EPIC: Efficient Asynchronous BFT with Adaptive Security

Chao Liu
UMBC

Sisi Duan
UMBC

Haibin Zhang
UMBC

***Abstract*—Asynchronous BFT protocols such as HoneyBadgerBFT and BEAT are inherently robust against timing, performance, and denial-of-service attacks. The protocols, however, achieve static security, where the adversary needs to choose the set of corrupted replicas before the execution of the protocol. The situation is in contrast to that of most of existing BFT protocols (e.g., PBFT) which achieve adaptive security, where the adversary can choose to corrupt replicas at any moment during the execution of the protocol.**

We present EPIC, a novel and efficient asynchronous BFT protocol with adaptive security. Via a five-continent deployment on Amazon EC2, we show that EPIC is slightly slower for small and medium-sized networks than the most efficient asynchronous BFT protocols with static security. We also find as the number of replicas is smaller than 46, EPIC's throughput is stable, achieving peak throughput of 8,000–12,500 tx/sec using t2.medium VMs. When the network size grows larger, EPIC is not as efficient as those with static security, with throughput of 4,000–6,300 tx/sec.

***Keywords*-Byzantine fault tolerance, asynchronous BFT, adaptive security, adaptively secure BFT, threshold cryptography**

I. INTRODUCTION

State machine replication (SMR) [52, 69] is a proven software technique to enable fault-tolerant and highly available services in critical distributed systems (e.g., Google's Spanner [17], Apache ZooKeeper [43]).

Byzantine fault-tolerant (BFT) SMR is the only known software solution for masking arbitrary failures and malicious attacks. BFT has been regarded as the model for building *permissioned blockchains*, where the distributed ledgers (i.e., replicas) know each other's identities but may not trust one another. BFT can be also used to improve the performance and deal with the lack of finality for *permissionless blockchains*, where enrollment is open to anyone and nodes may join and leave dynamically. These permissionless blockchains using BFT are also known as *hybrid blockchains* (e.g., [3, 30, 34, 47, 48, 57, 65, 77]).

BFT protocols can be roughly divided into three categories according to their timing assumptions: *asynchronous*, *synchronous*, or *partially synchronous* [33]. In asynchronous BFT, neither safety nor liveness relies on timing assumptions. In synchronous BFT systems, both safety and liveness may be violated if the synchrony assumption fails to hold. Partially synchronous BFT systems never violate safety but they achieve liveness when the network behaves synchronously only. (It was demonstrated in [60] that PBFT [25] would achieve zero throughput against an adversarial asynchronous network scheduler, echoing the celebrated FLP impossibility result [35].) For this reason, asynchronous BFT protocols are inherently robust against timing, performance, and denial-of-service (DoS) attacks and (arguably) the most appropriate solutions for mission-critical blockchain applications.

Due to its inherent robustness, asynchronous BFT (and atomic broadcast) have been extensively studied [4, 12, 19, 21, 28, 32, 51, 60, 63, 68]. Most notably, two recent asynchronous BFT systems, HoneyBadgerBFT [60] and BEAT [32], have comparable performance as partially synchronous BFT protocols and can scale to 100 replicas. In particular, HoneyBadgerBFT is an efficient asynchronous protocol with a modern implementation and a scalable real-world deployment, while BEAT offers various performance improvements for different application scenarios.

Despite the impressive performance and robustness for HoneyBadgerBFT and BEAT, the protocols have several major issues.

Static vs. Adaptive security. Depending on how the adversary decides to corrupt parties, there are two types of corruptions for BFT protocols:

- *Static corruptions*, where the adversary is restricted to choose its set of corrupted replicas at the start of the protocol and cannot change this set later on.
- *Adaptive corruptions*, where the adversary can choose its set of corrupted replicas at any moment during the execution of the protocol, based on the information it accumulated thus far (i.e., the messages observed and the states of previously corrupted replicas).

There is a strong separation result: statically secure protocols are not necessarily adaptively secure [23, 29]. HoneyBadgerBFT and BEAT (and a prior system SINTRA [21]) defend against static adversary only. The reason is that these protocols heavily rely on efficient but statically secure threshold cryptography. The situation is in contrast to that of partially synchronous BFT protocols, most of which achieve adaptive security [7, 25, 26, 31, 38, 42, 50, 59, 73].

Centralized vs. Distributed key generation. Both HoneyBadgerBFT and BEAT use threshold PRF and threshold encryption and require a trusted dealer to generate the cryptographic keys for individual replicas. The key generation procedure for HoneyBadgerBFT and BEAT is thus centralized. In contrast, most of the efficient BFT protocols rely on authenticated channels or digital signatures and do not need a trusted setup. It is desirable for efficient asynchronous BFT protocols to support a distributed key generation. In fact, we should build asynchronous BFT protocols with distributed key generation that is secure against adaptive corruptions.

Understanding the (exact) performance bottleneck(s) for asynchronous BFT. Asynchronous BFT protocols are complex, consisting of different distributed components and cryptographic building blocks. Both HoneyBadgerBFT and BEAT use reliable broadcast (RBC), asynchronous binary agreement (ABA), and threshold cryptography. Duan et al. showed in BEAT that asynchronous BFT can perform rather differently if using different RBC protocols [32]: Bracha's broadcast [15] results in a latency-optimized asynchronous BFT, AVID broadcast [22] leads to bandwidth-efficient and high-throughput ones, and AVID-FP [41] and its variant can be used to build bandwidth-optimal asynchronous BFT storage. In this paper, we find that not just RBC, the other two building blocks — ABA and threshold cryptography — can impact the performance significantly.

The case of ABA. Both BEAT and HoneyBadgerBFT utilize an ABA protocol proposed by Mostefaoui, Hammouma, and Raynal [64] (MHR ABA). The MHR ABA protocol is known as the most efficient ABA protocol that terminates in two rounds in expectation (completing within $\mathcal{O}(r)$ rounds with probability $1-2^{-r}$). In each round, the MHR protocol has two or three steps. (In contrast, the entire PBFT protocol has three steps only.)

The situation is exacerbated by a liveness issue recently reported [1]. Specifically, the MHR protocol assumes perfect random coins completely independent of the state of all correct nodes when they query the coin. The property is not guaranteed by any existing cryptographic common coin protocols. A malicious network scheduler can keep correct nodes entering the next round with inconsistent values, causing the protocol not to terminate. The best known solution to date is to use Cobalt ABA [58] which has one additional step for *each* round. The added cost, by percentage, could be significant, considering the MHR ABA terminates in two rounds in expectation and in each round there are only two or three steps.

While theoretically the resulting asynchronous BFT has much more rounds than PBFT, it was demonstrated that some ABA protocols may terminate faster than expected [62, 63]. The experimental result, however, was reported for ABA protocols terminating in an expected exponential number of rounds and for settings where the total number of replicas is at most ten. It is still unclear how the ABA protocols of our interests (terminating in an expected constant rounds) perform. It is interesting to evaluate the performance of asynchronous BFT using these ABA protocols for settings with more replicas.

The case of cryptography. It is well known that cryptography can be vital to the performance of BFT protocols. It was originally believed that signature-free BFT protocols such as PBFT are (much) more efficient than BFT protocols using signatures. It was later reported (e.g., BFT-SMaRt [73]) that with modern infrastructures, BFT protocols using signatures can be comparable to those without signatures. Some recent protocols such as HotStuff [76] and SBFT [37] using more expensive threshold signatures were also shown to be both efficient and scalable.

The situation for asynchronous BFT is arguably more complicated. First, HoneyBadgerBFT and BEAT use two (instead of one) threshold cryptographic primitives — threshold encryption and threshold PRF (pseudorandom function). Both protocols use them extensively. Second, the way of using threshold cryptography for them is quite different from that of partially synchronous BFT protocols. The threshold cryptographic operations in these asynchronous BFT protocols are evenly distributed among n replicas. The two features make it difficult to predict the exact bottlenecks for asynchronous BFT. In particular, if one aims to build a BFT protocol by modifying two primitives, one would have to implement and evaluate multiple protocols.

Besides, most known adaptively secure cryptographic schemes are known to be (much) more expensive than their statically secure ones [53–55]. It is unclear if asynchronous BFT protocols using adaptively secure schemes can be practical.

For both ABA and cryptography cases, we aim to understand the exact performance bottlenecks by mixing and matching different building blocks.

