1$ ones at least

$f+1$ represent correct processes that receive this value, $c \geq k$. Denote by x the number of ones in the remaining columns. Because each column has at most n ones we get:

$$x \geq n(n-f) - kn. \tag{1}$$

And because the remaining columns have at most $2f$ ones:

$$x \leq 2f(n-k). \tag{2}$$

Combining $(1), (2)$ we get:

$$2f(n-k) \geq n(n-f) - kn$$

$$2fn - 2fk \geq n^2 - fn - kn$$

$$(n-2f)k \geq n^2 - 3fn$$

$$k \geq \frac{n(n-3f)}{n-2f}.$$

Because $f = (\frac{1}{3} - \epsilon)n$ we get:

$$c \geq k \geq \frac{n(n-3(\frac{1}{3}-\epsilon)n)}{n-2(\frac{1}{3}-\epsilon)n} = \frac{n(1-1+3\epsilon)}{1-\frac{2}{3}+2\epsilon} = \frac{9\epsilon}{1+6\epsilon}n, \text{as required.}$$ ◀

Let $v_{min} \triangleq \min_{p_i \in \Pi}\{VRF_i(r)\}$. We prove that with a constant probability, it is common.

▶ **Lemma 4.** *$Prob[v_{min}$ is common$] \geq \frac{c}{n} - \frac{1}{3} + \epsilon$.*

Proof. Notice that we assume that the invocation of shared_coin(r) by each process is causally independent of its progress at other processes. Hence, for any two processes p_i, p_j, the messages $\langle \text{FIRST}, v_i \rangle$, $\langle \text{FIRST}, v_j \rangle$ are causally concurrent. Thus, due to our *delayed-adaptive adversary* definition, these messages are scheduled by the adversary regardless of their content, namely their VRF random values. Notice that the adversary can corrupt processes before they initially send their VRF values. Since the adversary cannot predict the VRF outputs of the processes, the probability that the process holding v_{min} is corrupted before sending its FIRST messages is at most $\frac{f}{n}$. The adversary is oblivious to the correct processes' VRF values when it schedules their first phase messages. Therefore, each of them has the same probability to become common. Since at most f common values originate at Byzantine processes, this probability is at least $\frac{c-f}{n-f}$. We conclude that v_{min} is common with probability at least $(1-\frac{f}{n})\frac{c-f}{n-f} = (1-\frac{(\frac{1}{3}-\epsilon)n}{n})\frac{c-(\frac{1}{3}-\epsilon)n}{n-(\frac{1}{3}-\epsilon)n} = (\frac{2}{3}+\epsilon)\frac{c-(\frac{1}{3}-\epsilon)n}{(\frac{2}{3}+\epsilon)n} = \frac{c-(\frac{1}{3}-\epsilon)n}{n} = \frac{c}{n} - \frac{1}{3} + \epsilon$. ◀

We next observe that if v_{min} is common, then it is shared by all processes.

▶ **Lemma 5.** *If v_{min} is common then each correct process holds v_{min} at the end of phase 2.*

Proof. Since v_{min} is common, at least $f+1$ correct processes receive it by the end of phase 1 and update their local values to v_{min}. During the second phase, each correct process hears from $n-f$ processes. This means that it hears from at least one correct process that has updated its value to v_{min} and sent it. ◀

▶ **Lemma 6.** *The coin's success rate is at least $\frac{18\epsilon^2+24\epsilon-1}{6(1+6\epsilon)}$.*

Proof. We bound the probability that all correct processes output $b \in \{0, 1\}$ as follows:

$Prob$[all correct processes output b] $\geq Prob$[all correct processes have the same v_i at the end of phase 2 and its LSB is b] $\geq Prob$[all correct processes have $v_i = v_{min}$ at the end of phase 2 and its LSB is b] $= \frac{1}{2} \cdot Prob$[all correct processes have $v_i = v_{min}$ at the end of phase 2] $\overset{\text{Lemma 5}}{\geq} \frac{1}{2} \cdot Prob[v_{min}$ is common] $\overset{\text{Lemma 4}}{\geq} \frac{1}{2}(\frac{c}{n} - \frac{1}{3} + \epsilon) \overset{\text{Lemma 3}}{\geq} \frac{18\epsilon^2+24\epsilon-1}{6(1+6\epsilon)}$. ◀

▶ Remark 7. Notice that for $\epsilon = \frac{1}{3}$ (i.e., $f = 0$) it holds that the coin's success rate is $\frac{1}{2}$ and we get a perfect fair coin.

We have shown a bound on the coin's success rate in terms of ϵ. Since $\epsilon > 0.109$, the coin's success rate is a positive constant. We next prove that the coin ensures liveness.

▶ **Lemma 8.** *If all correct processes invoke Algorithm 1 then all correct processes return.*

Proof. All correct processes send their messages in the first phase. As up to f processes may be faulty, each correct process eventually receives $n - f$ $\langle \text{FIRST}, x\rangle$ messages and sends a message in the second phase. As $n - f$ correct processes send their messages, each correct process eventually receives $n - f$ $\langle \text{SECOND}, x\rangle$ messages and returns. ◀

From Lemma 6 and Lemma 8 we conclude:

▶ **Theorem 9.** *Algorithm 1 implements a shared coin with success rate at least* $\frac{18\epsilon^2+24\epsilon-1}{6(1+6\epsilon)}$.

Complexity. In each shared coin instance all correct processes send two messages to all other processes. Each of these messages contains one VRF output (including a value and a proof), in addition to a constant number of bits that identify the message's type. Therefore, each message's size is a constant number of words and the total word complexity of a shared coin instance is $O(n^2)$.

We have presented a new shared coin in the asynchronous model that uses a VRF. This coin can be incorporated into the Byzantine Agreement algorithm of Mostefaoui et al. [31], to yield an asynchronous binary Byzantine Agreement with resilience $f = (\frac{1}{3} - \epsilon)n$, a word complexity of $O(n^2)$, and $O(1)$ expected time.

5 Committees

5.1 Validated committee sampling

With the aim of reducing the number of messages and achieving sub-quadratic word complexity, it is common to avoid all-to-all communication phases [21, 23]. Instead, a subset of processes is sampled to a committee and only processes elected to the committee send messages. As committees are randomly sampled, preventing the adversary from corrupting their members, each committee member cannot predict the next committee sample and send its message to all other processes. Potentially, if the committee is sufficiently small, this technique allow committee-based protocols to result in sub-quadratic word complexity.

Using VRFs, it is possible to implement *validated committee sampling*, which is a primitive that allows processes to elect committees without communication and later prove their election. It provides every process p_i with a private function $sample_i(s, \lambda)$, which gets a string s and a threshold $1 \leq \lambda \leq n$ and returns a tuple $\langle v_i, \sigma_i\rangle$, where $v_i \in \{true, false\}$ and σ_i is a proof that $v_i = sample_i(s, \lambda)$. If $v_i = true$ we say that p_i is *sampled* to the committee for s and λ. The primitive ensures that p_i is sampled with probability $\frac{\lambda}{n}$. In addition, there is a public (known to all) function, *committee-val*$(s, \lambda, i, \sigma_i)$, which gets a string s, a threshold λ, a process identification i and a proof σ_i, and returns *true* or *false*.

Consider a string s. For every i, $1 \le i \le n$, let $\langle v_i, \sigma_i \rangle$ be the return value of $sample_i(s, \lambda)$. The following is satisfied for every p_i:

- *committee-val*$(s, \lambda, i, \sigma_i) = v_i$.
- If p_i is correct, then it is infeasible for the adversary to compute $sample_i(s, \lambda)$.
- It is infeasible for the adversary to find $\langle v, \sigma \rangle$ s.t. $v \ne v_i$ and *committee-val*$(s, \lambda, i, \sigma) =$ *true*.

