Decentagram: Highly-Available Decentralized Publish/Subscribe Systems

Haofan Zheng*
UC Santa Cruz
hzheng6@ucsc.edu

Tuan Tran*
UC Santa Cruz
atran18@ucsc.edu

Roy Shadmon
UC Santa Cruz
rshadmon@ucsc.edu

Owen Arden
UC Santa Cruz
owen@soe.ucsc.edu

***Abstract*—This paper presents Decentagram, a decentralized framework for data dissemination using the publish/subscribe messaging model. Decentagram uses blockchain smart contracts to authenticate events that will be published using digital signatures or *self-attestation certificates* from code running in trusted execution environments (TEEs), both of which are verified on-chain. This approach permits any host with valid credentials to publish verified updates, increasing decentralization and availability of the system as a whole by simplifying compensation and incentivization, even for untrusted hosts running TEEs. Decentagram also supports on-chain subscribers where third-party contracts receive events immediately: within the same transaction as the published event. The same event will also be delivered to off-chain subscribing applications through an off-chain event broker. We provide an open-source implementation of Decentagram, and evaluate the gas cost of its on-chain components and the end-to-end latency of its off-chain component.**

I. Introduction

The goal of publish/subscribe (pub/sub) systems is the dissemination of information from *publishers* to interested *subscribers* quickly and efficiently. Several production systems have been developed (e.g., [1]–[5]), and pub/sub systems have been applied in a variety of contexts, from financial applications [6], [7] to health monitoring [8], as well as real-time vehicle detection [9], [10]. To declare interest in a category of published events, subscribers register with a *broker*. Publishers send their events to the broker, possibly including metadata indicating relevant categories, and the broker notifies registered subscribers of the new event.

Some recent systems [11], [12] integrate pub/sub systems with blockchains to provide Byzantine fault tolerance and auditability of published events. However, these systems rely on Byzantine fault tolerant (BFT) protocols for consensus among off-chain replicas before publishing events to the blockchain. Unfortunately, the trust assumptions between blockchain protocols and these off-chain BFT protocols are mis-matched. In addition to the blockchain protocol's trust assumptions (e.g., attackers control less than 51% of staked ETH [13]), subscribers must also trust an additional entity to manage membership in the BFT protocol. The membership management mechanism (MMM) is implicitly trusted to authenticate replicas, select the system parameter f that specifies the maximum number of Byzantine faults to tolerate, and maintain the required number of member replicas needed. Note that even if the MMM is distributed among the replicas in some fault-tolerant way, the subscriber must still trust that the parameter f is sufficient to prevent malicious events from being published, and that the non-faulty hosts—as viewed by the protocol—are trustworthy from the subscriber's perspective.

Publishing events to the blockchain means that the data will be visible to anyone, not just the subscribers. Some pub/sub systems, such as PUBSUB-SGX [14], use *secure enclaves* to protect sensitive data. The enclave is a secure hardware component that provides an application-level Trusted Execution Environment (TEE). Enclave programs, including their data, are cryptographically protected from all other components, including privileged OS components. However, PUBSUB-SGX is unable to ensure availability for event publication since it relies on direct TLS connections between publishers and subscribers, which are controlled by the host OS.

This paper presents Decentagram, a secure and decentralized pub/sub system that combines the core capabilities of both blockchain and TEE-based mechanisms to address the above issues of availability and confidentiality. Open-membership blockchains are only capable of providing *integrity* and *availability* guarantees[1] for smart contracts and their data. Since block proposers and validators must know the contents of each block, they cannot provide confidentiality. TEEs are only capable of providing *integrity* and *confidentiality* for the code and data they contain. Since the TEE's host controls all channels to and from the TEE, no availability guarantees are possible. By carefully integrating these two technologies, Decentagram provides the trifecta of information security guarantees, as well as extends the capabilities of the system. Decentagram is the first pub/sub system that supports decentralization of *data oracle*, *publishers*, *brokers*, and *subscribers*, and is also the first blockchain-based pub/sub system that can deliver events on- and off-chain simultaneously. Decentagram's on-chain smart contract framework allows TEEs running on untrusted hosts to authenticate with attestation credentials. Authenticating TEEs on-chain provides Decentagram with three significant advantages over previous systems:

- First, Decentagram enables *permissionless publishing*, allowing any host to run a data oracle without requiring direct attestations to subscribers.

* Authors contributed equally.

[1] In theory, confidentiality could be protected with advanced cryptographic protocols such as secure multi-party computation or fully-homomorphic encryption, but these are unlikely to sufficiently scale to blockchain settings.

- Second, Decentagram supports *incentivized event availability*, which compensates publishers for timely event publication and enables nullification of any (rational) benefit gained by delayed or suppressed events.
- Third, Decentagram replaces trust in third-parties (such as a quorum of replicas) with much weaker trust assumptions: other than standard TEE and blockchain assumptions, subscribers only need to trust the enclave code, which can be audited (or even verified) beforehand.

Decentagram also supports on-chain subscribers: third-party contracts that receive published events. Propagating events to third-party contracts in previous systems [11], [12], [15] requires off-chain subscribers to monitor blocks for new events and submit a transaction with the event to the third-party contract. This Monitor-and-React (M&R) approach introduces a delay between when an event is published and when the client's smart contract can react, making it challenging to process events consistently across on-chain and off-chain subscribers. During periods of network congestion, the time between on-chain and off-chain reactions could increase if the relayed events are not included in the next block.

Enabling timely authenticated on-chain notifications is especially useful for applications with non-deterministic smart contracts that require data from an external party to make decisions [16]. For example, in Decentralized trading markets [17] in which clients monitor an on-chain marketplace contract for the latest price updates to make bids, an on-chain client can react immediately (within the same block) and make time-sensitive bids much faster than M&R clients (at least 1 block away). As another example, applications that utilize smart contracts to manage the membership of a group of off-chain compute nodes [18] also benefit from on-chain notifications. In this case, if any of the compute nodes are compromised, its credentials need to be revoked, and immediate notification of the on-chain membership contract will prevent the compromised node from any further participation in on-chain activities. We present a case study in Section VI that demonstrates the effectiveness of decentralized revocation using Decentagram.

To the best of our knowledge, Decentagram is the first pub/sub system that delivers on-chain events directly. This functionality uses gas limits on cross-contract calls to safely execute callbacks to third-party contracts, preventing broken or malicious contracts from launching gas-exhaustion attacks [19]. Furthermore, using Decentagram's incentivized event availability, the increased cost of publish transactions incurred by these callbacks is paid with subscriber fees, not by data oracles.

Finally, we present a design for confidential event publishing in Decentagram, where events are encrypted before publication and authorized subscribers decrypt events off-chain. Unlike other TEE-based pub/sub systems, Decentagram supports an on-chain key-exchange protocol where a subscriber can obtain a subscription key without requiring a direct connection to the publishing oracle. We provide a design for this protocol that uses the Decentagram framework to distribute keys.

In summary, this paper makes the following contributions.

- We present the design and implementation of Decentagram, the first decentralized pub/sub framework with on-chain TEE attestation and authentication.
- Decentagram is resilient to Byzantine failures and is highly available, allowing any host to run as a data oracle as long as its enclave instance can authenticate itself to the on-chain broker contract.
- Decentagram's smart contracts support on-chain subscribers for clients, and provides atomic on-chain event notifications for subscribers to instantly react to events.
- An evaluation of the on-chain smart contracts shows that the gas cost to publish and subscribe is reasonable, and that the throughput of our off-chain components is more than sufficient to handle events in new blocks.

The rest of this paper is organized as follows. In Section II, we discuss the related work and how Decentagram compares against state-of-the-art pub/sub systems. Section IV contains the design of the on-chain and off-chain components in Decentagram, followed by the implementation details in Section V. Next, in Section VI, we present a case study that utilizes all the features provided by Decentagram. Then, we provide the evaluation of Decentagram in Section VII, and finally, we conclude the paper in Section VIII.

II. Related Work

Table I shows the features of representative Pub/sub systems and how they compare to Decentagram.

Early work in pub/sub systems such as Linda [20] and SIENA [21] described how, in loosely-coupled distributed systems, events can be generated and consumed by a set of processes, but did not consider the possibility of machine failures. To provide availability in the presence of failures, data can be replicated across a set of servers, as is done in ISIS [23], [25] and modern industrial pub/sub systems like Kafka [1] and RabbitMQ [3], where a subset of the servers can fail by crashing. One common feature between these crash fault tolerant Pub/sub systems is the ability to provide causal delivery of events to subscribers. Causal ordering of events ensures that events sent by multiple publishers to a subscriber are delivered in the same order that they were sent. Though sufficient for some applications, a stronger reliability guarantee such as publication total order [24], which ensures that all subscribers receive events in the same order, may be required for many applications (e.g., stock market data).

