RSS: Robust Signing Service using Threshold Signatures and TEEs

FILIP REZABEK, TUM School of CIT, Technical University of Munich, Germany
KILIAN GLAS, TUM School of CIT, Technical University of Munich, Germany
EBER CHRISTER, TUM School of CIT, Technical University of Munich, Germany
XINXIN FAN, IoTeX, USA
GEORG CARLE, TUM School of CIT, Technical University of Munich, Germany

Threshold signatures reduce the risk of single-key compromise by distributing signing authority, but each key share remains exposed to compromise of the software and infrastructure that execute the protocol. We present RSS, a threshold signing service that runs share generation and signing inside Trusted Execution Environments (TEEs). We integrate GG20 threshold ECDSA, FROST, and threshold BLS into the EnGINE experimentation framework and evaluate local and Google Cloud deployments using AMD SEV-SNP and Intel TDX. Our experiments separate distributed key generation (DKG), preprocessing, and online signing, and cover up to 40 logical protocol participants distributed across four physical hosts or confidential VMs (CVMs). In matched-platform comparisons, confidential execution adds limited overhead relative to protocol and deployment effects. DKG is the main scaling bottleneck: for 40 participants, it completes within seconds in the evaluated configurations, whereas signing completes in tens of milliseconds. Threshold BLS is approximately twice as slow as FROST for comparable values of n and t. These results establish the performance feasibility of executing threshold-signature workloads inside CVMs under benign-operation assumptions. The evaluation does not cover a complete attestation-bound provisioning lifecycle, persistent-state rollback protection, or Byzantine fault behavior.

Additional Key Words and Phrases: TEE, Threshold Cryptography, Methodology, Evaluation

1 Introduction

Recent research focusing on Threshold Signature Scheme (TSS) shifted towards applications in blockchain and cryptocurrencies [3, 40, 61]. Historically, TSSs were used for protecting, e.g., a Certificate Authority (CA)'s private key [42]. The TSSs provide cryptographic wallet protection [61], optimize consensus mechanism, or serve in various blockchain infrastructures such as bridging [67, 71] or oracles [18]. The consensus protocols rely on TSS to enhance their performance or security during the binary agreement of Byzantine Fault Tolerance (BFT) protocols [33, 50, 65, 87]. The replicas respond to a proposal with the partial signatures of their votes. Bridge designs often rely on a centralized set of peers that use TSS for signing transactions. Oracles allow off-chain data to be used as input to a function in smart contracts [18]. The data is processed automatically, so it is important to ensure its authenticity and validity [17]. Data authenticity can be achieved by TSS. Therefore, various TSSs have been proposed to improve on various aspects of previous designs [7, 39, 56, 60, 61, 70]. However, with the plethora of proposed applications and available TSSs, it is challenging to establish an overview of their use case requirements and properties. The National Institute of Standards and Technology (NIST) First Call for Multi-Party Threshold Schemes specifies a process for collecting threshold-scheme specifications, reference implementations, and experimental evaluations [15]. Even though threshold cryptography can mitigate a single point of failure, there have still been attacks on real-world deployments, especially in the blockchain domain [4, 25, 88]. Such attacks led to significant financial and reputation losses. Therefore, we introduce Robust Signing Service (RSS), which combines TSSs with Trusted Execution Environments (TEEs). Our work assesses the feasibility of RSS from theoretical and empirical perspectives.

Traditional hardware security modules can protect conventional signing keys, but generic support for the interactive protocols and auxiliary state required by threshold signature implementations is limited. In software deployments, a

key share and its protocol state reside in the memory of the guest that executes the protocol. A compromised host or hypervisor may therefore expose an individual share, even though compromising fewer than t shares should not enable a forgery under the security assumptions of the underlying threshold scheme.

RSS uses TEEs as a defense-in-depth layer: each threshold participant executes inside a confidential guest that is isolated from the underlying host or hypervisor, while the threshold scheme continues to distribute trust across participants. This combination does not prevent a malicious quorum of t authorized participants from signing, and it does not remove the TEE's trusted computing base risks. Remote attestation can authenticate a measured guest before share provisioning, but a production lifecycle also requires an attestation-bound channel, persistent-state protection, rollback handling, revocation, and recovery. Our experiments focus on the performance of threshold-protocol execution in Intel Trusted Domain Extensions (TDX) and AMD Secure Encrypted Virtualization (SEV)-Secure Nested Paging (SNP). Section 3.2 defines the corresponding security scope.

From the available TSSs, we select prominent schemes from the Elliptic-Curve DSA (ECDSA) family based on Gennaro et al. [39], Flexible Round-Optimized Schnorr Threshold (FROST) relying on Schnorr [56], and Boneh-Lynn-Shacham (BLS) [37]. Threshold ECDSA is especially relevant for legacy reasons and wide adoption in its non-threshold form. Schnorr's scheme aims to simplify signature generation in both its threshold and non-threshold forms and gained attention after its patent expired. Both schemes support a decoupled preprocessing phase of per-signature random nonces, resulting in a single online round when signing a message. Due to the reliance on pairing-based cryptography, BLS allows the aggregation of several signatures and possibly different public keys [10, 37]. The aggregation requires changes in the verification algorithm, increases implementation overhead, and might impact the privacy of partial signers. However, the aggregated signatures have a smaller memory footprint. We focus on TSSs that do not impact signature verification.

