Robust P2P Primitives Using SGX Enclaves

Yaoqi Jia[1] Shruti Tople[2] Tarik Moataz[3] Deli Gong[1] Prateek Saxena[4] Zhenkai Liang[4]

[1]ACM Member {jiayaoqijia, gnnnnng}@gmail.com [2]Microsoft Research t-shtopl@microsoft.com

[3]Brown University tarik_moataz@brown.edu [4]National University of Singapore {prateeks, liangzk}@comp.nus.edu.sg

Abstract

Peer-to-peer (P2P) systems such as BitTorrent and Bitcoin are susceptible to serious attacks from byzantine nodes that join as peers. Due to well-known impossibility results for designing P2P primitives in unrestricted byzantine settings, research has explored many adversarial models with additional assumptions, ranging from mild (such as pre-established PKI) to strong (such as the existence of common random coins). One such widely-studied model is the *general-omission* model, which yields simple protocols with good efficiency, but has been considered impractical or unrealizable since it artificially limits the adversary only to omitting messages.

In this work, we study the setting of a synchronous network wherein peer nodes have CPUs equipped with a recent trusted computing mechanism called Intel SGX. In this model, we observe that the byzantine adversary reduces to the adversary in the general-omission model. As a first result, we show that by leveraging SGX features, we eliminate any source of advantage for a byzantine adversary beyond that gained by omitting messages, making the general-omission model *realizable*. Second, we present new protocols that improve the communication complexity of two fundamental primitives — reliable broadcast and common random coins (or beacons) — in the synchronous setting, by utilizing SGX features. Our evaluation of 1000 nodes running on 40 DeterLab machines confirms theoretical efficiency claim.

1 Introduction

Peer-to-peer systems such as BitTorrent [2], Symform [15], CrashPlan [5], StorJ [14], Tor [16] and Bitcoin [1] are becoming popular among users due to ease of accessibility. In such P2P systems, online users can simply volunteer as peers (nodes) to join the network. However, this exact property allows adversarial or Sybil peers to be a part of the network and exhibit a *byzantine* (malicious) behavior. The presence of byzantine adversaries is a major security concern in P2P systems. For example, recently, researchers have demonstrated that in a popular cryptocurrency — Bitcoin — byzantine nodes can collude to eclipse or partition the honest nodes leading to double-spending and selfish mining attacks [60, 77]. Further, byzantine nodes in anonymous P2P networks can become the entry and exit nodes of an honest node's communication circuit, by advertising high-bandwidth connections and high-uptimes falsely [22]. These byzantine entry / exit nodes can selectively deny service or severely weaken the core anonymity properties of such systems as Tor, Cashmere and Hydra-Onions [16, 31]. In addition, byzantine nodes in the network can selectively forge, divert, delay or drop messages to disrupt the protocol execution. Therefore, designing robust P2P protocols continues to be an important research problem due to the attacks possible in a byzantine setting.

Researchers have extensively worked in the byzantine model to design solutions for fundamental P2P problems such as reliable broadcast and agreement among the peers [18, 19, 26, 28, 53, 54, 65, 81]. There are well-known impossibility results in the standard model of byzantine setting, such as the inability to achieve reliable broadcast or agreement when over $\frac{1}{3}$ of the network is byzantine [65, 81]. In a quest for efficient protocols that tolerate a larger fraction of malicious nodes, several failure models have been proposed which limit the capabilities of the byzantine adversaries. For instance, one such model is the *general-omission* model where the byzantine node can only omit messages that are either sent or received by it during the execution of a protocol [79, 82]. In this weaker adversarial model, it is possible to tolerate $\frac{N}{2}$ adversarial nodes and design relatively simple and efficient protocols for reliable broadcast [41, 59, 79, 82]. However, many of these models make strong assumptions, which are not always realistic and have not had a concrete basis for implementation.

Our approach. To this end, we study the possibility of using recent hardware-root-of-trust mechanisms for making previous adversarial models realizable in practical systems. We observe that emerging hardware, specifically Intel SGX, provides stronger trusted computing capabilities, which allow running hardware-attested user-level enclaves on commodity OSes [7–9, 46]. Enclaves provide hardware-isolated execution environment which guarantees that an application executing in an enclave is tamper-resistant and can be attested remotely. Assuming that SGX-like capabilities become commodity and widescale in end hosts, we ask if it is feasible to build robust P2P protocols. Our main observation is that by leveraging the capabilities of such a trusted hardware, one can restrict the behavior of byzantine adversaries to the *general-omission* model in synchronous networks [41, 59, 79, 82].

Specifically, we use four SGX features, i.e., enclave execution (F1), unbiased randomness (F2), remote attestation (F3) and trusted elapsed time (F4). Based on these hardware features, we enforce six security properties (P1 - P6). First, we enforce execution integrity (P1), message integrity & authenticity (P2) and blind-box computation (P3) to restrict the attacker to not forge messages or deviate from the execution of the given protocol. Thus, the adversarial node can only delay, replay and omit messages. We further leverage lockstep execution (P5) and message freshness (P6) to reduce the adversarial model to the general-omission model, where byzantine nodes have no additional advantage than omitting to send / receive messages. In such model, P3 disallows the adversary to selectively omit messages based on the content. Lastly, the halt-on-divergence (P4) allows us to detect and eliminate peers that selectively omit messages based on identities of senders / receivers, thus in turn reducing round complexity and "sanitizing" the network. Leveraging these properties we can further achieve improvement for the efficiency of protocols. We present efficient designs for reliably broadcasting messages called *Enclaved Reliable Broadcast (*ERB*)* protocol and an unbiased common random generator called *Enclaved Random Number Generator (*ERNG*)* protocol. Both ERB and ERNG primitives can be used as building blocks to solve a wide range of problems in distributed systems, such as random beacons [84], voting schemes [75], random walks [58], shared key generation [55, 56], cryptocurrency protocols [71] and load balancing protocols [47, 85] (details in Appendix H).

Results. Our work targets synchronous network where every machine is running an SGX-enabled CPU. Both of our protocols asymptotically reduce the round and communication complexity as compared to previous works in the byzantine model, and match with (or outperform) the results in general-omission model. For a network of size N, the round and communication complexity for ERB are $\min\{f+2, t+2\}$ and $O(N^2)$, where t / f ($f \le t < \frac{N}{2}$) is the number of byzantine peers / peers actually behaving maliciously for one execution of ERB. The communication complexity of the basic ERNG is $O(N^3)$, and the optimized ERNG further reduces the complexity to $O(N \log N)$. We have implemented our solution and the source code is available online [11]. We evaluate both ERB and ERNG, and our experimental results match our theoretical claims.

Contributions. We summarize the main contributions of this paper as below:

- *Realizable General-Omission Model.* We leverage SGX features to reduce byzantine model to general-omission model, where byzantine nodes have no extra advantage than omitting messages.
- *Better Synchronous P2P Protocols.* By enforcing our properties, we can improve the efficiency of P2P protocols. As the first attempt, we propose efficient protocols for reliable broadcast (ERB) and unbiased random number generation (ERNG).
- *Security Analysis & Evaluation.* We provide security analysis and proof for our protocol constructions. Our experimental evaluation confirms the theoretical expectations of our solutions.

2 Problem

Designing efficient solutions for P2P protocols in the byzantine setting is a widely-recognized problem with limited solutions [18, 19, 26, 53, 54, 81]. Our goal is to shed light on how SGX can aid to improve efficiency of synchronous P2P protocols. In this work, we take two fundamental problems as examples: 1) reliable broadcast and 2) common unbiased random number generator.

2.1 Problem Definition

In light of the previous works, we recall the standard definition of *reliable broadcast* [41, 79] and *common unbiased random number* [20] in the synchronous network:

Definition 2.1. (Reliable Broadcast). *A protocol for reliable broadcast in synchronous settings satisfies the following conditions:*

- *(Validity) If the sender is honest and broadcasts a message m, then all honest nodes eventually accept m.*
- *(Agreement) If an honest node accepts m, then all honest nodes eventually accept m.*
- *(Integrity) For any message m, every honest node accepts m at most once, if m was previously broadcast by the sender.*
- *(Termination) Every honest node eventually accepts a message (m or $\perp$).*

To define a *common unbiased random number* generator, we define the *bias* of any multi-variate function in a standard way [20].

Definition 2.2. (Unbiasedness). *Let $G: \{0,1\}^{k\times N} \to \{0,1\}^k$ be a deterministic multi-variate function that maps N elements in $\{0,1\}^k$ to one element in $\{0,1\}^k$. We define the* bias *of G, $\beta(G)$, as follows:*

$$\beta(G) = \max_{S\subseteq\{0,1\}^k}\left(\max\left(\frac{E[S]}{E_G[S]}, \frac{E_G[S]}{E[S]}\right)\right),$$

where $E_G[S]$ is the expected number of values in $G(x_1,\cdots,x_N) \in S$, and $E[S] = \frac{|S|}{2^k}$, which is the expected value when the output of G is distributed uniformly at random.

Definition 2.3. (Common Unbiased Random Number). *A protocol G generates a common unbiased random number r among N nodes if it satisfies the following conditions with high probability (w.h.p.):*

- *(Agreement) At the end of the protocol, all the honest nodes agree on the same value r.*
- *(Unbiasedness) The bias of $\beta(G) = 1$.*

For the analysis of protocols, we define the following complexities with respect to a single execution of the protocol.

- The *message / communication complexity* is defined as the total number of messages / bits transferred among all nodes in the worst case.
- The *round complexity* is defined as the number of executed rounds (or steps) in the worst-case.

2.2 Attacker Model

We consider a widely-studied standard synchronous model of P2P systems [18, 19, 26, 53, 54, 81]. In this model, our only new requirement is that every peer in the network uses an SGX-enabled CPU to run the P2P protocols. In a network of N nodes, the number of byzantine nodes t is strictly bounded under a fraction of $\frac{N}{2}$. The number of peers that actually behave maliciously for a particular execution of the protocol is $f(\leq t)$. Thus, a P2P network $\mathcal{P}$ is composed of N peers $\mathcal{P} = \{p_1, \cdots, p_N\}$ such that $N = 2t+1$. Every peer p_i in the P2P overlay has an identifier id_i and can communicate with other peers using their ids. The underlying TCP/IP substrate is assumed to provide reliable message delivery within a known bounded delay say Δ. Moreover, we consider a round-based *synchronous* model where each round is equal to the time an honest node requires to send a message and receive a response. Every peer is directly connected to all other peers in the network and knows the network size N. To summarize, we assume: the network size is N (S1); the protocol starts synchronously (S2); the round time is 2Δ (S3); the number of byzantine nodes is limited upto $\frac{N}{2}$ (S4); the peers are connected to each other (S5). This is a prominently used model in the previous literature of distributed P2P systems [18–20, 58, 79, 82]. We discuss the validity of these assumptions in Appendix G.

Our Model using SGX. In our model, a byzantine peer has a compromised or malware-ridden operating system but executes protocols using SGX enclaves [7, 8, 46]. Enclaves guarantee untampered execution in presence of malicious underlying software or co-processes. The byzantine nodes can take arbitrary software actions as long as it does not violate SGX guarantees.

Scope. Our focus is showing how to leverage SGX features to improve the efficiency of synchronous P2P protocols. Our model does not consider an adversary that can perform hardware attacks and break SGX security guarantees. We do not aim to prevent any information leakage through side-channels such as pagefaults, memory accesses or timing attacks to which SGX-enabled CPUs are known to be susceptible [66, 74, 90]. Indeed these problems are under investigation and recent research shows that defending against them is feasible. Existing solutions against these problems can directly apply to our work [72, 78, 86].

2.3 Strawman Solution & Attacks

Consider a strawman protocol for distributed random number generation using reliable broadcast, where the initiator

Algorithm 1: Strawman distributed random number generation protocol using reliable broadcast.

Input: A P2P network $\mathcal{P}$ composed of N nodes, an initiator node $\mathsf{id}_{\mathsf{init}}$
Output: A message $\hat{m}$

```
1  Initialization: m̂ ← ⊥; S_m ← ∅; rnd ← 1
2  upon self_id is initiator:
3  get(m) // m is a random number
4  m̂ ← m
5  add self_id to S_m
6  multicast INIT(m) to other peers
7  for rnd ≤ t+1 do
8      upon receiving INIT(m):
9      m̂ ← m
10     add self_id and sender_id to S_m
11     multicast ECHO(m) to other peers in round rnd+1
12     upon receiving ECHO(m):
13     if m̂ = ⊥ then
14         m̂ ← m
15         add self_id to S_m
16         multicast ECHO(m) to other peers in round rnd+1
17     end
18     if m = m̂ and sender_id ∉ S_m then
19         add sender_id to S_m
20         if |S_m| = N − t then
21             accept m̂
22         end
23     end
24     rnd ← rnd+1
25 end
26 if rnd > t+1 then
27     accept ⊥
28 end
```

broadcasts a random number m using an initialization message INIT to all the peers in a synchronous network (shown in Algorithm 1). If m is generated randomly and unbiasedly as well as reaches every honest node without being tampered, then all honest nodes will agree on the common unbiased random number m and the goal of the protocol is achieved. In Algorithm 1, upon receiving the INIT message, each peer further multicasts an ECHO message to all other peers. After receiving the ECHO messages from the majority of nodes, each peer accepts m as the final message $\hat{m}$. Note that if the initiator is honest, all honest nodes receive the message INIT during the first round and multicast ECHO messages at the beginning of the second round. In the second round, every honest node receives at least $N-t$ ECHO messages from $N-t$ honest nodes and maybe some byzantine nodes. Thus, after two rounds, every honest node will output the same value m from the initiator, which satisfies all the conditions of reliable broadcast in Definition 2.1. However, we show how a byzantine initiator and other byzantine peers can attack this protocol to violate Definitions 2.1 and 2.3.

Attacks by Byzantine Adversary. Byzantine initiator / peers can tamper with the execution of Algorithm 1 and forge the values of INIT and ECHO messages to perpetrate the following attacks.

A1 (Execution Deviation): For this attack, an adversary deviates from the control flow of the running program for the given protocol. The adversary can disregard essential conditions to jump to the desired instructions and execute them directly. For example, the adversary can skip all the conditions like Line 7 & 13 to directly multicast its ECHO value to parts of honest nodes but not all of them, to introduce equivocation to their final decisions. Moreover, the adversary can also repeat particular instructions to obtain an output she wants. For instance, if m is generated from a random source without being tampered during the execution of the protocol, an unbiased common random number can be agreed among all the peers in the network. A byzantine peer, however, can repeat the step that generates m (Line 3) from the random source until it returns a favorable random number. Hence, the output is biased as per Definition 2.3.

A2 (Message Forgery): Suppose that the adversary does not deviate from the execution of the given protocol, she can still alter the data flow (including input / output and intermediate states) of the program to forge messages. As per Definition 2.1, a reliable broadcast protocol requires that if one honest node accepts message m then all honest nodes accept m. The adversary can tamper with the INIT and ECHO messages to violate this agreement property of the protocol. A byzantine initiator colluding with other byzantine peers in the network can tamper with Line 6, 11 and 16 in the algorithm such that some honest nodes receive most ECHO messages with m' while others with m. This results in a fraction of honest nodes assigning $\hat{m}$ with m' and accepting m', while other honest nodes accept m as the final output, thereby causing inconsistency in the network.

A3 (Selective Omission): Assume that the adversary does not deviate from the control flow (i.e., the execution) of the given protocol or tamper with the data flow to forge messages, she can still omit, delay and replay messages in this restricted model. For an omission attack, it has two types: one is based on the content of the transmitted message and the other is dependent on the identity of the sender / receiver. For the first type, the adversary can observe its generated or received random number m and selectively decide to drop or forward it to other nodes based on its value, which introduces a bias in the final output for the honest nodes. For example, if the adversarial peers receive or initiate a message m, which is not the favorable one, they can omit to relay the message to the other nodes, thus all honest nodes may finally agree on $\perp$ instead of m. Further, to violate the agreement condition in Definition 2.1 and 2.3, the adversary can selectively decide to omit the message m depending on whether the destination peer is honest or malicious. It can broadcast m correctly to a few honest nodes and not send the message to the others for the last round. The honest nodes receiving m can multicast m to the others, but the others will not accept it as the execution ends. Thus, the honest nodes that do not receive a message will agree on $\perp$ while others will agree on m.

A4 (Message Delay): Alternatively, to generate an unbiased common random number, every peer can broadcast its random number to all other peers using Algorithm 1. All peers can then XOR the random numbers in the final set to generate the output. To bias this final output, a byzantine peer can intentionally hold its random number until it receives inputs

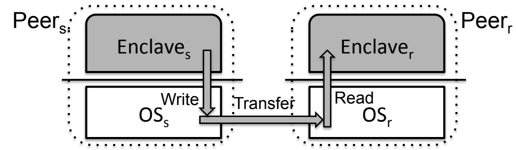


Figure 1: Each peer consists of two entities: an Enclave and an OS. The OS models the operating system and memory. The Enclave models the isolated memory and the secure execution of a program. The sender Enclave$_s$ can send a message via a secure channel to the receiver Enclave$_r$. The grey areas are secure against malicious OSes of byzantine nodes.

from all other honest peers [20]. In this way, the adversary can "look ahead" in the protocol, calculate the final output and then decide whether to participate in the protocol by sending its random number. If the final random number already favors the adversary then it does not participate in the protocol, otherwise it sends its message to all the peers. Note that, for $t < \frac{N}{2}$, all the byzantine adversaries can collude to introduce an exponential bias in the final value.