A. Our Contribution

We propose EPIC, an efficient asynchronous BFT protocol with adaptive security. We summarize our contributions as follows:

- We first characterize efficient BFT protocols using corruption models (adaptive vs. static corruptions).
- EPIC takes a new approach to adaptively secure asynchronous BFT. First, EPIC uses the LM-LJY adaptively secure threshold PRF scheme [53, 55] for common coins. Second, EPIC uses the Cobalt ABA protocol [58] which resolves the liveness issue of HoneyBadgerBFT and BEAT. Last, EPIC uses a hybrid method of the random transaction selection (as used in HoneyBadgerBFT and BEAT) and the FIFO transaction selection (as used in CKPS [19] and SINTRA [21]), eliminating the usage of expensive threshold encryption.
- EPIC instantiates and implements the distributed key generation protocol [53] which is also secure against adaptive corruptions. In comparison, HoneyBadgerBFT and BEAT do not have distributed key generation protocols, and most of the multi-party computation protocols simply do not instantiate ideal broadcast channels.

- As our new protocol modifies almost all building blocks for asynchronous BFT (including ABA, threshold PRF, and threshold encryption) but RBC, evaluating which component dominates the performance bottleneck is a difficult task. We therefore mix and match different building blocks to implement four asynchronous BFT protocols and evaluate their performance difference. Besides, to understand the cryptographic overhead, we implemented our baseline protocol with static security and EPIC with adaptive security. Our approach complements BEAT which essentially evaluated the performance difference using different RBC protocols.
- We show that EPIC is slightly slower than asynchronous BFT protocols with static security if the network size is small; however, if the network size grows larger, EPIC is not as efficient as those with static security. Besides, EPIC achieves its peak throughput when the network size is small, but even with 31 replicas, EPIC can still achieve throughput of 10,000 tx/sec for transactions of size 250 bytes.

II. Related Work

BFT assuming partial synchrony. Efficient partially synchronous BFT has been extensively studied [7, 9, 26, 31, 37, 38, 42, 50, 59, 73, 76]. Even for partially synchronous BFT protocols focusing on robustness [7, 26], their performance can drop 78%-99% in the presence of Byzantine replicas and/or clients [8]. It is demonstrated that PBFT would achieve zero throughput against an adversarial asynchronous scheduler [60].

Asynchronous binary agreement (ABA). ABA was introduced independently by Ben-Or [11] and Rabin [67]. ABA is a fundamental primitive to build most complex distributed system protocols [12, 18, 19, 21, 28, 61, 63]. For this reason, a significant number of ABA protocols have been proposed [13, 16, 20, 24, 36, 58, 64, 67, 72, 74, 75, 78]. Cachin, Kursawe, and Shoup (CKS) [20] proposed an efficient ABA which achieves optimal resilience and runs in $O(n^2)$ message complexity. The CKS ABA heavily uses RSA-based dual-threshold signatures [70] which are computationally expensive. Mostefaoui, Hamouma, and Raynal (MHR) [64] presented the first signature-free ABA that has the same message complexity as the CKS ABA [20]. The MHR ABA is used in HoneyBadgerBFT and BEAT. It is reported that the MHR ABA, however, has a liveness issue, if being instantiated using any existing cryptographic coin flipping protocols [1].

Asynchronous atomic broadcast and BFT. In an atomic broadcast, a broadcaster (one of the replicas) broadcasts messages to all replicas, and all replicas should deliver messages in the same order. Instead, BFT state machine replication specifies clients and replicas, and all replicas delivers client messsages in the same order. We do not distinguish Byzantine atomic broadcast and BFT and collectively call them BFT.

Asynchronous BFT protocols, such as SINTRA, HoneyBadgerBFT, and BEAT, follow the asynchronous common subset (ACS) framework [12, 19] which can be realized using RBC and ABA. The underlying ABA protocols are efficient, terminating in an expected constant number of rounds.

RITAS is a stack of randomized distributed protocols defending against Byzantine failures [63], RITAS consists of an efficient atomic broadcast implementation of Correia, Neves, and Verissimo [27]. While the protocol theoretically terminates in an expected exponential number of rounds, it was demonstrated that the protocol in practice may execute in only a few rounds for certain conditions. The RITAS protocol is shown to be efficient for a cluster of ten replicas.

Recently, Abraham, Malkhi, and Spiegelman provided an asynchronous BFT protocol that reduces message complexity to $O(n^2)$ [5], utilizing a similar workflow used in HotStuff [76]. Besides its higher latency, the protocol has theoretically lower throughput than HoneyBadgerBFT and BEAT which can select random transactions for high throughout.

Asynchronous hybrid BFT protocols. KS [51] and RC [68] are asynchronous hybrid BFT protocols guaranteeing both safety and liveness under asynchronous environments. Both protocols have an optimistic BFT protocol under "normal" circumstances (where there is no failure or the primary is correct) and a pessimistic BFT protocol under "rare" circumstances (e.g., asynchrony). KS [51] and RC [68] use PBFT-like protocols during normal operations and randomized asynchronous BFT for recovery in case of failures or asynchrony. KS proceeds in epochs and uses Bracha's broadcast [15] during the normal-operation phase, just like in PBFT. It suggests using randomized Byzantine agreement for backup and delivers some requests for liveness. It has the same efficiency as PBFT during gracious execution. RC replaces the reliable broadcast primitive in KS using consistent broadcast, a weaker primitive. RC is the first BFT protocol (atomic broadcast) with the message complexity only $O(n)$ in its normal case, while all other BFT (atomic broadcast) protocols have the message complexity $O(n^2)$. The improvement comes at the cost of more expensive recovery phase using heavy public-key cryptography. There is no implementation for either KS or RC.

Asynchronous MPC. Lu et al. [56] recently provided the first robust asynchronous multi-party computation system with guaranteed output delivery using HoneyBadgerBFT. The protocol achieves static security.

III. System and Threat Model

BFT. We consider a Byzantine fault-tolerant state machine replication (BFT) protocol, where f out of n replicas can fail arbitrarily (Byzantine failures) and a computationally bounded, adaptive adversary can coordinate faulty replicas.

The adaptive adversary can choose its set of dishonest parties at any moment during the execution of the protocol, based on the messages transmitted and the internal states of previously corrupted replicas. The BFT protocols considered in this paper tolerate $f \leq \lfloor \frac{n-1}{3} \rfloor$ Byzantine failures, which is optimal.

A replica *delivers operations*, each *submitted* by some client. The client should be able to compute a final response to its submitted operation from the responses it receives from replicas. We use operations, (client) requests, and transactions (blockchain terminology) interchangeably. Correctness of a BFT protocol is specified as follows.

- **Agreement**: If any correct replica delivers an operation m, then every correct replica delivers m.
- **Total order**: If a correct replica has delivered an operation m with a sequence number, and another correct replica has delivered an operation m' with the same sequence number, then $m = m'$.
- **Liveness**: If an operation m is submitted to $n-f$ correct replicas, then all correct replicas will eventually deliver m.

The liveness property has been referred to by other names (e.g., "fairness" [19],"censorship resilience" [60]).

Timing assumption. We can roughly divide BFT protocols into three categories according to their timing assumptions: *asynchronous*, *synchronous*, or *partially synchronous* [33]. An asynchronous BFT system makes no timing assumptions on message processing or transmission delays. If there is a known bound on message processing delays and transmission delays, then the corresponding BFT system is synchronous. Partially synchronous BFT lies in-between: messages are guaranteed to be delivered within a time bound, but the bound may be unknown to participants of the system or system designers.

Asynchronous BFT protocols are inherently more robust than other BFT protocols. Due to the celebrated FLP impossibility result [35] which rules out that deterministic protocols reach consensus in fully asynchronous environments, asynchronous BFT protocols must rely on randomization and be probabilistically live. This paper considers purely asynchronous systems making no timing assumptions on message processing or transmission delays. We assume synchrony for the distributed key setup phase, which is a one-time event. We will discuss the implication of the system choice.

IV. Primitives and Building Blocks

This section reviews the cryptographic and distributed systems building blocks for EPIC.

Threshold pseudorandom function (PRF). We describe threshold PRF with a decentralized key generation (e.g., [53]). A (t, n) threshold PRF scheme for a function F consists of the following algorithms (FGen, Eva, Vrf, FCom).

- An interactive key generation algorithm FGen involves n players $p_1, \cdots, p_n$. Each player p_i takes as input common public parameters, a security parameter l, the number n of total servers, and threshold parameter t. The output of the protocol is (pk, vk, sk), where pk is the public key, vk is the verification key, and $sk = (sk_1, \cdots, sk_n)$ is a list of private keys. Both pk and vk are known to anyone, and p_i only obtains sk_i.
- A PRF share evaluation algorithm Eva takes a public key pk, a PRF input m, and a private key sk_i, and outputs a PRF share σ_i.
- A share verification algorithm Vrf takes as input the verification key *vk*, a PRF input m, and a PRF share σ_i, and outputs a single bit.
- A combining algorithm FCom takes as input the verification key *vk*, a PRF input m, and a set of t valid PRF shares, and outputs a PRF value σ.