We present a methodological approach to understand each component in detail. For that, we investigate various frameworks for facilitating experiments for both TEE and TSSs. We extend an open-source framework called EnGINE/METHODA [74–76] introduced by Rezabek et al. Its implementation [73] relies on Ansible [72] and provides the foundation for infrastructure deployments, emphasizing fine-tuned experiment setups. EnGINE is a suitable base for extension towards large-scale deployments as it provides the foundation for infrastructure deployments, emphasizing fine-tuned experiment setups unlike other solutions [47, 66]. We extend EnGINE by cloud deployments relying on Google Cloud Platform (GCP), TSSs, and TEEs. We propose a methodological approach to assess various solutions on a common platform and observe their interactions and possible implications on scenario-specific Key Performance Indicators (KPIs). Across matched-platform comparisons, confidential execution generally introduces limited overhead or additional variance, while protocol and deployment choices dominate much of the observed End-to-End (E2E) cost.

Our key contributions (KCs) are as follows:

- **KC1** We define an RSS deployment model and threat model that separate threshold participants, TEE instances, and infrastructure failure domains, and that state the guarantees and non-goals of their combination.
- **KC2** We provide a common analytical methodology for comparing threshold ECDSA, FROST, and threshold BLS in terms of protocol phases, communication rounds, implementation-level cost, and security properties.
- **KC3** We extend EnGINE with cloud deployment, Confidential VMs (CVMs) computing, and full-stack TSS experiments.
- **KC4** We evaluate DKG, preprocessing, and signing in local and Google Cloud configurations using AMD SEV-SNP and Intel TDX, with up to 40 logical participants distributed across four physical hosts or CVMs.

2 Background

This section introduces key details of TEEs and selected TSSs, followed by a description of EnGINE.

2.1 Trusted Execution Technologies

TEEs establish secure enclaves within processors, safeguarding code and sensitive data from unauthorized access. There are two main categories: process-based and Virtual Machine (VM)-based solutions [58]. Process-based solutions, exemplified by Intel Software Guard Extensions (SGX) [85], enable the creation of secure enclaves with restricted interaction. VM-based TEEs, such as Intel TDX [54, 78] or AMD SEV-SNP [1, 58, 64], enhance virtual machine security through encrypting and isolating guest VMs from the hypervisor and supporting nested virtualization. These technologies cater to specific applications. Kata containers support deployments to VM-based TEEs solutions.

2.2 Threshold Signature Schemes

The idea of threshold signing was introduced by Desmedt [31, 32]. A t-out-of-n TSS distributes signing-key material among n participants such that at least t participants must cooperate to produce a valid signature.

The security and availability properties of a TSS must be distinguished. Unforgeability commonly assumes that an adversary corrupts strictly fewer than t participants, although the precise corruption threshold and adversarial model depend on the protocol. Availability instead requires at least t responsive participants. Consequently, a basic t-out-of-n deployment can tolerate at most $n - t$ unavailable participants, and possibly fewer during a particular signing attempt when the protocol uses a fixed signer set or is not robust to aborts. Protocol robustness describes whether missing or malformed responses can be identified and bypassed. We keep unforgeability, availability, and robustness distinct throughout this work. The special case $t = n$ requires all registered participants to cooperate.

We distinguish four operations: key generation, optional preprocessing, online signing, and verification. Key generation creates the public key and distributes a secret-key share to each participant. The shares may be generated collaboratively using Distributed Key Generation (DKG), or distributed by a trusted dealer using, for example, Shamir's Secret Sharing (SSS) [82]. Depending on the scheme, key generation may also produce auxiliary protocol data. When supported, Distributed Key Refresh (DKR) updates existing shares without reconstructing the secret key [19].

For nondeterministic signature schemes, an optional preprocessing phase generates message-independent state, such as per-signature nonces or nonce commitments. Performing this work offline can reduce the number of message-dependent communication rounds. During online signing, the selected participants generate partial signatures, verify partial signatures where required, and combine at least t valid contributions into the final signature.

Within each evaluated signature family, the resulting threshold signature can be verified using the corresponding ordinary verification algorithm. The verifier therefore need not know whether the signature was generated by a single signer or by a threshold protocol.

2.2.1 Schnorr. The FROST scheme [56] is based on the Schnorr signature scheme [79]. FROST prioritizes efficiency over robustness. Robustness in this context refers to the behavior of FROST, where a predetermined set of peers is selected from which a response is expected, and if any single peer does not respond, no signature is generated. Successor schemes, e.g., ROAST [77] or FROST2 [26], are robust but introduce communication overhead. Like plain Schnorr signatures, FROST is non-deterministic because it requires random nonces for each signature. The nonce can be precomputed, requiring only one communication round for multiple *Signing* rounds in a batch processing fashion [56].

Consecutively, a single communication round is required in the *Signing* step, with each peer computing and returning the partial signature share. For our work, we extend the frost-dalek Rust library [57].

2.2.2 ECDSA. ECDSA is popular in its non-threshold form, thus, we see many threshold schemes based on ECDSA [38, 48, 70]. We employ *GG20* scheme [39], which introduced an identifiable abort and precomputation of random nonces. The identifiable abort identifies misbehaving parties during the DKG or *Signing* phase. *GG20* builds upon the previous *GG18* scheme [38], which is lacking the nonce precomputation and identifiable aborts. We extend the ZenGo-X ECDSA project [90]. We also implemented a dedicated DKR protocol for GG20 and GG18.

2.2.3 BLS. BLS, unlike Schnorr or ECDSA, is a deterministic scheme [10, 37] and relies on pairings and pairing-friendly elliptic curves, such as Weil curves [12]. BLS needs a single round of communication for *Signing*. BLS has a higher computational cost for signature generation and verification, which can be up to 10x higher than for ECDSA [9]. We selected the threshold BLS library provided by ZenGo-X [89], which implements the Galindo et al. scheme [37].