Some recent systems make use of blockchain technology to eliminate centralization of the event broker, tolerate Byzantine failures, and provide auditability for publications [11], [12]. HyperPubSub [12] keeps a record for each pub/sub operation on the blockchain, but the system does not fully tolerate Byzantine faults because the broker is implemented using Kafka which, as mentioned earlier, can only provide fault tolerance against crashes. Trinity [11] solves the issue of Byzantine brokers by using a Byzantine fault tolerant

Systems	Fault Tolerance	Fault Threshold	Auditability	Confidentiality	On-chain Notification	Publishing Incentive
Linda [20], SIENA [21]	✗	✗	✗	✗	✗	✗
PUBSUB-SGX [14]	✗	✗	✗	✓	✗	✗
ISIS [22], [23], Kafka [1], RabbitMQ [3]	Crash	$\lfloor \frac{n-1}{2} \rfloor$	✗	✗	✗	✗
HyperPubSub [12]	Crash	$\lfloor \frac{n-1}{2} \rfloor$	✓	✗	✗	✗
Trinity [11]	Byzantine	$\lfloor \frac{n-1}{3} \rfloor$	✓	✗	✗	✗
Chios [24]	Byzantine	$\lfloor \frac{n-1}{3} \rfloor$	✗	✓	✗	✗
Chainlink [18]	Byzantine	$\lfloor \frac{n-1}{3} \rfloor$	✓	✓	✗†	✓
Decentagram	Byzantine	Blockchain	✓	✓	✓	✓

TABLE I: Representative Pub/sub systems. †Most Chainlink clients access oracle data off-chain, but some on-chain processing for special *aggregator* contracts is possible.

consensus protocol, and a Byzantine quorum of brokers (i.e., more than two thirds) need to agree on an operation before a transaction is sent to the blockchain to record the operation. While Byzantine fault tolerant consensus improves resilience against malicious brokers, they are limited by the fault threshold $f = \lfloor \frac{n-1}{3} \rfloor$, where n is the number of brokers. This means that if there are n brokers, at least $n - f$ brokers must be operational for the system to function, and the system will be unavailable if the number of accumulated faults surpasses f. Chainlink [18] describes a system with an on-chain oracle contract that can validate reports generated by a set of off-chain oracles. A designated off-chain leader oracle collects attestations to form a report and submits it to the oracle contract, which then determines the oracles that contributed to the report and compensates them. Each client can then monitor the oracle contracts for events that are emitted, and then send transactions to update their contract. The system, however, requires that the oracle contract belongs to an administrator who can not only add or remove oracles, but also set the compensation amount for the oracles. Clients can also interact with the oracles directly, essentially becoming their own on-chain broker, but this requires paying oracles per request.

In Decentagram, the broker is implemented as a smart contract, so the system inherits the blockchain's availability guarantees. For public blockchains, these are typically much stronger than what is possible to achieve with traditional BFT protocols. In Ethereum, there are currently over 974K validators and 24M ether staked in the Ethereum mainnet [26], meaning that at least one-third [13] of them must be compromised to prevent the blockchain from proceeding. The system remains available as long as the blockchain is available and at least one oracle can provide authenticated publication. BFT protocols have been demonstrated to scale only up to the low hundreds of nodes [27], implying blockchain availability guarantees are at least three orders of magnitude higher. Thus Decentagram remains available under significantly more faults than systems such as Chainlink, Trinity, and Chios. Decentagram also improves on monitor-and-check approaches such as Trinity [11] by delivering events to subscribing contracts immediately. In Section VII-D we compare the average, minimum, and maximum latency experienced by an application based on Decentagram and one based on the M&R approach.

Several prior pub/sub systems works have considered security and, more specifically, confidentiality for event subscriptions and notifications [14], [28]–[30]. One approach to providing confidentiality is through using access control policies for publishers and subscribers [28]–[30]. PUBSUB-SGX [14], like Decentagram, uses TEEs to provide confidentiality and integrity for subscriptions and to enforce subscription policies. Subscribers directly connect to event brokers (called matchers) and authenticate them via remote attestation. Unlike Decentagram, publishers do not execute in TEEs and are not incentivized to publish events, meaning that dishonest publishers may choose to delay or drop events to their own advantage. PUBSUB-SGX does not offer fault tolerance for event brokers or publishers, although the authors propose a replication scheme for brokers at the cost of duplicated event delivery. Unfortunately, this scheme does not eliminate the possibility of message loss [14].

Some works have presented smart contracts that run inside TEEs, such as enclaves. However, to authenticate the enclave, FastKitten [31] and LucidiTEE [32] require the participants to communicate with the enclave directly to perform remote attestation (RA). This prevents other smart contracts from verifying the results generated by the enclave. Ekiden [33] and PDO [34] require a customized consensus layer with the ability to verify the enclave's certificate chain, making it incompatible with existing blockchain networks, and expensive to deploy TEE-based contracts. To the best of our knowledge, Decentagram is the first system that supports on-chain RA using EVM (Ethereum Virtual Machine) smart contracts. Therefore, TEE-generated events can be verified on-chain, and Decentagram can be deployed on any EVM-compatible blockchains.

Like other large-scale pub/sub systems such as Kafka [1] and RabbitMQ [3], Decentagram provides a topic-based subscription model. Adapting Decentagram to other models, such as content-based subscriptions [35], may be possible, but resolving the usual tension between scalability and content filtering in this new context is beyond the scope of this paper, though we expect that minimizing on-chain filter processing would be a key to reducing publishing costs.

III. System and Threat Model

System Model. Decentagram integrates with the smart contract framework of a permissionless blockchain to benefit from its *integrity* and *availability* properties. Our implementation targets the Ethereum Virtual Machine (EVM), so Decentagram's smart contracts can operate on any EVM-compatible blockchain. Any node can join the network as a blockchain participant or as a publisher, oracle, broker, or client in Decentagram. Decentagram uses TEEs to provide *confidentiality* and *integrity* for event data. TEE message authentication is based on Remote Attestation (RA), where a key provisioned to the TEE by the hardware manufacturer is used to sign a new public key whose private key is only known to the TEE. Specifically, TEEs create self-attestation certificates [36], [37], which contain third-party verifiable credentials generated by remotely attesting the TEE to itself. These certificates are used to create authenticated channels between TEEs and for smart contracts to verify transactions from TEEs.

Threat Model. We assume attackers control at most the fraction of the blockchain network tolerated by the underlying protocol for its availability and integrity. In Ethereum, this is no more than $\frac{1}{3}$ of staked ETH (319k validators) to prevent halting progress, and no more than $\frac{1}{2}$ of staked ETH (478k validators) to fork the blockchain or force a reorganization. Nodes controlled by attackers may participate in or deviate from the blockchain protocol arbitrarily, and may send arbitrary transactions to any smart contract or blockchain address.

We also make the usual assumptions regarding TEE platforms: the attacker has complete control of the physical host executing the TEE and may delay, drop, or reorder messages to and from the TEE. Furthermore, the code and data contained in a TEE are cryptographically protected by a key provisioned by the manufacturer and (subsequently) known only to the TEE, and the TEE implementation is correct in the sense that it successfully enforces the security abstraction it claims to. Key-extraction and other side-channel attacks against a specific realization of TEE hardware (e.g., Intel SGX, AMD SEV, etc.) are beyond the scope of this paper. Attackers are also assumed incapable of breaking cryptographic algorithms employed by Decentagram, the blockchain protocol, or the TEE platform.

Hosts of data oracles are assumed to be rational in that they will transmit new events via transactions when profitable, and that compensation from publication is a sufficient incentive. For any TEE component, fraudulent (local) blockchain forks are detectable through eclipse attack detection [38] schemes, but we do not explicitly adapt and implement such techniques.

For encrypted event messages protected by a shared key, we assume protecting the message and its key is in the interest of both the data oracle and the subscribers. This is an appropriate assumption, for example, when the message contains the private information of the subscriber, but is *not* appropriate for applications such as digital rights management.

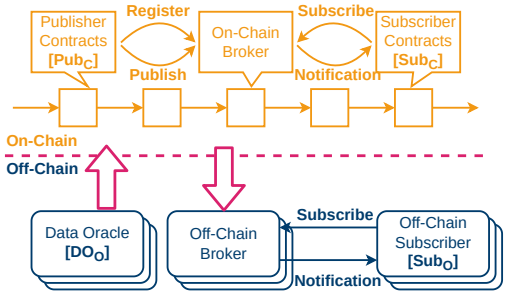


Fig. 1: Overview of Decentagram

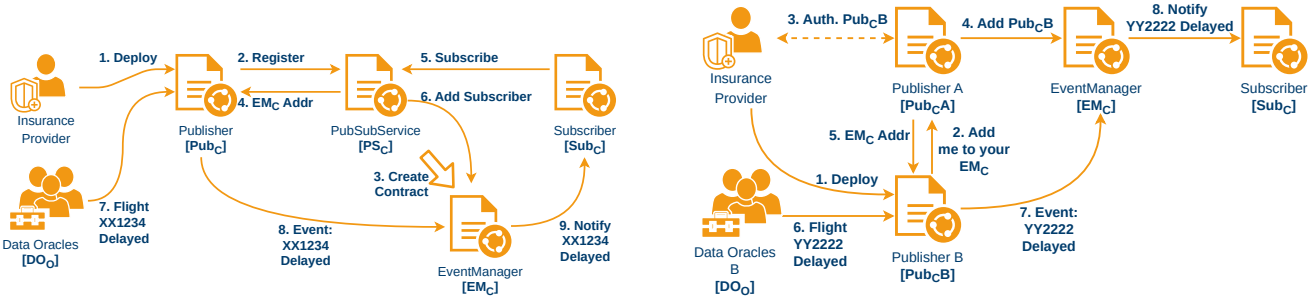
IV. Decentagram Design

Figure 1 shows the architecture of the Decentagram framework, which consists of off-chain data oracles (DO_O)[2] and on-chain publisher (Pub_C), on-chain brokers, off-chain brokers, on-chain subscribers (Sub_C), and off-chain subscribers (Sub_O). DO_O is the source of event data, so it knows what and how to collect data needed by the application. DO_O publishes a new event by making transactions, containing the event data, to the blockchain. The on-chain broker is implemented by a set of smart contracts that incentivize the publication of events and disseminate these events to Sub_C. Sub_O uses the off-chain broker to subscribe to events and receive notifications. Upon receiving a new event, the on-chain broker emits an EVM event, which includes the DO_O's event data in the current block. The off-chain broker then filters the EVM events in each block for those from the on-chain broker contract address.