We require the threshold PRF value to be unpredictable against an adversary that controls up to $t-1$ servers. We also rely on an additional uniqueness property, which guarantees that for a given public key *pk*, there exists exactly one valid signature on each message m. One can consider both static and adaptive adversaries just as in BFT protocols. In the adaptive corruption model, the adversary can corrupt players and query signing oracles at any moment of the protocol, based on the information collected so far.

Byzantine reliable broadcast (RBC). In RBC, a sender (one of the replicas) sends a message to all other replicas. An asynchronous RBC protocol [52] satisfies the following properties:

- **Agreement**: If two correct replicas deliver two messages m and m' then $m = m'$.
- **Totality**: If some correct replica delivers a message m, all correct replicas deliver m.
- **Validity**: If a correct sender broadcasts a message m, all correct replicas deliver m.
- **Integrity**: Every correct replica delivers a message m from sender p at most once. If p is correct, then m was previously broadcast by p.

Bracha's broadcast [15] is a well-known implementation of RBC. To broadcast a message m, its communication complexity is $\mathcal{O}(n^2|m|)$. Cachin and Tessaro [22] proposed an erasure-coded RBC (AVID broadcast) reducing the bandwidth to $\mathcal{O}(n|m|)$. EPIC is compatible with any RBC and implements AVID broadcast as in HoneyBadgerBFT and BEAT.

Asynchronous binary agreement (ABA). In an ABA protocol, each replica has a binary value as an initial input v_{input} (also known as a *vote*). ABA allows replicas to agree on the value of a single bit and deliver the value. ABA should satisfy the following properties:

- **Validity**: If all correct replicas have the same input value v, correct replicas will deliver v.

- **Agreement**: If a correct replica delivers v and another correct replica delivers v', then $v = v'$.
- **Termination**: All correct replicas eventually deliver a binary value with probability 1.

An ABA protocol proceeds in rounds, where for a round r, a replica has an input est_r. ABA protocols considered in the paper have an expected constant number of rounds. In each round, there are a small number of steps (two to four steps for the ABA protocols we consider), and replicas query the common coin (realized using threshold cryptography) and decide to either terminate the protocol or propose some values for the next round.

V. Problems and Technical Overview

A. Review of Efficient Asynchronous BFT Protocols

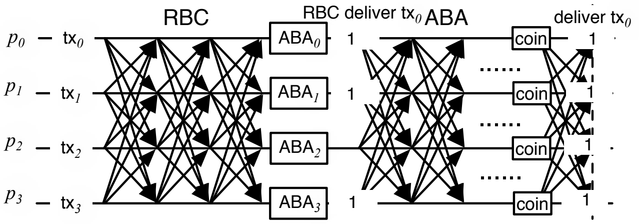

Figure 1. The ACS consensus workflow.

We consider the ACS (asynchronous common subset) framework for asynchronous BFT. In the framework, servers propose a subset of transactions in their transaction pool and deliver the union of the transactions in the agreed-upon vector. The framework was used by Ben-Or et al. [12], CKPS [19], SINTRA [21], HoneyBadgerBFT [60], and BEAT [32].

Figure 1 reviews the framework for HoneyBadgerBFT and BEAT (with threshold encryption ignored). The protocols proceed in epochs. Each epoch consists of two phases: an RBC phase and an ABA phase. In the RBC phase, each replica proposes a subset of transactions from its transaction pool (a proposal) and uses RBC to broadcast the transactions. In the ABA phase, replicas run n parallel ABA instances, the i-th of which is used to agree on whether the proposal from replica p_i has been RBC-delivered.

In Figure 1, replicas p_i ($i \in [0..3]$) propose transactions tx_i, respectively. When a replica delivers a value from an RBC instance $j \in [0..3]$, it inputs 1 to the j-th ABA instance. HoneyBadgerBFT and BEAT follow Ben-Or et al. [12] and ask each replica to abstain from proposing 0 until $n - f$ ABA instances have been delivered by the replica, which guarantees system throughput.

Also for high throughput, HoneyBadgerBFT ensures that each replica proposes mostly disjoint sets of transactions. Thus, replicas in HoneyBadgerBFT propose randomly selected transactions. To prevent an adversary from censoring a particular transaction, HoneyBadgerBFT requires replicas to use threshold encryption to encrypt transactions. After delivering transactions in ciphertext, replicas collectively decrypt them.

HoneyBadgerBFT uses the bandwidth-efficient AVID broadcast for RBC [22] and the MHR protocol for ABA [64]. BEAT is a family of five asynchronous BFT protocols, providing a series of improvements to HoneyBadgerBFT. In particular, the baseline protocol in BEAT (hereinafter BEAT) eliminates the usage of pairing-based cryptography and leverages more efficient threshold cryptography using elliptic curves. Other protocols in BEAT mainly explore the performance impact using different RBC or information dispersal protocols.

B. Characterizing BFT Using Corruption Models

We characterize existing BFT protocols using corruption models. For static security, the adversary needs to decide which replicas to corrupt before the execution of the system, whereas for adaptive security, the adversary can adaptively choose which replicas to corrupt, based on information the adversary has accumulated thus far.

Common-coin asynchronous BFT. Efficient asynchronous BFT systems, such as SINTRA, HoneyBadgerBFT, and BEAT, use the ACS framework and rely on cryptographic common coins.

SINTRA uses the RSA-based dual threshold signature scheme of Shoup [70] and the Diffie-Hellman problem based threshold PRF scheme of Cachin, Kursawe, and Shoup [20] implemented using finite fields. HoneyBadgerBFT uses the pairing-based threshold encryption of Baek and Zheng [44] and the pairing-based threshold signature of Boldyreva [14]. BEAT uses the threshold encryption scheme of Shoup and Gennaro [71] and the threshold PRF scheme of Cachin, Kurasawe, and Shoup, both of which are implemented using elliptic curves. All of the above threshold cryptographic schemes are proven secure against static corruptions only. Therefore, the corresponding asynchronous BFT systems achieve static security.

Local-coin asynchronous BFT. RITAS is a stack of randomized distributed protocols defending against Byzantine failures [63] in the adaptive corruption model. It consists of an efficient atomic broadcast protocol of Correia, Neves, and Verissimo (CNV) [27]. Instead of using common coins, the CNV protocol relies on local coins. While the protocol terminates in an expected exponential number of rounds, it was demonstrated that the protocol in practice may execute in only a few rounds for certain conditions. In a different scenario, the classic Bracha's ABA protocol [15] was shown to have better performance than the ABA protocol of Cachin, Kursawe, and Shoup [20] in the LAN environment where the total number of replicas is at most ten [62].

The CNV atomic broadcast protocol is much more bandwidth and round expensive than both HoneyBadgerBFT and BEAT. This makes it theoretically less efficient in a scalable WAN environment. No known experimentation was conducted for the CNV protocol with more than ten replicas. Moreover, as the CNV protocol terminates in an expected exponential number of rounds, its performance with an asynchronous network scheduler would be considerably worse than HoneyBadgerBFT and BEAT.

Partially synchronous BFT. Most of the existing BFT protocols in partially synchronous environments [7, 26, 38, 42, 50, 59, 73] achieve adaptive security. Protocols such as SBFT [37] and HotStuff [76] rely on statically secure threshold signatures and achieve static security only.

Committee-based (BFT) protocols. Some scalable hybrid blockchain protocol [66] or Byzantine agreement protocol [46] does not use threshold cryptography but involves the selection of a small committee among all replicas for consensus, which makes adaptive security a non-trivial task for them. This is not our concern, as we study conventional BFT protocols where all replicas, not just a fraction of them, need to participate in the consensus process.

C. Achieving Adaptive Security for EPIC

EPIC follows HoneyBadgerBFT and BEAT and uses a novel combination of new primitives to achieve adaptive security.

As we mentioned above, both HoneyBadgerBFT and BEAT use threshold common coin and threshold encryption schemes which are statically secure. Intuitively, to achieve adaptive security, one would have to replace both statically secure primitives using adaptively secure ones.

First, we adopted and implemented the adaptively secure threshold PRF scheme of Loss and Moran [55] which is built from the adaptively secure threshold signature scheme of Libert, Joye, and Yung [53].[1] The adaptively secure threshold PRF scheme (hereinafter the LM-LJY threshold PRF) requires four pairing computation for signature verification, twice more expensive than the threshold PRF scheme in HoneyBadgerBFT which requires two pairing computation. The LM-LJY scheme is much more expensive than the pairing-free threshold PRF scheme in BEAT. It is natural to explore the performance penalty of using adaptively secure threshold PRF protocol.

To handle threshold encryption, one could use an adaptive secure one as well. To our knowledge, the scheme of Libert and Yung [54] is the most efficient threshold encryption scheme. The scheme, however, relies on a bilinear group of composite order, which is much less efficient than a regular, prime-order bilinear group [39]. We take a different approach without using threshold encryption. In our approach, replicas maintain a transaction buffer. Replicas select a random subset of T transactions in plaintext for most epochs. They periodically switch to select the first T transactions in their buffer. Doing so can keep the efficiency as HoneyBadgerBFT and BEAT, while ensuring any transaction cannot be censored for too long.