We refer to the set of processes sampled to the committee for s and λ as $C(s, \lambda)$. In this paper we set λ to $8 \ln n$. Let d be a parameter of the system such that $\max\{\frac{1}{\lambda}, 0.0362\} < d < \frac{\epsilon}{3} - \frac{1}{3\lambda}$. Our committee-based protocols can no longer wait for $n - f$ processes. Instead, they wait for $W \triangleq \lceil (\frac{2}{3} + 3d)\lambda \rceil$ processes. We show that whp at least W processes will be correct in each committee sample and hence waiting for this number does not compromise liveness. In addition, instead of assuming f Byzantine processes, our committee-based protocols assume that whp the number of Byzantine processes in each committee is at most $B \triangleq \lfloor (\frac{1}{3} - d)\lambda \rfloor$. The following claim is proven in the full version of the paper [16] using Chernoff bounds.

▷ Claim 10. For a string s and $\lambda = const \cdot \ln n$ the following hold with high probability:
(S1) $|C(s, \lambda)| \le (1 + d)\lambda$.
(S2) $|C(s, \lambda)| \ge (1 - d)\lambda$.
(S3) At least W processes in $C(s, \lambda)$ are correct.
(S4) At most B processes in $C(s, \lambda)$ are Byzantine.

If a protocol uses a constant number of committees, then with high probability, Claim 10 holds for all of them. If, however, a protocol uses a polynomial number of committees then it does not guarantee the properties of this claim. The following corollaries are derived from Claim 10 and are used to ensure the safety and liveness properties of our protocols that use committees (a proof is in the full version). Intuitively, S3 allows the protocol to wait for W messages without forgoing liveness. Property S5 below shows that if two processes wait for sets P_1 and P_2 of this size, then they hear from at least $B + 1$ common processes of which, by S4, at least one is correct.

▶ **Corollary 11** (S5). *Consider $C(s, \lambda)$ for some string s and some $\lambda = const \cdot \ln n$ and two sets $P_1, P_2 \subset C(s, \lambda)$ s.t $|P_1| = |P_2| = W$. Then, $|P_1 \cap P_2| \ge B + 1$.*

The following property is used to show that if $B + 1$ correct processes hold some value, and some correct process waits for messages from W processes, then it hears from at least one correct process that holds this value.

▶ **Corollary 12** (S6). *Consider $C(s, \lambda)$ for some string s and some $\lambda = const \cdot \ln n$ and two sets $P_1, P_2 \subset C(s, \lambda)$ s.t $|P_1| = B + 1$ and $|P_2| = W$. Then, $|P_1 \cap P_2| \ge 1$.*

5.2 WHP Coin

We now employ committee sampling to reduce the word complexity of our shared coin. Our new protocol is called whp_coin. As before, we assume that for every $r \in \mathbb{N}$, the invocation of whp_coin(r) by some process p is causally independent of its progress at other processes. We now define the *WHP coin* abstraction:

▶ **Definition 13** (WHP Coin). *A WHP coin with success rate ρ is a shared object exposing whp_coin(r), $r \in \mathbb{N}$ at each process. If all correct processes invoke whp_coin(r) then, whp (1) all correct processes return, and (2) all of them output the same value b with probability at least ρ, for any value $b \in \{0, 1\}$.*

The whp_coin protocol is presented in Algorithm 2. It samples two committees, one for each communication step. In each step, only the processes that are sampled to the committee send messages. However, since the committee samples are unpredictable, messages are sent to all processes. With committees, processes can no longer wait for $n - f$ messages. Instead they wait for W messages. Since a constant number of committees is sampled in the protocol, Claim 10 holds for all of them and by S3, all processes receive W messages, ensuring liveness.