A5 (Message Replay): In the restricted model, the adversarial node can use a message m_{prev} from an instance of the protocol running in parallel, or which was run in the past to one (or more) honest node(s) and forward the correct message m to other honest nodes [69]. This results in an inconsistency where few honest nodes agree on m_{prev} and others agree on m, thereby violating the agreement condition.

3 Solution Overview

In this section, we put forward ideas using SGX features to enforce six security properties to restrict the capabilities (A1 - A5) of a byzantine adversary, as shown in Section 2.

3.1 SGX Features and Security Properties

We first start by recalling Intel SGX features (supported in both simulation and hardware modes in the latest version [7, 9]), which can also be provided by other trusted hardware.

F1: *Enclaved Execution* - SGX supports hardware-isolated memory region called enclaves such that a compromised underlying OS cannot tamper the execution of the code running inside this enclave.

F2: *Unbiased Randomness* - SGX provides a function `sgx_read_rand` that executes the `RDRAND` instruction to generate hardware-assisted unbiased random numbers.

F3: *Remote Attestation* - SGX allows a remote party to verify that an application is running in an enclave on an SGX-enabled CPU.

F4: *Trusted Elapsed Time* - SGX provides a function `sgx_get_trusted_time` that returns a trusted elapsed time in seconds relative to a reference point.

Abstractly, a peer can be considered as the composition of two entities: an OS and an Enclave as shown in Figure 1. The OS models the untrusted entity including the operating system and memory. It has access to all the system resources such as file system and network. The OS can arbitrarily invoke an enclave program and start its execution. The Enclave models the isolated memory space that loads the program and executes it securely. Thus, Enclave corresponds to the trusted entity of a peer. We illustrate how to enforce P1 - P6 properties using SGX features to thwart A1 - A5 attacks.

P1 (Execution Integrity): With remote attestation (F3), an enclave in one peer can verify the correctness of the running program for the given protocol on the other nodes and whether it is executing on a valid SGX-enabled CPU or not. Moreover, F1 ensures that the execution in an enclave cannot be tampered with by the OS. F1 and F3 together enforce the execution integrity against A1. Hence, an adversary cannot deviate from the execution of the protocol in an enclave arbitrarily by skipping / repeating instructions to violate the control flow of the running program.

P2 (Message Integrity & Authenticity): In designing our protocols, we first perform a setup phase where each peer connects to every other in the network and then performs a series of steps. Analogous to P1, every enclave first uses F3 to verify the correctness of the protocol executing on other peers. Next, they generate public / private key pairs inside the enclaves and exchange the public keys with each other. Then all the messages transmitted between any two enclaves can be signed to ensure the integrity and authenticity against A2. Moreover, the internal states of the program are also protected using F1. Therefore, the integrity of all messages including input / output / intermediate states is guaranteed. In this case, it is clear that an adversary cannot forge valid messages to bias the honest nodes to make inconsistent decisions.

P3 (Blind-box Computation): F1 ensures that all intermediate states of the protocol's computation are hidden from the OS. Leveraging F2, the provided randomness is also hidden from the OS. This guarantees that the input state is hidden along with the intermediate states of the protocol's execution We say in this case that the computation is a *blind-box computation*. As the adversarial node does not know the random number and given that the output of the computation is encrypted between the Enclave and the OS, she cannot selectively omit or drop messages based on their contents. Note that an important part of instantiating such a blind-box computation is the ability to instantiate a secure channel between two or more enclaves. In fact, enclaves can agree on a shared key to establish a secure channel using Diffie-Hellman key exchange. Nodes can then encrypt all the messages (including program's intermediate input / output) transmitted between each other to provide confidentiality against malicious OSes. Note that, establishing such a shared key in the enclaved setting is slightly weaker than the standard byzantine model, as the malicious operating system cannot access the shared secret keys and decrypt the exchanged messages due to F1. With P1 - P3, we can reduce the byzantine model to a restricted model, where an adversarial node can only replay, omit and delay messages.

P4 (Halt-on-Divergence): To mitigate selective omission based on nodes' identities (A3), we enforce a security mechanism called halt-on-divergence. This property halts any malicious node deviating from the protocol under some given condition. As an instance, if an adversarial node sends a message, but does not receive adequate responses, it will be forced to leave the current protocol execution. Halt-on-divergence mechanism should be incorporated through a specific acknowledgment protocol instantiation in such a way that every malicious node will be forced to leave if the acknowledgment is not verified. In particular, we introduce an acknowledgment scheme where every receiver acknowledges the sender on receiving every valid message. A message sent over a secure channel is considered valid only if it contains the expected sequence and round number. Naturally, an acknowledgment is not sent for a replayed, omitted or delayed message. Since all honest receivers will reply with acknowledgment (ACK) messages on receiving valid messages, an honest sender should at least receive $t+1$ ACK messages. Any node receiving less than $t+1$ ACK messages will halt and leave the network.

The key idea here is to penalize any deviating adversary by churning the node out of the network. This effectively "sanitizes" the network. Thus, to remain a part of the network, every peer should send valid messages to the majority of the network. This property also aids honest nodes in the protocol to decide the final output early and finish the execution immediately.

P5 (Lockstep Execution): F4 allows us to realize a synchronized network across all rounds of a protocol. Each peer uses F4 to decide the correct value of the ongoing round and inserts this round number in all the sent messages. To detect *delay* attacks (A4), a peer simply matches the round number present in an incoming message with the current round number. This defense is hard in the byzantine model with public-key infrastructure even if it supports F1, since the OS can tamper with the relative time to either increase or decrease the rounds of a node. Therefore, having access to a trusted elapsed time functionality allows to perform lockstep execution and detect delay attacks in the restricted model.

P6 (Message Freshness): Similar to [69], we use sequence numbers to ensure message freshness and therefore defend against replay attacks (A5). The main challenge lies in ensuring secure exchange of the initial sequence numbers for each peer and ensuring that the sequence number remains untampered with during the entire intermediate states of the protocol execution. Using the secure channel, the peers securely exchange a nonce or a *sequence number*, which is incremented sequentially by the peer. The nonce is generated using F2 supported by SGX. This prevents the malicious adversary from tampering the initial nonce value to its own advantage. Note that the keys and initial sequence numbers exchange occur only once during the setup phase. If an adversarial node restarts or relaunches its enclave, all the data in the enclave will be removed. Since the enclave does not have the valid sequence number and round number, it cannot re-join the same or any on-going execution, which is equivalent to be considered as a new node for the protocol.

3.2 Overview of Our Results

In this work, we achieve the following results.

R1: *By enforcing (P1 - P6), we reduce the byzantine model to the general-omission model.*

By enforcing P1 - P3, we first reduce byzantine model to a restricted model, in which byzantine nodes can only delay / omit / replay messages. Due to space constraints, we defer the formalization and proof to Appendix A. We believe that the formalization, while based on traditional cryptographic primitives, provides a new conceptual framing of SGX-enabled CPUs security features, and may be of independent interest. By applying P5 and P6, we further confine the adversarial nodes into the general-omission model.

R2: *We propose an efficient reliable broadcast protocol (*ERB*) with early stopping, which improves communication complexity from $O(N^3)$ to $O(N^2)$ (refer to Section 4).*

For this result, we leverage four properties. First, P1 - P3 ensure that the adversarial nodes cannot forge messages and deviate from the execution of the protocol. Second, we leverage P4 to show that ERB can broadcast a message to the entire network in $\min\{f+2, t+2\}$ rounds with better performance as shown in Table 1. We further illustrate that our properties are generic and can improve the efficiency of traditional protocols of reliable broadcast. Due to space limitations, we detail our findings in Appendix B.

R3: *We propose a new unbiased random number generation protocol (*ERNG*) with communication complexity $O(N^3)$ for the basic version, or $O(N \log N)$ for the optimized one, as shown in Table 2 (refer to Section 5).*

With P3 and P5, our unoptimized ERNG solution directly runs our ERB protocol as a sub-routine on the entire network to agree on a random number generated using F2. It has round and communication complexity of $O(N)$ and $O(N^3)$, respectively. We present an optimized version of ERNG by reducing the byzantine fraction from $\frac{N}{2}$ to $\frac{N}{3}$, and forming a cluster of peers within the network. Leveraging the trusted randomness F2 and P3, we can sample a small set of nodes forming a representative cluster. The ERB protocol is executed within this small cluster to generate the final unbiased random number. The round and communication complexity of this optimized ERNG is further reduced to $O(\log N)$ and $O(N \log N)$. Note that the optimized version of ERNG only applies when the size of the network is large enough.

4 Enclaved Reliable Broadcast Protocol

We propose an *enclaved reliable broadcast* (ERB) in the synchronous model using SGX features. The transmitted message, val, between any two peers has the format: $\mathsf{val} := \langle \mathsf{type}, \mathsf{id}, \mathsf{seq}, m, \mathsf{rnd} \rangle$, where $\mathsf{type} \in \{\mathsf{INIT}, \mathsf{ECHO}, \mathsf{ACK}\}$ and rnd represents the current round of the ERB protocol. If

Protocol	Attacker Model	Network Size	Round Complexity	Comm. Complexity
PT [82]	Omission	$t+1$	$\min\{f+2, t+1\}$	$O(N^3)$
PR [79]		$2t+1$	$\min\{f+2, t+1\}$	
CT [41]			$2t+1$	$O(N^2)$
PSL [81]	Byzantine	$3t+1$	$t+1$	O(exp (N))
BGP [28]			$\min\{f+2, t+1\}$	
BG [26]		$4t+1$	$t+1$	O(poly(N))
GM [53, 54]		$3t+1$	$\min\{f+5, t+1\}$	
AD15 [18]			$\min\{f+2, t+1\}$	
AD14 [19]	Byzantine	$2t+1$	$3t+4$	$O(N^4)$
ERB	Byz. + SGX	$2t+1$	$\min\{f+2, t+2\}$	$O(N^2)$

Table 1: Round complexity and communication complexity for reliable broadcast in synchronous network.

Protocol	Network Size	Round Complexity	Comm. Complexity
AS [20]	$6t+1$	$O(N)$	$O(N^3)$
AD14 [19]	$2t+1$	$O(N)$	$O(N^4)$
Basic ERNG	$2t+1$	$O(N)$	$O(N^3)$
Optimized ERNG	$3t+1$	$O(\log N)$	$O(N \log N)$

Table 2: Round / communication complexity for random number generation protocols in synchronous distributed systems.

$\mathsf{type} = \mathsf{INIT}$, then the initiator peer $\mathsf{id}_{\mathsf{init}}$ is initiating the broadcast by sending the message m with sequence number $\mathsf{seq}_{\mathsf{init}}$ at round rnd. If $\mathsf{type} = \mathsf{ECHO}$, it means that its sender knows that $\mathsf{id}_{\mathsf{init}}$ has sent m, as it has already received either a value with INIT or ECHO for the first time. Finally, if $\mathsf{type} = \mathsf{ACK}$, it means that the peer acknowledges that it has already received either INIT or ECHO values from the sender. We introduce three functions Halt, Multicast and Wait:

- $\mathsf{Halt}(\mathsf{st})$: is a function that sets the state st to $\perp$.
- $\mathsf{Multicast}(\mathsf{id}_i, \mathsf{val})$: is a functionality that multicasts the value val from the sender p_i to the receiver p_j, for all $j \in [N] \setminus \{i\}$.
- $\mathsf{Wait}(\tau)$: is a function that has as an input the current elapsed time τ in the ongoing round, and suspends the protocol for $(2\Delta - \tau)$ seconds.

Note that Halt function enforces the *halt-on-divergence* property (P4) that we have introduced in Section 3. When the state of the node is set to $\perp$ the node halts on-divergence and is ejected from the P2P network $\mathcal{P}$. For the sake of exposition, we write $\mathsf{Wait}(\mathsf{rnd})$ in the code description, we say in this case that the protocol waits until the end of the round rnd.

4.1 ERB details

Prior to running the very first instance of the ERB protocol, there is a setup phase. The setup is performed whenever the program (ERB) needs to be updated or changed. We detail the setup phase followed by the explanation of our algorithm.

Setup Phase: Every pair of sender and receiver peer use remote attestation (F3) along with enclaved execution (F1) to verify the correctness of the execution, and therefore enforcing P1 - P3. Then they establish a secure channel using Diffie-Hellman key exchange. This setup enforces P1 - P3, which restricts the byzantine nodes to only omit, replay and delay messages. Next, each peer picks at random a sequence number such that $\mathsf{seq}_s, \mathsf{seq}_r \xleftarrow{\$} \{0,1\}^k$ and send it to each other. That is, every node has to store the sequence numbers of all other nodes in $\mathcal{P}$. Finally, every node sets the variable rnd to the value 1. The overhead of the setup is in $O(N^2)$ while the storage overhead per node is in $O(N)$.

Initialization Phase: An initiator node first multicasts the value $\mathsf{val} = \langle \mathsf{INIT}, \mathsf{id_{init}}, \mathsf{seq_{init}}, m, \mathsf{rnd} \rangle$, where $\mathsf{seq_{init}}$ is the sequence number of the initiator node, and rnd is the round number. The round rnd is first initialized to 1, the enclave will now increment the rnd after every 2Δ seconds—we take advantage of the elapsed time feature of SGX to tie a round to an interval of 2Δ seconds.

Echo Phase: Until round $t+2$, if a node receives an INIT or ECHO message for the first time, it performs the following actions: (1) start the local clock and initialize the round rnd to 1, the round will increment every 2Δ seconds, (2) if both rnd and seq are consistent with the expected values, it will store the message m, else it just ignores it and treats it as an omitted message. If there is no delay or replay detected, then it multicasts an ECHO message to all nodes at the end of the current round. If the node has already received a valid ECHO message from a distinct node, it will only add the sender's identifier into the set $\mathsf{S_{echo}}$. Recall that at the end of the setup phase, all honest nodes have the same copy of the sequence number of all honest nodes. After every valid instance of the protocol, nodes will increase all sequence numbers by 1.

Decision Phase: If the node has received at least $t+1$ correct ECHO messages from distinct nodes, i.e., $|\mathsf{S_{echo}}| = t+1$, then the node accepts $\hat{m}$. After $t+2$ rounds, if the node has not received adequate distinct ECHO messages, it accepts $\hat{m} := \perp$. Every multicast requires the node to receive at least $t+1$ ACK messages, else the node churns out itself using Halt.

4.2 Analysis

In Algorithm 2, if a byzantine sender decides to omit a message, it will not receive a corresponding ACK message as the sent messages never reach the receiver peer. The sender $\mathsf{Enclave}_s$ detects that the underlying OS_s is byzantine if it does not receive at least $t+1$ ACK messages. On failing to receive majority ACK messages, $\mathsf{Enclave}_s$ executes the Halt function as per our algorithm and churns itself out of the network based on our halt-on-divergence property (P4). By leveraging the P4, any node can actively detect its own anomalous behavior instead of relying on other nodes to send messages every round to passively identify the anomaly. This results in communication complexity for anomaly detection decreased from $O(N^2)$ to $O(N)$ and the overall complexity is reduced to $O(N^2)$, compared to previous passive-detection approaches, e.g., Perry *et al.*'s work [82]. Here we state our main theorem below and defer the detailed proof to Appendix C.

Theorem 4.1. *If* $N \geq 2t+1$, ERB *is a* reliable broadcast protocol *as defined in Definition 2.1.*

ERB **Performance Analysis.** Algorithm 2 has a worst-case round complexity equal to $t+2$ with communication complexity in $O(N^2)$ and $t < \frac{N}{2}$ byzantine nodes. This only occurs

Algorithm 2: ERB: Enclaved reliable broadcast protocol (for a node id_i with the initiator $\mathsf{id_{init}}$ sending a message m and a sequence number $\mathsf{seq_{init}}$).