On the one hand, EPIC eliminates the usage of any threshold encryption scheme, thereby potentially improving the performance of asynchronous BFT. On the other hand, the periodical switch for selecting transactions in a FIFO manner may reduce performance. Therefore, we must experimentally verify which of the above two factors will dominate the performance overhead.

Loss and Moran [55] claimed that they obtained the first adaptively secure ABA protocol running in $O(n^2)$ communication complexity by using the LM-LJY threshold PRF to obtain common coins for the MHR ABA. The scheme, unfortunately, has the same liveness issue as reported [1]. In contrast, EPIC combines the LM-LJY threshold PRF and the Cobalt ABA to obtain an adaptively secure ABA protocol running in $O(n^2)$ communication complexity.

As the LM-LJY threshold PRF scheme is the only threshold cryptographic scheme used in EPIC, we just need to build a decentralized key generation protocol for the LM-LJY threshold PRF scheme. A decentralized key generation protocol has already been described in the same paper by Libert, Joye, and Yung [53]. The key generation algorithm assumes a broadcast channel and private and authenticated pairwise channels. Most cryptographic and multi-party computation systems assuming broadcast channels simply use best-effort broadcast (see [40] and references therein) and therefore do not provide the fault tolerance needed. We provide a concrete instantiation using Bracha's broadcast [16] and an implementation for the protocol. Our key generation process works in synchronous environment, tolerating up to $n/2$ Byzantine faulty replicas.

VI. EPIC

In this section, we describe the design of EPIC. EPIC follows the ACS framework and has an RBC phase and an ABA phase. Different from HoneyBadgerBFT and BEAT, EPIC achieves adaptive security and decentralized key generation. Figure 2 describes the EPIC protocol using RBC and ABA in a black-box manner.

We begin with a high-level overview. The protocol proceeds in epochs numbered by s (initialized as 0). In each epoch, replicas select a subset of transactions as a proposal from their transaction buffer and agree on a set of transactions containing the union of the proposals of at least $n-f$ replicas. Let B be the batch size of the transactions for an epoch. In an epoch, each replica proposes transactions of size $b = \lceil B/n \rceil$ (the batch size for a replica). In the RBC phase, replicas use RBC to broadcast the proposals. In the ABA phase, n parallel ABA instances are run. The i-th ABA instance is used to agree on whether the transactions from replica p_i have been delivered in the RBC phase. If a correct replica p_j terminates the i-th ABA instance

[1] Specifically, Loss and Moran [55] proved that the signature scheme of Libert, Joye, and Yung satisfies the uniqueness property. It is therefore trivial to derive a threshold PRF scheme in the random oracle model.

```
Initialization
  buf ← $\emptyset$                                   {transaction buffer}
  $B$                                                 {batch size}
  $\mu$, $\delta$                   {parameters for transaction selection}
  $s \leftarrow 0$                                     {epoch number}
  $i$                                                   {replica id}
  $\{\text{RBC}_j\}_n$   {$n$ RBC instances where $j$ is the sender of $\text{RBC}_j$}
  $\{\text{ABA}_j\}_n$                                {$n$ ABA instances}
epoch s
  if $s = 0, \cdots, \mu - 1 \bmod (\mu + \delta)$
    let $value$ be a random selection $\lceil B/n \rceil$ of transactions for the first $B$ elements in buf
  else let $value$ be the first $\lceil B/n \rceil$ transactions in buf
  input $value$ to $\text{RBC}_i$
  upon delivery of $value_j$ from $\text{RBC}_j$
    if $\text{ABA}_j$ has not yet been provided input, input 1 to $\text{ABA}_j$
  upon delivery of 1 from $\text{ABA}_j$ and $value_j$ from $\text{RBC}_j$
    $output \leftarrow output \cup value_j$
  upon delivery of 1 from at least $n - f$ ABA instances
    for each $\text{ABA}_j$ instance that has not been provided input
      input 0 to $\text{ABA}_j$
  upon termination of all the $n$ ABA instances
    deliver $output$
  $s \leftarrow s + 1$
```

Figure 2. EPIC algorithm for p_i.

with 1, the transactions from p_i are delivered. Otherwise, the transactions will not be included. We follow Ben-Or et al. [12] (and HoneyBadgerBFT and BEAT), ensuring at least $n - f$ ABA instances terminate with 1, and thus the union of the transactions from at least $n-f$ replicas are delivered. To this goal, every replica abstains from proposing 0 until $n - f$ ABA instances have been delivered by the replica. Each ABA instance terminates with probability 1/2 for each round. As EPIC must wait for all ABA instances to finish, the running time of EPIC is $O(\log N)$ in expectation.

As in HoneyBadgerBFT and most BEAT instances, EPIC uses an adaptively secure RBC — AVID broadcast [22]. In the following, we specify other building blocks for EPIC:

- the transaction selection approach in EPIC,
- the decentralized key distributed algorithm for EPIC,
- EPIC's ABA protocol—the Cobalt ABA, and
- the adaptively secure common coin protocol from the LM-LJY threshold PRF scheme.

Transaction selection for EPIC. EPIC uses a novel transaction selection approach which is distinguished from prior asynchronous BFT protocols such as SINTRA, HoneyBadgerBFT, and BEAT.

In SINTRA, replicas maintain a log of transactions according to the order they are received and replicas select as input the first subset of transactions in the transaction buffer. We call the approach FIFO selection. The approach can be easily shown to achieve liveness but lead to low system throughput. This is because the transactions delivered are unions of the transactions selected by replicas, and replicas tend to select the same transactions in each epoch.

In HoneyBadgerBFT (and BEAT), replicas propose randomly selected sets of transactions to improve throughput. Doing so directly causes a liveness issue, as a network adversary can censor certain transactions so that they will not be delivered. Thus, HoneyBadgerBFT (and BEAT) choose to use threshold encryption to avoid censorship. In their approach, replicas first encrypt the proposals and then decrypt them collectively when transactions in ciphertext are delivered. We call this approach ETD (standing for "encrypt-then-decrypt"). HoneyBadgerBFT uses the pairing-based threshold encryption scheme of Baek and Zheng, while BEAT uses the threshold encryption scheme of Shoup and Gennaro [71]. Both schemes are efficient but statically secure only.

In EPIC, we take a different approach without using threshold encryption. We ask replicas to select random transactions *in plaintext* for most epochs (e.g., $\frac{4}{5}$ of the total epochs) and periodically switch to the FIFO selection. The strategy is performed deterministically for all replicas.

More formally, let s be the epoch number initially numbered 0. Let μ and δ be two system parameters determining how often the protocol switches between the two modes. In EPIC, replicas can first perform random selection for μ epochs and then the FIFO selection for δ epochs. Our approach can be generalized to many other scenarios, as long as the switching strategy is deterministic (and known to all replicas), and the system can perform the FIFO selection for a non-negligible fraction of epochs (to ensure liveness).

Our transaction selection approach can keep the efficiency of HoneyBadgerBFT and BEAT, while avoiding censorship. Since our approach can be somewhat viewed as a hybrid of HoneyBadgerBFT and SINTRA, we call it HYB (standing for "hybrid").

The HYB approach eliminates the usage of (expensive) threshold encryption scheme, which would improve efficiency. The periodical transaction switch to the FIFO selection may reduce performance. Besides, the approach provides trade-offs between latency and throughput. Roughly, if μ is reasonably larger than δ, the system favors throughput over latency, and otherwise the opposite is the case.

The Cobalt ABA. HoneyBadgerBFT and BEAT use the MHR protocol in the ABA phase. It was reported that the MHR protocol is not live if being instantiated using any existing cryptographic common coin protocols [1]. (Essentially, the MHR protocol makes a strong common coin assumption which existing cryptographic common coin protocols fail to satisfy.) Besides, the MHR protocol in HoneyBadgerBFT and BEAT achieve static security due to the use of statically secure common coin protocols.