Decentagram is made up of four types of smart contracts: PubSubService (PS_C), EventManager (EM_C), Publisher (Pub_C), and Subscriber (Sub_C). With respect to traditional Pub/sub systems, the PS_C and EM_C together form the on-chain broker, which indirectly connects publishers and subscribers. PS_C is the first contract deployed on the blockchain, after which all publishers and subscribers can use it by referencing its address. For each new event type that it registers, PS_C will deploy a new EM_C that handles the stream of events for that type. Each Pub_C is the entry point for off-chain data oracles (DO_O). When the DO_O sends transactions with data to its designated Pub_C, the Pub_C is responsible for authenticating the DO_O's data. If DO_O is a trusted source or is providing digitally signed data from a trusted source, the Pub_C can verify the digital signature on the data directly. Otherwise, when DO_O is operated by an untrusted host, Pub_C authenticates the DO_O's TEE to verify it is running known and trusted code, ensuring its content is trustworthy.

Among these components, DO_O, Pub_C, Sub_C, and Sub_O are application dependent, and implemented by application developers. We assume these known and trusted implementations correctly use Decentagram libraries, so all authenticated TEEs contain correctly implemented Decentagram components.

[2]In the following, we subscript off-chain Decentagram components with O for *off-chain*, and on-chain components with C, for *contract*.



(a) Registering a new Publisher and adding a Subscriber.

(b) Registering a new Publisher to another Publisher's EventManager.

Fig. 2: Decentagram's on-chain workflow.

Figure 2a illustrates the interactions between these contracts for the travel insurance example. At deployment, (1) the Pub_C registers itself with the PS_C (2), and receives the address of the new EM_C created by PS_C (4). To subscribe to these on-chain events, a Sub_C calls PS_C (5), specifying the address of the Pub_C it wishes to receive events from and a deposit to compensate the transaction cost of their event notifications. PS_C calls the relevant EM_C (6) to add the new Sub_C's address and callback function to its list of on-chain subscribers. For each update received and *verified* by the Pub_C (7), the Pub_C calls into the EM_C (8) for distribution to the subscribers, who are notified by executing Sub_C's callback function (9).

A. Publisher Contracts for the Same Event.

Allowing more than one Pub_C for the same event type is dangerous since the new Publisher can accept data from a DO_O that the original Publisher would reject. To add an additional Pub_C safely, the initial Pub_C mediates access to its EM_C by other Pub_C. This approach provides a mechanism for expanding DO_O authentication mechanisms or policies and prevents the Pub_C from anticipating such mechanisms at deployment. For example, Figure 2b illustrates Publisher B's contract being added to Publisher A's EM_C. At deployment (1), instead of calling PS_C to create a new EM_C, Publisher B calls into Publisher A to request to be added to its EM_C (2). If the insurance company has authorized Publisher B (3), Publisher A adds the new publisher (4) and returns the address of its EM_C for Publisher B to use (5). Subsequent updates from Publisher B's DO_O (6) are sent to A's event manager (7), and distributed to the same list of subscribers (8).

B. Incentivizing Publications.

In Decentagram, Sub_C rely on the EM_C to notify them of events by invoking their callback function, and the EM_C in turn relies on Pub_C to notify it when these events happen. This series of cross-contract calls is initiated by a transaction from a DO_O, which invokes the Pub_C. Because these cross-contract calls happen within the same transaction, the transaction sender must include the cost of executing the target functions in the Pub_C, EM_C, and each Sub_C registered with the EM_C.

To ensure fair delivery of notifications, the EM_C requires each Pub_C call to include sufficient gas to execute all registered Sub_C callbacks. Without this check, the Pub_C could intentionally under-fund the call causing the transaction to be reverted or only a prefix of Sub_C to be notified. In either case, the Sub_O would be able to observe the transaction included in the block,[3] potentially allowing them to react to the event without Sub_C being notified.

The DO_O initiating the Publisher call is compensated for the cost of executing the transaction and successfully notifying the subscribing contracts by transferring the funds deposited by Sub_C at registration to the DO_O. The fee amount is specified by Pub_C during registration. If a Subscriber's deposit is insufficient, the Subscriber is removed from the list of subscribers before the callback is executed. The oracle may also be rewarded for each valid event submitted to the Pub_C, as specified by the Pub_C at deployment. For example, an application developer could offer a bug bounty program via Pub_C, where the first oracle to report evidence of a compromised component is rewarded. We discuss such an example in more detail in Section VI. Our incentivization mechanism does not guarantee a particular fee schedule results in an incentive-compatible system since externalities may impact whether triggering an event is in the best interest of a DO_O. It does however provide system designers with a tool to compensate for such considerations.

C. Publications Authentication with TEEs.

To our knowledge, Decentagram is the only decentralized pub/sub system that supports permissionless publishing: any entity can act as a DO_O as long as they meet the data verification requirements of Pub_C. When the publication source is independently verifiable, such as a message digitally signed by the original data source, any DO_O can send a transaction containing the signed message. To incentivize timely publication, Pub_C may choose to reward the first successfully committed transaction with the proceeds of notifying the subscribers; remaining successfully committed transactions are refunded minus transaction fees without notifying subscribers.

Some publication data may not be independently verifiable. For example, since data transmitted over a TLS connection is

[3] A reverted transaction does not emit events, but the transaction (including the event data) would still be present in the committed block.

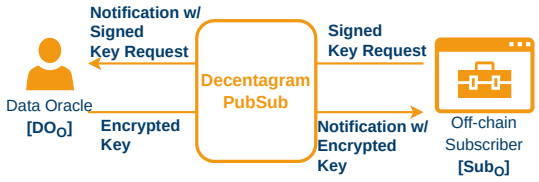


Fig. 3: Key Exchange for Encrypted Event Notifications

protected by an ephemeral shared key, participating parties can authenticate the data, but a third party is unable to guarantee after the fact which party generated the data. For these cases, Decentagram supports DO_O running in TEEs. Running an oracle in a TEE was first used by Town Crier [39], which ensures a webpage is retrieved properly via trusted code running in a TEE. Town Crier nodes are authenticated by off-chain subscribers directly using an interactive remote attestation protocol. Unfortunately, this design scales poorly since the operators of both on-chain and off-chain subscribers would need to independently execute this protocol for each data oracle that may publish events.

In contrast, Decentagram's TEE oracles are verified by Pub_C using self-attestation certificates [36], [37], which authenticate TEEs to independent third parties without requiring an additional attestation round. Since the code authorized to produce updates is specified by the Publisher, Subscribers can assess the trustworthiness of the data sources before subscribing without connecting to specific instances.

D. Supporting Off-Chain Subscribers

The lower right portion of Figure 1 shows the overview of the off-chain portion of Decentagram. All nodes in a blockchain network have access to the blockchain data and the same view of confirmed blocks. The off-chain portion of Decentagram builds on this decentralized design by replicating the off-chain broker: each replica has the same copy of confirmed blockchain data.

Any application can subscribe to the events emitted by a Pub_C by authenticating the closest off-chain broker and subscribing to the events, specifying the address of the Publisher contract. The corresponding off-chain broker will be responsible for notifying the application when the relevant Pub_C emits an event.

E. Encrypted Event Notifications

Here we describe our design for confidential events in Decentagram. To protect event data, a DO_O will encrypt the data using a symmetric key and publish the ciphertext as an event. For highly-available and scalable access to keys, we present a key-exchange protocol for authorized subscribers implemented on top of Decentagram.

Figure 3 shows the overview of this key exchange protocol. Sub_O sends a signed key request, including a public encryption key, to a pre-deployed key-exchange contract, triggering a notification to the DO_O. Based on the signed request, the DO_O will authenticate the requesting Sub_O, and encrypt the current key using the Sub_O's public key. The encrypted key is then sent to the key-exchange contract, which notifies the Sub_O. To mitigate *unintentionally* leaked keys by subscribers, keys should be refreshed periodically and stored in a secure hardware keychain hosted by a TEE or TPM [40]. As discussed in Section III distributing symmetric keys is appropriate when the subscriber can be expected to protect the confidentiality of the data. Encrypted events are only supported for off-chain subscribers since all contract inputs, intermediate values, and outputs are public to all nodes.

F. Dealing with Chain Reorganizations

A chain reorganization happens when two (or more) groups of miners in a blockchain network disagree on the sequence of blocks that form the canonical chain. Eventually, one sequence is extended enough to be accepted by the entire network, but the nodes that initially selected the abandoned chain must roll-back the effects of the abandoned blocks. Recent blockchain protocol designs such as proof-of-stake (PoS) are significantly less susceptible to chain reorganizations [41] than proof-of-work (PoW) protocols, but they are nevertheless possible under the right adversarial and network conditions [42]. Chain reorganizations could cause Sub_O in a pub/sub system to see and react to events in a sequence of blocks, only for those blocks to be rolled back and their transactions committed in a different order on the reorganized chain. Sub_C are unaffected by reorganizations since, like all contracts, they are reverted and re-executed on the new blocks.