Algorithm 2 Protocol whp_coin(r): code for process p_i.

```
1: Initially $first\text{-}set, second\text{-}set = \emptyset$, $v_i = \infty$
2: if $sample_i(\text{FIRST}, \lambda) = true$ then
3:     $v_i \leftarrow VRF_i(r)$
4:     send $\langle \text{FIRST}, v_i \rangle$ to all processes

5: upon receiving $\langle \text{FIRST}, v_j \rangle$ with valid $v_j$
       from validly sampled $p_j$ do
6:     if $sample_i(\text{SECOND}, \lambda)$ then
7:         if $v_j < v_i$ then $v_i \leftarrow v_j$
8:         $first\text{-}set \leftarrow first\text{-}set \cup \{j\}$
9:         when $|first\text{-}set| = W$ for the first time
10:            send $\langle \text{SECOND}, v_i) \rangle$ to all processes

11: upon receiving $\langle \text{SECOND}, v_j \rangle$ with valid $v_j$
       from validly sampled $p_j$ do
12:     if $v_j < v_i$ then $v_i \leftarrow v_j$
13:     $second\text{-}set \leftarrow second\text{-}set \cup \{j\}$
14:     when $|second\text{-}set| = W$ for the first time
15:         return $LSB(v_i)$
```

In the full version of the paper [16] we adapt the coin's correctness proof given in Section 4 to the committee-based protocol, proving the following theorem:

▶ **Theorem 14.** *Algorithm 2 implements a WHP coin with a constant success rate.*

Complexity. In each whp_coin instance using committees all correct processes that are sampled to the two committees (lines 2,6) send messages to all other processes. Each of these messages contains a VRF output (including a value and a proof), a VRF proof of the sender's election to the committe and a constant number of bits that identify the type of message that is sent. Therefore, each message's size is a constant number of words and the total word complexity of a WHP coin instance is $O(nC)$ where C is the number of processes that are sampled to the committees. Since each process is sampled to a committee with probability $\frac{1}{\lambda}$, we get a word complexity of $O(n\lambda) = O(n \log n) = \widetilde{O}(n)$ in expectation.

6 Asynchronous sub-quadratic Byzantine Agreement

We adapt the Byzantine Agreement algorithm of Mostefaoui et al. [31] to work with committees. Our protocol leverages an *approver* abstraction, which we implement in Section 6.1 and then integrate it into a Byzantine Agreement protocol in Section 6.2.

6.1 Approver abstraction

The *approver* abstraction is an adaptation of the Synchronized Binary-Value Broadcast (SBV-broadcast) primitive in [31]. It provides processes with the procedure *approve(v)*, which takes a value v as an input and returns a set of values.

▶ **Assumption 1.** *Correct processes invoke the approver with at most* 2 *different values.*

Under this assumption, an approver satisfies the following:

▶ **Definition 15** (Approver). *In an approver instance the following properties hold whp:*
- *Validity. If all correct processes invoke approve(v) then the only possible return value of correct processes is* $\{v\}$.
- *Graded Agreement. If a correct process* p_i *returns* $\{v\}$ *and another correct process* p_j *returns* $\{w\}$ *then* $v = w$.
- *Termination. If all correct processes invoke approve then approve returns with a non-empty set at all of them.*

Our approver uses different committees for different message types, as illustrated in Fig. 1. Importantly, the protocol satisfies the so-called *process replaceability* [21] property, whereby a correct process selected for a committee C broadcasts at most one message in its role as a member of C. Thus, our delayed-adaptive adversary can learn of a process's membership in a committee only after that process ceases to partake in the committee. This allows us to leverage the sampling analysis in the previous section. For clarity of the presentation, we discuss the algorithm here under the assumption that properties S1-S6 hold for all sampled committees. As shown above, these hold whp for each committee, and the algorithm employs a constant number of committees, so they hold for all of them whp.