EPIC thus enhances the ABA choice in HoneyBadgerBFT and BEAT in two aspects. First, we use the Cobalt ABA protocol [58] instead of the MHR ABA to resolve the liveness issue. Second, we instantiate the Cobalt ABA protocol using

```
Initialization
  $r \leftarrow 0$                                        {round}
  $est_0 \leftarrow v_{input}$        {set input for round 0 to initial input}
round $r$
  broadcast bval($est_r$)                       {broadcast input}
  upon receiving bval($v$) from $f+1$ replicas
    if bval($v$) has not been sent, broadcast bval($v$)
  upon receiving bval($v$) from $2f+1$ nodes
    $bin\_values \leftarrow bin\_values \cup \{v\}$
  wait until $bin\_values \neq \emptyset$            {move to the second step}
    broadcast aux($v$) where $v \in bin\_values$
  upon receiving $n-f$ aux() such that the set of values $vals$ the
messages is a subset of $bin\_values$
      broadcast conf$_r$($vals$)                {move to the third step}
    upon receiving $n-f$ conf$_r$() such that the set of values $vals$
  is a subset of $bin\_values$
    $c \leftarrow Coin()$                              {obtain common coin}
    if $vals = \{\rho\}$
      $est_{r+1} \leftarrow \rho$
      if $\rho = c$, deliver $b$                   {terminate the protocol}
    else $est_{r+1} \leftarrow c$                   {enter the next round}
  $r \leftarrow r+1$
```

Figure 3. The MHR and Cobalt ABA protocol. The Cobalt ABA includes the boxed code, while the MHR ABA does not.

the adaptively secure LM-LJY threshold PRF scheme.

The MHR and Cobalt ABA protocols are illustrated in Figure 3, where the Cobalt ABA includes the boxed code but the MHR ABA does not include. Specifically, the MHR protocol has two to three steps in each round. In the first step, all replicas broadcast their input. If a replica receives $f+1$ matching input value that is different from its input, it triggers the second step by broadcasting the value. In the last step, if a replica receives $2f+1$ matching value v, it broadcast an aux(v). Next, if a replica receives $2f+1$ aux() messages, it either uses the only available binary value from the aux() messages or the common coin value to enter the next round. The livenss issue for MHR protocol is due to the usage of the cryptographic common coin. The adversary (and network scheduler) can learn the value of the common coin and *manipulate* the sequence of messages received by other replicas to make the protocol never terminate. To solve the issue, the Cobalt ABA protocol introduces one more step in each round. In the Cobalt ABA protocol, each replica needs to additionally broadcast the values received in the aux() step. We use the Cobalt ABA to obtain an adaptively secure ABA protocol running in $O(n^2)$ communication complexity.

The additional step in the Cobalt ABA protocol can be significant, as the Cobalt ABA only has three or four steps in each round and it may run in several rounds (two on average). On the other hand, it was shown that some ABA protocols may terminate faster than expected (in a small-scale setting) [62, 63]. It is natural to ask what the performance penalty of both BEAT and EPIC using the Cobalt ABA would be.

Distributed key generation. EPIC uses one threshold cryptographic primitive only, the LM-LJY adaptively secure threshold PRF. The threshold PRF scheme is the adaptively secure threshold PRF scheme of Loss and Moran [55] which is built from the adaptively secure threshold signature scheme of Libert, Joye, and Yung [53]. If the key generation for the threshold PRF is decentralized, so is EPIC.

In fact, a decentralized key generation protocol has been described in the same paper by Libert, Joye, and Yung [53], but no implementation is provided. The key generation algorithm assumes a broadcast channel and private and authenticated pairwise channels.

While (most of) existing distributed cryptographic and multi-party computation systems [40] relying on broadcast channels simply use best-effort broadcast, we do provide a concrete instantiation using Bracha's broadcast [16]. Bracha's broadcast has three steps, achieving adaptive security in asynchronous environments. It is straightforward to build a synchronous version for it. We describe the EPIC distributed key generation protocol in Figure 4.

Though a distributed key generation protocol should ideally work in asynchronous environments, our protocol works in synchronous environments only. We made the system decision, in part because the protocol is simpler to implement than existing asynchronous protocols [45, 49]. Another reason is that the key generation procedure is a one-time event, and replicas can set an adequately large timeout value to ensure safety. Note that key generation protocol can tolerate up to $\frac{n}{2}$ Byzantine failures, while an asynchronous key generation protocol only tolerates up to $\frac{n}{3}$ Byzantine failures.

Common coin protocol. For the adaptively secure common coin protocol, we use the LM-LJY threshold PRF scheme of Loss and Moran [55] based on the adaptively secure threshold signature scheme of Libert, Joye, and Yung [53] (Figure 5). The threshold PRF scheme requires four pairing computation for signature verification. It is thus twice more expensive than the threshold PRF scheme in HoneyBadgerBFT and much more expensive than the pairing-free threshold PRF scheme in BEAT. The threshold LM-LJY PRF scheme provides adaptive security under the Symmetric eXternal Diffie-Hellman (SXDH) assumption.

Finally, we illustrate in Figure 6 the common coin protocol based on the LM-LJY $(f+1, n)$ threshold PRF scheme $(\mathsf{Eva}, \mathsf{Vrf}, \mathsf{FCom})$.

VII. Implementation

The entire EPIC library includes 13,000 lines of Python code, among which 900 lines of code are written for key distribution and 1,200 lines of code are used for evaluation. In total, we implemented four asynchronous BFT protocols summarized in Table I. For both EPIC and EPIC-MHR, we use the BN256 pairing curve to understand the performance overhead incurred by threshold cryptography.

Common public parameter setup: Let $\mathcal{BG} = (q, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ be an asymmetric bilinear group, where $\mathbb{G}$, $\hat{\mathbb{G}}$, and $\mathbb{G}_T$ are cyclic groups of prime order q, g and $\hat{g}$ are generators for $\hat{\mathbb{G}}$, and $e\colon \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ is an efficiently computable bilinear map.

- p_i chooses random polynomials for $k \in \{1, 2\}$: $A_{ik}[X] = a_{ik0} + a_{ik1}X + \cdots + a_{ikf}X^f$ and $B_{ik}[X] = b_{ik0} + b_{ik1}X + \cdots + b_{ikf}X^f$ of degree f. p_i runs RBC to broadcast $C_{ikd} = g^{a_{ikd}}\hat{g}^{b_{ikd}}$ for $d \in [0..f]$. p_i sends $\{A_{ik}(j), B_{ik}(j)\}_{k=1}^2$ to p_j for $j \in [1..n]$.
- p_i sends a complaint against p_j for any of the conditions:
  - p_i received $\{A_{jk}(i), B_{jk}(i)\}_{k=1}^2$ from p_j and checks
    $$g^{A_{jk}(i)}\hat{g}^{B_{ik}(i)} = \prod_{d=0}^{f} C_{jkl}^{i^d} \qquad \text{for} \quad k = 1, 2 \qquad (1),$$
    but these equalities do not both hold,
  - p_i did not receive values from p_j, or
  - p_i received more than one set of values.
- Upon receiving a complaint from p_j, p_i runs RBC to broadcast $\{A_{ik}(j), B_{ik}(j)\}_{k=1}^2$ that satisfy (1).
- p_i marks p_j as disqualified if
  - p_i received more than f complaints against p_j, or
  - p_j answered a complaint with values that falsify (1).

p_i then builds the set of non-disqualified replicas Q.

- p_i computes $(pk, vk_1, \cdots, vk_n, sk_i)$ as follows:
  - $$pk \leftarrow \prod_{i \in Q} C_{ik0},$$
  - $$vk_j \leftarrow \left(\prod_{u \in Q} \prod_{d=0}^{t} C_{u1l}^{i^d}, \prod_{u \in Q} \prod_{d=0}^{t} C_{u2l}^{i^d} \right) \quad \text{for} \quad j \in [1..n]$$
  - $$sk_i \leftarrow \left\{ \sum_{j \in Q} A_{jk}(i), \sum_{j \in Q} B_{jk}(i) \right\}_{k=1}^{2}$$

Figure 4. The distributed key generation algorithm FGen for $(f+1, n)$ threshold PRF for replica p_i, where the output of p_i is $(pk, vk_1, \cdots, vk_n, sk_i)$. The algorithm is executed only once and tolerates $\frac{n}{2}$ corruptions. The first three steps are interactive, while the last two steps involve local computation only.