2.3 EnGINE

The ***En****vironment for* ***G****eneric* ***I****n-vehicular* ***N****etworking* ***E****xperiments* (EnGINE) framework was initially developed for conducting Time Sensitive Networking (TSN) experiments. It is an open-source and adaptable experiment coordination tool implemented in Ansible, easily integrated with Commercial off-the-Shelf (COTS) hardware. EnGINE offers a comprehensive experimentation approach designed for consistent, replicable, and verifiable TSN networking experiments [13]. It primarily leverages the Linux networking stack in experiments and utilizes open-source software, such as Iperf3 [34], for essential functions like traffic generation.

Experiments conducted in EnGINE follow a structured process orchestrated through a management node, encompassing four key phases. The initial stages, namely the **install** and **setup** phases, involve the preparation of participating nodes. In the **install** phase, the chosen Operating System (OS) is deployed on the machines, while the **setup** phase focuses on installing all necessary dependencies. Following the setup, the experiments are conducted in the **scenario** phase, covering all experiments within a given campaign. This involves the configuration of networks, as well as the preparation and execution of applications for traffic generation and data collection. Each experiment is initiated and concluded, with this cycle repeating until all defined experiments for the scenario are finished. The results are then processed in the final **process** phase.

3 Analysis

RSS combines TSSs with TEEs. This section demonstrates the advantage of the additional security provided by RSS. First, we categorize deployment strategies based on the available literature, which motivates RSS usage. Also, we introduce an evaluation methodology for RSS. Lastly, to assess the RSS, we outline the selected evaluation approach and suitable framework behavior.

3.1 Threshold Schemes as a Middleware

Various applications rely on threshold cryptography. We systematize application families and argue how TSS are used as middleware and how the RSS architecture combined with TEEs can be deployed.

3.1.1 Byzantine fault-tolerant consensus protocols: Various consensus protocols rely on threshold cryptography to enhance their performance or security [33, 50, 65, 87]. The HoneyBadger protocol [65], BEAT [33], and DUMBO [50]

utilize Threshold Public-Key Encryption (TPKE) and threshold signing, with the latter being the focus of this work. TSSs are often considered during the binary agreement of BFT protocols. The replicas respond to a proposal with the partial signatures of their votes. This approach reduces the overall authenticator complexity [16, 49, 87] or, for leader-based approaches, the message size overhead [23, 87]. The DFINITY project utilizes TSSs for various applications. First, a Verifiable Random Function (VRF) that relies on the BLS TSS to collaboratively derive a random number used for a leader election [52]. Next, it allows for the issuance of signatures and data encryption [21], relying on BLS. For private computation in smart contracts, the You Only Speak Once (YOSO) model [8, 20, 22, 43, 44] was recently introduced. YOSO relies on running threshold cryptography among the consensus committee. An open research challenge is making YOSO communication and computationally efficient for large-scale deployment with new blocks created every few seconds in modern blockchains [22, 44]. Such private computations apply to private memory pools, aiming to mitigate negative externalities of Maximal Extractable Value (MEV) and prevent transaction censorship [6]. Discussed solutions rely on TEEs [80], but not in combination with threshold cryptography. Overall, TSS are used in BFT and blockchain, coupled with the consensus or exposed to applications building on top of them.

3.1.2 Blockchain Infrastructure: This includes bridging, oracles, and block-building solutions. Bridging provides interoperability and asset exchange among various blockchain protocols [67, 71]. Bridge designs often rely on a centralized set of peers that use TSS for signing transactions. Oracles allow off-chain data to be used as input to a function in smart contracts [18]. The data is processed automatically, so it is important to ensure its authenticity and validity [17]. Data authenticity can be achieved by TSS. Each data feed creates a partial signature, but eventually, the smart contract can verify a single signature corresponding to a set of peers. Recent optimization covers scenarios with a larger set of data feeds and their dynamic size of signer quorums [3]. We see standalone deployments that interact with other systems.

3.1.3 Hardware Trojans and Secure Storage: Threshold cryptography offers mitigation strategies for both secure and hardware trojans, as discussed in [62]. They mitigate this issue by combining vendor hardware to form a cluster. There is a centralized controller for interaction with the devices that cannot forge signatures but stage a Denial-of-Service (DoS) attack, if compromised. The individual commands are duplicated across multiple quorum sets to ensure hardware redundancy. Similar approaches are considered for cryptographic wallet protection with various security considerations in [11, 59, 61]. These solutions focus on methods purely in software, making them more versatile.

3.1.4 Internet Infrastructure Protocols: An extension to DNS Security Extensions (DNSSEC) using TSS is proposed in [28], which provides additional security to users requiring to collect multiple queries from DNS servers. The authors in [83] extend the Resource Public Key Infrastructure (RPKI) used in BGP Security (BGPSec) protocol by a TSS. RPKI leverages Regional Internet Registry (RIR) to allocate IP prefixes and authorize specific Autonomous Systems (ASs) as route origins. The hierarchical structure of RPKI grants RIR the authority to revoke and invalidate any objects they have issued. Overall, DNSSEC and RPKI deployments integrate TSSs as middleware.