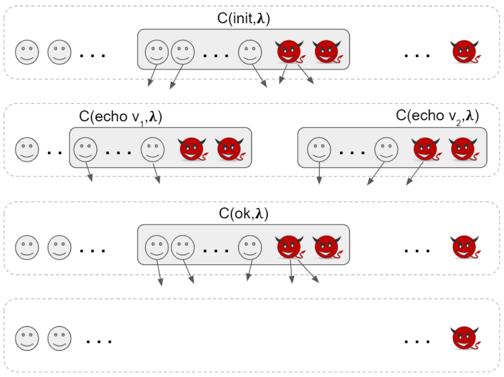

Figure 1 Committees sampled in Algorithm 3.

The approver's pseudo-code appears in Algorithm 3. It consists of three phases – init, echo, and ok. In each phase, committee members broadcast to all processes. Messages are validated to originate from legitimate committee members using the *committee-val* primitive; this validation is omitted from the pseudo-code for clarity. In the first phase, each init committee member broadcasts its input value to all processes.

The role of the echo phase is to "boost" values sent by sufficiently many processes in the init phase, and make sure that *all* correct processes receive them. "Sufficiently many" here means at least $B + 1$, which by S4 includes at least one correct process. Ensuring process replaceability in the echo phase is a bit tricky, since committee members must echo every value they receive from least $B+1$ processes, and there might be two such values. (Recall that we assume that correct processes invoke the protocol with at most two different values, so there cannot be more than two values that exceed this threshold). To ensure that each committee member broadcasts at most once, we sample a different committee for each value. That is, the value v is part of the string passed to the sample function for this phase.

When a member of the ok committee receives $\langle \text{ECHO}, v\rangle$ messages from W different members of the same echo committee for the first time, it broadcasts an $\langle \text{OK}, v\rangle$ message. Note that the process sends an ok message only for the first value that exceeds this threshold. An $\langle \text{OK}, v\rangle$ message includes, as proof of its validity, W signed $\langle \text{ECHO}, v\rangle$ messages. Again, the proof and its validation are omitted from the pseudo-code for clarity. Once a correct process receives W valid OK messages, it returns the set of values in these messages.

Algorithm 3 Protocol approve(v_i): code for process p_i.

```
1: if $sample_i(\text{INIT}, \lambda) = true$ then broadcast $\langle \text{INIT}, v_i\rangle$

2: upon receiving $\langle \text{INIT}, v\rangle$ from $B+1$ different processes do
3:     if $sample_i(\langle \text{ECHO}, v\rangle, \lambda) = true$ then broadcast $\langle \text{ECHO}, v\rangle$

4: upon receiving $\langle \text{ECHO}, v\rangle$ from $W$ different processes do
5:     if $sample_i(\text{OK}, \lambda) = true \wedge$ haven't sent any $\langle \text{OK}, *\rangle$ message then
6:         broadcast $\langle \text{OK}, v\rangle$

7: upon receiving $\langle \text{OK}, *\rangle$ from $W$ different processes do
8:     return the set of values received in these messages
```

We next prove that Algorithm 3 implements an approver. The following three lemmas are stated here and their proofs appear in the full version [16]:

▶ **Lemma 16** (Validity). *If all correct processes invoke* approve(v) *then the only possible return value of correct processes is* $\{v\}$ *whp.*

▶ **Lemma 17** (Graded Agreement). *If a correct process* p_i *returns* $\{v\}$ *and another correct process* p_j *returns* $\{w\}$ *then* $v = w$ *whp.*

▶ **Lemma 18** (Termination). *If all correct processes invoke approve then at every correct process approve returns with a non-empty set whp.*

From Lemmas 16,17,18, we conclude the following theorem:

▶ **Theorem 19.** *Algorithm 3 implements an approver.*

Complexity. In each approver instance correct processes that are sampled to the four committees (lines 1,3,5) send messages to all other processes. The committee size is $O(\lambda) = O(\log n)$ whp. Messages contain values, VRF proofs of the sender's election to the committee, signatures of $O(\lambda)$ committee members, and a constant number of bits that identify the type of message that is sent. Therefore, each message's size is at most $O(\lambda)$ words and the total word complexity of a shared coin instance is $O(n\lambda^2) = O(n \log^2 n) = \widetilde{O}(n)$ in expectation. The λ^2 appears in the expression due to the signatures of $O(\lambda)$ processes sent along the ok messages.