Common public parameter setup: The same as Figure 4; besides, define two hash functions $H\colon \{0,1\}^* \to \mathbb{G}^2$, $H'\colon \mathbb{G}^2 \to \{0,1\}$.

```
Eva($pk, m, sk_i$)
  $(h_1, h_2) \leftarrow H(m)$
  parse $sk_i$ as $\{A_k[i], B_k[i]\}_{k=1}^2$
  $z_i \leftarrow \prod_{k=1}^2 h_k^{-A_k[i]}$; $v_i \leftarrow \prod_{k=1}^2 h_k^{-B_k[i]}$
  return $\sigma_i \leftarrow (z_i, v_i)$.
Vrf($vk, m, vk_i, y_i$)
  parse $\sigma_i$ as $(z_i, v_i)$ and $vk_i$ as $(V_{1i}, V_{2i})$
  $(h_1, h_2) \leftarrow H(m)$
  if $e(z_i, g) \cdot e(v_i, \hat{g}) \cdot \prod_{k=1}^2 e(h_1, v_{ki}) = 1$ return 1
  else return 0
FCom($vk, m, \{\sigma_j\}_{j \in S}$)
  upon receiving $f + 1$ valid PRF shares $\{\sigma_j\}_{j \in S}$
    for $j \in S$, parse $\sigma_j$ as $(z_j, v_j)$
    $z \leftarrow \prod_{j \in S} z_i^{\Delta_{j,S}(0)}$; $v \leftarrow \prod_{j \in S} v_i^{\Delta_{j,S}(0)}$
    $\{\Delta_{j,S}$ denotes the Lagrange polynomial for $j \in S\}$
    return $\sigma \leftarrow H'(z, v)$.
```

Figure 5. The LM-LJY $(f+1, n)$ threshold PRF scheme (Eva, Vrf, FCom).

```
upon receiving $Coin$(sid)
  $m \leftarrow$ sid
  $\sigma_i \leftarrow$ Eva($pk, m, sk_i$)
  broadcast $(i, m, \sigma_i)$
upon receiving $f + 1$ valid threshold PRF shares $\{\sigma_k\}_{k \in S}$ on $m$
  return $\sigma \leftarrow$ FCom($vk, m, \{\sigma_k\}_{k \in S}$)
```

Figure 6. The common coin protocol from the LM-LJY $(f+1, n)$ threshold PRF scheme, where sid is a session identifier and consists of an epoch number s, a round number r, and an ABA instance number $j \in [1..n]$.

BEAT is our baseline protocol. We used the BEAT0 protocol which leverages more efficient cryptography than HoneyBadgerBFT. While using BEAT0, we call our baseline protocol BEAT for simplicity.

We use the BN256 curve [10] for the LM-LJY threshold PRF scheme. BN256 and BN254 curves achieve 110-bit security and have been widely used in many other BFT and blockchain systems (e.g., SBFT [37]). We modified the Charm Python library [6] to wrap a version of the relic C++ library [2] and then implemented the LM-LJY threshold PRF scheme using the modified Charm library.

VIII. Evaluation

Overview. We evaluate the protocols on Amazon EC2 using up to 91 virtual machines (VMs) in different regions across five continents. Each VM is the *t2.medium* type with two vCPUs and 4GB memory, running Ubuntu 16.04. We evaluate the protocols in both LAN and WAN settings, where the VMs are launched in the same EC2 region in the LAN setting, and the VMs are evenly distributed in different regions in the WAN setting. We evaluate the protocols using different network sizes and batch sizes.

We evaluate the protocols with different network sizes. We use f to represent the network size, and the total number of replicas is $n = 3f + 1$. Recall B and $b = \lceil B/n \rceil$ are the batch size for the protocol and the batch size for transactions proposed by each replica, respectively. *All transactions are of size 250 bytes.*

When evaluating latency only, we have $b = 1$, where each replica proposes a single transaction. When evaluating throughput, we vary the size of b until the throughput reaches its peak and stabilizes.

The system throughput is evaluated according to the actual delivered transactions using real transaction buffers. In particular, overlapping transactions delivered are counted once. Unless stated otherwise, we let the transaction buffer at each replica be $10 \cdot b$. Our approach is more precise than HoneyBadgerBFT and BEAT. In their approaches, replicas use local random coins to generate independently distributed transaction sets from a large space, and the probability of

Protocol	ABA	Common Coin	Transaction Selection	Liveness	Adaptive Security
BEAT	MHR [64]	CKS [20]	ETD	No	No
BEAT-Cobalt	Cobalt [58]	CKS [20]	ETD	Yes	No
EPIC-MHR	MHR [64]	LM-LJY [53, 55]	HYB	No	Yes
EPIC	Cobalt [58]	LM-LJY [53, 55]	HYB	Yes	Yes

Table I
THE FOUR IMPLEMENTED ASYNCHRONOUS BFT PROTOCOLS.

any two sets being overlapping is negligible. Our approach is needed to understand the exact impact of using various transaction selection approaches.

We run each experiment five times, and each experiment runs 20 epochs. We then calculate the average results. For HYB in EPIC, we let μ be four and δ be one. Namely, we first run four epochs using random selections and then one epoch using the FIFO selection.

Our results show that EPIC achieves adaptive security with low overhead when f is small. In the WAN setting, EPIC has only 2%, 5%, 21% lower throughput than BEAT-Cobalt when $f = 1, 2, 5$, respectively. When f further grows, the performance overhead for EPIC compared with BEAT-Cobalt is significantly higher. When $f = 30$, EPIC achieves 68% lower peak throughput than BEAT-Cobalt.

Besides the conventional metrics (latency, throughput, and scalability), our evaluation also aims to identify the performance bottlenecks using a variety of experiments.

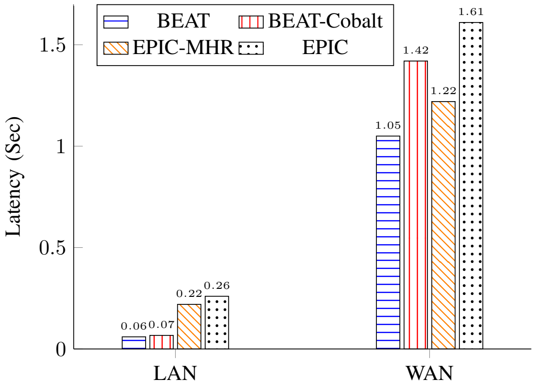

Figure 7. Latency for $f = 1$ in both LAN setting and WAN setting under no contention.

MHR ABA vs. Cobalt ABA. Both HoneyBadgerBFT and BEAT use the MHR ABA. The Cobalt ABA solves the liveness problem at the cost of an additional step in each round. We find that in all of our experiments, BEAT (using MHR) outperforms BEAT-Cobalt and EPIC-MHR outperforms EPIC (using Cobalt) in terms of both latency and throughput. We also find that the performance degradation caused by the extra step in the LAN setting is small but certainly noticeable, while it becomes more visible in the WAN setting. This is expected, as the network latency caused by the extra step has a more significant impact in the WAN setting.

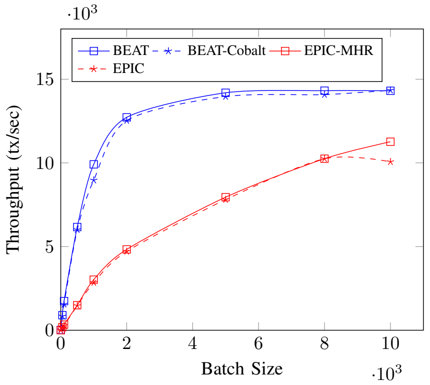

Figure 8. Throughput of BEAT and EPIC for $f = 1$ in the LAN setting.

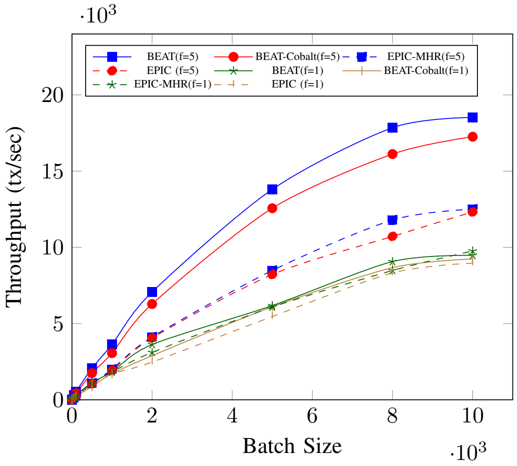

Figure 9. Throughput for $f = 1$ and $f = 5$ in the WAN setting.

Figure 8 and Figure 9 show the throughput of the BFT protocols for $f = 1$ in LAN and WAN environments, respectively. In the LAN setting, BEAT-Cobalt achieves 0.02% lower throughput than BEAT, and EPIC achieves 1% lower throughput than EPIC-MHR. In the WAN setting, BEAT-Cobalt achieves 2% lower throughput than BEAT and EPIC achieves 8% lower throughput than EPIC-MHR. As for latency, we show in Figure 7 BEAT-Cobalt has 16%-34% higher latency than BEAT and EPIC has 18%-31% higher latency than EPIC-MHR.

Adaptive vs. Static security. We compare EPIC with

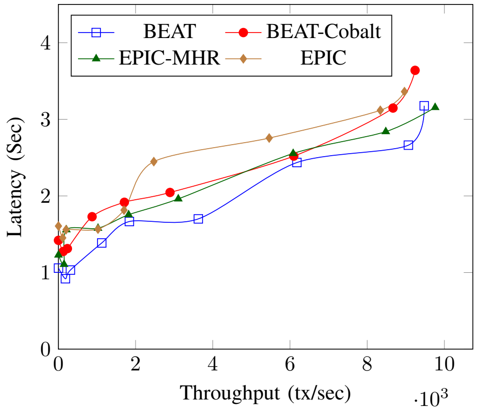

Figure 10. Latency vs throughput for $f = 1$ in the WAN setting.