Input: A P2P network $\mathcal{P}$ composed N nodes, a message m and a sequence number $\mathsf{seq_{init}}$ for the initiator $\mathsf{id_{init}}$
Output: A message $\hat{m}$

- initialization: $\hat{m} \leftarrow \perp; \mathsf{S_{echo}} \leftarrow \emptyset; \mathsf{rnd} \leftarrow 1$
- upon $\mathsf{id}_i = \mathsf{id_{init}}$ and $\mathsf{st}_i \neq \perp$:
  $\hat{m} \leftarrow m$;
  $\mathsf{S_{echo}} \leftarrow \mathsf{S_{echo}} \cup \{\mathsf{id_{init}}\}$;
  $\mathsf{Multicast}(\mathsf{id_{init}}, \langle \mathsf{INIT}, \mathsf{id_{init}}, \mathsf{seq_{init}}, m, \mathsf{rnd} \rangle)$;
- **for** $\mathsf{rnd} \leq t+2$ **do**
  - upon receiving $\langle \mathsf{INIT}, \mathsf{id_{init}}, \mathsf{seq}, m, \mathsf{rnd}' \rangle$ from $\mathsf{id_{init}}$:
    if $\mathsf{rnd}' = \mathsf{rnd}$ and $\mathsf{seq} = \mathsf{seq_{init}}$ **then**
      send $\langle \mathsf{ACK}, \mathsf{id_{init}}, \mathsf{seq}, H(m), \mathsf{rnd} \rangle$ to $\mathsf{id_{init}}$;
      $\hat{m} \leftarrow m$;
      $\mathsf{S_{echo}} \leftarrow \mathsf{S_{echo}} \cup \{\mathsf{id_{init}}\} \cup \{\mathsf{id}_i\}$;
      $\mathsf{Wait}(\mathsf{rnd})$ then $\mathsf{Multicast}(\mathsf{id}_i, \langle \mathsf{ECHO}, \mathsf{id_{init}}, \mathsf{seq}, m, \mathsf{rnd}+1 \rangle)$;
    end
  - upon receiving $\langle \mathsf{ECHO}, \mathsf{id_{init}}, \mathsf{seq}, m, \mathsf{rnd}' \rangle$ from peer id_j:
    if $\mathsf{rnd}' = \mathsf{rnd}$ and $\mathsf{seq} = \mathsf{seq_{init}}$ **then**
      send $\langle \mathsf{ACK}, \mathsf{id_{init}}, \mathsf{seq}, H(\mathsf{val}), \mathsf{rnd} \rangle$, where $\mathsf{val} = \langle \mathsf{ECHO}, \mathsf{id_{init}}, \mathsf{seq}, m, \mathsf{rnd} \rangle$ to peer id_j;
      if $\hat{m} = \perp$ **then**
        $\hat{m} \leftarrow m$;
        $\mathsf{S_{echo}} \leftarrow \mathsf{S_{echo}} \cup \{\mathsf{id}_i\}$;
        $\mathsf{Wait}(\mathsf{rnd})$ then
        $\mathsf{Multicast}(\mathsf{id}_i, \langle \mathsf{ECHO}, \mathsf{id_{init}}, \mathsf{seq}, m, \mathsf{rnd}+1 \rangle)$;
      end
      if $\mathsf{id}_j \notin \mathsf{S_{echo}}$ **then**
        $\mathsf{S_{echo}} \leftarrow \mathsf{S_{echo}} \cup \{\mathsf{id}_j\}$
        if $|\mathsf{S_{echo}}| = N - t$ **then**
          accept $\hat{m}$;
        end
      end
    end
  - upon $\mathsf{Multicast}(\mathsf{id}_i, \mathsf{val})$:
    send val to id_k, for all $k \in [N] \setminus \{i\}$;
    receive N_{ack} acknowledgements $\langle \mathsf{ACK}, \mathsf{id_{init}}, \mathsf{seq}, H(\mathsf{val}), \mathsf{rnd}' \rangle$, where $\mathsf{rnd}' = \mathsf{rnd}$ and $\mathsf{seq} = \mathsf{seq_{init}}$;
    if $N_{\mathsf{ack}} < t$ **then**
      $\mathsf{Halt}(\mathsf{st}_i)$;
    end
  - $\mathsf{rnd} \leftarrow \mathsf{rnd} + 1$;
  end
- **if** $|\mathsf{S_{echo}}| < N - t$ **then**
  $\hat{m} \leftarrow \perp$;
  accept $\hat{m}$;
  end
- $\mathsf{seq_{init}} \leftarrow \mathsf{seq} + 1$;

if the byzantine peers delay the instance for t rounds before sending the message to at least one honest node. However, in this case, the round complexity is equal to $f+2$ rather than $t+2$ as the delay is only in function of the number of byzantine nodes f. On the other hand, byzantine nodes can also decide to not send the message to any honest node, and then the round complexity is $t+2$ with $O(t)$ communication complexity.

5 Enclaved Random Number Generation

We present our algorithm that generates an unbiased common random number called *enclaved random number generation (*ERNG*).*

5.1 Unoptimized ERNG

We detail our unoptimized ERNG in Algorithm 3. At a higher level, every node generates a random number from the enclave, and then performs ERB protocol to broadcast to every

Algorithm 3: Unoptimized-ERNG: Unoptimized enclaved unbiased random number generation protocol executed by peer p_i.

Input: A P2P network $\mathcal{P}$ composed of N nodes
Output: A unbiased random number r

- initialization: $\mathsf{S}_{\mathsf{final}} \leftarrow \emptyset; \mathsf{rnd} \leftarrow 1$
  for $\mathsf{rnd} \leq t+2$ **do**
  - **if** $\mathsf{rnd} = 1$ **then**
        initiate ERB with inputs $m_i \xleftarrow{\$} \{0,1\}^k$ and seq_i;
    end
    if $2 \leq \mathsf{rnd} \leq t+2$ **then**
        execute ERB instances and wait for the output ($\mathsf{M}_i = \{\hat{m}_1, \cdots, \hat{m}_{l_i}\}$);
    end
    $\mathsf{rnd} \leftarrow \mathsf{rnd} + 1$;
  end
- $\mathsf{S}_{\mathsf{final}} \leftarrow \mathsf{M}_i$;
  $\mathsf{seq}_j \leftarrow \mathsf{seq}_j + 1$, for all $j \in [N]$
  accept $\mathrm{r} = \bigoplus_{v \in \mathsf{S}_{\mathsf{final}}} v$.

node. According to Theorem C.1, all honest nodes in this case will receive the random numbers from all honest nodes after $t+2$ rounds, and may eventually receive several random numbers from other byzantine nodes. According to the validity requirement, for each ERB instance, every honest node will accept a random number from its initiator or $\perp$ so that all honest nodes have the same final set $\mathsf{S}_{\mathsf{final}}$ of random numbers. By performing exclusive disjunction (or XOR) of all received random numbers, every honest node obtains an *unbiased common* random number eventually.

Unbiasedness and Randomness Analysis. We describe the main intuition behind the common unbiasedness and randomness of our ERNG's output and defer formal details to Appendix E. To bias the random value, the adversary may perform several attacks. It can first try to directly forge the random number, however, this is restricted as per execution integrity (P1) and message integrity (P2) enforced by F1 and F3. An adversary can force the program to generate a local random number of its choice. However, each enclave generates an unbiased random number from SGX-enabled CPU instruction `RDRAND` using F2. It is not possible to bias the source of randomness based on the hardware guarantees.

Our blind-box computation (P3) together with the secure channel guarantee that an adversary cannot selectively omit its random number based on its value with the goal to bias the output. Therefore, the adversary cannot infer the random numbers submitted by other honest peers during the execution. Note that, the defense against replay attacks is already provided by the ERB protocol.

One adversarial strategy is to learn the final output and then decide whether to participate or not in the protocol, as in Attack A4. From Algorithm 3, all honest nodes output the final value after round $t+2$. In order to bias the final value, the adversary should perform the following steps within round number $t+2$: (1) learn the XOR of random numbers from honest nodes, (2) decide whether to participate or not based on the final value, (3) and multicast its number to honest nodes. In Algorithm 3, the final XOR operation executes only when $\mathsf{rnd} > t+2$. The execution integrity (P1) ensures sequential execution of our protocol. This property restricts the adversary from directly jumping to the step that computes the XOR operation and learn the result before other honest nodes generate the final output. Next, the lockstep execution (P5) enforced by the elapsed time feature (F4) allows us to bound the time for each round, even on a byzantine peer. Therefore, the adversary cannot look ahead and compute the final output before the last round. If the adversary decides to delay its own random number based on the computed final value, the adversarial random number will be neglected by all honest peers as it will reach after $t+2$ round. Combining P1, P5 and P3, it is not possible for the byzantine adversary to achieve steps (1) and (3) simultaneously.

For clarity and without any loss of generality, we model Algorithm 3 as a multi-variate function $G: \{0,1\}^{k \times N} \rightarrow \{0,1\}^k$ that maps N elements in $\{0,1\}^k$ to one element in $\{0,1\}^k$ such that $G(x_1, \cdots, x_N) = \bigoplus_{i=1}^{N} x_i$.

Theorem 5.1. *The bias of* G $\beta(G) = 1$.

We defer the proof to Appendix E.

5.2 Optimized ERNG

Next, we illustrate the main steps behind our optimized ERNG and defer the pseudo-code details to Appendix F.1. In this section, we consider that at most $t \leq \frac{N}{3}$ nodes of the network can be byzantine. In this case, ERNG terminates after $\gamma + 4$ rounds, where γ is a statistical parameter. The intuition behind our optimization can be formulated as follows: we first notice that if we select uniformly at random a subset of nodes from $\mathcal{P}$, we can still guarantee w.h.p. the existence of an honest majority within this smaller representative cluster. By leveraging F2 to generate a random number and blind-box computation (P3), we can sample a set of peers forming a representative cluster. The main remaining question, therefore, is how large this cluster should be. As a starting point, note that if the cluster size is equal to $\frac{2N}{3}$, the probability of having an honest majority is equal to one. This already suggests that the cluster size can be chosen to be smaller. Conceptually, the protocol can be decomposed into three main steps:

Cluster Selection: The purpose of this step is to construct a representative cluster of the entire P2P network. The cluster will consist of nodes selected uniformly at random from $\mathcal{P}$. At round 1, every node picks uniformly at random a number from $\{0, \cdots, \frac{N}{2\gamma} - 1\}$ using SGX (F2). This operation is protected leveraging property P3 in such a way that the computation is hidden from the OS. If the random number equals 0, then the node is *chosen* to be part of the cluster, and then it multicasts a CHOSEN message to all nodes in $\mathcal{P}$. Upon receiving the CHOSEN message, every chosen node adds the identifier of the sender to its own set $\mathsf{S}_{\mathsf{chosen}}$. The size of the set $\mathsf{S}_{\mathsf{chosen}}$ represents the size of the cluster.

ERB **Instances:** We first detail a pseudo-solution and then detail our main construction in Algorithm 6 in Appendix F.1. In round 2, the nodes constituting the cluster will each generate a random number and broadcast it *only* to the nodes constituting the cluster (i.e., peers' identifiers in $\mathsf{S}_{\mathsf{chosen}}$). That is, every node in the cluster will run an independent ERB instance. The intuition behind these multiple instances is the following: for the broadcast to be effective, at least one broadcast instance has to succeed in that the accepted message is different from $\perp$. However, the complexity of such solution is cubic in $O(|\mathsf{S}_{\mathsf{chosen}}|^3)$ which can be a handicap in term of efficiency. As a solution, we incorporate a two-phases clustering. The idea behind this choice is the following: in order to generate a random number we only require one honest node to output a random number r (otherwise the ERNG protocol may output $\perp$). We can then proceed to select just a few number of nodes to perform the ERB protocol. As long as at least one of these nodes is honest, the correctness of our ERNG holds. Concretely, to generate the second representative cluster, we perform the following: from nodes in $\mathsf{S}_{\mathsf{chosen}}$, we uniformly pick at random a value from $\{0,\cdots,\gamma'-1\}$, where γ' is a parameter in function of γ that verifies $\gamma' \leq \gamma$. The peers that output a random number equal to zero will be the only peers able to initiate the ERB protocol. We will show that this strategy will greatly decrease the communication complexity and defer its analysis to Appendix F. Note that this phase lasts for $\gamma+2$ rounds when all ERB instances terminate.

Selection Decision: At the end of the broadcast phase, the node of the clusters will have each a set containing eventually several random numbers. Note that, as ERB is a reliable broadcast primitive, we know that all honest peers in the cluster will have the same set of random numbers. Once a node in $\mathcal{P}$ receives at least $\gamma+1$ sets of random numbers, M_κ, originating from the nodes in the cluster, it will output the set M_κ as $\mathsf{S}_{\mathsf{final}}$. All honest nodes will output the same set under the assumption that there is a majority of honest nodes in the cluster. Finally, the random number equals the XOR value of all random numbers in $\mathsf{S}_{\mathsf{final}}$.

5.3 Analysis

We present the proofs for the Lemma and Theorems below in Appendix F.

Lemma 5.1. *If up to $t = \frac{N}{3}$ nodes are byzantine, then with at least $1-\mathsf{negl}(\gamma)$ probability, the representative cluster has more than γ honest nodes, and less than γ byzantine nodes.*

Theorem 5.2. Agreement: *All honest nodes eventually agree on the same common set $\mathsf{S}_{\mathsf{final}}$ in* ERNG.

Theorem 5.3. Unbiasedness: *The output of the* ERNG *protocol is an unbiased random number.*

ERNG **Performance Analysis.** Note that in ERNG, $O(\gamma)$ nodes will be chosen to form the first representative cluster and therefore run $O(\gamma)$ Multicast functions. The communication complexity of this first step is $O(\gamma^2)$. Then, among this first representative cluster, a second cluster will be composed such that all nodes of this cluster will run each an ERB instance. If the size of the second representative cluster is $O(\sqrt{\gamma})$ (as shown in Corollary F.1 in Appendix F), then the communication complexity of this step is $O(\gamma^2\cdot\sqrt{\gamma})$. Finally, the member of the first representative cluster will multicast the output of the ERB instances to all peers in $\mathcal{P}$. The communication complexity of this final step is $O(N\cdot\gamma)$. That is, overall, the communication complexity of ERNG equals $O(N\cdot\gamma+\gamma^{\frac{5}{2}})$. Based on Lemmas F.1 and F.2, if N is large such that it verifies $\gamma\in o(N)$, then we can set $\gamma\in O(\log N)$. In this case, the communication complexity and round complexity of ERNG are equal to $O(N\log N)$ and $O(\log N)$.

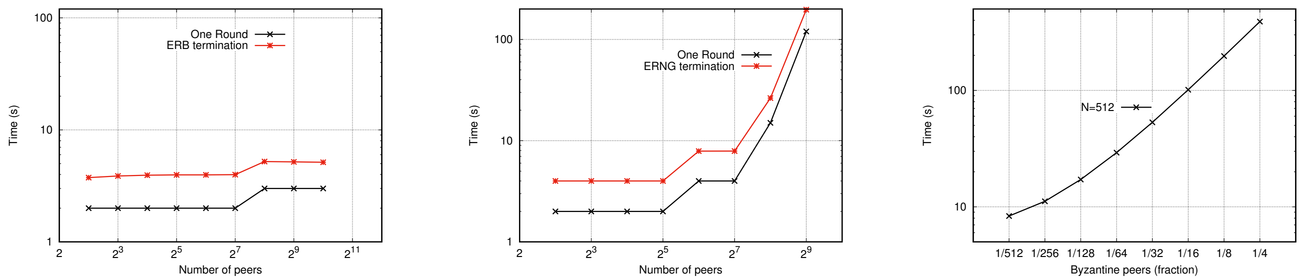
6 Evaluation

Implementation. We have implemented a prototype of ERB, unoptimized ERNG and ERNG in C/C++ using Intel SGX's Linux SDK [8]. The implementation contains 4030 lines of code (LOC) measured using CLOC tool [4]. Our prototype implementation is open source and available online [11]. We re-use the ported OpenSSL library including cryptographic utilities (`libcrypto` available with Intel SDK), to perform Diffie-Hellman key exchange and AES encryption/decryption. We use `boost` [3] library to implement the communications between any two nodes and use Google `protobuf` libraries [12] and `rapidjson` [13] to serialize transferred data.

Experimental Setup. We use the DeterLab network testbed for our experiments [6]. It consists of 40 servers running Ubuntu 14.04 with dual Intel(R) Xeon(R) hexacore processors running at 2.2 GHZ with 24 cores and 24 GB of RAM. All machines are connected and share the same link with the bandwidth of 128MBps. Every node in our protocol takes up to 1 - 800 MB memory which limits the maximum number of nodes to 2^{10} in our experiments. Due to the limited number of machines in our testbed, we have to run multiple nodes on each machine, thus we use SGX simulation mode[1] for our program and use a simulated Intel attestation service (IAS).

Evaluation Methodology. To evaluate the correctness of our protocols, we measure the round complexity (time to terminate) and communication complexity (network traffic) for ERB, unoptimized-ERNG and ERNG, by varying the number of nodes from 2^2 to 2^{10}. We have highly optimized our system to handle dynamic ports allocations to handle a larger number of nodes within one machine (order of 25 nodes per machine). Part of our results reported in this section are for the *optimistic* case where all nodes behave honestly. We evaluate the round complexity of ERB while varying the number of byzantine nodes in the network up to $\frac{1}{4}$ of the entire network composed of 512 nodes. We also compare our experiment results for the traffic size with theoretical ones to verify if they match our asymptotic analysis.

[1] All SGX features we use are supported in the simulation mode and F4 is supported in seconds.