6.2 Byzantine Agreement WHP

Our next step is solving Byzantine Agreement whp, formally defined as follows:

▶ **Definition 20** (Byzantine Agreement WHP)**.** *In Byzantine Agreement WHP, each correct process $p_i \in \Pi$ proposes a binary input value v_i and decide on an output value $decision_i$ s.t. with high probability the following properties hold:*

- *Validity. If all correct processes propose the same value v, then any correct process that decides, decides v.*
- *Agreement. No two correct processes decide differently.*
- *Termination. Every correct process eventually decides.*

We present the pseudo-code for our algorithm in Algorithm 4. Our protocol executes in rounds. Each round consists of two approver invocations and one call to the WHP coin. Again, we discuss the algorithm assuming S1-S6 hold. We will argue that the algorithm decides in a constant number of rounds whp, and so these properties hold for all the committees it uses. The local variable est_i holds p_i's current estimate of the decision value. The variable $decision_i$ holds p_i's irrevocable decision. It is initialized to $\perp$ and set to a value in $\{0, 1\}$ at most once. Every process p_i begins by setting est_i to hold its initial value. At the beginning of each round processes execute the approver with their est values. If they return a singleton $\{v\}$, they choose to invoke the next approver with v as their proposal and otherwise they invoke the next approver with $\perp$. By the approver's graded agreement property, different processes do not return different singletons. Thus, at most two different values ($\perp$ and one in $\{0, 1\}$) are given as an input by correct processes to the next approver, satisfying Assumption 1.

At this point, after all correct processes have chosen their proposals, they all invoke the WHP coin in line 8 in order to select a fall-back value. Notice that executing the WHP coin protocol after proposals have been set prevents the adversary from biasing proposals based on the coin flip. Then, in in line 9, all processes invoke the approver with their proposals. If a process does not receive $\perp$ in its return set, it can safely decide the value it received. It does so by updating its $decision$ variable in line 13. If it receives some value other than $\perp$ it adopts it to be its estimated value (line 18), whereas if it receives only $\perp$, it adopts the coin flip (line 16). If all processes receive $\perp$ in line 4 then the probability that they all adopt the same value is at least 2ρ, where ρ is the coin's success rate. If some processes receive v, then the probability that all the processes that adopt the coin flip adopt v is at least ρ. With high probability, after a constant number of rounds, all correct processes have the same estimated value. By validity of the approver, once they all have common estimate, they decide upon it within 1 round.

We now prove our main theorem:

▶ **Theorem 21.** *Algorithm 4 when using an approver (Definition 15) and a WHP coin (Definition 13) solves Byzantine Agreement whp (Definition 20).*

We first show that Algorithm 4 satisfies the approver and WHP coin primitives' assumptions whp. Proving this allows us to use their properties while proving Theorem 21.

▶ **Lemma 22.** *For every round r of Algorithm 4 the following hold:*

1. *All correct processes invoke approve with at most 2 different values.*
2. *The invocation of $whp\_coin(r)$ by a correct process p is causally independent of its progress at other processes.*

Algorithm 4 Protocol Byzantine Agreement(v_i): code for process p_i.

```
1:  $est_i \leftarrow v_i$
2:  $decision_i \leftarrow \bot$

3:  for $r = 0, 1, \ldots$ do
4:      $vals \leftarrow$ approve($est_i$)
5:      if $vals = \{v\}$ for some $v$ then
6:          $propose_i \leftarrow v$
7:      otherwise $propose_i \leftarrow \bot$

8:      $c \leftarrow$ whp_coin($r$)
9:      $props \leftarrow$ approve($propose_i$)
10:     if $props = \{v\}$ for some $v \neq \bot$ then
11:         $est_i \leftarrow v$
12:         if $decision_i = \bot$ then
13:             $decision_i \leftarrow v$
14:     else
15:         if $props = \{\bot\}$ then
16:             $est_i \leftarrow c$
17:         else                              ▷ $props = \{v, \bot\}$
18:             $est_i \leftarrow v$
```