An application may be indifferent to the order of events as long as they are eventually delivered (and not duplicated), but some may require a stricter ordering. While individual transactions cannot be replayed due to Ethereum's built-in per-address nonce for replay protection, enforcing a linear ordering of events from multiple oracles requires a nonce per event type, maintained by the contract and incremented for each event. The Publisher contract can require data oracles to include the current nonce. Since the publisher only accepts new events from an authenticated TEE, malicious hosts cannot replay previous events by changing the nonce.

In the event of a reorganization, transactions from different oracles may be reordered in a way that causes a nonce to be invalid and dropped by the Publisher contract. In this case, the oracle should revert its own local state and retry the transaction when the nonce is once again valid. A consequence of the initial transaction being rejected is that a competing oracle may successfully publish an event with that nonce before the original oracle's second transaction succeeds. Once a block is finalized (currently after at most 64 blocks), it can no longer be reorganized, and is considered safe after at most 32 blocks. Luckily, block reorganizations are infrequent (a couple dozen out of over 7000 blocks per day), and almost always have a depth of only one block [41].

V. Implementation

We implemented the Decentagram smart contracts described in Section IV in Solidity, a language for the Ethereum

blockchain. Solidity was chosen because of its maturity and rich set of features. Specifically, we used the setting of gas limits on cross-contract calls, function access controls, access to the transaction cost and value, and digital signature verifications in our on-chain Pub/sub service implementation. Off-chain, we relied on the logging mechanism provided by smart contract event emissions together with bloom filters, transaction receipts, and merkle trees supported by an Ethereum client to filter blocks for events to notify subscribers. All implementations for on-chain and off-chain components are available on GitHub at https://github.com/lsd-ucsc/Decentagram.

A. Decentagram Smart Contracts

The core on-chain components of Decentagram are PS_C and EM_C, which create new event streams for Pub_C and register new Sub_C. Publisher and Subscriber contracts are application-dependent, but we present examples in Section VI.

1) Securing against Problematic Subscribers: Subscribing contracts are untrusted, so we must isolate the execution of on-chain subscriber callbacks to prevent a buggy or malicious callback from causing an entire event notification to fail. All callback invocations are wrapped in *try-catch* blocks (exceptions are ignored), and the gas usage of the call is calculated after the call returns or an exception is caught.

To prevent gas exhaustion attacks [19], the EM_C limits gas usage by callback functions. If gas usage exceeds the remaining gas in the Sub_C's balance or a predefined gas limit set by the Publisher contract, the function is interrupted, and the EM_C continues notifying other Sub_C.

2) On-chain Subscribers and the Gas Limit: Blocks in Ethereum have a size limit that is defined by the maximum amount of computation required to execute the transactions contained in the block. Currently, the gas limit is 30 million [43]; a single transaction using as much as 30 million gas would take up the entire block. Our EM_C contract limits the amount of gas used by callback functions, but the overall transaction gas limit constrains the number of on-chain subscribers a specific EM_C can have.

The gas limit for callback functions is set by the (initial) Pub_C when a new EM_C is created. For reference, the cost to make cross-contract calls (such as to subscriber callback functions) is around 3347 gas, the cost to set a persistent (stored in the contract) boolean flag is 3050 gas, and the cost to add an element to a hash map is 22,200 gas. Assuming Pub_C consumes 2 million gas (the worst case when authenticating a new data source), this leaves 28 million gas remaining as an upper bound for executing subscriber callbacks. Balancing the tradeoff between the number of Sub_C and the callback gas limit is left to Pub_C. Our default gas limit per subscriber is set to 200,000, which allows for about 140 Sub_C.

One way of surpassing the maximum number of Sub_C would be to split subscriber notifications over multiple transactions. The publisher could register and deploy an additional EM_C for each new Sub_C, and notify each EM_C in separate transactions, each with a separate gas limit. The drawback of this approach is that some Sub_C will be notified later than others, and it may be necessary to create additional incentives to ensure the transaction sender notifies all Sub_C (e.g., requiring an up-front deposit that covers the total notification cost).

Only Sub_C actions triggered by a Pub_C transaction through a callback function are subject to these gas limits. For M&R style workflows, a user can use Sub_O to monitor the on-chain events. When a new event is received, the Sub_O relays that event to the desired smart contract in a subsequent transaction. Even more effective is a hybrid approach where a minimal Sub_C performs critical updates to the Sub_C's state, such as revoking access or setting flags. Following this, the Sub_O can send a transaction to complete any remaining tasks related to the event. For example, invoking a callback that just sets a boolean flag costs about 6500 gas. An EM_C with a subscriber gas limit of 6500 can support up to around 4300 Sub_C.

3) Authenticating TEEs On-Chain: Decentagram oracles and off-chain brokers are implemented as decentralized components in the Decent Application Framework [37]. Decent enclaves authenticate themselves to on-chain contracts using a self-attestation certificate: an X.509 certificate chain generated by the secure enclave during the self-attestation process [36]. The root of trust for any TEE remote attestation protocol is the hardware manufacturer's certificate, in this case the Intel Attestation Service (IAS) certificate for Intel SGX enclaves. The certificate chain (see [37] for details) is used to authenticate the Decent enclave's certificate, which contains the enclave component's hash and public key. The hash is used to verify the intended code is running in the enclave, and will be the same for any host executing that Decent component. On startup, each component generates new public/private key pairs and initiates the self-attestation process to generate the certificate it uses to authenticate itself to on-chain contracts, clients, or other Decent components. Note that only the enclave has access to the private keys and mediates all uses of the keys in cryptographic processes. We provide a Solidity library function and pre-deployed smart contracts to verify the validity of this certificate chain. The verification process involves both RSA and ECDSA signature verification, validity period checking, and enclave attestation report parsing, but the Pub_C interface is just a single function call with the self-attestation certificate and the expected code hash. As far as we are aware, this is the first mechanism that verifies the authenticity of a secure enclave from within a smart contract.

Permissionless blockchains are Sybil-resistant [44] by design, but Decentagram's on-chain message authentication and verification prevents Sybils from being reimbursed for publishing spurious events. While any Ethereum client may send transactions to Decentagram contracts, only those produced by authenticated Decentagram components are accepted and compensated with subscriber fees. Thus only messages produced by authentic Decentagram data oracles, which are assumed to be valued by subscribers, will be accepted for publication.

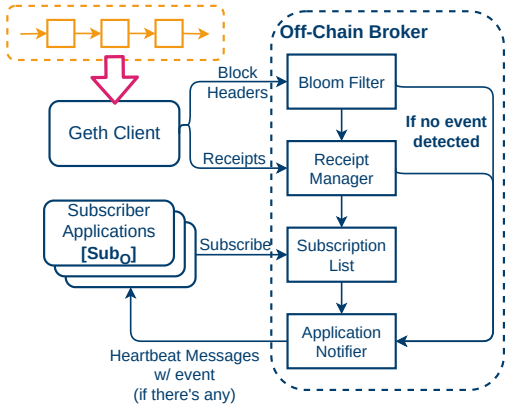


Fig. 4: Overview of the Off-Chain Broker Implementation

B. Sending Events to Off-Chain Subscribers

Figure 4 shows an overview of the off-chain broker's workflow. The Geth Client [45] communicates with other Ethereum nodes to maintain the blockchain data locally. The off-chain broker constantly polls the Geth client for new block headers and checks for new events emitted by the addresses of EM_C contracts it is monitoring. When a new event is received, the off-chain broker consults its list of Sub_O for that event type and sends each Sub_O the new event.

1) Source of Blockdata: The blockchain data is usually generated and stored by the Ethereum nodes in the network. Different nodes in the network communicate to synchronize their view of the chain. Instead of implementing these logics in the off-chain broker, we directly retrieve the blockchain data from the Geth client using RPC calls via TCP connections. Therefore, the Geth client is responsible for maintaining the blockchain data and synchronizing with other nodes in the network. Alternatively, developers may choose to use other Ethereum clients that follow the standard Ethereum protocol and provide compatible RPC interfaces.

To support applications using secure enclaves, our off-chain broker can also run on the Intel SGX secure enclave platform, but additional protection is needed to ensure the authenticity of the blockchain data received from the Geth client running outside the enclave. Being isolated from the OS means that enclave programs have no control over the OS resources, such as the network connections. They have to rely on the programs running outside of the enclave to feed in the blockchain data. An adversary may perform eclipse attacks [46] by crafting fake blockchain data and passing it to the enclave in order to suppress the revocation events. To prevent such attacks, in a PoW blockchain, an eclipse attack detection scheme [38] can be applied. This is less of a concern for PoS blockchains, since recent work suggests that it requires at least 2.4 years to complete the attack [47]. Therefore, proper verification of the blockchain data inside of the enclave is sufficient to prevent eclipse attacks in PoS blockchains like Ethereum.

2) On-Chain Events: After receiving blockchain data, the off-chain broker checks for monitored contract events. Ethereum contracts may emit named tuples called (helpfully) *events*, which are included in the transaction receipt. Decentagram uses these events to communicate with off-chain brokers.