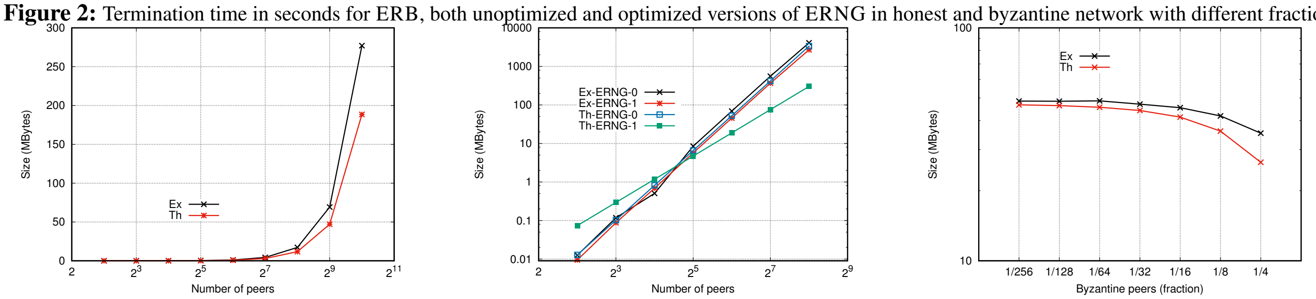


(a) Termination of ERB slightly increase with the number of peers.

(b) Termination time of ERNG in function of the number of nodes in $\mathcal{P}$.

(c) Time termination of ERB linearly increase with the number of byzantine nodes in $\mathcal{P}$.

Figure 2: Termination time in seconds for ERB, both unoptimized and optimized versions of ERNG in honest and byzantine network with different fractions.



(a) Communication of ERB in function of the number of nodes in $\mathcal{P}$.

(b) Communication of ERNG in function of the number of nodes in $\mathcal{P}$.

(c) Communication of ERB in function of different byzantine peers in $\mathcal{P}$.

Figure 3: (Th) theoretical and (Ex) experimental comparisons off network overall communication bandwidth in MB for ERB, both unoptimized (ERNG-0) and optimized (ERNG-1) versions of ERNG in honest and byzantine network with different fractions.

6.1 ERB **Evaluation**

Honest Termination: Constant Scalability. Determining the termination of ERB is essential to validate our reliable broadcast primitive. Fig. (2a) shows that the termination time, in the case of an honest initiator, is nearly equal to twice the value of one round. This validates our theoretical results where we show that ERB finishes in 2 rounds when the initiator is honest. The small increase at 2^8 is purely due to the bandwidth bottleneck of our testbed, as the nodes share the same link.

Traffic Size: Quadratic Scalability. Fig. (3a) demonstrates that the communication complexity quadratically increases in function of the number of peers in $\mathcal{P}$ (note that the x-axis is logarithmic). The message size of INIT and ACK is around 100 Bytes and 80 Bytes, respectively. For 1024 nodes in $\mathcal{P}$, the traffic size equals 277 MB. We show that this result matches our theoretical expectation.

6.2 ERNG **Evaluation**

Honest Termination: Limited Scalability. We show in Fig (2b) that ERNG termination remains slightly constant from 2^2 to 2^7 and then increases afterwards. Unfortunately, this does not reflect our theoretical findings and this is mainly due to the limitation of our testbed, namely, the upper bound on the communication link of 128MBps that all nodes have to share. For small values of peers N, the communication complexity of the unoptimized ERNG is cubic in N, while the optimized version is also (nearly) cubic for smaller values of N. Given a fixed bandwidth, this explains why the termination increases for larger values of N to reach 103 s for one instance of ERNG.

Traffic Size: Cubic Scalability. Fig. (3b) demonstrates that the communication complexity cubically increases in function of the number of peers in $\mathcal{P}$ for the unoptimized ERNG. Our theoretical results back up our experimental result. For ERNG as the bandwidth links get overflowed much faster, we limited our experiments to 512 nodes. For the optimized ERNG, small values of the number of peers in the network did not allow us to optimally select a cluster size that can guarantee w.h.p. the agreement. In this case, we fix the cluster to be $\frac{2}{3}$ of the network and we show that the traffic size decreases and has a 60% improvement over the unoptimized one. Note that this result can get much better for a larger number of peers in realistic settings. Here, we draw our theoretical curve for the ideal evaluation which can be guaranteed only for larger N.

6.3 **Byzantine case**

In Fig (2c), we show that the termination time of ERB linearly increases with the number of byzantine nodes behaving maliciously in the current instance. We gradually increase the fraction of byzantine nodes from $\frac{1}{512}$ to $\frac{1}{4}$. As a strategy of byzantine nodes, we have taken into consideration the worst-case where byzantine nodes create a chain (a byzantine sends its message to only one byzantine node each round and then gets eliminated) in order to delay the termination as much as possible. In the case of $\frac{1}{4}$ byzantine fraction, the ERB termination takes 389 seconds while it only takes 4 seconds in the honest case. For traffic size, if the number of byzantine nodes increases, the communication complexity of ERB decreases as shown in Fig. (3c). This is mainly due to the halt-on-divergence property that will eject the nodes whenever it behaves maliciously. That is when an honest node multicasts a message, the eliminated byzantine node will not acknowledge this message which greatly reduces the communication complexity. For example, for $\frac{1}{4}$ byzantine fraction in a 512-node network, the traffic size equals 35 MB, while in an honest node instance, it is equal to 69 MB, a 50% decrease.

7 Related Work

Reliable broadcast has been extensively investigated in various adversarial models. In our work, we show how Intel SGX improves the efficiency of existing protocols in these models, renewing interest in studying these protocols with SGX-based implementations.

Reliable Broadcast: Reliable broadcast has been extensively studied since the 1980s, and is closely related to the problem of byzantine agreement (BA). Several excellent surveys on the problem are available [64, 88]. Byzantine agreement can also achieve reliable broadcast [32, 35, 37, 61, 73, 76, 83, 88]. For the asynchronous network, Bracha's classic reliable broadcast protocol requires $O(N^2)$ communication complexity and tolerates up to $\frac{N}{3}$ byzantine nodes [33,34]. Cachin and Tessaro [38] leverage erasure codes to improve efficiency and reduce communication complexity. However, as the time is not bounded, messages may incur arbitrary delays, and most protocols do not guarantee terminating runs, except under some special assumptions such as sharing a "common coin" [32, 83].

Without any extra assumptions, reliable broadcast and byzantine agreement in the synchronous setting can tolerate $\frac{N}{3}$ byzantine nodes at most, and with $\min\{f+2, t+1\}$ round complexity [48, 65, 81]. Lamport *et al.* and Pease *et al.* propose protocols terminating within $t+1$ rounds and tolerating up to $\frac{N}{3}$ byzantine nodes, but with exponential communication complexity [65, 81]. Berman *et al.* achieve $O(\mathsf{poly}(N))$ communication complexity but only tolerating upto $\frac{N}{4}$ byzantine nodes [26]. Garay *et al.* later present a BA protocol terminating within $\min\{f+5, t+1\}$ rounds [53, 54].

To tolerate a larger fraction of byzantine nodes, additional assumptions are often needed. A common assumption is that of having a one-time trusted dealer that pre-deploys PKI in the infrastructure. This assumption, for instance, allows digital signatures to be used for *authentication*, wherein a message claimed to be sent by a node A can be assured to be originating from A [49, 52, 62, 65]. This weakens the capabilities of the byzantine adversary, which cannot forge messages on behalf of honest nodes. Researchers have proposed protocols to use digital signatures to boost the resilience from $\frac{N}{3}$ to $N-1$, but the communication complexity is still large, i.e., $O(\exp(N))$ and $O(N^3)$ [49, 65]. Katz *et al.* extend the work of Feldman and Micali [51] to employ authenticated channels, and present protocols tolerating $\frac{N}{2}$ byzantine nodes with $O(\mathsf{poly}(N))$ complexity [62]. Fitzi *et al.* also give an authenticated BA protocol that beats this bound ($\frac{N}{2}$) but under specific number-theoretic assumptions [52]. Abraham *et al.* provide a solution with early stopping ($\min\{f+2, t+1\}$) and polynomial complexity [18]. In this work, we use SGX features to reduce the byzantine model to the general omission model, and further propose ERB to achieve $\min\{f+2, t+2\}$ round complexity and $O(N^2)$ communication complexity.

Researchers also have proposed byzantine fault-tolerant algorithms using trusted services, such as by using trusted computing primitives, primarily focusing on making PBFT more efficient [23, 40, 42, 44, 45, 67, 70, 89]. These works have observed similar relation to crash-fault-tolerant protocols, as we have. For example, Chun *et al.* introduce an attested append-only memory (A2M) to remove the ability of adversarial replicas to equivocate without detection, which helps to increase the resilience from $\frac{N}{3}$ to $\frac{N}{2}$ [42]. However, these works have concentrated on handling asynchronous protocols with weak time assumptions like PBFT. In this paper, in contrast to previous approaches, we work on the round-based synchronous model. Our work extends these ideas to detecting and remediating failures of synchronous network assumptions (e.g. our lockstep execution and halt-on-divergence). Additionally, we investigate the use of our blind-box execution primitive in our new distributed RNG protocol which is bias-resistant, and more efficient using secure sampling for cluster creation. We leave the extension of applying our properties and primitives to asynchronous protocols for future work.

Distributed RNG: Generating common coins in a distributed manner for randomized BA in asynchronous networks can also be used for generating unbiased random numbers [27, 36, 83]. However, these protocols either require a trusted dealer to set up the initial states of different nodes or pre-distribute data to the nodes in the network. Other works employing asynchronous verifiable secret sharing (AVSS) protocols do not have the trusted dealer, but can probabilistically execute with errors [24, 32, 39, 87]. Most of these works employ some cryptographic primitives that, in most case, can be considered heavy-weight and performance unfriendly. Awerbuch *et al.* propose a solution that tolerates up to $\frac{N}{6}$ byzantine nodes, with $O(N)$ round complexity and $O(N^3)$ communication complexity [20] to generate a random number with a constant bias. Other works, such as Andrychowicz *et al.*'s one, generate a common random number based on proof of work [19] with $O(N^4)$ communication complexity, but the output can eventually be biased. Moreover, the large communication cost for most of these approaches prevents scalability to a large number of nodes. We present more efficient (with $O(N \log N)$ communication complexity) and unbiased RNG generation for the synchronous network case.

8 Conclusion

The recent availability of Intel SGX in commodity laptops and servers provides a promising research direction for advancing the area of P2P systems. Our main observation is that leveraging SGX features can restrict a byzantine model to a general-omission model in synchronous systems. We highlight that using SGX we can improve the efficiency of P2P protocols such as reliable broadcast and unbiased random number generator in synchronous settings.

References

[1] Bitcoin. https://bitcoin.org/en/, Accessed: 2017.

[2] BitTorrent. http://www.bittorrent.com/, Accessed: 2017.

[3] Boost C++ library. http://www.boost.org/, Accessed: 2017.

[4] CLOC. http://cloc.sourceforge.net/, Accessed: 2017.

[5] CrashPlan. http://www.code42.com/crashplan/, Accessed: 2017.

[6] DeterLab. https://www.isi.deterlab.net/index.php3, Accessed: 2017.

[7] Intel Software Guard Extensions. https://software.intel.com/en-us/sgx, Accessed: 2017.

[8] Intel Software Guard Extensions for Linux OS. https://01.org/intel-softwareguard-extensions, Accessed: 2017.

[9] Intel software guard extensions sdk for linux os. https://download.01.org/intel-sgx/linux-1.9/docs/Intel_SGX_SDK_Developer_Reference_Linux_1.9_Open_Source.pdf, Accessed: 2017.

[10] NIST randomness beacon. https://www.nist.gov/programs-projects/nist-randomness-beacon, Accessed: 2017.

[11] P2P using SGX. https://bitbucket.org/P2PUsingSGX/p2pusingsgx, Accessed: 2017.

[12] Protocol Buffers - Google's data interchange format. https://github.com/google/protobuf, Accessed: 2017.

[13] RapidJSON. http://rapidjson.org/, Accessed: 2017.

[14] Storj.io. http://storj.io/, Accessed: 2017.

[15] Symform. http://www.symform.com/, Accessed: 2017.

[16] Tor. https://www.torproject.org/, Accessed: 2017.

[17] True Random Number Service. https://www.random.org/, Accessed: 2017.

[18] I. Abraham and D. Dolev. Byzantine agreement with optimal early stopping, optimal resilience and polynomial complexity. In *STOC*, 2015.

[19] M. Andrychowicz and S. Dziembowski. Distributed cryptography based on the proofs of work. *IACR*, 2014.

[20] B. Awerbuch and C. Scheideler. Robust random number generation for peer-to-peer systems. In *PODC*, 2006.

[21] L. Babai. Trading group theory for randomness. In *STOC*, 1985.

[22] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource routing attacks against tor. In *WPES*, 2007.

[23] J. Behl, T. Distler, and R. Kapitza. Hybrids on steroids: Sgx-based high performance bft. In *EuroSys*, 2017.

[24] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In *PODC*, 1994.

[25] I. Bentov, A. Gabizon, and D. Zuckerman. Bitcoin beacon. *arXiv*, 2016.

[26] P. Berman and J. A. Garay. Cloture votes: n/4-resilient distributed consensus in t+ 1 rounds. *STOC*, 1993.

[27] P. Berman and J. A. Garay. Randomized distributed agreement revisited. In *FTCS*, 1993.

[28] P. Berman, J. A. Garay, and K. J. Perry. Optimal early stopping in distributed consensus. In *WDAG*, 1992.

[29] J. Bonneau, J. Clark, and S. Goldfeder. On bitcoin as a public randomness source. *IACR*, 2015.

[30] N. Borisov. Computational puzzles as sybil defenses. In *P2P*, 2006.

[31] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. Denial of service or denial of security? In *CCS*, 2007.

[32] G. Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *PODC*, 1984.

[33] G. Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 1987.

[34] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *JACM*, 1985.

[35] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, 2001.

[36] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptology*, 2005.

[37] C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the internet. In *DSN*, 2002.

[38] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In *SRDS*, 2005.

[39] R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC*, 1993.

[40] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, 1999.

[41] T. D. Chandra and S. Toueg. Time and message efficient reliable broadcasts. In *WDAG*, 1990.

[42] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested append-only memory: Making adversaries stick to their word. In *OSR*, 2007.

[43] J. Clark and U. Hengartner. On the use of financial data as a random beacon. *EVT/WOTE*, 2010.

[44] M. Correia, N. F. Neves, and P. Verissimo. How to tolerate half less one byzantine nodes in practical distributed systems. In *SRDS*, 2004.

[45] M. Correia, P. Veríssimo, and N. F. Neves. The design of a cots real-time distributed security kernel. In *EDCC*, 2002.

[46] V. Costan and S. Devadas. Intel SGX explained. *IACR*, 2016.

[47] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *JPDC*, 1989.

[48] D. Dolev, R. Reischuk, and H. R. Strong. Early stopping in byzantine agreement. *JACM*, 1990.

[49] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SICOMP*, 1983.

[50] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *CACM*, 1985.

[51] P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SICOMP*, 1997.

[52] M. Fitzi and J. A. Garay. Efficient player-optimal protocols for strong and differential consensus. In *PODC*, 2003.

[53] J. A. Garay and Y. Moses. Fully polynomial byzantine agreement in t+ 1 rounds. In *STOC*, 1993.

[54] J. A. Garay and Y. Moses. Fully polynomial byzantine agreement for processors in rounds. *SICOMP*, 1998.

[55] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT*, 1999.

[56] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology*, 2007.

[57] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In *STOC*, 1986.

[58] R. Guerraoui, F. Huc, and A.-M. Kermarrec. Highly dynamic distributed computing with byzantine failures. In *PODC*, 2013.

[59] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In *Distributed Systems (2nd Ed.)*, 1993.

[60] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security*, 2015.

[61] B. M. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. *TALG*, 2010.

[62] J. Katz and C.-Y. Koo. On expected constant-round protocols for byzantine agreement. In *CRYPTO*, 2006.

[63] J. Katz and Y. Lindell. *Introduction to modern cryptography*. 2014.

[64] V. King and J. Saia. Scalable byzantine computation. *SIGACT News*, 2010.

[65] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *TOPLAS*, 1982.

[66] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado. Inferring fine-grained control flow inside sgx enclaves with branch shadowing. *arXiv*, 2016.

[67] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *NSDI*, 2009.

[68] F. Li, P. Mittal, M. Caesar, and N. Borisov. Sybilcontrol: practical sybil defense with computational puzzles. In *STC*, 2012.

[69] Y. Lindell, A. Lysyanskaya, and T. Rabin. On the composition of authenticated byzantine agreement. *JACM*, 2006.

[70] J. Liu, W. Li, G. O. Karame, and N. Asokan. Scalable byzantine consensus via hardware-assisted secret sharing. *arXiv*, 2016.

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

[72] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun. Rote: Rollback protection for trusted execution. *IACR*, 2017.

[73] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of bft protocols. In *CCS*, 2016.

[74] Ming-Wei-Shih, S. Lee, T. Kim, and M. Peinado. T-sgx: Eradicating controlled-channel attacks against enclave programs. 2017.

[75] T. Moran and M. Naor. Split-ballot voting: everlasting privacy with distributed trust. *TISSEC*, 2010.

[76] A. Mostefaoui, H. Moumen, and M. Raynal. Signature-free asynchronous Byzantine consensus with $t < \frac{n}{3}$ and $O(n^2)$ messages. In *PODC*, 2014.