3.1.5 RSS Architecture. As a result of our analysis, we present Figure 1, which shows how TSS can be deployed as middleware supporting other solutions. As a part of RSS, we focus on the integration with TEEs ① represented on the Infrastructure Layer (①a). In simplified Communication Layer ②, we consider any stack building on top of Internet Protocol (IP) using TCP, UDP, QUIC, or other protocol. The TSSs serve as the Middleware Layer ③, where we see solutions of BFT-State Machine Replication (SMR) or broadly consensus (③a) that can utilize TSS for various purposes (③b). Similarly, TSSs might utilize BFT-SMR or other primitives for robust peer communication. Nevertheless, as shown in the literature use cases, TSS (③b) can be used individually. The Application Layer ④ covers previously introduced

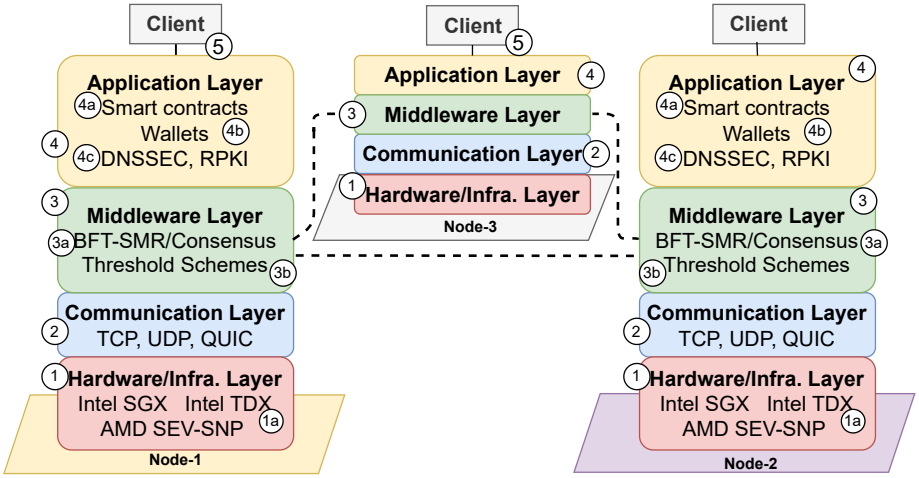


Fig. 1. RSS Middleware and Infrastructure Layers

applications based on individual protocols. This includes smart contracts (④ⓐ) and wallets (④ⓑ) in the case of blockchain deployments or other network infrastructure protocols, e.g., DNSSEC or RPKI (④ⓒ). The Client ⑤ can interact with the underlying stack that provides the application's relevant features irrespective of the request entry, where the system handles the communication among individual peers. A modified architecture, where the client can interact with a standalone signing service, is also possible, but based on the literature, we see more scenarios of threshold services as middleware. RSS combines the Infrastructure (①) and Middleware (③) Layers.

3.2 Threat Model and Security Scope

System model. RSS consists of n logical threshold participants. Each participant executes a protocol implementation and holds one key share inside a TEE. A logical participant is not necessarily an independent failure domain: multiple participants may share one physical host, CVM, guest kernel, or software image. A compromise of such a shared domain counts as a compromise of every participant colocated in that domain.

Adversary. We consider an infrastructure adversary that may control the host operating system or hypervisor, persistent storage, and network scheduling. It may observe, delay, drop, replay, or modify network traffic; protocol-specific authenticated channels are expected to detect unauthorized modification. The adversary may also corrupt participant instances. RSS preserves the corruption threshold of the underlying scheme rather than strengthening it: a compromise of fewer than t independent share-holding instances should not enable signature forgery, subject to the scheme's stated adversary model. DoS remains possible.

Trusted computing base. We trust the processor's TEE implementation, firmware and microcode, the guest boot chain and kernel, the RSS application, the cryptographic libraries, and the attestation-verification logic. Compromise of the guest or RSS process is treated as compromise of that threshold participant. Remote attestation provides evidence about measured software state; it does not by itself provision or bind a key share, prove application-policy correctness, or prevent rollback.

Table 1. Defense-in-depth interpretation of RSS. Detailed assumptions and non-goals are defined in Section 3.2.

Property	RSS interpretation
Share confidentiality	A malicious host or hypervisor need not learn a participant's share while the TEE and guest TCB remain trustworthy. Security against forgery still requires that fewer than t independent shares are compromised under the adversary model of the underlying threshold scheme. Colocation and common-mode vulnerabilities reduce this independence.
Attestation and state	Remote attestation can identify an approved measured guest and TEEs can protect protocol state while it is in use. Share provisioning must be cryptographically bound to the attestation evidence, while persistent state additionally requires sealing, freshness, and rollback protection.
Authorized quorum	Any authorized set of at least t participants can produce a valid signature. RSS therefore does not by itself prevent an authorized but undesirable signing request; request policy, auditing, and accountability are separate service-layer requirements.
Availability	At least t suitable participants must remain responsive, and tolerance of missing or malformed responses is protocol-specific. TEEs provide no intrinsic availability improvement, and we claim no denial-of-service protection.

Security goals. RSS aims to protect the confidentiality and integrity of an individual key share and its in-use auxiliary state from a malicious host or hypervisor. If one or more TEE instances are compromised, the threshold scheme continues to protect the signing key only while the total number of compromised shares remains below t. Availability requires at least t suitable and responsive participants and also depends on the robustness and signer-selection behavior of the chosen protocol.

Non-goals. The current evaluation does not establish resistance to physical attacks, microarchitectural side channels, a malicious TEE vendor, common-mode firmware or software vulnerabilities, guest-root compromise, persistent-state rollback, or DoS. It also does not prevent t authorized participants from jointly signing an undesired request; request authorization, policy enforcement, and accountability are separate service-layer requirements. The experiments measure benign-operation performance and do not empirically evaluate Byzantine fault handling.

3.3 Security Assessment of RSS

RSS combines threshold signing with TEE isolation as a defense-in-depth construction rather than as a set of unconditional, additive guarantees. Table 1 summarizes the resulting security interpretation. Threshold signing distributes exposure across participants, while a TEE can isolate one participant's share from its infrastructure operator. If one or more participant instances are compromised, the underlying threshold scheme continues to protect the signing key only while fewer than t shares are compromised in total.