Proof. **1.** It is easy to see, by induction on the number of rounds, that since the processes' inputs are binary and we use a binary coin, the *est* of all processes is in $\{0, 1\}$ at the beginning of each round. Hence, the approver in line 4 is invoked with at most two different values. Due to its graded agreement property, all processes that update their propose to $v \neq \bot$ in line 6 update it to the same value whp. Thus, whp, in line 9 approver is invoked with either v or $\bot$.

2. Correct processes call *whp_coin*(r) without waiting for indication that other processes have done so. ◀

The following lemma shows that for any given round of the algorithm, (1) whp all processes complete this round, and (2) with a constant probability, they all have the same estimate value by its end. Its proof is in the full version of the paper.

▶ **Lemma 23.** *If all correct processes begin round r of Algorithm 4 then whp:*
1. *All correct processes complete round r, i.e. they're not blocked during round r.*
2. *With probability greater than ρ, where ρ is the success rate of the WHP coin, all correct processes have the same est value at the end of round r.*

The following lemmas indicate that the Byzantine Agreement whp properties are satisfied, which completes the proof of Theorem 21.

▶ **Lemma 24.** *(Validity) If at the beginning of round r of Algorithm 4 all correct processes have the same estimate value v, then whp any correct process that has not decided before decides v in round r.*

Proof. If all correct processes start round r then by Lemma 23 they all complete round r. Since they all being with the same estimate value v, they all execute approve(v) in line 4. Hence, by approver's validity and termination, whp they all return the non-empty set $\{v\}$ and update their *propose* values to v. Then, they all execute approve(v) for the second time in line 9, and due to the same reason, they all return $\{v\}$ whp. Any correct process that has not decided before decides v in line 13. ◀

▶ **Lemma 25.** *(Termination) Every correct process decides whp.*

Proof. By Lemma 23, for every round r of Algorithm 4, with probability greater than ρ, where ρ is the success rate of the WHP coin, all correct processes have the same *est* value at the end of r whp. Hence, by Lemma 24, with probability greater than ρ, all correct processes decide by round $r+1$ whp. It follows that the expected number of rounds until all processes decide is bounded by $\frac{1}{\rho}$, which is constant. Thus, by Chebyshev's inequality, whp all correct processes decide within a constant number of rounds. ◀

▶ **Lemma 26.** *(Agreement) No two correct processes decide different values whp.*

Proof. Let r be the first round in which some process p_i decides on some value $v \in \{0,1\}$. Thus, p_i's invocation to approver in line 9 of round r returns $\{v\}$. If another correct process p_j decides w in round r then its approver call in line 9 of round r returns $\{w\}$. By approver's graded agreement, $v = w$ whp. Consider a correct process p_k that does not decide in round r. By the definition of r, p_k hasn't decided in any round $r' < r$. By approver's graded agreement, whp, p_k returns $\{v, \bot\}$ in line 9 of round r, and p_k updates its est_k value to v in line 18. It follows that whp all correct processes have v as their estimate value at the beginning of round $r+1$. By Lemma 24, every correct process that has not decided in round r decides v in round $r+1$ whp. ◀

Complexity. In each round of the protocol, all correct processes invoke two approver calls and one WHP coin instance. Due to the constant success rate of the WHP coin, the expected number of rounds before all correct processes decide is constant. Thus, due to the word complexity of the WHP coin and approver, the expected word complexity is $O(n\log^2 n) = \widetilde{O}(n)$ and the time complexity is $O(1)$ in expectation.