[77] K. Nayak, S. Kumar, A. Miller, and E. Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *EuroS&P*, 2015.

[78] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious multi-party machine learning on trusted processors. In *USENIX Security*, 2016.

[79] P. R. Parvédy and M. Raynal. Optimal early stopping uniform consensus in synchronous systems with process omission failures. In *SPAA*, 2004.

[80] R. Pass, E. Shi, and F. Tramèr. Formal abstractions for attested execution secure processors. *IACR*, 2016.

[81] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JACM*, 1980.

[82] K. J. Perry and S. Toueg. Distributed agreement in the presence of processor and communication faults. *TSE*, 1986.

[83] M. O. Rabin. Randomized byzantine generals. In *FOCS*, 1983.

[84] M. O. Rabin. Transaction protection by beacons. *JCSS*, 1983.

[85] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. In *IPTPS*, 2003.

[86] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena. Preventing page faults from telling your secrets. In *ASIACCS*, 2016.

[87] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford. Scalable bias-resistant distributed randomness. *IACR*, 2016.

[88] V. Vaikuntanathan. *Randomized algorithms for reliable broadcast*. PhD thesis, 2009.

[89] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo. Efficient byzantine fault-tolerance. *TC*, 2013.

[90] Y. Xu, W. Cui, and M. Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Security and Privacy*, 2015.

A Primitives and Formal Definitions

In this section, we first start by formally defining the syntax of the communication protocol between two peers, that we denote by Peer channel. Using this definition, we next define various failure modes and primitives. Using SGX, we assume that *execution integrity (P1)* is enforced. We then show that the following properties: *message integrity & authenticity (P2)*, *blind-box computation property (P3)* can be emulated based on the Blinded channel, executing on a particular program. Then we go ahead and formally define the *halt-on-divergence (P4)* property for any program running between two peers. Finally, we show how to reduce the byzantine model to a model where a peer can only replay, omit and delay, dubbed ROD , given that a Blinded channel exists.

A.1 Peer **Channel**

Abstractly, a peer can be considered as the composition of two entities: an Enclave and an OS. The OS models the untrusted entity including the operating system and memory. It has access to all the system resources such as file system, network and others. The OS can arbitrarily invoke an enclave program and start its execution. The Enclave models the isolated memory space that loads the program and executes it securely. Thus, Enclave corresponds to the trusted entity of a peer. A concurrent work provides a formal study to show that SGX enclaves can be considered as a trusted entity [80]. The Enclave of the two Peers can interact with each other via their OSs. We formally define a Peer channel as a protocol, $\mathsf{Peer}^{\mathsf{ch}}$, between a sender $\mathsf{Peer}_s = (\mathsf{Enclave}_s, \mathsf{OS}_s)$ and a receiver $\mathsf{Peer}_r = (\mathsf{Enclave}_r, \mathsf{OS}_r)$. A Peer channel can be seen as a generalization of the traditional secure communication channel between two parties. The main difference is that the definition of $\mathsf{Peer}^{\mathsf{ch}}$ protocol is augmented with the program π running within the trusted Enclave. Before defining the Peer channel, we first provide a definition of a program π.

Definition A.1. (Program.) *A program π is a sequence of instructions i.e., $\pi = (\pi_1, \cdots, \pi_n)$ such that the i^{th} instruction π_i takes as an input the state st_i and a message m_i and outputs a message m_{i+1} along with an updated state st_{i+1}. By convention, we write for all $m_i \in \{0,1\}^*$, $(\mathsf{st}_{i+1}, m_{i+1}) \leftarrow \pi_i(\mathsf{st}_i, m_i)$. The initial state is st_1.*

Based on the above definition, for a program π with n instructions the output $\mathtt{out}$ of π is $(\mathsf{st}_{out}, \mathtt{out}) \leftarrow \pi_n(\mathsf{st}_n, m_n)$ where st_{out} is the final state of the program. We denote the set of all such programs by Π. Note that, in a program π, an instruction with $\perp$ state as input always outputs $\perp$ i.e., $(\perp, \perp) \leftarrow \pi_i(\perp, m_i)$. Hence, if $\exists i$ such that $(\perp, \perp) \leftarrow \pi_i(\mathsf{st}_i, m_i)$, then the output of the program π is always $\perp$.

Definition A.2. (Program Transcript.) *Let $\pi \in \Pi$ and messages $m_1, \cdots, m_n \in \{0,1\}^*$ such that $\mathbf{m} = (\mathbf{m_i})_{\mathbf{i} \in [\mathbf{n}]}$, for all initial states $\mathsf{st}_1 \in \{0,1\}^*$ and for all $i \geq 1$ such that $(\mathsf{st}_{i+1}, m_{i+1}) \leftarrow \pi_i(\mathsf{st}_i, m_i)$, a transcript of π with inputs st_1 and $\mathbf{m}$ denoted by $\mathsf{trans}_\pi^{\mathbf{m}}$ equals:*

$$\mathsf{trans}_\pi^{\mathbf{m}} = \big(\pi_1(\mathsf{st}_1, m_1), \cdots, \pi_i(\mathsf{st}_i, m_i), \cdots, \pi_n(\mathsf{st}_n, m_n)\big).$$

Definition A.3. (Transcript Types.) *Let $\pi \in \Pi$ and $\mathsf{trans}_\pi^{\mathbf{m}}$ its transcript for a fixed message $\mathbf{m} = (\mathbf{m_i})_{\mathbf{i} \in [\mathbf{n}]}$. We say that the transcript is:*

- valid, *if $\forall i \in [n]$, $\mathsf{st}_i \neq \perp$,*
- invalid, *if $\exists i \in [n]$, $\mathsf{st}_i = \perp$,*

where $(\mathsf{st}_i, m_i) \leftarrow \pi_{i-1}(\mathsf{st}_{i-1}, m_{i-1})$.

We denote by $\mathcal{V}_\pi$ and I_π, the set of all n-messages for which the transcript is valid and invalid, respectively.

Definition A.4. (Peer **Channel.**) *Given $\pi_s, \pi_r \in \Pi$ are programs executing in $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ with st_s and st_r as respective initial states. A Peer channel between $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ is tuple of four possibly interactive algorithms $\mathsf{Peer}^{\mathsf{ch}} = (\mathsf{Init}, \mathsf{Write}, \mathsf{Transfer}, \mathsf{Read})$ such that:*

- $(K_s, K_r) \leftarrow \mathsf{Init}\big((1^k, \mathsf{st}_s, \pi_s), (1^k, \mathsf{st}_r, \pi_r)\big)$*: is a probabilistic interactive algorithm between $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$. $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ take as inputs a security parameter k, a program π_s and π_r and the initial state st_s and st_r, and outputs keys K_s and K_r for the sender and receiver, respectively.*
- $(\mathsf{st}'_s, \mathsf{data}'_s) \leftarrow \mathsf{Write}\big((\mathsf{st}_s, K_s, m, \pi_s), \mathsf{data}_s\big)$*: is a probabilistic interactive algorithm between $\mathsf{Enclave}_s$ and OS_s. $\mathsf{Enclave}_s$ has as inputs a state st_s, a key K_s, a message m and a program π_s; the OS_s has as the input a data block data_s; the algorithm outputs an updated state st'_s for $\mathsf{Enclave}_s$ and the updated data block data'_s for OS_s.*
- $(\mathtt{null}, \mathsf{data}'_r) \leftarrow \mathsf{Transfer}\big(\mathsf{data}'_s, \mathsf{data}_r\big)$*: is a probabilistic interactive algorithm between OS_s and OS_r that takes as input the data block data'_s and data_r respectively, and outputs $\mathtt{null}$ for OS_s and an updated data block data'_r for OS_r*
- $\big((\mathsf{st}'_r, \mathrm{r}), \mathtt{null}\big) \leftarrow \mathsf{Read}\big((\mathsf{st}_r, K_r, \pi_r), \mathsf{data}'_r\big)$*: is a probabilistic interactive algorithm between $\mathsf{Enclave}_r$ and OS_r. $\mathsf{Enclave}_r$ has as inputs a state st_r, a key K_r and the program π_r; the OS_r has as the input a data block data'_r; the algorithm outputs an updated state st'_r and a response* r *for $\mathsf{Enclave}_r$ and $\mathtt{null}$ for OS_r.*

When $\pi_s = \pi_r = \pi$, we can write $\mathsf{Peer}_\pi^{\mathsf{ch}}$ to denote that $\mathsf{Peer}^{\mathsf{ch}}$ is parametrized with the program π.

A.2 Failure Modes

We define four progressively stronger failure modes: *honest*, *general omission*, ROD and *byzantine* modes of $\mathsf{Peer}^{\mathsf{ch}}$. Here we introduce a ROD model as an intermediate model, wherein the adversary can only a) Replay b) Omit c) or Delay messages during a protocol, or follow it as prescribed. We particularly focus on the sender behavior for simplicity, but our definition extends to both sender and receiver. Note that to capture *delay*, we super-script the Transfer algorithm with Δ such that $\mathsf{Transfer}^{\Delta}$, to denote that the Transfer can take time Δ to complete. We denote by Replay_{π}, the set containing all values generated by Write in polynomial number of executions of program π running concurrently or earlier in time [69].

Definition A.5. *(***Failure Modes.***) Given a* Peer *channel* $\mathsf{Peer}^{\mathsf{ch}} = (\mathsf{Init}, \mathsf{Write}, \mathsf{Transfer}, \mathsf{Read})$ *between two* Peers, Peer_r *and* Peer_s *, for all security parameters* $k \in \mathbb{N}$ *and for all programs* $\pi, \pi_s, \pi_r, \pi' \in \Pi$ *such that*

- $(K_s, K_r) \leftarrow \mathsf{Init}\big((1^k, \mathsf{st}_s, \pi_s), (1^k, \mathsf{st}_r, \pi_r)\big)$.

For all messages $m \in \{0,1\}^*$*, for all state* $\mathsf{st}_s \in \{0,1\}^*$*, for all data block* $\mathsf{data}_s, \mathsf{data}_r \in \{0,1\}^*$ *such that* $|m| \leq |\mathsf{data}_s|$ *and* $|\mathsf{data}_s| = |\mathsf{data}_r|$,

- $(\mathsf{st}'_s, \mathsf{data}'_s) \leftarrow \mathsf{Write}\big((\mathsf{st}_s, K_s, m, \pi_s), \mathsf{data}_s\big)$*;*
- $(\bot, \mathsf{data}'_r) \leftarrow \mathsf{Transfer}^{\Delta}\big(\mathsf{data}'_s, \mathsf{data}_r\big)$*;*
- $\big((\mathsf{st}'_r, \mathrm{r}), \bot\big) \leftarrow \mathsf{Read}\big((\mathsf{st}_r, K_r, \pi_r), \mathsf{data}'_r\big)$.

We say that

- $\mathsf{Peer}^{\mathsf{ch}}$ *is in an* **honest mode***, if we have*
  - $\mathsf{data}'_s = \mathsf{data}'_r$ *and,*
  - $\pi_s = \pi$*,*
  - Δ *is bounded.*
- $\mathsf{Peer}^{\mathsf{ch}}$ *is in a* **general omission mode** *, if we have*
  - $\mathsf{data}'_s = \begin{cases} \bot & or, \\ \mathsf{data}'_r & ; \end{cases}$
  - $\pi_s = \pi$*,*
  - Δ *is bounded.*
- $\mathsf{Peer}^{\mathsf{ch}}$ *is in a* **ROD mode***, if we have*
  - $\mathsf{data}'_s = \begin{cases} \bot & or, \\ \mathsf{data} \leftarrow \mathsf{Replay}_{\pi} & or, \\ \mathsf{data}'_r & ; \end{cases}$
  - $\pi_s = \pi$*,*
  - $\Delta < \infty$.
- $\mathsf{Peer}^{\mathsf{ch}}$ *is in a* **byzantine mode***, if we have*
  - $\mathsf{data}'_s = \begin{cases} \phi(\mathsf{data}'_r) \text{ where} & \\ \phi \in \{\{0,1\}^* \rightarrow \{0,1\}^*\} & or, \\ \mathsf{data} \leftarrow \mathsf{Replay}_{\pi} & or, \\ \bot; & \end{cases}$
  - $\pi_s = \begin{cases} \pi & or, \\ \pi' & \text{where } \pi' \neq \pi; \end{cases}$
  - $\Delta < \infty$

A.3 Core Primitives

We define two primitives: a) Blinded channels and b) halt-on-divergence. Theorem A.2, below, uses the Blinded channel primitive to demonstrate that byzantine mode reduces to the ROD mode. As shown in Section 4, we can further leverage additional SGX features, namely properties (P5) and (P6), to reduce the ROD model to the general-omission model. Informally, a Blinded channel guarantees confidentiality and integrity of a message over a Peer channel $\mathsf{Peer}^{\mathsf{ch}} = (\mathsf{Init}, \mathsf{Write}, \mathsf{Transfer}, \mathsf{Read})$.

Definition A.6. **(Blinded Channels.)** *We say that* $\mathsf{Peer}^{\mathsf{ch}}$ *is* Blinded *if for all p.p.t adversaries* $\mathcal{A}$ *we have:*

$$\Pr[\mathsf{Exp}^{\mathsf{EX}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda), \textit{and},$$

$$\Pr[\mathsf{Exp}^{\mathsf{Priv}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda), \textit{and},$$

$$\Pr[\mathsf{Exp}^{\mathsf{Auth}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{EX}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$, $\mathsf{Exp}^{\mathsf{Priv}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$, $\mathsf{Exp}^{\mathsf{Auth}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$ *are:*

$\underline{\mathsf{Exp}^{\mathsf{EX}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)}$:

- two parties generate keys K_s and K_r such that $(K_s, K_r) \leftarrow \mathsf{Init}\big(1^k, \pi\big)$. The entire interaction between both of the parties is saved in a transcript $\mathcal{T}$;
- compute $b \xleftarrow{\$} \{0,1\}$, if $b = 0$, then output $K = (K_s, K_r) \xleftarrow{\$} \{0,1\}^k$, otherwise output $K = (K_s, K_r) \leftarrow \mathsf{Init}\big(1^k, \pi\big)$.
- Given K and $\mathcal{T}$, $\mathcal{A}$ outputs b' and wins if $b' = b$.

$\underline{\mathsf{Exp}^{\mathsf{Priv}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)}$:

- generate keys K_s and K_r such that $(K_s, K_s) \leftarrow \mathsf{Init}\big(1^k, \pi\big)$;
- $\mathcal{A}$ has access to $O^{\mathsf{write}(K_s,.)}(.)$ and $O^{\mathsf{read}(K_r,.)}(.)$;
- $\mathcal{A}$ chooses two equal-length messages m_0 and m_1;
- compute $\mathsf{Write}\big((\mathsf{st}_s, K_s, m_b, \pi), \mathsf{data}_s\big)$ where $b \xleftarrow{\$} \{0,1\}$, and output data;

- $\mathcal{A}$ has again access to $O^{\mathsf{write}(K_s,.)}(.)$ and $O^{\mathsf{read}(K_r,.)}(.)$;
- $\mathcal{A}$ outputs b', if $b' = b$, the experiment outputs 1, and 0 otherwise.

$\underline{\mathsf{Exp}^{\mathsf{Auth}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)}$:

- generate keys K_s and K_r such that $(K_s, K_r) \leftarrow \mathsf{Init}(1^k, \pi)$;
- $\mathcal{A}$ has access to $O^{\mathsf{write}}(K_s,.)$. $\mathcal{A}$ queries a polynomial number of messages m and eventually outputs ct, we denote by Q the set of all queries that $\mathcal{A}$ sent to the oracle;
- Given ct, $O^{\mathsf{write}}(K_s,.)$ outputs r. If $m \notin Q$ and $\mathrm{r} \neq \perp$. $\mathcal{A}$ outputs 1.

Attaching a program π while defining a $\mathsf{Peer}^{\mathsf{ch}}$ enables us to introduce the *halt-on-divergence* primitive as follows.

Definition A.7. (Halt-on-divergence.) *Let $\pi \in \Pi$ be a program and $\mathsf{trans}^{\mathbf{m}}_{\pi}$ its transcript for a fixed n-messages* $\mathbf{m}$*, we say that* $\mathsf{Peer}^{\mathsf{ch}}_{\pi}$ halts on-divergence *if $\mathsf{trans}^{\mathbf{m}}_{\pi}$ is invalid, i.e.,* $\mathbf{m} \in I_{\pi}$

A.4 Implementing Blinded Channel using SGX

We show how we build a $\mathsf{Peer}^{\mathsf{ch}}$ channel using SGX where $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ are trusted entities. Theorem A.1 shows that such a $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ channel is a Blinded channel, and therefore enforces both (P2) and (P3) propertie. In particular, we consider that there is a KeyEx_{π} protocol between $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ that is used to generate a session key for a program π. Whenever there is a new program the key has to be re-generated. The key exchange protocol can be instantiated using Diffie-Hellman key exchange, referring to [63] Chapter 9. We use SGX remote attestation to verify that both parties run their code inside an Enclave. While this step is neither required nor captured in the $\mathsf{Peer}^{\mathsf{ch}}$ definition, it is mandatory to guarantee our *execution integrity (P1)*. We detail our instantiation in Figure 4, in our case, we consider that $\pi_r = \pi_s = \pi$. We denote by *parse* and *compute* the actions of decoding a string and running a particular algorithm, respectively.