Many TSSs are introduced for individual families [7, 37, 39, 56, 60, 61, 70]. For our work, we select a subset of schemes for which we observe adoption and provide publicly available libraries that were considered state of the art (SotA). For nondeterministic schemes, we prefer constructions that separate message-independent preprocessing from the online signing phase. Threshold BLS is deterministic and therefore has no nonce-preprocessing phase. The schemes' assessment combines theoretical and empirical evaluations. These steps apply to future TSSs. For the theoretical step, we focus on the runtime complexity and properties of the schemes, including security ($\mathbf{P_{Sec}}$), robustness ($\mathbf{P_{Rob}}$), preprocessing ($\mathbf{P_{Prcom}}$), and total number of communication rounds ($\mathbf{P_{\#Rounds}}$) during signing. $\mathbf{P_{Sec}}$ describes if the scheme is secure against a static adversary, which decides which nodes to corrupt upfront and support the dishonest majority, where more than half of the peers can be malicious [81]. $\mathbf{P_{Rob}}$ defines the case where a single party cannot

Table 2. Overview of Threshold Schemes

Family	Scheme	$\mathbf{P_{Sec}}$	$\mathbf{P_{Rob}}$	$\mathbf{P_{Prcom}}$	$\mathbf{P_{\#Rounds}}$ T | O*
ECDSA	GG18 [38]	SA†	✓	×	9 | -
	GG20 [39]	SA†, DM‡	✓	✓	8 | 1
Schnorr	FROST [56]	SA†	×	✓	2 | 1
BLS	Galindo [37]	SA†	✓	-	1 | -

†*Static* Adversary; ‡*Dishonest* Majority
**Total* | *Online* Rounds, ✓ - support, × - no support

Table 3. Runtime Complexity of Threshold Schemes

Scheme	GG18	GG20	FROST	BLS
DKG	$O(t \cdot n^2)$	$O(t \cdot n^2)$	$O(t \cdot n^2)$	$O(t \cdot n^2)$
Preprocess	$O(t)$	$O(t)$	$O(n)^*$	N/A
Signing	$O(t)$	$O(t)$	$O(n^2)$	$O(t^2)$

*FROST is not robust, expects responses from all n peers

stage a denial-of-service attack on the scheme. $\mathbf{P_{Prcom}}$ denotes if the scheme supports a preprocessing phase and $\mathbf{P_{\#Rounds}}$ represents how many communications rounds are required for the total and online number of rounds.

Table 2 summarizes the properties of the schemes. We observe differences in security, robustness, and differentiation between online and offline rounds. The differentiation between types of rounds only applies to non-deterministic schemes, such as GG20, GG18, and FROST, as they require random nonces for each issued signature. FROST and GG20 support the pre-processing of nonces, resulting in only a single communication round that must be online. All are TSSs where $t < n$. Based on the overview in Table 2, Galindo et al. is the most efficient in terms of communication.

The bounds in Table 3 describe the evaluated implementations and their selected communication patterns. Unless stated otherwise, they should not be read as universal protocol lower bounds. In particular, the observed FROST dependence on n includes distribution and hashing of signer and nonce lists and the fixed selected-set behavior of our implementation. The threshold BLS aggregation cost includes computation of Lagrange coefficients in the selected library; implementations may precompute some values when the signer set is fixed. We therefore use the complexity table to explain observed trends and report protocol rounds, per-participant work, aggregator work, message count, and critical-path latency separately where relevant.

Starting with the *DKG*, we observe the same runtime complexity for all implemented schemes, as they use a similar approach based on Pedersen's commitments [41, 55]. All schemes require two round trips of communication with varying levels of computation happening on each peer relevant for auxiliary data computation. Ideally, the communication requires a reliable, secure channel. This can be achieved by, for example, a reliable broadcast echo protocol, which we implemented in the GG20 protocol [14]. For FROST, we implemented a secure channel with the Noise protocol [69]. For GG20 we use the reliable broadcast. The complexity hints at a challenge with scalability, which is the main focus in [30, 84] offering possible improvements. It provides a theoretical analysis of current solutions and proposes optimized solutions for BLS DKG and Verifiable Secret Sharing (VSS) approaches. The *Preprocess* phase depends mainly on t for all schemes except FROST. Since FROST is not robust, all selected peers for the *Signing* are expected to be available during

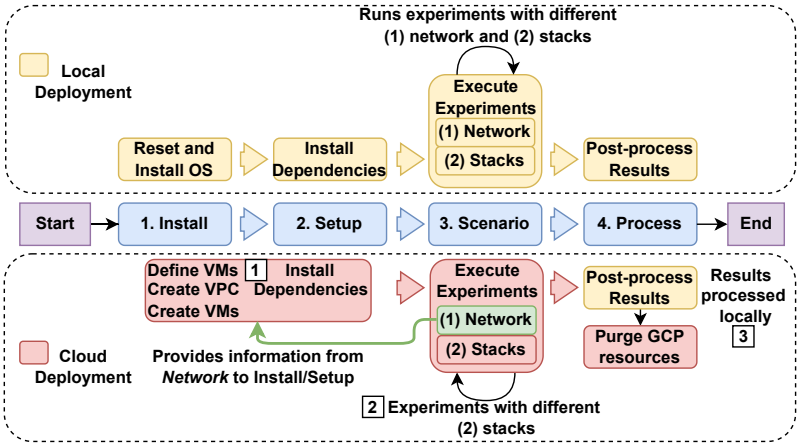


Fig. 2. EnGINE extension local and cloud deployments

the next phases and respond. The behavior of GG18 and GG20 differs in comparison to FROST. Unlike GG18 and GG20, where the communication happens among n parties, only t parties are expected to behave according to the protocol. The required extensions of the libraries allow for the deployment of a full-stack realistic system, enabling the collection of relevant insights for existing solutions.