7 Conclusions and Future Directions

We have presented the first sub-quadratic asynchronous Byzantine Agreement algorithm. To construct the algorithm, we introduced two techniques. First, we presented a shared coin algorithm that requires a trusted PKI and uses VRFs. Second, we formalized VRF-based committee sampling in the asynchronous model for the first time.

Our algorithm solves Byzantine Agreement with high probability. It would be interesting to understand whether some of the problem's properties can be satisfied with probability 1, while keeping the sub-quadratic communication cost. In addition, in order to achieve the constant success rate of the coin and guarantee the committees' properties, we bounded ϵ from below by a constant. This bound prevented us from achieving optimal resilience. The question whether it is possible to relax this bound to allow better resilience remains open.

References

1 Ittai Abraham, TH Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 317–326, 2019.

2 Ittai Abraham, Guy Golan-Gueta, and Dahlia Malkhi. Hot-stuff the linear, optimal-resilience, one-message bft devil. *CoRR*, abs/1803.05069, 2018.

3 Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2019.

4 Mohamad Ahmadi, Abdolhamid Ghodselahi, Fabian Kuhn, and Anisur Rahaman Molla. The cost of global broadcast in dynamic radio networks. *Theoretical Computer Science*, 806:363–387, 2020.

5 Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. Helix: A scalable and fair consensus algorithm resistant to ordering manipulation. *IACR Cryptology ePrint Archive*, 2018:863, 2018.

6 Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. John Wiley & Sons, 2004.

7 Baruch Awerbuch and Christian Scheideler. A denial-of-service resistant dht. In *International Symposium on Distributed Computing*, pages 33–47. Springer, 2007.

8 Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. State machine replication in the libra blockchain. *The Libra Assn., Tech. Rep*, 2019.

9 Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30. ACM, 1983.

10 Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. Cryptology ePrint Archive, Report 2020/851, 2020.

11 Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

12 Gabriel Bracha and Sam Toueg. Resilient consensus protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 12–26. ACM, 1983.

13 Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

14 Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC*, volume 93, pages 42–51. Citeseer, 1993.

15 Jing Chen, Sergey Gorbunov, Silvio Micali, and Georgios Vlachos. Algorand agreement: Super fast and partition resilient byzantine agreement. *IACR Cryptology ePrint Archive*, 2018:377, 2018.

16 Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement whp. *arXiv preprint arXiv:2002.06545*, 2020.

17 Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

18 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, January 1985.

19 Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

20 Matthew Franklin and Haibin Zhang. Unique ring signatures: A practical construction. In *International Conference on Financial Cryptography and Data Security*, pages 162–170. Springer, 2013.

21 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

22 Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

23 Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. *Journal of the ACM (JACM)*, 58(4):1–24, 2011.

24 Valerie King and Jared Saia. Byzantine agreement in polynomial expected time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 401–410. ACM, 2013.

25 Marek Klonowski, Dariusz R Kowalski, and Jarosław Mirek. Ordered and delayed adversaries and how to work against them on a shared channel. *Distributed Computing*, 32(5):379–403, 2019.

26 Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1(11), 2014.

27 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

28 Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

29 Silvio Micali. Very simple and efficient byzantine agreement. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 6:1–6:1. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.ITCS.2017.6`.

30 Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 120–130. IEEE, 1999.

31 Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *Journal of the ACM (JACM)*, 62(4):31, 2015.

32 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

33 Oded Naor, Mathieu Baudet, Dahlia Malkhi, and Alexander Spiegelman. Cogsworth: Byzantine view synchronization. *arXiv preprint arXiv:1909.05204*, 2019.

34 Michael O Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409. IEEE, 1983.

35 Peter Robinson, Christian Scheideler, and Alexander Setzer. Breaking the $\Omega(\sqrt{n})$ barrier: Fast consensus under a late adversary. In *30th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2018*, pages 173–182. ACM New York, 2018.

36 Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

37 Alexander Spiegelman. In search for a linear byzantine agreement. *arXiv preprint arXiv:2002.06993*, 2020.