The constructor of PS_C emits a `Deploy` event marking the initialization of the on-chain broker. The off-chain broker then begins to monitor for `Register` events. Each time a publisher registers, a `Register` event announces the address of its corresponding EM_C. With this information, the off-chain broker can maintain a mapping of publisher addresses to event manager addresses, and start to monitor for `Publish` events, in order to know the occurrence of events emitted by that event manager. By knowing the address of PS_C, the off-chain broker can determine the event manager addresses it wants to monitor for `Publish` events.

These events emitted by the on-chain contract will then be visible in the log data of transaction receipts. A naive approach to monitor these events will be retrieving all the receipts of every block and checking their logs to see if any of them contain the events of interest. However, parsing and reading all the receipts incurs a significant amount of overhead, especially when validating the receipts root hash is necessary. We discuss more about this overhead in our evaluation of the off-chain broker in Section VII-C. Instead, we can look at the bloom filter contained in the block header to probabilistically determine if a block contains events we are interested in.

In Ethereum [48], each block header contains a bloom filter that is constructed using the address of contracts that have emitted event(s), the event's signature, and indexed event's arguments. To efficiently filter events, the Broker only has to check the header bloom against the contract addresses and the event signatures. If the filter signals a positive result, the rlp-encoded receipts will be fetched and processed. False positives can occur, but the false positive rate on Ethereum's main chain is promisingly low (about 0.5% [49]).

3) Subscriber Services: The off-chain broker could start at any point of time. Some instances may be started before PS_C is deployed, while other instances may be started after deployment and some events have already been published by EM_C. Similarly, the Sub_O may start subscribing to the Pub_C before or after the Pub_C has published any events. For instance, a Sub_O wants to subscribe to a Pub_C that emits events when a new item is added to a revocation list. If the Sub_O starts subscribing after Pub_C has already published some events, the Sub_O will also want to know what items are already in the revocation list. Because of this, the off-chain broker must maintain a record of all the events that have been published since the deployment of PS_C.

As described above, the off-chain broker knows the beginning of the on-chain Pub/sub services and all the addresses of the existing event manager contracts. During the boot phase of the off-chain broker, it will retrieve history blocks from the Geth client, and record all the past events. So, in addition to notifying the Sub_O of the new events, the off-chain broker will also log these events in storage. At runtime, the off-chain broker accepts event requests from Sub_O, which specify the address of Pub_C at the beginning of a TCP connection. The

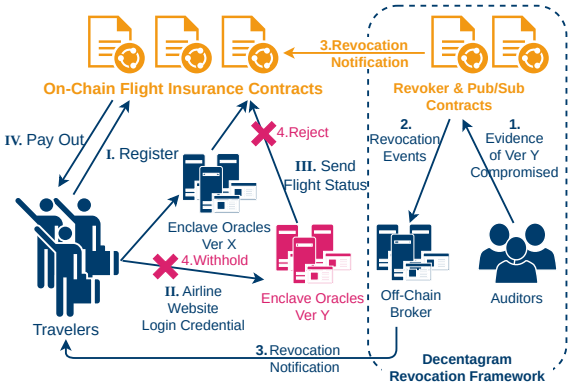


Fig. 5: One implementation of an enclave oracle in the Figure 2 workflow is proven vulnerable by evidence from the auditors.

Broker replies with all the past events and corresponding block numbers from that Pub_C. The Sub_O receives new events over the TCP connection as they occur.

To notify the Sub_O of the new events, the off-chain broker can simply send the event data via the opened TCP connection. However, this approach is not secure enough in case the network connection between the Broker and the Sub_O is untrusted. For instance, an adversary may suppress the TCP connection between them to pretend that the publisher has not published any new events. A subscriber that is listening to the flight delayed events from the off-chain broker may be misled to believe that the publisher has not published any delayed flight since the subscription.

To address this issue, the Broker will send heartbeat messages through the opened TCP connection periodically. Each heartbeat message contains the latest block number of the blockchain. In case a new event is published, the event data will be sent along with the heartbeat message. If the Sub_O does not receive any heartbeat message for a certain period of time, it may assume that the connection is broken, and take appropriate actions to prevent possible losses.

VI. Case Study: Revocation of Enclaves

Decentagram is useful for more than just web oracles that import data from off-chain sources. In the flight insurance example discussed in Section IV, consider an oracle that uses login credentials to an airline's booking system to automatically re-book a client's travel when a cancellation or delay occurs. These credentials are secure from a malicious host if the oracle is running within a TEE, but a vulnerability in the enclave code could cause the client's credentials to be leaked or fraudulent claims disbursed. Once discovered, quickly revoking the credentials of all oracles running the vulnerable component is critical to mitigate exploitation.

Timely revocation is a challenging problem: instantaneous revocation is not possible in almost any distributed system, but is even more challenging in a decentralized system. We present a decentralized solution for timely revocation as a Decentagram service. The insurer's contract receives immediate on-chain notification when a valid revocation is published, preventing fraudulent claim payouts, and off-chain subscribers receive notification when the relevant block is received.

A. State of the Art

A few existing works have discussed using smart contracts to govern the membership of a system. SCPKI [50] is a PKI system that uses smart contracts to manage attributes of participants in the system. IKP [51] provides a smart contract that can parse X.509 certificates to check whether it is verified for a domain. Unlike Decentagram, neither IKP nor Proofchain [52] provide a way for other contracts that depend on certificate revocations to be notified.

In a native enclave model, such as Intel SGX, a platform revocation mechanism is provided by the hardware manufacturer [53], [54], to revoke a compromised hardware platform. However, security vulnerabilities are also commonly found in the software implementations.

Several existing works, such as SCONE [55], Graphene-SGX [56], MAGE [57], Decent [37], and CCF [58], proposed distributed enclave application frameworks, where different enclave components can be deployed on different machines and communicate with each other. SCONE, Graphene-SGX, and MAGE have not explicitly discussed the means to revoke problematic components. In the framework proposed by Decent, replicas of a specialized enclave component distribute the revocation list to other enclave components; but it is unclear how those replicas reach the consensus on what components to revoke. CCF utilizes a permission-based ledger to manage the membership of enclave components. Participants in the system can propose and vote to add or remove enclave components. This approach may not be suitable for applications used by the general public. Additionally, participant interactions may cause unnecessary delays in the revocation decision for certain simple but critical situations, such as when the private key of an enclave application is leaked. Therefore, a more *effective* and *flexible* revocation framework is needed.

B. Decentagram Revocation Framework

We present a revocation framework built on top of Decentagram, with three types of revokers in the role of on-chain publishers as examples: the voting revoker, the conflicting message revoker, and the compromised key revoker. It should be noted that the revocation mechanism is not limited to these three types, and applications can define their own revokers based on their needs. These revokers act as publishers (as in Figure 2a), so each registers with PS_C to obtain an event manager. The Sub_C uses these revokers for timely revocation events. Meanwhile, the off-chain broker listens to the on-chain events emitted by EM_C, and notifies Sub_O.

The conflicting-message and compromised-key revokers require messages to be signed with a key only known to the to-be-revoked enclave in order to prove it has been compromised. To achieve this, we use the same mechanism described in Section V-A3 for authenticating DO_O.

Voting Revoker Contract. The voting revoker is similar to the propose-and-vote scheme used in CCF. During contract construction, the transaction sender needs to specify a list of stakeholders. Then, during runtime, any stakeholder can vote to add a component to the revocation list. If the number of votes reaches a specified threshold (e.g., 2/3 of the stakeholders), the contract adds the component to the revocation list, and publishes a corresponding revocation event using EM_C. Stakeholders can also vote to add more stakeholders, or remove existing stakeholders.

Conflicting Message Revoker Contract. In a distributed application, it is often the case that components need to communicate with each other or with the users. For example, in replicated systems, components may need to vote to elect a leader or decide on a value. If replicas can be Byzantine, these systems can be vulnerable to equivocation from malicious replicas that send conflicting votes [59]–[61]. For this type of malicious behavior, the voting revoker becomes inefficient. Instead, we can use a conflicting message revoker to determine the presence of such behavior and quickly revoke the malicious component, without waiting for votes from stakeholders.

Unlike the regular message signing scheme, where the sender signs on one hash calculated based on the entire message, the message sender here is required to compute a hash of a session ID (e.g., leader selection session X), and a hash of the message content (e.g., vote for node C); then, the sender signs on a single hash that is calculated from both of these hashes. An outside auditor can monitor the session ID hash, the message content hash, and the signature. When two messages have the same session ID hash but different message content hashes, it indicates that the sender has sent conflicting messages; and the auditor can report the corresponding hashes and signatures to the revoker contract. The contract will then check if the hashes and signatures are valid, and add the corresponding component into the revocation list in case of a conflict. By calculating two separate hashes for session ID and message content, neither the auditor nor the blockchain participants can know the actual message content.

Compromised Key Revoker Contract. Another revocation scheme is to revoke a component when its private key is compromised. The private key of an enclave application is stored inside the enclave memory, and is only accessible by the enclave component. Therefore, the revoker contract can quickly revoke a component when it receives a signature over a well-known revocation message (e.g., `"REVOKE THIS KEY"`), signed by a private key that should have been kept secret by the enclave. By verifying the signature of a revocation message, the contract can revoke an enclave component not only when its private key is completely exposed, but also when the private key is partially exposed, such that it is sufficient for an adversary to forge a signature.

Non-Enclave Version. Often times, these revocation schemes also make sense for regular applications that do not use secure enclaves. For example, the voting revoker can be used to propose and vote on a public key, of which the corresponding private key is compromised. Or the conflicting message revoker can be used to detect conflicting messages signed by the same private key. Therefore, we also provide a set of revokers using the same logic as described above, but are used to revoke general EC keys.