For the *Signing* phase, we observe comparable performance that depends on the value of t for all schemes except FROST depending on n. FROST is $O(n^2)$ due to the effect of transmitting or hashing the message signatures and list of nonces, which can be cheap operations for even moderate hardware. It relies on n due to the missing robustness. As a part of *Signing* phase, we also consider the aggregation of partial signatures to a final signature. For GG18, GG20, and FROST, we consider $O(t)$ complexity for aggregations as it is only the aggregation (additions) of the verified partial signatures. However, for BLS, we compute the t Lagrange basis polynomials, each of which takes t multiplications to compute, resulting in t^2.

3.4 Deployment Strategies

Based on Section 3.1, we observe deployments in local networks such as cryptographic wallets or applications that rely on broad internet infrastructure that is possibly hosted in the cloud. Therefore, the support of cloud environments allows us to create performance estimates that are realistic and close to the target scenario. The combination of local and cloud provides complementary views and extends the capabilities and set of experiments. For easy porting of the defined **scenarios**, we integrate GCP to deploy the same definition of stacks locally and in the cloud. An EnGINE extension introduces support for VM-based TEEs, i.e., Intel TDX and AMD SEV-SNP. Besides, the cloud provides additional virtualization forms covering full VMs or more lightweight containers. When deploying the confidential VMs in the cloud, the containers running inside of the VM inherit the features of the confidential VM and can run in their TEE enclave.

Figure 2 presents the default **Start** to **End** of the EnGINE workflow and illustrates the differences for local and the newly added cloud deployment. For the local deployment, we can refer to Section 2.3 for a general overview. The cloud extension requires a modification of the first steps of **install** and **setup** with additional details from the **scenario** [1]. In

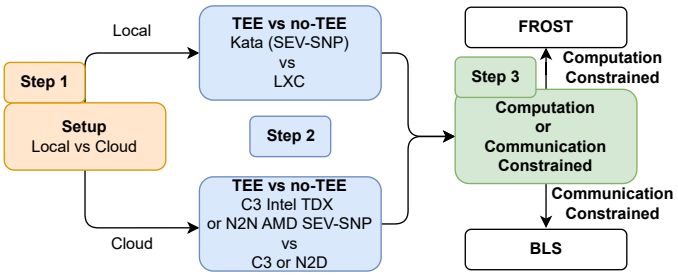


Fig. 3. Selection Process of RSS Building Blocks

local deployments we boot the machines and later on set the network configuration based on the physical connections. However, in GCP, we must define a Virtual Private Cloud (VPC) during the boot time of the VM and its corresponding interfaces [46]. Therefore, we extract these details from the network configuration in a **scenario**. For the definition of the VM we can specify various Hardware (HW) parameters and also specify if the whole VM should run in a TEE. Once the VM is spawned, we install the corresponding dependencies and start with the execution of the experiments 2. Since the network is fixed, we can only configure existing virtual connections, e.g., using queuing discipline (qdisc) or others. This might pose a challenge, but if the scenarios are defined properly, a subset of the overall virtual connections can be used for individual experiments. Once experiments are completed, we store the artifacts and process them locally 3. This serves as a cost optimization. Lastly, the resources of GCP are purged.

4 Design & Implementation

After covering the aspects of the deployment approaches, we cover the experiment design, including the network topologies and experiment campaigns. We provide a decision tree in Figure 3 with the various approaches an operator of an RSS can consider. The individual aspects of the decision path are covered in our experiments.

4.1 Experiment Structure

As shown in Section 3, there are many options for TSSs, setups, and TEEs architectures. Therefore, we provide a simplified decision tree in Figure 3 that should be beneficial for deciding what approach best fits their needs. The consecutive experiments follow these steps and provide empirical evidence for the decisions. Figure 3 considers **Step 1 - Setup**, where we support both local and cloud deployments. Of course, a hybrid model with some nodes running locally and in the cloud is possible, but we focus only on separate modes for our evaluations. Depending on the **Setup**, one can decide between a TEE and no-TEE deployment as a part of **Step 2**. We cover various TEEs and virtualization technologies as a part of the evaluation. Lastly, depending on the HW capabilities, one can choose broadly between communication (e.g., nodes with bad uplink/downlink or IoT scenarios) or computation-constrained (e.g., smartphones or embedded devices) environments within **Step 3**. The security properties covered in Section 3.3 are part of the considerations.

4.2 Step 1 - Experiment Setup

Based on **Step 1**, we differentiate between the local and cloud deployments for our evaluations and aim to use comparable nodes' HW specifications. Table 4 provides an overview of the HW details, corresponding to the host OS.

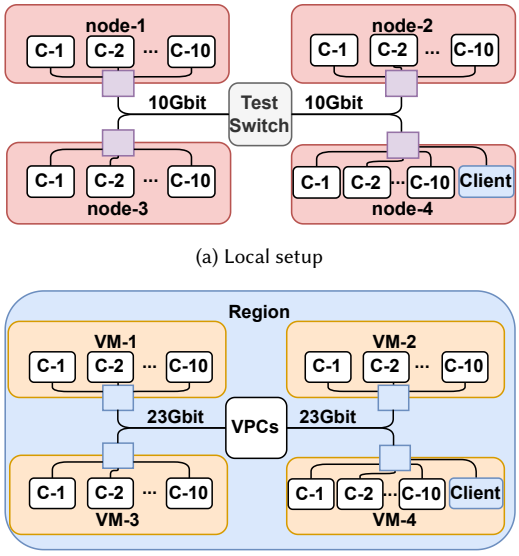
Table 4. HW Specifications for Local and Cloud Deployments