Theorem A.1. *If* KeyEx *is a secure key exchange protocol,* SKE *is CPA secure encryption schemes,* MAC *a secure message authentication code, then* $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ *is a* Blinded Peer *channel.*

Proof Sketch. First, we want to show that the Init algorithm is a secure key exchange. Note that both parties run *two* instances of a KeyEx protocol to generate two session keys. That is, if KeyEx is a secure key exchange then $\Pr[\mathsf{Exp}^{\mathsf{EX}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(k) = 1] \leq \frac{1}{2} + \mathsf{negl}(k)$.

Second, we need to show that $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ is a secure communication channel. Note that, we use the variant *encrypt-then-mac*

Let $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a private encryption scheme, $\mathsf{MAC} = (\mathsf{Gen}, \mathsf{Auth}, \mathsf{Vrfy})$ be a message authentication code, KeyEx a key exchange algorithm, and H be a hash function. We define $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}} = (\mathsf{Init}, \mathsf{Write}, \mathsf{Transfer}, \mathsf{Read})$ as follows:

- $\mathsf{Init}((1^k, \mathsf{st}_s, \pi), (1^k, \mathsf{st}_r, \pi))$:
  1. $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ fetch the hardware-embedded private keys $\mathrm{sk}_s, \mathrm{sk}_r$ from $\mathsf{st}_s, \mathsf{st}_r$, respectively;
  2. compute $(\mathsf{key}_1, \mathsf{key}_2) \leftarrow \mathsf{KeyEx}_{\pi}(\mathrm{sk}_s, \mathrm{sk}_r)$;
  3. $\mathsf{Enclave}_s$ outputs $K_s = (\mathsf{key}_1, \mathsf{key}_2)$ and $\mathsf{Enclave}_r$ outputs $K_r = (\mathsf{key}_1, \mathsf{key}_2)$.
- $\mathsf{Write}((\mathsf{st}_s, K_s, m, \pi), \mathsf{data}_s)$:
  1. parse $K_s = (\mathsf{key}_1, \mathsf{key}_2, \mathrm{sk}_s)$;
  2. set $(\mathsf{st}'_s, \mathsf{val}) \leftarrow \pi(\mathsf{st}_s, m)$
  3. compute $\mathsf{ct}_1 = \mathsf{SKE.Enc}(\mathsf{key}_1, \langle \mathsf{val}, H(\pi) \rangle)$ and $\mathsf{ct}_2 = \mathsf{MAC.Auth}(\mathsf{key}_2, \mathsf{ct}_1)$;
  4. set $\mathsf{data}_s = (\mathsf{ct}_1, \mathsf{ct}_2)$
  5. $\mathsf{Enclave}_s$ outputs st'_s and OS_s outputs $\mathsf{data}'_s = \mathsf{data}_s$.
- $\mathsf{Transfer}(\mathsf{data}'_s, \mathsf{data}_r)$:
  1. OS_r sets $\mathsf{data}_r = \mathsf{data}'_s$;
  2. OS_s outputs $\perp$ and OS_r outputs $\mathsf{data}'_r = \mathsf{data}_r$.
- $\mathsf{Read}((\mathsf{st}_r, K_r, \pi), \mathsf{data}'_r)$:
  1. parse $K_r = (\mathsf{key}_1, \mathsf{key}_2, \mathrm{sk}_r)$ and $\mathsf{data}'_r = (\mathsf{ct}_1, \mathsf{ct}_2)$;
  2. if $\mathsf{MAC.Vrfy}(\mathsf{key}_2, \mathsf{ct}_1) := \mathsf{ct}_2$ and $\mathsf{st}_r \neq \perp$, $\mathsf{Enclave}_r$ computes
     - $\langle \mathrm{r}_1, \mathrm{r}_2 \rangle = \mathsf{SKE.Dec}(\mathsf{key}_1, \mathsf{ct}_1)$;
     - if $\mathrm{r}_2 = H(\pi)$, then compute $(\mathsf{st}'_r, \mathrm{r}) \leftarrow \pi(\mathsf{st}_r, \mathrm{r}_1)$, output $(\mathsf{st}_r, \perp)$ otherwise.
  3. if $\mathsf{MAC.Vrfy}(\mathsf{key}_2, \mathsf{ct}_1) \neq \mathsf{ct}_2$ or $\mathsf{st}_r = \perp$, $\mathsf{Enclave}_r$ outputs $\mathrm{r} = \perp$ and $\mathsf{st}'_r = \mathsf{st}_r$

Figure 4: $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$: SGX-based Peer channel.

which is shown in [63] Chapter 9 to provide a secure communication channel if SKE is CPA secure and MAC a secure message authentication. This ends our proof sketch. □

Theorem A.2. *Assuming that* $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ *is a* Blinded *channel, then* $\mathsf{Peer}^{\mathsf{ch}}$ *in byzantine is equivalent to* $\mathsf{Peer}^{\mathsf{ch}}$ *in* ROD *mode.*

Proof Sketch. For clarity, we assume that the sender is byzantine while the receiver is not. We can apply an analogous proof for the remaining combinations as well. To prove the theorem, we need to show that the view of the honest node in the ROD and byzantine modes are the same w.h.p. under the assumption that $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ is a Blinded Peer channel. For this, it is sufficient to show the following two steps: first, that any forged message for any $\phi \in \{\{0,1\}^* \to \{0,1\}^*\} \setminus \{\mathcal{C}\}$ will not change the state of the receiver st_r, i.e., that the forged message is equivalent to receiving nothing, $\perp$, where $\mathcal{C}$ is the set composed of all functions that maps $\mathsf{data}_r^{'}$ to one of the messages in $\mathsf{Replay}_\pi \bigcup \{\mathsf{data}_r^{'}\}$. Second, we need to show that, for any valid data $\mathsf{data}_s^{'}$ output by Write, the receiver state will not change if $\pi_s \neq \pi_r$ (recall that we are assuming the receiver honest and in this case means that $\pi_r = \pi$). We detail below the two steps of the reduction:

Step 1. If $\mathsf{data}_s^{'} = \phi(\mathsf{data}_r^{'})$ where $\phi \in \{\{0,1\}^* \to \{0,1\}^*\} \setminus \{\mathcal{C}\}$ such that $\langle \mathsf{ct}_1, \mathsf{ct}_2 \rangle = \mathsf{data}_s^{'}$. Then, we have that $\Pr[\mathsf{MAC.Vrfy}(\mathsf{key}_2, \mathsf{ct}_1) \neq \mathsf{ct}_2] \geq 1 - \mathsf{negl}(k)$ under the assumption that $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ is a Blinded channel. Based on the $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ in Figure 4, if $\mathsf{MAC.Vrfy}(\mathsf{key}_2, \mathsf{ct}_1) \neq \mathsf{ct}_2$, then $\mathsf{st}_r' = \mathsf{st}_r$ w.h.p. Note that this is valid for any program π_s. The view of the receiver is now equal:

- $$\mathsf{data}_s^{'} = \begin{cases} \mathsf{data}_r^{'} & \forall \pi_s \\ \mathsf{data} \leftarrow \mathsf{Replay}_{\pi_s} & \forall \pi_s \\ \perp \end{cases}$$

Step 2. Now, if the node is running a new program $\pi_s \neq \pi$ such that $(\mathsf{st}_s', \mathsf{data}_s^{'}) \leftarrow \mathsf{Write}\big((\mathsf{st}_s, K_s, m, \pi_s), \mathsf{data}_s\big)$. In this case, $\mathsf{data}_r^{'} = \mathsf{data}_s^{'} = \langle \mathsf{ct}_1, \mathsf{ct}_2 \rangle$. However, based on collision-resistance assumption of the hash function H, the malicious node cannot find any program π_s such that $H(\pi_s) = H(\pi)$.[2] In this case, if $H(\pi_s) \neq H(\pi)$, then based on the $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ protocol, $\mathsf{st}_r' = \mathsf{st}_r$, i.e., the state of the receiver does not change, which is therefore equivalent to receiving nothing, $\perp$. the view of the receiver is then equal:

- $$\mathsf{data}_s^{'} = \begin{cases} \mathsf{data}_r^{'} & \text{for } \pi_s = \pi \\ \mathsf{data} \leftarrow \mathsf{Replay}_{\pi_s} & \text{for } \pi_s = \pi \\ \perp \end{cases}$$

Finally, we emphasize that the delay constraint ($\Delta < \infty$) remains valid for both byzantine and ROD modes. Note that this final view is exactly the same of the ROD model. Note that the same holds when we consider the receiver byzantine, or both sender and receiver byzantine. This concludes our proof. □

[2]This can also be done by signing the program for every message output by the SGX-enabled program.

B Rethinking Reliable Broadcast Protocols

In this section, we explain the shortcomings of classic protocols for reliable broadcast. Reliable broadcast or byzantine generals problem is formally defined in Definition 2.1. The crux of such protocol is that all honest nodes eventually agree on the same value, which is the one proposed by the sender (or initiator) if the initiator is honest. Reliable broadcast was first proposed by Lamport *et al.* in 1982, which has $O(N^t)$ message complexity and $t+1$ round complexity. The proposed protocol was also resilient upto $\frac{N}{3}$ byzantine nodes [65]. Since 1980s, reliable broadcast has been extensively studied and various protocols have been developed, which are well summarized in several excellent survey papers [64, 88]. As byzantine nodes can behave arbitrarily, these protocols have to use different techniques to prevent the impact of the proposed biased values by the byzantine nodes, which generally leads to high (like exponential or polynomial) message complexity. Moreover, it has been shown that the optimal resilience cannot exceed third the size of the network [81]. To reduce communication complexity and increase resilience, several ways have been proposed, and using digital signatures is the primary one.

B.1 Digital Signature Schemes

Using digital signatures denotes that a node appends its signature (signed with its private key) to every message it sends. This guarantees the integrity and authenticity of the message, which can be easily verified by the other nodes using the sender's public key. It is well known that no nodes can forge the signature of another node w.h.p. This results in restricting the behavior of byzantine nodes, which can in this case only omit to relay messages, or construct different values as an initiator. We present a reliable broadcast protocol using digital signatures in Algorithm 4 adapted from Lamport *et al.*'s work [65].

In RB_{sig}, each node signs every message it multicasts. In the first round, the initiator sends a signed message to the other nodes. Then for any round rnd, a node that receives a *valid* message will sign and forward it in the next round. A message received by a node id_i in round rnd is valid if it contains signatures from rnd different nodes except id_i. In Algorithm 4, we use $[m : \mathsf{id}_1 : \mathsf{id}_2 : \ldots : \mathsf{id}_j]$ to denote a message, in which m is the value signed by the initiator id_1 and $[m : \mathsf{id}_1]$ is signed by id_2, and so on. This means that id_1 sent the signed message $[m : \mathsf{id}_1]$ to a node id_2 in the first round, and id_2 sent $[m : \mathsf{id}_1 : \mathsf{id}_2]$ in the second round, until id_j sent the signed message $[m : \mathsf{id}_1 : \mathsf{id}_2 : \ldots \mathsf{id}_j]$ in the rnd^{th} round.

Algorithm 4: RB_{sig}: Reliable broadcast protocol using digital signatures (for a node id_i with the initiator $\mathsf{id}_{\mathsf{init}}$ sending a message m).

Input: A P2P network $\mathcal{P}$ composed N nodes, a message m for the initiator $\mathsf{id}_{\mathsf{init}}$
Output: A message $\hat{m}$

```
• initialization: m̂ ← ⊥; S_m ← ∅; rnd ← 1
• for rnd ≤ t+1 do
      if rnd = 1 and id_i = id_init then
          m̂ ← m
          Multicast [m : id_init] to all the other nodes
      end
    • upon receiving [m' : id_init] from id_init:
      S_m ← {m'}
      Multicast [m' : id_init : id_i] to all nodes except id_init, id_i in round
      rnd + 1
    • upon receiving [m' : id_init : id_1 : ... : id_j] from peer id_j:
      if m' ∉ S_m then
          S_m ← S_m ∪ {m'}
          if j < t+1 then
              Multicast [m' : id_init : id_1 : ... : id_j : id_i] to all nodes
              except id_init, ..., id_i in round rnd + 1
          end
      end
    • rnd ← rnd + 1;
  end
  if rnd > t+1 then
      if |S_m| = 1 then
          m̂ ← m where S_m = {m}
      end
      return m̂
  end
```

In RB_{sig}, the initiator id_{init} signs and sends its value to every node in the first round. If any node receives the message, it stores the value in S_m, signs and sends it to the other nodes for the second round. For round $\mathsf{rnd} < t+1$, every node receives a valid message from other node. In the case where the received value does not belong to S_m, then the node adds the value to S_m and multicasts the signed message to the other nodes. After $t+1$ rounds, every node verifies whether S_m consists of a unique value $\hat{m}$, if that holds then the node outputs $\hat{m}$, otherwise he output is the default value $\perp$.

Based on digital signatures property, the byzantine nodes cannot forge a honest-like message. Therefore, every honest node only requires one valid message sent from either one byzantine or one honest node to determine the value from the initiator. If the initiator is honest, every honest node will receive the correct value from the initiator during the first round, and will discard invalid messages forged by byzantine nodes for the remaining rounds. If the initiator is byzantine, it can send different values to different honest nodes to bias the result. To ensure the validity of the message, after $t+1$ rounds, at least one signature in the message is from an honest node, and the honest node will broadcast the signed message to the other honest nodes, thus all honest nodes will receive the same message. Eventually, all honest nodes received the same set of values. If multiple values are received, all honest nodes will agree on a default value, otherwise they agree on the only received value.

Using digital signatures improves network's resilience from $\frac{N}{3}$ to $N-1$, but communication complexity remains the same $O(N^t)$. Later, an optimized algorithm using digital signatures was proposed to reduce communication complexity to $O(N^3)$ [49]. At a higher level, this improvement is achieved through a new strategy that only retransmits values which have not been previously sent. Even in this case, every node has to relay $O(N)$ messages and a message can contain $O(N)$ signatures, which results in $O(N^3)$ communication complexity for the protocol. Meanwhile, the verification of $O(N^2)$ signatures may lead to a non-negligible performance cost for the honest nodes, especially when the byzantine nodes construct and send enormous number of invalid messages to the honest nodes.

Efficiency Improvement. In the following, we discuss how our properties can lead to better asymptotics. First, by enforcing P1 - P3 and P5 - P6, we confine the byzantine nodes into the general-omission model only allowing to omit messages. We can further use P3, and secure channels in particular, to guarantee the confidentiality of the transmitted messages. In this way, when a node relays a message to the others in this model, it can append its identity instead of signing the message with its private key, which achieves the same effect of using signatures. Therefore, we circumvent the transmission of multi-signature messages and the process of verifying signatures, which reduces the communication complexity from $O(N^3)$ to $O(N^2)$ and avoids the significant computation cost (as the symmetric decryption is much cheaper than signature verification).

B.2 Early Stopping Schemes

Apart from reducing communication complexity, SGX can also aid to decrease round complexity. In the general-omission model, several protocols have been proposed to reduce the round complexity. We recall a classic example of reliable broadcast protocol with early stopping in $min\{f+2, t+1\}$ rounds in Algorithm 5 adapted from Perry et al.'s work [82]. When $f < t$ omission faults take place, then all honest nodes will stop by the end of round $f+2$.

In Algorithm 5, $\mathsf{M}_i^{\mathsf{rnd}}(j)$ represents the message received by id_i from id_j in round rnd. $\mathsf{QUIET}_i^{\mathsf{rnd}}$ denotes the set of nodes from which id_i has not received a message from round 1 through round rnd. In the first round, the initiator sends a message to the other nodes and halts. For any round, if a node receives a message from another node, it stores the value in $\mathsf{M}_i^{\mathsf{rnd}}(j)$. If a node id_i does not receive any message from another node id_j for round rnd, id_j will be added into $\mathsf{QUIET}_i^{\mathsf{rnd}}$. When a node has not decided the value and it receives a value, it will set the decision as the new value and broadcasts the value to all nodes in the next round ($\mathsf{rnd}+1 \leq t+1$). If it does not receive any value and $\mathsf{rnd} = t+1$, the node will decide the default value $\perp$. If the round number $\mathsf{rnd} < t+1$ is larger than the size of $\mathsf{QUIET}_i^{\mathsf{rnd}}$, the node will

Algorithm 5: RB_{early}: Reliable broadcast protocol with early stopping (for a node id_i with the initiator $\mathsf{id}_{\mathsf{init}}$ sending a message m).