After the revoker contracts have verified the evidence and added the corresponding components into the revocation list, an event will be emitted to the transaction receipt by their EM_C. The events included in the receipts will be monitored by the off-chain broker as described in Section V-B. Once an event is found in the receipt, Sub_O will be notified. The Sub_O could be enclave components from a distributed enclave application or an application that communicates with enclave components. After receiving the notification, these subscribers can take appropriate actions to protect themselves from the compromised components such as updating access control lists or purging potentially corrupted data.

By utilizing public blockchain and smart contracts, our approach not only provides a means to distribute the component revocation list, but also provides a decentralized mechanism to add components into the list. The ability to define different types of revokers shows the *flexibility* of the framework, and the timely revocation events delivery provided by Decentagram highlights the *effectiveness* of the framework.

VII. Evaluation

A. Publisher, Subscriber, and On-Chain Broker Contracts

We evaluate the gas cost of our implementation of the on-chain broker with minimal publisher and subscriber contracts. Pub_C registers with PS_C and exposes a function to publish a fixed-payload event. Sub_C subscribes to Pub_C events and verifies it receives the expected payload for each event. To evaluate gas cost, we deployed our contracts on Ganache, a local Ethereum blockchain testing environment [62].

Based on the transaction receipt, the gas cost of deploying PS_C is 623,330. In addition, we evaluated the gas costs of *using* the on-chain broker, including three major operations - registering, subscribing, and publishing. In this experiment, we measured the cost of each operation, and repeated the process with the number of publishers and subscribers increasing from 1 to 20. For event registration, we calculated the average gas used per publisher. As the number of on-chain publishers increases, the average gas used per publisher stays relatively constant, at around 570 thousand gas, with negligible fluctuations (less than 50 gas). Ethereum's mapping data structure allows data to be fetched and stored with constant gas cost, so the gas cost of managing the address of Pub_C and their EM_C does not increase with the map size.

A similar process is used to evaluate the gas cost of subscribing. We deployed the Sub_C, with each of them subscribing to a different Pub_C, and recorded the gas used. As the number of Sub_C increases, the average gas used per subscriber also stays constant, at around 160 thousand gas, with negligible fluctuations (less than 10 gas). Like the `register` operation, the `subscribe` operation uses the mapping data structure to look up the address of the event manager, which results in a constant gas cost.

Operations	Enclaves	secp256k1 Keys
Revocation by Voting		
Deploy	840,815	852,279
Vote (average)	81,386	81,267
Revocation by Conflicting Messages		
Deploy	1,874,373	814,601
Report	1,515,330	70,047
Revocation by Compromised Keys		
Deploy	1,882,836	762,302
Report	1,507,982	66,015

TABLE II: Gas Costs for Revocation Contracts

Next, we evaluated the cost of publishing by deploying multiple Sub_C that subscribe to a single Pub_C. With one subscriber, the publishing cost is 134,768 gas. As the number of Sub_C increases, publishing cost increases linearly, with the marginal cost around 76,000 gas for each additional subscriber. That is because the `publish` operation iterates through the list of Sub_C to invoke their callback functions.

Today, the cost per gas in the Ethereum Mainnet is around 0.001 cents. Thus, the base cost to publish a message is \$1.35 increasing by \$0.76 for each Sub_C. Our contracts are also deployable on EVM-compatible chains. We have deployed and tested Decentagram contracts on Avalanche as well as "Layer 2" (L2) chains Polygon, BNB, and Optimism. L2 chains execute contracts and transactions on a local chain and commit checkpoints to the main Ethereum chain. For L2 chains, off-chain components in Figure 1 would monitor and interact directly with the L2 chain.

Deploying Decentagram to such a chain greatly reduces the cost to publish messages while still benefiting from the security of the Ethereum mainnet. For example, the cost per gas in Polygon network is $4 \cdot 10^{-6}$ cents, and $6 \cdot 10^{-5}$ cents in BNB network. At these prices, the (base, per-subscriber) cost is (\$0.0053, \$0.0030) and (\$0.086, \$0.049), respectively. Note that subscriber fees are expected to reimburse publishing costs, and these prices represent the "break-even" cost for publishers. Higher transaction fees require higher subscriber fees to incentivize publisher participation, so reducing these fees enables a wider range of Decentagram applications.

B. Revocation Contracts

The evaluation of the on-chain revocation contracts consists of two sets of revokers: one is for revoking enclave components, as described in Section VI, and the other one is for revoking general secp256k1 Elliptic Curve (EC) keys; and the results are shown in Table II.

We initialized the revocation-by-voting contracts with three stakeholders, with two votes being sufficient to revoke an enclave image or key. The gas cost of the `vote` operation is calculated by averaging the gas used in these two votes. As shown in the table, both of the voting revokers have similar gas costs since counting votes is similar for each situation.

The gas cost of the conflicting messages revoker, as well as the compromised key revoker, are significantly higher for enclaves than for general EC keys. This is because the two enclave revokers need to verify the remote attestation reports from the running enclave component, since the conflicting messages revoker needs to ensure that the messages are generated by the same instance of an enclave, and the compromised key revoker needs to verify that the proposed keys are held by an instance of the enclave component. The verification of the remote attestation reports includes one verification of RSA-signed X.509 certificates, one verification of RSA signatures, and one JSON message decoding, which are expensive in terms of gas costs. High gas cost is typical for frameworks that parse and manage X.509 certificates on-chain (e.g., [63]). Since revocations occur infrequently and are critical to security, these gas costs are justifiable. A vulnerable Decent component only needs to be revoked once: even if a new host loads the component and generates a new signing key, the attestation certificate will be rejected due to the revoked component identity. Enclave application developers could provide bug bounties to incentivize the disclosure of vulnerable components and cover the cost of the revocation process.

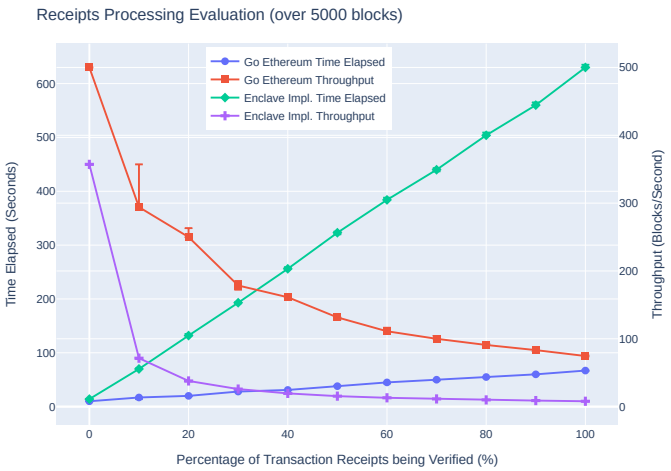


Fig. 6: Off-chain Receipts Processing Evaluation. Error bars (where visible) represent the max and min of three runs.

C. Off-chain Block Processing

In the off-chain evaluation, we focus on block processing efficiency, so that the Sub_O can be notified of the events in these blocks in a timely manner. Among those steps described in Section IV, we have identified the major overhead in block processing as being the parsing and validating of transaction receipts. We evaluated two versions of receipt processing implementations: one is our implementation built and run inside of the enclave environment, required by the framework described in Section VI, and the other version uses Geth under the normal execution environment. The experiment is conducted on a PC running the Ubuntu operating system, equipped with an i3-7100T CPU, and 32G of RAM. The results are shown in Figure 6.

The off-chain broker uses the bloom filter in the block header to skip processing of blocks that return a negative

	$\mathbf{DO}_O \rightarrow \mathbf{Pub}_C$	$\mathbf{Pub}_C \rightarrow \mathbf{Sub}_C$	$\mathbf{DO}_O \rightarrow \mathbf{Sub}_C$
Decentagram	12 s (Max:35, Min:5)	0 s (Max:0, Min:0)	12 s (Max:35, Min:5)
M&R	12 s (Max:46, Min:10)	12.5 s (Max:35, Min:9)	25 s (Max:58,Min:22)

TABLE III: Median End-to-End Latency Comparison.

result. In our experiment, we simulate positive bloom filter results for 0% to 100% of 5000 blocks in 10% increments. These blocks were selected from the range 8,875,000 to 8,880,000, in the Ethereum Goerli testnet. We can see that the time taken by both implementations increases linearly as the possibility increases. Even when receipts from every block have to be parsed and verified, the enclave implementation shows a throughput of 7.94 blocks/second, which is around 95 times faster than the Ethereum block arrival rate of 12 seconds/block. In case of Geth, the throughput is 74.63 blocks/second, which is around 896 times faster than the block arrival rate. Hence, regardless of the off-chain broker version the Sub_O uses, there should not be any backlog of new blocks, and Sub_O will be notified of new events in a timely manner.