	Local	**Cloud**	
	AMD	AMD	Intel
CPU	EPYC 7543	EPYC 7B13	Xeon 8481C
Core/Thread	32C/64T	32C/64T	22C/44T
TEE	SEV-SNP	SEV-SNP	TDX
Family	**Kata/LXC**	**N2D**	**C3**

These specifications correspond to the host OS for the local node. On the other hand, for the cloud, these specifications do not necessarily match what is provided to the guest VM, as other nodes might be deployed on the same physical host. We rely on Linux Container (LXC), Kata, and QEMU virtualization solutions to allocate fixed resources to guest nodes for scalability and to assess the network behavior. We deploy Ubuntu Focal 20.04 LTS on the host OS for local Kata container deployments, as this is the required version to operate the containers. For the rest of the experiments relying on LXC and cloud deployments, we use Ubuntu Jammy 22.04 LTS as host OS and inside of the containers and VM. We limit the resources to two vCPUs for local deployments and four vCPUs for the cloud. Four vCPUs is the lowest value due to the GCP constraints. For both we use 4 GB RAM. As outlined in Section 3.4, all experiments are deployed using the extended EnGINE framework.

Nodes are interconnected in a mesh topology, where each peer can communicate with other peers via a centralized switch or VPC, as shown in Figure 4. For both setups, we deploy four nodes where each hosts up to 10 containers (LXC or Kata). The `node-4` always deploys a `client` container that interacts with the system and orchestrates the issuing of commands to run DKG or generate signatures. It is also used to measure the E2E latency from when the command is issued until it receives the response. Due to limitations of Intel TDX and GCP, we can only accommodate a single region [45]. Therefore, all of the VMs are deployed in a single region with up to ten runC containers, as shown in Figure 4b. Unlike local deployment, we only scale to ten or eight containers for some experiments due to budget. This is partially due to budget limitations and resource exhaustion for the experiment $n > 32$ for BLS. The VM specification corresponds to the instance family as shown in Table 4, and either the whole VM runs natively or in a TEE (Intel TDX or AMD SEV-SNP). The family handles are important for the evaluation, as we use them as experiment identifiers to compare baseline solution and TEE deployment. Table 4 presents the resources available to the bare metal or VM specifications, but for our experiments, we limit the container resources to either 2 or 4 vCPUs for local or cloud experiments, respectively. All of the experiments are run inside the containers. This is a baseline behavior and overhead introduced by the VM. Due to limitations of GCP during the time of the experiments, we run all of the VMs in the same region for the experiments with Intel TDX. Therefore, we also run baseline evaluation for the single and multiple regions along with AMD SEV-SNP experiments.

Logical participants and failure domains. We distinguish logical protocol participants from independent infrastructure failure domains. Each logical participant runs one threshold-protocol instance and holds one share. In the local configuration, up to 40 participants are placed in containers across four physical hosts. In the cloud configuration, up to 40 participants are placed in runC containers across four VMs. When a cloud VM is confidential, all containers inside that VM share the same TEE instance, guest kernel, and VM-level failure domain. The experiments therefore scale the protocol to 40 logical participants but do not emulate 40 independently attested machines. Production deployments



(a) Local setup

(b) Cloud setup in a single region

Fig. 4. Comparison of local and cloud deployments

should place shares so that compromise of one host, VM, guest kernel, or administrative domain cannot expose a threshold number of shares.

Interpretation of platform comparisons. We use matched pairs to discuss confidential-computing overhead: C3 versus C3-TDX, N2D versus N2D-SEV-SNP, and native Kata versus Kata-SEV-SNP in the microbenchmarks. Comparisons between local and cloud environments, between Intel and AMD platforms, or between LXC and Kata-SEV-SNP also change processor model, vCPU allocation, operating-system version, virtualization stack, network, or timing method. We treat those comparisons as configuration-level observations, not as estimates of a single causal TEE effect. In particular, the local TSS comparison between LXC and Kata-SEV-SNP combines virtualization and TEE effects.

4.3 Experiment Design

To evaluate the RSS, we introduce experiments assisting the decision for **Step 2** and **Step 3** in Figure 3.

4.3.1 Step 2 - TEE Evaluation. First, we analyze memory-bound and computationally-bound CPU performance by measuring the Mega Floating-Point Operations Per Second (MFLOPS) of the nodes. We use the *triad* operation based on the STREAM benchmark suite for our memory-bound benchmark [63]. The form of the computation is $\vec{a} = \vec{b} + \vec{d} \odot \vec{c}$, with increasing vector values from 2^5 to 2^{27}. The bottleneck lies in the speed at which the vectors are loaded into the CPU, emphasizing memory performance. For the computationally-bound CPU evaluation, we use multiple matrix operations as $A_{k+1} = A_k + \boldsymbol{BC}$, with the matrices B and C being initialized once. We assess the matrix sizes from 10 to 2300. Each of the experiments is executed in a dedicated container. Triad has 4096, and Matrix has 96 repetitions. We showcase an average over the whole run for each data point.

Understanding the network performance and the impact of TEEs is crucial due to the nature of distributed deployments in blockchain. We focus on network throughput and network latency with varying payload sizes. For latency evaluation,

we rely on *tcpdump* and *Iperf3*. The *Iperf3* client runs inside of the corresponding container that is limited in resources, and the *Iperf3* server runs on a dedicated bare metal server or a VM. This roughly resembles the Figure 4 with certain switching logic in between. We run a dedicated *tcpdump* on both client and server. To ensure the clocks between the source and sink are in sync, we use Network Time Protocol (NTP) for GCP and Precision Time Protocol (PTP) for local evaluations instead. For both scenarios, we rely on Chrony [24] for container deployments, where locally, we can provide a virtual PTP clock to the container. This ensures a higher precision for our evaluation and is an advantage of the local deployments. For GCP, we run the evaluation in the same region and subnet. Each benchmark application is single-threaded. For the latency and throughput experiments, we run *iperf3* for 10 s and take an average over all the points.