BEAT-Cobalt, a live, asynchronous BFT protocol with static security. As shown in Figure 7-13, EPIC protocols consistently achieve lower throughput and higher latency than BEAT-Cobalt. This is mainly due to the fact that LM-LJY involves more expensive cryptographic computation. The performance difference in the WAN environment between the two protocols is relatively small: EPIC has 13% higher latency and 5% lower throughput than BEAT-Cobalt. In the LAN setting, the difference is considerably larger where the peak throughput of EPIC is 29% lower than that of BEAT-Cobalt. Besides, we show latency vs. throughput in Figure 10. We observe that for both BEAT and EPIC, their latency increases dramatically when the throughput is close to 9,000 tx/sec.

Transaction selection. We evaluate the transaction selection approach (HYB) for EPIC in both LAN and WAN environments. In our experiments, we let δ be one and run 100 epochs with different μ values. We run the experiments using $f = 1$ and varying b sizes. As we observe similar results for different b sizes, we selectively present the results for $b = 1000$ in Table II. Our experiment shows our HYB strategy provides efficient trade-offs between latency and throughput: if μ is small, the system achieves lower throughput in both LAN and WAN settings.

μ	LAN	WAN
2	5684.10	1147.02
3	6650.40	1341.90
4	7104.66	1433.63
5	7388.17	1490.97
10	7955.31	1605.40
15	8182.70	1651.24
20	8239.33	1662.64
50	8409.91	1697.20
100	8466.72	1708.36

Table II
THROUGHPUT OF EPIC IN BOTH LAN SETTING AND WAN SETTING WHEN $f = 1$ AND $\delta = 1$. EACH EXPERIMENT IS RUN FOR 100 EPOCHS.

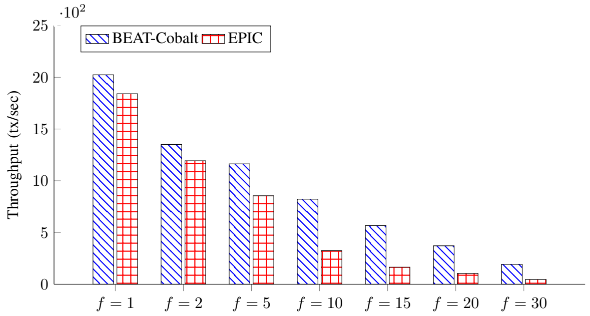

Figure 11. Average throughput per replica of BEAT-Cobalt and EPIC when $b = 5000$ in the WAN setting as f increases.

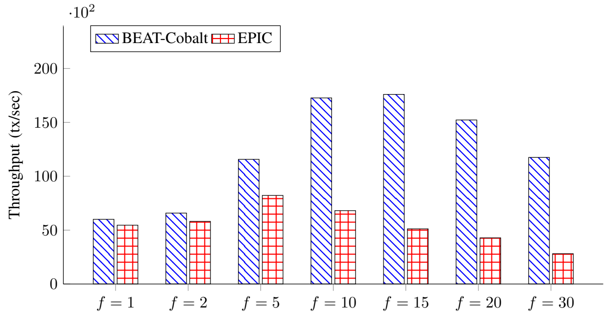

Figure 12. Throughput of BEAT-Cobalt and EPIC when $b = 5000$ in the WAN setting as f increases.

Scalability. We evaluate the scalability of all our implemented protocols by varying f from 1 to 30. To better present our results, we further distinguish (total) throughput TH in the conventional sense and the average throughput per replica AT. The latter is defined as the actual average delivered transactions per second proposed by each replica. In other words, the total throughput is the aggregation of average throughput per replica. Existing protocols report different throughput. Specifically, HoneyBadgerBFT and BEAT report different throughput numbers where HoneyBadgerBFT reported for TH (without considering overlapped transactions proposed by different replicas) and BEAT reported for AT. Although the (total) throughput represents the actual system throughput, we do find that both total throughput and average throughput are worth reporting. Specifically, the average throughput can better illustrate the performance downgradation when the network size grows. In comparison, the total number of proposed transactions grows as the network size increases since replicas all propose transactions concurrently. Therefore, the total throughput does not necessarily downgrade as f increases. We report the average throughput in Figure 11 and the (total) throughput in Figure 12 and Figure 13.

We evaluate the throughput by varying b and observe a similar trend in all protocols. We report the throughput of BEAT-Cobalt and EPIC as b increases in Figure 13. We also report the average and total throughput for BEAT-Cobalt and

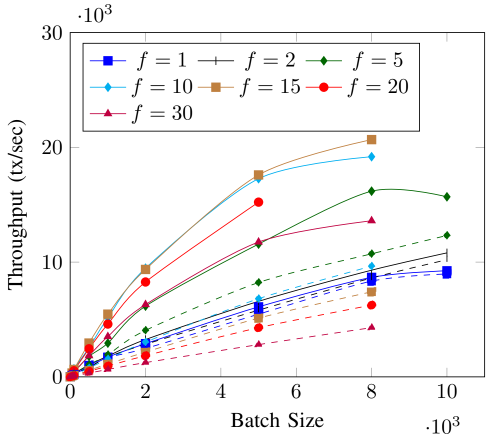

Figure 13. Throughput for BEAT-Cobalt and EPIC in the WAN setting as f increases. BEAT-Cobalt and EPIC are represented in solid and dashed lines, respectively.

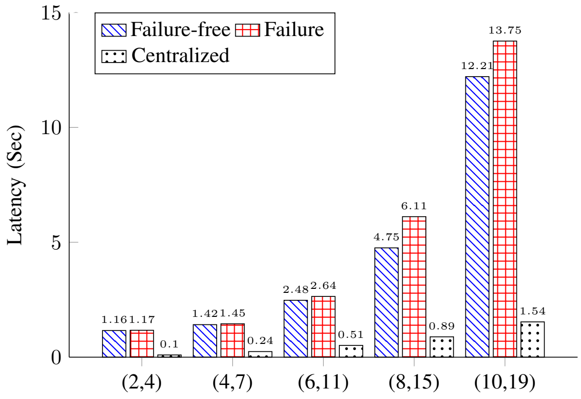

Figure 14. Latency of (t, n) distributed key generation vs. centralized key generation in the LAN setting.

EPIC for $b = 5000$. First, all the protocols achieve lower average throughput per replica when f grows. When $f = 1$, EPIC has 2% lower average throughput per replica than BEAT-Cobalt. When $f = 2$ and 5, EPIC has about 5% and 21% lower average throughput per replica, respectively. But if f further increases, the difference for average throughput becomes significantly larger. (When $f = 10$, the peak (total) throughput of EPIC is around $10,000$ tx/sec.)

For the total throughput, however, we find that as f increases, the throughput for all asynchronous BFT protocols first increases and then decreases. This is (quite) expected, though it has been formally reported by HoneyBadgerBFT or BEAT. Indeed, when f first grows, the number of concurrently proposed transactions grows significantly, making the system throughput higher than that with smaller network sizes. When f further grows, the average throughput per replica becomes much smaller; even if the average throughput per replica gets multiplied by a large $dtx \geq (n - f)$ (the number of ABA instances that deliver 1), the total throughput remains smaller. In our experiments (using $b = 5000$), we observe that BEAT-Cobalt achieves the largest throughput when $f = 15$, and the throughput of EPIC is the largest when $f = 5$.

Distributed key generation. Figure 14 summarizes the latency of our $(f+1, n)$ distributed key generation protocol in LAN environments. We evaluate the failure-free scenario, where all replicas are correct and the non-disqualified set includes all replicas, and the failure scenario, where there exists a single malicious replica. We also compare the two scenarios with a centralized key generation scenario where there is no interaction. Since the distributed key generation protocol runs in synchronous environments, we test only the optimal scenario where the timer equals the message processing and message transmission delays. Therefore, our evaluation result can be used to guide the timer setup for the protocol. (Of course, one should setup much larger timers in practice.) As shown in Figure 14, the distributed key generation incurs (much) higher latency compared to the centralized approach. The performance difference between the failure scenarios and the failure-free scenarios is noticeable but comparatively small.

IX. Conclusion

We design and implement EPIC, an efficient asynchronous BFT protocol achieving adaptive security and decentralized key distribution. To build EPIC, we use a combination of new primitives, including a hybrid approach for transaction selections. To evaluate EPIC, we mix and match system building blocks to implement various protocol variants and identify the performance bottlenecks via extensive evaluation in LAN and WAN environments. Overall, EPIC is not much slower than asynchronous BFT protocols with static security.

X. Acknowledgment

We thank Christian Cachin for answering our questions for adaptive security. We thank the DSN reviewers for their helpful comments.