D. End-to-End Latency Comparison

Decentagram only requires that a candidate event be signed by an authenticated component, avoiding the need for an off-chain consensus round prior to publication as in Chainlink and Chios. Therefore, we evaluate the latency of Decentagram by measuring the time from when events are published to when they are delivered. The M&R approach serves as a good baseline for comparison since it also does not require consensus prior to publication. To compare the efficiency of Decentagram with the traditional M&R approach, we conducted an experiment to measure the end-to-end latency of notification delivery. In both test cases, a DO_O publishes new data to a Pub_C, and a Sub_C waits to be notified of the new data. Additionally, a Sub_O monitors new blocks to be notified of the new data as well as the confirmation from the Sub_C showing that it has processed the data. The Ethereum Goerli testnet, which has a block arrival rate of 12 seconds/block, was used in this experiment; and all the test cases were repeated 20 times, with the median, minimum, and maximum values reported in Table III. The first column shows the time elapsed from DO_O publishes data until Sub_O receives it via the event emitted by Pub_C. The result between the two approaches are almost the same, since both of them require the DO_O to make a transaction to Pub_C, and then Sub_O will be notified when they receive the event. The second column shows the time required for the Sub_O to receive the subsequent confirmation from Sub_C after it has been notified of the event from Pub_C.

As expected, the difference in latency between the two approaches is significant. With Decentagram, the Pub_C was able to notify the Sub_C via cross-contract call within the same transaction. Thus, when Sub_O received the event, it also received the confirmation that Sub_C has finished processing the data. While in the M&R approach, the Sub_O has to react to the event by making another transaction to Sub_C, which results in a longer latency. The third column shows the total time elapsed from event publication to the confirmation from Sub_C. Decentagram is able to complete the entire process using only one transaction, reducing the latency by half.

During the experiment, we encountered network fluctuations, with some time slots being skipped causing the new block to arrive later than expected, and some blocks being empty causing our transactions to be delayed until the next block. The time it takes for data to propagate from DO_O to Sub_C took even longer when these two situations occurred simultaneously. Such situations are common on the testnet but rare on the mainnet. However, the mainnet is also more congested, which would also cause similar effects. In both cases, Decentagram is less affected by these fluctuations since it only requires one transaction to notify subscriber contracts compared to two transactions required by the M&R approach.

E. Application Domain

Topics in Decentagram are identified by the Pub_C address, so the number of channels is not limited in any practical way (there are 2^{160} distinct contract addresses). The number of Sub_C addresses per channel is capped by the block gas limit (see Section V-A2). In our default configuration, each channel can support up to 140 on-chain subscribers. Note this limit only applies to *on-chain* subscribers—there are no limitations on the number of *off-chain* subscribers. In fact, since the off-chain brokers are decentralized, the number of off-chain subscribers scales indefinitely.

Message delivery latency relies on the block arrival rate of the underlying blockchain network. The 12 seconds/block rate is the result of our use of the Ethereum blockchain. EVM-compatible L2 networks such as Polygon or BNB have faster block rates (2 or 3 seconds, respectively). Taking these aspects into account, Decentagram is most suitable for applications that have event-generation rates on the order of seconds, need to scale to massive numbers of off-chain subscribers, with on-chain subscribers reasonably distributed over many topics.

VIII. Conclusion

We presented Decentagram, a framework for instant delivery of on-chain events and timely delivery of off-chain events using the pub/sub messaging model. Our Publisher, Subscriber, and Broker contracts are resilient to Byzantine failures and provide incentives for event publications. We motivate the framework with a decentralized revocation case study, demonstrating its benefits over the state-of-the-art in revocation speed for on-chain subscribers. We evaluated Decentagram's gas cost for contract deployment and execution, and demonstrated receipts processing throughput of the off-chain broker is more than sufficient for processing new blocks at the rate they arrive.

Acknowledgements

Partial funding for this research was provided by NSF CAREER grant CNS-1750060.

REFERENCES

[1] J. Kreps, N. Narkhede, and J. Rao, "Kafka: a distributed messaging system for log processing," in *Proceedings of the NetDB*, ser. NetDB'11, vol. 11. New York, NY, USA: Association for Computing Machinery, Jun. 2011, pp. 1–7. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2017/09/Kafka.pdf

[2] Google Cloud, "What is Pub/Sub?" May 2023. [Online]. Available: https://cloud.google.com/pubsub/docs/overview

[3] RabbitMQ, "RabbitMQ," https://www.rabbitmq.com/, 2023. [Online]. Available: https://www.rabbitmq.com/

[4] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout, "Java message service," Sun Microsystems, Santa Clara, CA, Tech. Rep., Dec. 2002. [Online]. Available: https://download.oracle.com/otn-pub/jcp/jms-2_0-pr-spec/JMS20.pdf

[5] Amazon Web Services, "Amazon SNS," https://aws.amazon.com/sns/, 2023. [Online]. Available: https://aws.amazon.com/sns/

[6] A. Machanavajjhala, E. Vee, M. Garofalakis, and J. Shanmugasundaram, "Scalable ranked publish/subscribe," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 451–462, Aug. 2008. [Online]. Available: https://doi.org/10.14778/1453856.1453906

[7] B. Chandramouli and J. Yang, "End-to-end support for joins in large-scale publish/subscribe systems," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 434–450, Aug. 2008. [Online]. Available: https://doi.org/10.14778/1453856.1453905

[8] B. Eze, C. Kuziemsky, L. Peyton, G. Middleton, and A. Mouttham, "Policy-based data integration for e-health monitoring processes in a B2B environment: Experiences from canada," *Journal of Theoretical and Applied Electronic Commerce Research*, vol. 5, no. 1, pp. 56–70, Apr. 2010. [Online]. Available: https://doi.org/10.4067/S0718-18762010000100006

[9] S. Kul, S. Eken, and A. Sayar, "Distributed and collaborative real-time vehicle detection and classification over the video streams," *International Journal of Advanced Robotic Systems*, vol. 14, no. 4, Jul. 2017. [Online]. Available: https://doi.org/10.1177/1729881417720782

[10] S. Kul, I. Tashiev, A. Şentaş, and A. Sayar, "Event-based microservices with apache kafka streams: A real-time vehicle detection system based on type, color, and speed attributes," *IEEE Access*, vol. 9, pp. 83 137–83 148, Jun. 2021. [Online]. Available: https://doi.org/10.1109/ACCESS.2021.3085736

[11] G. S. Ramachandran, K.-L. Wright, L. Zheng, P. Navaney, M. Naveed, B. Krishnamachari, and J. Dhaliwal, "Trinity: A byzantine fault-tolerant distributed publish-subscribe system with immutable blockchain-based persistence," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. Institute of Electrical and Electronics Engineers, May 2019, pp. 227–235. [Online]. Available: https://doi.org/10.1109/BLOC.2019.8751388

[12] N. Zupan, K. Zhang, and H.-A. Jacobsen, "Hyperpubsub: A decentralized, permissioned, publish/subscribe service using blockchains: Demo," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos*, ser. Middleware '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 15–16. [Online]. Available: https://doi.org/10.1145/3155016.3155018

[13] C. Smith, S. Supreme, C. Badhe, and T. Pfledderer, "Ethereum proof-of-stake attack and defense," https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/attack-and-defense/, Jun. 2023.

[14] S. Arnautov, A. Brito, P. Felber, C. Fetzer, F. Gregor, R. Krahn, W. Ozga, A. Martin, V. Schiavoni, F. Silva, M. Tenorio, and N. Thümmel, "PubSub-SGX: Exploiting trusted execution environments for privacy-preserving publish/subscribe systems," in *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, ser. SRDS '18. New York, NY, USA: Institute of Electrical and Electronics Engineers, Oct. 2018, pp. 123–132. [Online]. Available: https://doi.org/10.1109/SRDS.2018.00023

[15] K.-L. Wright, M. Martinez, U. Chadha, and B. Krishnamachari, "Smartedge: A smart contract for edge computing," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. Institute of Electrical and Electronics Engineers, Jul. 2018, pp. 1685–1690. [Online]. Available: https://doi.org/10.1109/Cybermatics_2018.2018.00281

[16] M. Alharby and A. Van Moorsel, "Blockchain-based smart contracts: A systematic mapping study," *arXiv preprint arXiv:1710.06372*, 2017.

[17] A. Esmat, M. de Vos, Y. Ghiassi-Farrokhfal, P. Palensky, and D. Epema, "A novel decentralized platform for peer-to-peer energy trading market with blockchain technology," *Applied Energy*, vol. 282, p. 116123, 2021.

[18] L. Breidenbach, C. Cachin, A. Coventry, A. Juels, and A. Miller, "Chainlink off-chain reporting protocol," https://research.chain.link/ocr.pdf, Chainlink Labs, Tech. Rep., Feb. 2021. [Online]. Available: https://research.chain.link/ocr.pdf

[19] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings 6*. Springer, 2017, pp. 164–186.

[20] N. Carriero and D. Gelernter, "Linda in context," *Communications of the ACM*, vol. 32, no. 4, pp. 444–458, Apr. 1989. [Online]. Available: https://doi.org/10.1145/63334.63337

[21] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, Aug. 2001. [Online]. Available: https://doi.org/10.1145/380749.380767

[22] K. P. Birman and T. A. Joseph, "Reliable communication in the presence of failures," *ACM Transactions on Computer Systems*, vol. 5, no. 1, pp. 47–76, Jan. 1987. [Online]. Available: https://doi.org/10.1145/7351.7478

[23] K. P. Birman, "Replication and fault-tolerance in the ISIS system," in *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, ser. SOSP '85. New York, NY, USA: Association for Computing Machinery, Dec. 1985, pp. 79–86. [Online]. Available: https://doi.org/10.1145/323647.323636

[24] S. Duan, C. Liu, X. Wang, Y. Wu, S. Xu, Y. Yesha, and H. Zhang, "Intrusion-tolerant and confidentiality-preserving publish/subscribe messaging," in *2020 International Symposium on Reliable Distributed Systems (SRDS)*. Institute of Electrical and Electronics Engineers, Sep. 2020, pp. 319–328. [Online]. Available: https://doi.org/10.1109/SRDS51746.2020.00039

[25] R. S. Kazemzadeh and H.-A. Jacobsen, "Reliable and highly available distributed publish/subscribe service," in *2009 28th IEEE International Symposium on Reliable Distributed Systems*. Institute of Electrical and Electronics Engineers, Sep. 2009, pp. 41–50. [Online]. Available: https://doi.org/10.1109/SRDS.2009.32

[26] Beaconscan, "Statistics & charts, mainnet beacon chain (phase 0) ethereum 2.0 explorer," https://beaconscan.com/statistics, 2023.