Input: A P2P network $\mathcal{P}$ composed N nodes, a message m for the initiator $\mathsf{id}_{\mathsf{init}}$
Output: A message $\hat{m}$

```
• initialization: m̂ ← ?; QUIET_i^rnd ← ∅, M_i^rnd(j) ← ∅; rnd ← 1
• upon id_i = id_init:
  m̂ ← m;
  Multicast m to all the other nodes
  return m̂
• for rnd ≤ t+1 do
    • upon receiving ⟨m'⟩ from peer id_j:
      M_i^rnd(j) ← M_i^rnd(j) ∪ {m'}
    • QUIET_i^rnd ← QUIET_i^(rnd−1) ∪ {id_j | M_i^rnd(j) = ∅}
    if m̂ = ? and ∃id_j where M_i^rnd(j) ≠ ∅ then
        m̂ ← M_i^rnd(j)
        if rnd < t+1 then
            Multicast m̂ to all the other nodes in round rnd+1
        end
    else if m̂ = ? and ∄id_j where M_i^rnd(j) ≠ ∅ then
        if rnd < t+1 then
            if rnd > |QUIET_i^rnd| then
                m̂ ← ⊥
            end
            Multicast m̂ to all the other nodes in round rnd+1
        else
            m̂ ← ⊥
        end
    else if m̂ ≠ ? then
        return m̂
    • rnd ← rnd+1
end
```

send $\perp$ to all nodes in round $\mathsf{rnd}+1$, otherwise it will send $\perp$. Finally, once the node decides its value, it halts.

The early-stopping protocol requires every node to broadcast its decision for every round, to inform the other nodes about its liveness. In this way, honest nodes can detect abnormal behaviors of malicious nodes for each round. Based on the detection, all honest nodes can halt and agree on the same value by the end of round $f+2$, where f nodes behave maliciously (e.g., omit to replay messages). The detailed proof can be found in the work [82]. Based on the proposed broadcast detection mechanism, the protocol can early-stop. However, the communication complexity increases to be in $O(N^3)$, as every node broadcasts its value every round.

Efficiency Improvement. By leveraging the *halt-on-divergence* property (P4), we can actively stop nodes behaving maliciously, which eliminates the t-round broadcasting and reduces the communication complexity to $O(N^2)$ as well as sanitizes the network by removing the malicious nodes. For instance, if a malicious node sends a message to other nodes but omit to receive messages from over half of the nodes in the network, the node will be forced to leave the network. Therefore, any node can actively detect its own anomalous behavior instead of relying on other nodes to send messages every round to passively identify the anomaly. This can lead to reduce communication complexity for anomaly detection from $O(N^2)$ to $O(N)$.

C ERB Analysis

In this section, we use the same terminology used in Appendix A, namely, we assume that between any two nodes of the network, an $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ instantiation of the Blinded Peer channel is enabled. In particular, it provides us with both *message Integrity & authenticity (P2)* and *blind-box computation (P3)* properties. Throughout this section, we implicitly consider that the program is publicly available, and therefore its *execution integrity (P1)* is enforced.

Theorem C.1. *If $N \geq 2t+1$ where t is the upper bound on the number of byzantine peers, and $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ is a* Blinded Peer *channel, then* ERB *is a* reliable broadcast protocol *as defined in Definition 2.1 with worst-case round complexity equal to $t+2$ and communication complexity equal to $O(N^2)$.*

Proof. We are going to gradually prove the five requirements of terminating reliable broadcast. Note that the assumption that the peers communicates using $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ implies that a byzantine node can only delay, omit or replay messages, as we have shown in Theorem A.2. As long as the network is synchronous with a fixed time interval for a round to complete, delaying is then equivalent to omitting a message, as the message will not be considered by honest nodes past the round, enforcing therefore the *lockstep execution (P5)* property. Replaying a message is also ineffective as every peer is identified by a sequence number as well, that is generated by the trusted enclave in the Peer channel, and therefore enforcing the *message freshness (P6)* property. Under the assumption that $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ is a Blinded channel, we can replace all occurrences of Multicast by communication between two trusted parties. To sum up, and throughout the proof, it is valid to consider that if there is a delay, omission or replay, this will be equivalent to considering that the first party does not send any message.

Lemma C.2. Validity: In ERB, if the sender is honest and accepts message m, then all honest nodes eventually accept m, otherwise if the sender is byzantine, after round $t+2$, all honest nodes either accept the same message m or $\perp$.

Proof. In the following, we consider two different types of initiators: an honest and a byzantine peer.

(1) Let the sender be the peer p_{init} with identifier $\mathsf{id}_{\mathsf{init}}$. If p_{init} is honest, according to ERB, the sender multicasts its message m in an INIT message for the first round. All honest nodes will receive m in the first round and multicast ECHO to all nodes in the second round, as every node at this stage is going to receive m for the first time. At the end of these two rounds, every honest node will receive at least $t+1$ ECHO messages for m from all honest nodes. According to ERB, each honest node will accept m.

(2) If the initiator is byzantine, we proceed to show the validity by contradiction. Suppose that the lemma does not hold in the byzantine case, which means that at the end of round

$t+2$, not all honest nodes agree on the same value, i.e., only a strict subset of honest nodes agree on m, but the remaining peers agree on $\perp$. According to the protocol, any node accepting m must have received at least $t+1$ ECHO messages from different nodes. The upper bound of byzantine nodes is t, thus at least one honest node should have multicasted an ECHO message to nodes accepting m during $t+2$ rounds. For the proof to go through, we introduce the following claims and then proceed with the contradiction case.

Claim C.1. *There is at least one honest node that does not receive any* ECHO *message after* $t+1$ *rounds.*

Proof. We prove this claim by contradiction. Suppose that all honest nodes receive an ECHO message during $t+1$ rounds, then they multicast ECHO after receiving it. Therefore, all honest nodes will receive $t+1$ ECHO before the end of round $t+2$, which means that all of them accept m. This contradicts our assumption that only a strict subset of honest nodes agree on the same value m. □

Claim C.2. *No honest nodes receives* ECHO *messages after round* t.

Proof. This claim can be also shown by contradiction. Suppose one honest node receives an ECHO message before round t, it must multicast ECHO to all nodes, and all honest nodes receive it before the end of round t. However, this contradicts our Claim C.1. □

We can now proceed by induction where the two claims holds for any $i \in [t-1]$. That is, we can show:

- There is at least one honest node that does not receive any ECHO message after $t+1-i$ rounds.
- All honest nodes do not receive ECHO messages after round $t-i$.

In this case, for any $i \in [t]$, we can show that all honest nodes do not receive ECHO messages after round i. That is the only way an honest node can receive a message m is in round $t+1$ transmitted by a byzantine node. However, this is a contradiction as this event cannot occur. A byzantine node holding a message at round $t+1$ means that through all rounds the message was transmitted between byzantine nodes, *only*. Knowing that if a node does not receive $t+1$ ACK it will halt, this means that the best strategy is to transfer the message to only one byzantine node at a time. This means that there is a need to $t+1$ byzantine nodes in the network which contradicts our assumption.

□

Lemma C.3. Agreement: If an honest node accepts m, then all honest nodes eventually accept m.

Proof. If the sender of m is honest, then all honest nodes accept m according to Lemma C.2. If the sender is byzantine, then all honest nodes either accept m or $\perp$ according to Lemma C.2. Therefore, if an honest node accepts m, all honest nodes accept m, no matter the sender is honest or byzantine. □

Lemma C.4. Integrity: For any message m, every honest node accepts m at most once, and only if m was broadcast by the sender.

Proof. According to Algorithm 2, every honest node only accepts m once, while receiving $t+1$ ECHO messages. If $m \neq \perp$, all honest nodes accept the message broadcasted by the sender, no matter if the sender is honest or byzantine, Lemma C.2. □

Lemma C.5. Early Stopping: Every honest node will terminate at round $\min\{f+2, t+2\}$.

Proof. According to Algorithm 2, if the initiator is honest, then all honest nodes accept m from id_{sint} after two rounds. If it is a byzantine intiatior and f nodes violate the protocol (e.g., receiving less than $t+1$ acknowledgement responses after sending a message), any of these f nodes can exist in the network for at most f rounds. After f rounds, if the message m is sent from any of the f nodes to the other nodes, then the other nodes will follow the protocol and multicast m to all honest nodes. After two rounds, all honest nodes will agree on the same value m. Otherwise, all honest nodes will wait until the end of round $t+2$ and accept the default value $\perp$. Therefore, all honest nodes will terminate at round min f + 2, t + 2. Lemma C.2, C.3 and C.4 also hold in the early-stopping case. □

Lemma C.6. Termination: Every honest node eventually accepts m or $\perp$.

Proof. According to Algorithm 2, if an honest node receives $t+1$ ECHO or INIT messages during $\min\{f+2, t+2\}$ rounds, it will accept m immediately; otherwise, it will accept $\perp$ at the end of round $t+2$. □

Lemma C.7. Efficiency: For any sender, the communication complexity is $O(N^2)$ for one instance of ERB.

Proof. For ERB, every node only broadcasts to all nodes once when receiving INIT or ECHO for the first time, thus every node sends N messages. To reply requests from other nodes with ACK messages, every node sends at most another N messages. There are N nodes in the network, so the communication complexity for one run of ERB is at most $2N^2$ or $O(N^2)$ in total. □

This concludes our proof. □

D P2P Sanitization & Analysis

In ERB, we introduce the concept of *network sanitization* or *faulty node elimination* captured by the *halt-on divergence (P4)* property, or the Halt function for short. This process has an important impact on the P2P topology as whenever a byzantine node misbehaves, the enclave of the node will deterministically stop the node. Thus, the byzantine node gets ejected from the network. This byzantine node cannot generate any new message as its enclave halts. We say that this *sanitizes* the P2P topology.

A byzantine OS gets churned out if it deviates from the sequential execution of ERB. Since it cannot infer the content of each message due to our blinded channel, one of the possible strategy is to behave maliciously uniformly at random. We present in the following our analysis that details the sanitization impact on ERB in this particular scenario, and shows that after a polynomial number of instances, the expected round complexity of the protocol becomes constant[3]. First, we give a characterization of the pace of sanitization considering that for every instance, a byzantine node can behave malicious with a probability that can be independently tuned. We also consider the effect of a new node joining the network before the start of every instance of the protocol to replace eventually an eliminated node.

Theorem D.1. *Let F_r denotes the random variable counting the number of byzantine nodes after r instances of Algorithm 2, then*

$$\Pr[F_r \geq 1] \leq e^{-\lambda},$$

where $\lambda = \frac{rp}{2} - \ln(t)$, t the upper bound of byzantine nodes in the P2P network and p the fraction of activated byzantine nodes at any instance.

Proof. It is easy to see that the number of byzantine nodes in the P2P network at the $(i+1)$th instance of Algorithm 2 equals: $F_{i+1} = F_i - R_i + A_i$, where R_i represents the number of byzantine nodes that have arbitrarily misbehaved and therefore are eliminated from the network, and A_i represents the number of new peers that have joined the P2P network. We can then write: $R_i = \sum_{j=1}^{F_i} X_j^{(1)}$, and $A_i = \sum_{j=1}^{F_i} X_j^{(2)}$, where $X_j^{(1)} \sim \mathcal{B}_p$ and $X_j^{(2)} \sim \mathcal{B}^p_{\frac{1}{2}}$ is a conditional Bernoulli random variables such that , $X_j^{(2)} = 1$ if $X_j^{(1)} = 1$ for $j \in [F_i]$.

$X_j^{(1)}$ is a Bernoulli random variable with parameter p that captures the fact that a node can misbehave in a particular instance of Algorithm 2 with probability p, while the second random variable $X_j^{(2)}$ captures the fact that whenever a node is eliminated from the network, it can be replaced with either a honest or malicious node. This is in phase with our assumption that we can handle a honest majority at the beginning.

Note that both R_i and A_i are both a *random sum* of random variables. As the number of failures at some iteration can be considered independent of the sum of failures occurred throughout all iterations of Algorithm 2, we can consider that both $X_j^{(1)}$ and $X_j^{(2)}$ are independent of F_i.

Based on Wald's equation, we have $E[F_{i+1}] = (1-\frac{p}{2}) \cdot E[F_i]$. By induction we can show that $E[F_{i+1}] = (1-\frac{p}{2})^{i+1} \cdot E[F_0]$, where $E[F_0] = E[t] = t$, the initial number of byzantine nodes in the network.

Based on Markov inequality, we show that

$$\Pr[F_r \geq 1] \quad \leq \quad t(1-\frac{p}{2})^r \leq e^{-\frac{rp}{2}+\ln(t)}.$$

Setting $\frac{rp}{2} - \ln(t) = \lambda$ concludes the proof.

□

For example, for $\lambda = 30$ and $t = \frac{N}{2} - 1$ for $N = 2^{10}$ $p = 2^{-5}$, the number of rounds before the P2P gets sanitized with high probability equals to $r \approx 2500$.

We are now interested in computing the expected number of rounds in average of Algorithm 2 while taking into consideration our sanitization protocol. Theorem D.1 shows that the P2P can get sanitized w.h.p. after a particular number of rounds, however, throughout the different instances, the number of byzantine nodes decreases as well, which suggests that the round complexity can get better. We will show in the following theorem that Algorithm 2 has a constant round complexity in average after a polynomial number of instances.

Theorem D.2. *Algorithm 2 has a constant round complexity in average for a number of instances $r = \mathsf{poly}(N)$ w.h.p.*

In this theorem, we consider the same setting of Theorem D.1 where the number of byzantine nodes at the $(i+1)$th instance equals $F_{i+1} = F_i - R_i + A_i$, where $R_i = \sum_{j=1}^{F_i} X_j^{(1)}$, and $A_i = \sum_{j=1}^{F_i} X_j^{(2)}$, $X_j^{(1)} \sim \mathcal{B}_p$ and $X_j^{(2)} \sim \mathcal{B}^p_{\frac{1}{2}}$ is a conditional Bernoulli random variables such that , $X_j^{(2)} = 1$ if $X_j^{(1)} = 1$ for $j \in [F_i]$.

We have shown that in this case the expected value of $E[F_i] = (1-\frac{p}{2})^i \cdot t$.

To compute the expected number of rounds per instance, we need to count first the total number of possible byzantine nodes to ever occur after r instances, T_r. This equals $T_r = \sum_{i=1}^{r} \sum_{j=1}^{F_i} (X_i^{(1)} + X_i^{(2)})$, Moreover, we also define the average number of rounds per instance as a random variable , R, equal to $R = 2 \cdot \frac{(r-T_r)}{r} + t \cdot \frac{T_r}{r}$, where during $r - T_r$ rounds the protocol is optimal and equals 2, and in T_r rounds the protocol has a worst-case round complexity and equals to f.

We then have that, leveraging Wald's equation, $E[T_r] = \frac{3t}{2} \cdot (1-(1-\frac{p}{2})^{r+1})$. Then, $E[R] - 2 \sim \frac{3t^2}{2r} \cdot (1 - e^{\frac{-pr}{2}})$. By Markov, we have $\Pr(R \geq 3) \leq \frac{3t^2}{2r} \cdot (1 - e^{\frac{-pr}{2}})$.

That is, if $r = \mathsf{poly}(N)$ and $p = O(\frac{1}{t})$, then $\Pr(R \geq 3) \in O(\frac{1}{\mathsf{poly}(N)})$.

[3]We believe that the network sanitization asymptotic improvement can apply independently of the malicious nodes' strategy

E Unoptimized ERNG Analysis

In this section, similar to Appendix A, we denote by the ROD mode, a mode where peers in a network $\mathcal{P}$ can only replay, omit and delay messages.