References

[1] Bug in ABA protocol's use of common coin. https://github.com/amiller/HoneyBadgerBFT/issues/59.

[2] Relic crypto library. https://github.com/relic-toolkit.

[3] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman. Solida: A blockchain protocol based on reconfigurable Byzantine consensus. In *OPODIS*, 2017.

[4] I. Abraham, D. Malkhi, and A. Spiegelman. Validated asynchronous Byzantine agreement with optimal resilience and asymptotically optimal time and word communication. *arXiv preprint arXiv:1811.01332*, 2018.

[5] I. Abraham, D. Malkhi, and A. Spiegelman. Asymptotically optimal validated asynchronous Byzantine agreement. In *Proceedings of the Symposium on Principles of Distributed Computing*, pages 337–346. ACM, 2019.

[6] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptographic Engineering*, 3(2):111–128, 2013.

[7] Y. Amir, B. Coan, J. Kirsch, and J. Lane. Prime: Byzantine replication under attack. *IEEE Transactions on Dependable and Secure Computing*, 8(4):564–577, 2011.

[8] P. Aublin, S. B. Mokhtar, and V. Quéma. Rbft: Redundant Byzantine fault tolerance. In *ICDCS*, pages 297–306, 2013.

[9] J.-P. Bahsoun, R. Guerraoui, and A. Shoker. Making BFT protocols really adaptive. In *IPDPS*, pages 904–913. IEEE, 2015.

[10] P. S. L. M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In *Proceedings of the 12th International Conference on Selected Areas in Cryptography*, 2006.

[11] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC*, pages 27–30, 1983.

[12] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In *Proceedings of the 13th annual symposium on Principles of distributed computing*, pages 183–192. ACM, 1994.

[13] P. Berman and J. A. Garay. Randomized distributed agreement revisited. In *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*, pages 412–419. IEEE, 1993.

[14] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC*, pages 31–46, 2003.

[15] G. Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162. ACM, 1984.

[16] G. Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[17] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350. USENIX Association, 2006.

[18] C. Cachin, D. Collins, T. Crain, and V. Gramoli. Byzantine fault tolerant vector consensus with anonymous proposals. *arXiv preprint arXiv:1902.10010*, 2019.

[19] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

[20] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

[21] C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the internet. In *DSN*, pages 167–176. IEEE, 2002.

[22] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In *SRDS*, pages 191–201. IEEE, 2005.

[23] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *STOC '96*, 1996.

[24] R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *STOC*, volume 93, pages 42–51. Citeseer, 1993.

[25] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[26] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making Byzantine fault tolerant systems tolerate Byzantine faults. In *NSDI*, volume 9, pages 153–168, 2009.

[27] M. Correia, N. F. Neves, and P. Verissimo. How to tolerate half less one Byzantine nodes in practical distributed systems. In *SRDS*, pages 174–183. IEEE, 2004.

[28] M. Correia, N. F. Neves, and P. Veríssimo. From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1):82–96, 2006.

[29] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT*, 1999.

[30] C. Decker, J. Seidel, and R. Wattenhofer. Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, page 13. ACM, 2016.

[31] S. Duan, H. Meling, S. Peisert, and H. Zhang. BChain: Byzantine replication with high throughput and embedded reconfiguration. In *OPODIS*, pages 91–106, 2014.

[32] S. Duan, M. K. Reiter, and H. Zhang. BEAT: Asynchronous bft made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2028–2041. ACM, 2018.

[33] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[34] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *NSDI*, pages 45–59, 2016.

[35] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. Technical report, Massachusetts Inst of Tech Cambridge lab for Computer Science, 1982.

[36] R. Friedman, A. Mostefaoui, and M. Raynal. Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46–56, 2005.

[37] G. Golan-Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu. SBFT: A scalable and decentralized trust infrastructure. In *DSN*, pages 568–580, 2019.

[38] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić. The next 700 bft protocols. *ACM Transactions on Computer Systems*, 32(4):12:1–12:45, 2015.

[39] A. Guillevic. Comparing the pairing efficiency over composite-order and prime-order elliptic curves. In *ACNS*, 2013.

[40] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic. Sok: General purpose compilers for secure multi-party computation. In *S&P*, 2019.

[41] J. Hendricks, G. R. Ganger, and M. K. Reiter. Verifying distributed erasure-coded data. In *PODC*, 2007.

[42] J. Hendricks, S. Sinnamohideen, G. R. Ganger, and M. K. Reiter. Zzyzx: Scalable fault tolerance through Byzantine locking. In *DSN*, pages 363–372. IEEE, 2010.

[43] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX annual technical conference*, volume 8. Boston, MA, USA, 2010.

[44] Joonsang Baek and Yuliang Zheng. Simple and efficient threshold cryptosystem from the gap diffie-hellman group. In *GLOBECOM '03*, 2003.

[45] A. Kate, Y. Huang, and I. Goldberg. Distributed key generation in the wild. *IACR Cryptology ePrint Archive*, 2012.

[46] V. King and J. Saia. Breaking the $o(n^2)$ bit barrier: scalable Byzantine agreement with an adaptive adversary. *Journal of the ACM (JACM)*, 58(4):18, 2011.

[47] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *USENIX Security*, pages 279–296, 2016.

[48] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger. *IACR Cryptology ePrint Archive*, 2017:406, 2017.

[49] E. Kokoris-Kogias, A. Spiegelman, D. Malkhi, and I. Abraham. Bootstrapping consensus without trusted setup: Fully asynchronous distributed key generation. *IACR Cryptology ePrint Archive*, 2019.

[50] R. Kolta, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative Byzantine fault tolerance. *ACM Transactions on Computer Systems*, 27(4):7:1–7:39, 2009.

[51] K. Kursawe and V. Shoup. Optimistic asynchronous atomic broadcast. In *ICALP*, pages 204–215, 2005.

[52] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[53] B. Libert, M. Joye, and M. Yung. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theoretical Computer Science*, 645:1–24, 2016.

[54] B. Libert and M. Yung. Adaptively secure non-interactive threshold cryptosystems. In *ICALP*, 2011.

[55] J. Loss and T. Moran. Combining asynchronous and synchronous Byzantine agreement: The best of both worlds. *IACR Cryptology ePrint Archive*, 2018:235, 2018.

[56] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *CCS '19*, 2019.

[57] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In *CCS*, pages 17–30. ACM, 2016.

[58] E. MacBrough. Cobalt: Bft governance in open networks. *arXiv preprint arXiv:1802.07240*, 2018.

[59] J.-P. Martin and L. Alvisi. Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.

[60] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of bft protocols. In *Proceedings of the SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016.

[61] H. Moniz, N. F. Neves, and M. Correia. Byzantine fault-tolerant consensus in wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 12(12):2441–2454, 2012.

[62] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo. Experimental comparison of local and shared coin randomized consensus protocols. In *SRDS*, pages 235–244, 2006.

[63] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo. Ritas: Services for randomized intrusion tolerance. *IEEE transactions on dependable and secure computing*, 8(1):122–136, 2008.

[64] A. Mostefaoui, M. Hamouma, and M. Raynal. Signature-free asynchronous Byzantine consensus with $t < n/3$ and $o(n^2)$ messages. In *PODC*, pages 2–9. ACM, 2014.

[65] R. Pass and E. Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, 2017.

[66] R. Pass and E. Shi. Thunderella: blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.

[67] M. O. Rabin. Randomized byzantine generals. In *SFCS*, pages 403–409. IEEE, 1983.

[68] H. V. Ramasamy and C. Cachin. Parsimonious asynchronous Byzantine-fault-tolerant atomic broadcast. In *OPODIS*, 2005.

[69] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[70] V. Shoup. Practical threshold signatures. In *EUROCRYPT 2000*.

[71] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptol.*, 15(2):75–96, Jan. 2002.

[72] Y. J. Song and R. van Renesse. Bosco: One-step Byzantine asynchronous consensus. In *DISC*, pages 438–450. Springer, 2008.

[73] J. Sousa, E. Alchieri, and A. Bessani. State machine replication for the masses with bft-smart. In *DSN*, pages 355–362, 2014.

[74] T. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

[75] S. Toueg. Randomized byzantine agreements. In *PODC*, pages 163–178. ACM, 1984.

[76] M. Yin, D. Malkhi, M. Reiterand, G. G. Gueta, and I. Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *38th ACM symposium on Principles of Distributed Computing (PODC)*, 2019.

[77] M. Zamani, M. Movahedi, and M. Raykova. Rapidchain: A fast blockchain protocol via full sharding. In *CCS*, pages 931–948, 2018.

[78] P. Zielinski. Optimistically terminating consensus: All asynchronous consensus protocols in one framework. In *ISPDC*, pages 24–33. IEEE, 2006.