[27] C. Berger, S. B. Toumia, and H. P. Reiser, "Does my bft protocol implementation scale?" in *Proceedings of the 3rd International Workshop on Distributed Infrastructure for the Common Good*, 2022, pp. 19–24.

[28] J. Bacon, D. M. Eyers, J. Singh, and P. R. Pietzuch, "Access control in publish/subscribe systems," in *Proceedings of the Second International Conference on Distributed Event-Based Systems*, ser. DEBS '08. New York, NY, USA: Association for Computing Machinery, Jul. 2008, pp. 23–34. [Online]. Available: https://doi.org/10.1145/1385989.1385993

[29] L. I. W. Pesonen and J. Bacon, "Secure event types in content-based, multi-domain publish/subscribe systems," in *Proceedings of the 5th International Workshop on Software Engineering and Middleware*, ser. SEM '05. New York, NY, USA: Association for Computing Machinery, Sep. 2005, pp. 98–105. [Online]. Available: https://doi.org/10.1145/1108473.1108495

[30] Y. Zhao and D. C. Sturman, "Dynamic access control in a content-based publish/subscribe system with delivery guarantees," in *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*. Institute of Electrical and Electronics Engineers, Jul. 2006, pp. 60–60. [Online]. Available: https://doi.org/10.1109/ICDCS.2006.32

[31] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, "FastKitten: Practical smart contracts on bitcoin," in *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Aug. 2019, pp. 801–818. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/das

[32] S. Gaddam, R. Kumaresan, S. Raghuraman, and R. Sinha, "LucidiTEE: Scalable policy-based multiparty computation with fairness," in *Cryptology and Network Security*, J. Deng, V. Kolesnikov, and A. A. Schwarzmann, Eds., vol. 14342. Singapore: Springer Nature Singapore, Oct. 2023, pp. 343–367. [Online]. Available: https://doi.org/10.1007/978-981-99-7563-1_16

[33] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *2019 IEEE*

European Symposium on Security and Privacy (EuroS&P). Institute of Electrical and Electronics Engineers, Jun. 2019, pp. 185–200. [Online]. Available: https://doi.org/10.1109/EuroSP.2019.00023

[34] M. Bowman, A. Miele, M. Steiner, and B. Vavala, "Private data objects: an overview," Intel Labs, Tech. Rep., Nov. 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1807.05686

[35] H. Shen, *Content-Based Publish/Subscribe Systems*. Boston, MA: Springer US, Oct. 2009, pp. 1333–1366. [Online]. Available: https://doi.org/10.1007/978-0-387-09751-0_49

[36] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing, and M. Vij, "Integrating remote attestation with transport layer security," Intel Corporation, Tech. Rep., Jan. 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1801.05863

[37] H. Zheng and O. Arden, "Secure distributed applications the decent way," in *Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems*, ser. ASSS '21. New York, NY, USA: Association for Computing Machinery, Jun. 2021, pp. 29–42. [Online]. Available: https://doi.org/10.1145/3457340.3458304

[38] H. Zheng, T. Tran, and O. Arden, "Total eclipse of the enclave: Detecting eclipse attacks from inside TEEs," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, ser. ICBC '21. New York, NY, USA: Institute of Electrical and Electronics Engineers, May 2021, pp. 1–5. [Online]. Available: https://doi.org/10.1109/ICBC51069.2021.9461081

[39] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 270–282. [Online]. Available: https://doi.org/10.1145/2976749.2978326

[40] T. C. Group, https://trustedcomputinggroup.org/, 2023.

[41] Etherscan, "Forked blocks," https://etherscan.io/blocks_forked/, 2023.

[42] M. Neuder, D. J. Moroz, R. Rao, and D. C. Parkes, "Low-cost attacks on ethereum 2.0 by sub-1/3 stakeholders," *arXiv preprint arXiv:2102.02247*, 2021.

[43] YCHARTS, "Ethereum average gas limit," https://ycharts.com/indicators/ethereum_average_gas_limit, 2023. [Online]. Available: https://ycharts.com/indicators/ethereum_average_gas_limit/

[44] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[45] Ethereum Foundation, "Go ethereum - official go implementation of the ethereum protocol," https://github.com/ethereum/go-ethereum, 2023. [Online]. Available: https://github.com/ethereum/go-ethereum

[46] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th USENIX Security Symposium (USENIX Security 15)*. Online Proceedings: USENIX Association, Aug. 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[47] S. Zhang and J.-H. Lee, "Eclipse-based stake-bleeding attacks in PoS blockchain systems," in *Proceedings of the 2019 ACM International Symposium on Blockchain and Secure Critical Infrastructure*, ser. BSCI '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 67–72. [Online]. Available: https://doi.org/10.1145/3327960.3332391

[48] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Foundation, Tech. Rep., 2022. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[49] Ryan Schneider, "Bloom filter false positive rate w/ ERC-20/721," https://github.com/ethereum/go-ethereum/issues/17613, 2018. [Online]. Available: https://github.com/ethereum/go-ethereum/issues/17613

[50] M. Al-Bassam, "SCPKI: A smart contract-based PKI and identity system," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, ser. BCC '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 35–40. [Online]. Available: https://doi.org/10.1145/3055518.3055530

[51] S. Matsumoto and R. M. Reischuk, "Ikp: Turning a pki around with decentralized automated incentives," in *2017 IEEE Symposium on Security and Privacy (SP)*. Institute of Electrical and Electronics Engineers, May 2017, pp. 410–426. [Online]. Available: https://doi.org/10.1109/SP.2017.57

[52] T. Saleem, M. U. Janjua, M. Hassan, T. Ahmad, F. Tariq, K. Hafeez, M. A. Salal, and M. D. Bilal, "Proofchain: An x.509-compatible blockchain-based pki framework with decentralized trust," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 213, no. C, Aug. 2022. [Online]. Available: https://doi.org/10.1016/j.comnet.2022.109069

[53] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel software guard extensions: EPID provisioning and attestation services," Intel Corporation, Tech. Rep., Mar. 2016. [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/671370?fileName=ww10-2016-sgx-provisioning-and-attestation-final.pdf

[54] V. Scarlata, S. Johnson, J. Beaney, and P. Zmijewski, "Supporting third party attestation for Intel SGX with Intel data center attestation primitives," Intel Corporation, Tech. Rep., Apr. 2019. [Online]. Available: https://cdrdv2-public.intel.com/671314/intel-sgx-support-for-third-party-attestation.pdf

[55] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure linux containers with Intel SGX," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 689–703. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov

[56] C. che Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A practical library OS for unmodified applications on SGX," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, Jul. 2017, pp. 645–658. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai

[57] G. Chen and Y. Zhang, "MAGE: Mutual attestation for a group of enclaves without trusted third parties," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 4095–4110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/chen-guoxing

[58] M. Russinovich, E. Ashton, C. Avanessians, M. Castro, A. Chamayou, S. Clebsch, M. Costa, C. Fournet, M. Kerner, S. Krishna, J. Maffre, T. Moscibroda, K. Nayak, O. Ohrimenko, F. Schuster, R. Schwartz, A. Shamis, O. Vrousgou, and C. M. Wintersteiger, "CCF: A framework for building confidential verifiable replicated services," Microsoft Research, Tech. Rep., Apr. 2019. [Online]. Available: https://github.com/microsoft/CCF/blob/0d43355/CCF-TECHNICAL-REPORT.pdf

[59] I. Abraham, G. Gueta, D. Malkhi, L. Alvisi, R. Kotla, and J.-P. Martin, "Revisiting fast practical byzantine fault tolerance," Dec. 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1712.01367

[60] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel, "Cheapbft: Resource-efficient byzantine fault tolerance," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: Association for Computing Machinery, Apr. 2012, pp. 295–308. [Online]. Available: https://doi.org/10.1145/2168836.2168866

[61] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Third Symposium on Operating Systems Design and Implementation (OSDI 99)*. USENIX Association, Feb. 1999, pp. 173–186. [Online]. Available: https://www.usenix.org/legacy/publications/library/proceedings/osdi99/castro.html

[62] Truffle Suite, "What is ganache?" https://trufflesuite.com/docs/ganache/, 2023. [Online]. Available: https://trufflesuite.com/docs/ganache/

[63] A. S. Ahmed and T. Aura, "Turning trust around: Smart contract-assisted public key infrastructure," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. Institute of Electrical and Electronics Engineers, Aug. 2018, pp. 104–111. [Online]. Available: https://doi.org/10.1109/TrustCom/BigDataSE.2018.00026