Theorem E.1. *If $\mathcal{P}$ operates in the* ROD *mode, then the bias of G $\beta(G) = 1$.*

Proof. Note that while G can be modeled as a multi-variate function, it does not capture the sequencing of inputs. For our proof to go through, we need to first show that the sequencing of ERNG is guaranteed and a node can only participate with its input if it starts synchronously with all nodes. For this, we have the following two cases:

- *early start*: if a byzantine node transmits its INIT at $\mathsf{rnd} = 1$, then based on Lemma C.2, the node outputs (either m or $\perp$) will be considered as an input for G,
- *late start*: if a byzantine holds the INIT message until seeing the output, then its input will not be added to $\mathsf{S}_{\mathsf{final}}$ as the message will be considered delayed. The output of G in this case equals $\perp$

Note that for both cases, the nodes have to start the protocol at $\mathsf{rnd} = 1$ if they want to participate with their inputs in the final output. Moreover, based on the $\mathsf{Blinded}$ channel, we know that nodes can only obtain the final output of G while not viewing any internal state of G, which enforce the *blind-box computation (P3)* property. That is, it is valid to consider G as a multi-variate function that is fed all inputs at once. Let us denote by X the random variable that captures the output of G such that $X = X_1 \oplus \cdots X_N$, where X_i's are random variables that capture the input provided by every node in $\mathcal{P}$, for all $i \in [N]$. As $\mathcal{P}$ operates in the ROD mode, Lemma C.2 demonstrates that all honest nodes receive the same set $\mathsf{S}_{\mathsf{final}}$ at the end of the protocol. We then can rewrite X such that $X = \bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N} X_i$, where $\kappa = |\mathsf{S}_{\mathsf{final}}|$. In the following, we need to show that $E_G[S] = E[S] = \frac{|S|}{2^k}$, for all $S \subseteq \{0,1\}^k$. Note that $E_G[S] = \Pr[X \in S]$, and therefore it is sufficient to compute $\Pr[X \in S]$.

$$\Pr[X \in S] \quad = \quad \Pr[\bigcup_{x \in S} (X = x)] = \sum_{x \in S} \Pr[X = x]$$

The second equality follows from the fact that all events are disjoint. Now for a given $x \in S$, $\Pr[X = x]$ equals:

$$\begin{aligned}
&= \Pr[\bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N} X_i = x] \\
&= \sum_{x_N \in \{0,1\}^k} (\Pr[\bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N-1} X_i = x \oplus x_N | X_N = x_N] \\
&\cdot \Pr[X_n = x_N]) \\
&= \sum_{x_N \in \{0,1\}^k \setminus \{0\}} (\Pr[\bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N-1} X_i = x \oplus x_N | X_N = x_N] \\
&\cdot \Pr[X_n = x_N]) + \Pr[\bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N-1} X_i = x | X_N = 0] \cdot \Pr[X_n = 0] \\
&= \Pr[\bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N-1} X_i = x | X_N = 0] \\
&= \Pr[\bigoplus_{i=1}^{\kappa} X_i = x | X_{\kappa+1} = 0, \cdots, X_N = 0] \\
&= \sum_{x_\kappa \in \{0,1\}^k} \Pr[\bigoplus_{i=1}^{\kappa} X_i = x \oplus x_\kappa | X_\kappa = x_\kappa, X_{\kappa+1} = 0, \cdots, X_N = 0] \\
&\cdot \Pr[X_\kappa = x_\kappa] \\
&= \frac{1}{2^s} \sum_{x_\kappa \in \{0,1\}^k} \Pr[\bigoplus_{i=1}^{\kappa} X_i = x \oplus x_\kappa | X_\kappa = x_\kappa, X_{\kappa+1} = 0, \cdots, X_N = 0] \\
&= \frac{1}{2^{s \cdot (k-1)}} \sum_{x_2, \cdots, x_\kappa \in \{0,1\}^k} \Pr[X_1 = x \oplus \bigoplus_{i=2}^{\kappa} x_i | \\
&\quad X_2 = x_2, \cdots, X_\kappa = x_\kappa, X_{\kappa+1} = 0, \cdots, X_N = 0] \\
&= \frac{1}{2^{s \cdot k}} |\{x_2, \cdots, x_\kappa \in \{0,1\}^k\}| = \frac{1}{2^s}
\end{aligned}$$

Thus, $\Pr[X \in S] = \frac{|S|}{2^s}$. This concludes our proof. □

F Optimized ERNG

F.1 Optimized ERNG pseudo-code

We present a pseudo-solution of our optimized ERNG in Algorithm 6.

F.2 Proofs

Lemma F.1. *If up to $t = \frac{N}{3}$ nodes are byzantine, then with at least $1 - \mathsf{negl}(\gamma)$ probability, the representative cluster has more than γ honest nodes, and less than γ byzantine nodes.*

Proof. In ERNG at round 1, every node picks uniformly at random a value from $\{0, \cdots, \frac{N}{2\gamma} - 1\}$. That is, every node has a probability equal to $q = \frac{2\gamma}{N}$ to be chosen as a representative. Let H_i and B_i be two random variable that equal 1 if the i^{th} honest and byzantine node is chosen respectively, otherwise they equal zero. Let us denote by $\mathsf{H} = \sum_{i=1}^{2t} H_i$ and $\mathsf{B} = \sum_{i=1}^{t} B_i$ the number of selected honest and byzantine nodes in the cluster. Then both H and B are distributed following a binomial distribution with a number of trials equal to $2t$ and t, respectively. We have $E[\mathsf{H}] = \sum_{i=1}^{2t} E[H_i] = 2t \cdot \frac{2\gamma}{N} = \frac{4t \cdot \gamma}{N}$. Similarly, $E[Y] = \frac{2t \cdot \gamma}{N}$. Based on two variations of Chernoff bound, considering $t = \frac{N}{3}$, we obtain that

$$\Pr[\mathsf{H} > (1 - \delta_1)\frac{4\gamma}{3}] \geq 1 - e^{-\frac{2\delta_1^2 \cdot \gamma}{3}},$$

similarly, $\Pr[\mathsf{B} < (1 + \delta_2)\frac{2\gamma}{3}] \geq 1 - e^{-\frac{2\delta_2^2 \cdot \gamma}{9}}$, where $\delta_1, \delta_2 < 1$. For a choice of $\delta_1 = \frac{1}{4}$ and $\delta_2 = \frac{1}{3}$, we obtain,

$$\Pr[\mathsf{H} > \gamma] \geq 1 - e^{-\frac{\gamma}{24}},$$

Algorithm 6: ERNG: Enclaved unbiased random number generation protocol executed by peer p_i.

Input: A P2P network $\mathcal{P}$ composed of N nodes
Output: A unbiased random number r

- initialization: $\mathsf{S_M} \leftarrow \emptyset; \mathsf{S_{final}} \leftarrow \emptyset; \mathsf{S_{chosen}} \leftarrow \emptyset; \mathsf{rnd} \leftarrow 1$

```
for rnd ≤ γ+4 do
    if rnd = 1 then
        • every peer p_i compute r_i ←$ {0, ···, N/(2γ) − 1};
        if r_i = 0 then
            Multicast(id_i, val), where
            val = ⟨CHOSEN, id_i, seq_i, ⊥, 1⟩;
            S_chosen ← {id_i};
        end
        • upon receiving val = ⟨CHOSEN, id_j, seq_j, m_j, rnd_j⟩
        if type = CHOSEN and rnd_j = 1 and seq_j = seq_i then
            S_chosen ← S_chosen ∪ {id_j};
        end
    end
    • if r_i = 0 and rnd = 2 then
        compute r'_i ←$ {0, ···, γ' − 1};
        if r'_i = 0 then
            initiate ERB with inputs m_i ←$ {0,1}^k, seq_i and
            peers in S_chosen;
        end
        seq'_j ← seq_j, for all id_j ∈ S_chosen;
    end
    if r_i = 0 and 3 ≤ rnd ≤ γ+2 then
        execute ERB instances and wait for the output;
    end
    if r_i = 0 and rnd = γ+3 then
        Wait(rnd) then obtain M_i = {m̂_1, ···, m̂_{l_i}};
        seq_j ← seq'_j, for all id_j ∈ S_chosen;
    end
    if rnd = γ+4 then
        • if r_i = 0 then
            S_M ← S_M ∪ {M_i};
            Multicast(id_i, ⟨FINAL, id_i, M_i, seq_i, γ+4⟩);
        end
        • upon receiving val = ⟨FINAL, M_j, seq'_j, rnd_j⟩:
        if rnd_j = γ+4 and seq'_j = seq_j then
            S_M ← S_M ∪ {M_j};
            if # of M_κ ≥ γ+1 where M_κ ∈ S_M then
                S_final ← M_κ;
                accept r = ⊕_{v∈S_final} v.
            end
        end
    end
    rnd ← rnd + 1;
end
```

- $\mathsf{seq}_j \leftarrow \mathsf{seq}_j + 1$, for all $j \in [N]$;

and,

$$\Pr[\mathsf{B} < \gamma] \geq 1 - e^{-\frac{\gamma}{41}}.$$

□

Lemma F.2. *If $\gamma' = \sqrt{\gamma}$, then the probability that $\Omega(\sqrt{\gamma})$ honest nodes are selected to be in the second representative cluster is at least $1 - \mathsf{negl}(\gamma)$.*

Proof. Based on Algorithm 6, every node in the cluster has a probability of $\frac{1}{\gamma'}$ to be chosen. Let us denote by X_i the random variable equal to one if the node is selected. We then denote by, $\mathsf{H}' = \sum_{i=1}^{\mathsf{H}} X_i$, the random variable that counts the number of honest node in the second cluster. Based on Wald's equation, we obtain $E[\mathsf{H}'] = \frac{E[\mathsf{H}]}{\gamma'} = \frac{4\gamma}{3\gamma'}$. Then, based on Chernoff bound, we obtain for $\delta < 1$,

$$\Pr[\mathsf{H}' > (1-\delta) \cdot \frac{4\gamma}{3\gamma'}] \geq 1 - e^{-\frac{4\delta^2 \cdot \gamma}{3\gamma'}}$$

if we set $\delta = 1 - \frac{1}{\gamma}$ and $\gamma' = \sqrt{\gamma}$, then we obtain

$$\Pr[\mathsf{H}' > \frac{4\sqrt{\gamma}}{3}] \geq 1 - e^{-\sqrt{\gamma}}.$$

This ends out proof. □

Note that we can obtain better bounds if we consider computing the pmf of H' as it follows a binomial distribution with a binomial number of trials

Corollary F.1. *If $\gamma' = \sqrt{\gamma}$, then the size of the first and second representative clusters is in $O(\gamma)$ and $O(\sqrt{\gamma})$ w.h.p*

The proof of the corollary directly follows from Lemma F.2.

Theorem F.1. Agreement: *All honest nodes eventually agree on the same common set $\mathsf{S_{final}}$ in* ERNG.

Proof. In round 1, $|\mathsf{S_{chosen}}|$ nodes are uniformly at random selected to be part of the representative cluster. Based on Lemma F.1, we have shown that the cluster contains *strictly* more than γ honest nodes, and *strictly* less than γ byzantine nodes w.h.p. when $t < \frac{N}{3}$ That is, we have created a new smaller P2P network $\mathsf{S_{chosen}}$ in which the honest nodes represent the majority. In the cluster, all honest nodes know each other, but byzantine nodes may deliberately not contact honest nodes on purpose. In this case, the cluster will be more robust with less byzantine nodes. Thus, all the results introduced for ERB will hold for this cluster of nodes.

From round 2 to round $\gamma + 3$, the second cluster has more than $\sqrt{\gamma}$ honest nodes w.h.p. according to Lemma F.2. For each instance of ERB— whether initiated by an honest or byzantine node, the honest representative nodes will agree on a same message according to Lemma C.3. Since there is at least one honest sender, all honest nodes will accept the honest sender's message for its run of ERB based on Lemma C.2. After around $O(\sqrt{\gamma})$ runs, all honest nodes will agree on the same set of random numbers. Since the number of honest representative nodes is larger than γ and all of them will multicast FINAL messages for the same set of messages in round $\gamma + 4$, then all honest nodes will receive adequate FINAL messages to accept the common set $\mathsf{S_{final}}$. □

In ERNG, since the message m_i is a random number generated by the SGX and proposed by the peer p_i, then eventually every honest node accepts the same set $\mathsf{S_{final}}$ of random numbers according to Theorem F.1. By performing exclusive disjunction (or XOR) of all the random numbers in S_{final}, every honest node can obtain a common random number r. We demonstrate next that the random number r is unbiased against byzantine nodes.

Theorem F.2. Unbiasedness: *The output of the* ERNG *protocol is an unbiased random number.*

sketch. Given Theorem F.1, we know that all honest nodes agree on the same set S_M. On the other hand, leveraging $Peer^{Ch}_{sgx}$ Peer channel, we know that all random numbers in the ERNG protocol are generated within the SGX enclave and never tempered with as the network is in the ROD model, based on Corollary A.2. Finally, it is sufficient to show that if all random numbers generated in SGX are random then the output of ERNG is an unbiased random number, which holds given SGX primitive generates unbiased random number against the operating system according to Theorem E.1. □

G Are our Assumptions Reasonable?

S1: Network Size. We start with a fixed size network $\mathcal{P}$ with N peers. We assume that there exists information that publicly identifies every node in $\mathcal{P}$, this can be for example a node IP address. This assumption is reasonable under some common conditions. For example, for banking systems, all involved machines should be registered and publicly available. Fortunately, we can weaken such as assumption and we can extend our setting to work within a variable size network based on the following technique: whenever a node wants to join $\mathcal{P}$, the joining node contacts another neighbor node and communicates both its sequence number and identifier. The contacted node will use ERB to reliably broadcast the pair to all peers in $\mathcal{P}$ and then send the joining peer a message containing all existing identifiers of $\mathcal{P}$. We can leverage the same technique in a recursive way to even start with a one node network $\mathcal{P}$. Note that in this case the identifier need not to be publicly known.

S2: Synchronous Start. Before initiating the ERB primitive, we assume that any honest node in $\mathcal{P}$ can be triggered at the same reference time. This reference time can be provided in different ways such as periodic execution from a fixed reference date, or simply by starting at a time posted in public servers. Once synchronized, every node uses the trusted elapsed time from SGX to maintain a relative time from the reference time. This therefore will maintain an internal clock within every node's enclave. As the enclaves in all honest nodes will have the (nearly) same internal clock, all nodes will start the next instance of the protocol at the same time. If any byzantine node deviates by omitting or delaying the oracle message, its elapsed time will be different from the one honest nodes have. Consequently, all the byzantine node messages will be delayed as they are going to have a different round number.

S3: Round time 2Δ seconds. The round time (2Δ seconds) is adequately determined to allow any honest round trip message to complete within 2Δ seconds. The round increments are managed using the trusted elapsed time, which implies that even if the OS is byzantine, the round number will be always incremented inside the enclave every 2Δ seconds. We also emphasize that the time interval between any two internal clocks for honest nodes is negligible compared to 2Δ seconds. As ERB does not use any underlying heavy cryptographic primitive, we assert that any sent message will be received in the same round. The 2Δ seconds is mostly dedicated for network latency reasons.

S4: Number of byzantine nodes less than $\frac{N}{2}$. To join a network $\mathcal{P}$, an adversary is required to control machines with SGX-enabled CPUs, in which the number of possible launched enclaves is bounded [46]. To control $\frac{N}{2}$, the adversary needs to control a number of SGX machines. Meanwhile, we can also employ existing sybil defenses in our network to control the number of byzantine nodes, e.g., defenses using computation puzzles or proof of work [30, 68]. The details of deploying these sybil defenses are beyond the scope of this paper.

S5: Connected Peers. For simplicity of design and to follow the standard model used in previous works, we assume that all the peers in the network are connected to each other. However this assumption can be relaxed such that the network is a sparse but expander or random graph. This will guarantee that there is a path in between any two honest nodes. Thus, the direct point-to-point broadcast in our protocol can be replaced with a flooding algorithm to broadcast messages.

H Applications

Both ERB and ERNG primitives can be used as building blocks to solve a wide range of problems in distributed systems. In the following, we review some of the most prominent applications.

Random Beacons. A random beacon protocol [84] offers a way to generate uniformly random strings that are unknown to the nodes before their generation. Random beacons have been extensively studied as they have numerous applications in cryptography and information security, such as secure contract signing protocols [50, 84], voting schemes [75], zero-knowledge protocols [21, 57], and cryptocurrency protocols [71]. Building random beacons is a difficult task. Practical solutions usually leverage a trusted third party [10, 17], or utilize public data available on the Internet such as financial data [43]. However, the data from these services has to be trusted and certified, which unfortunately represents a strong assumption in practice. Recently, researchers have also proposed several protocols to generate random beacons by using Bitcoin as a source of publicly-verifiable randomness [25, 29]. However, the adversary can bias the beacon by introducing a new monetary cost. With ERNG, the underlying system can easily generate a common unbiased random number in the network.

Random Walks. In order to build a more robust P2P topology, random walk is an essential primitive to distribute nodes uniformly in the network to maintain an expander topology. Guerraoui *et al.* [58] build a virtual overlay on top of the physical nodes, in order to maintain a robust P2P topology. Each virtual node represents a cluster that consists of a set of physical nodes such that at least $\frac{2}{3}$ of the nodes are honest. This guarantees that decisions or agreements of the cluster hold on the behalf of the entire physical nodes of the network. Ensuring that the virtual nodes are honest will guarantee the correctness of the random walks against byzantine nodes. However, this is not sufficient and in order to determine the next hop in the random walk, an unbiased random number is required. With ERNG, we present an efficient solution for this issue in such a way that physical nodes in the cluster can generate a common unbiased random number to designate the next hop, and therefore maintain a robust topology.

Shared Key Generation. By performing ERNG, every honest node will share a common unbiased random number that can be used as a key, salt or initialization vector for symmetric cryptography. ERNG can also be used as a building block for distributed key generation (DKG) where the peers want to compute a shared public and private key. DKG has several applications and in particular in threshold cryptography, we refer the reader to the works by Gennaro *et al.* [55, 56].

Random Load Balancing. Random load balancing is generally performed by a centralized server to distribute tasks to slave servers [47, 85]. A centralized server is often considered as a single point of failure, which is usually the primary target of attackers. Once the centralized server is compromised, the whole load balancing system fail as well. With ERNG, we distribute the decision generation process to a cluster of nodes instead of a centralized server. When a new request or a task comes to any node, the cluster of nodes evaluate ERNG to generate an unbiased common random number and send the decision to the target slave server. Once the slave server receives adequate confirmations from (say) half of the nodes, it can take upon the task and evaluate it. This way, even if half of the nodes are either compromised/failed, the load balancing system can still work correctly. Note that the nodes can a-priori pre-process many random numbers to speed-up the process. The random numbers can be generated and stored in the hard drive using *sealing* technique enabled by the SGX.