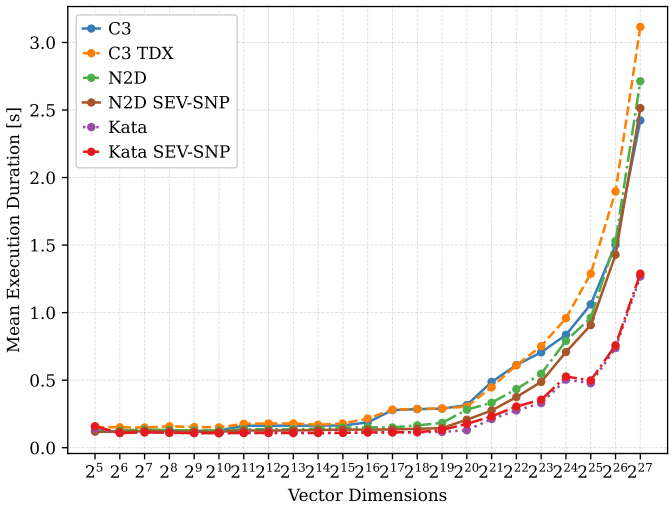
4.3.2 Step 3 - TSSs. We mainly study the E2E latencies for all operations with varying values of n and t. The latency is measured between the client issuing the command and its completion. This includes *Signing* step as shown in Table 3. Such outcomes might hint at the theoretical assessments shown in Table 3 but might not be visibly prominent for fewer peers. As outlined in Section 2, we selected the ZenGo-X ECDSA library [90], frost-dalek for FROST [36], and ZenGo-X for BLS [89]. The used libraries are written in Rust and provide only basic functionality covering an implementation of the cryptographic primitives. Therefore, we must extend each library by a dedicated communication stack to emulate full-stack realistic blockchain ecosystem peer-to-peer deployments. We implement the Noise protocol [69] for FROST and BLS. For DKG we require reliable broadcast [14]. The experiments are split into *DKG*, *Preprocess*, and *Signing*. An example microbenchmark is shown for FROST covering the individual steps in Figure 8. We choose FROST as it contributes to the root cause analysis of the successive results. For *DKG*, we include GG18 and GG20 and FROST to investigate the impact of auxiliary data in a local setup. For *Preprocess*, we cover only FROST as it requires fewer communication rounds for the offline phase and becomes more popular in the industry with a shift towards it from ECDSA. For *Signing*, we cover FROST and BLS as popular schemes showcasing the cost associated with the online phase.

We scale the experiments to at most 40 logical participants distributed across four physical hosts or cloud VMs. This scale is intended to expose protocol and communication trends rather than to represent 40 independent TEE failure domains. For each studied end-to-end parameter configuration, we execute at least 500 iterations.

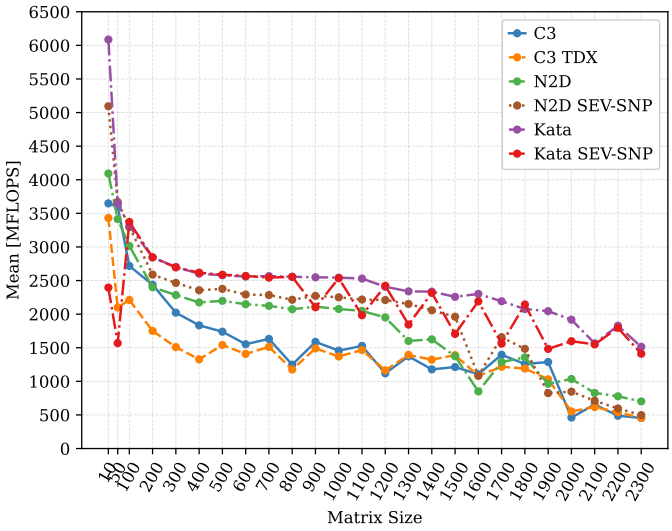
We evaluate benign, failure-free protocol execution. We do not inject unexpected participant crashes, malformed shares, adversarial message scheduling, attestation failures, or persistent-state rollback. Accordingly, the experiments measure feasibility and latency, not empirical Byzantine robustness. A t-out-of-n service can proceed only while at least t suitable participants remain responsive, and a particular signing attempt may tolerate fewer failures when its selected signer set is not robust to aborts.

5 Evaluation

The threat model motivates RSS as a defense-in-depth construction; this section evaluates the benign-operation performance and usability of its individual and combined components. We begin with microbenchmarks to establish infrastructure-level baselines for interpreting the end-to-end latency results. The evaluation follows Figure 3 and examines the Infrastructure and Middleware Layers introduced in Figure 1. We then evaluate threshold-signature execution inside the selected confidential-computing configurations.



(a) STREAM triad execution time (memory-bandwidth bound)



(b) Matrix benchmark throughput (compute bound)

Fig. 5. CPU microbenchmarks for local and cloud configurations. (a) STREAM triad execution time, which is primarily memory-bandwidth bound. (b) Matrix benchmark throughput in MFLOPS, which is primarily compute bound. Cross-platform comparisons include hardware and software differences; TEE overhead is assessed using matched platform pairs.

5.1 Individual RSS Layers Evaluation

To assess individual layers covered in Figure 1, we provide an overview impact of TEEs and TSSs. For that, we analyze the overhead of individual layers.

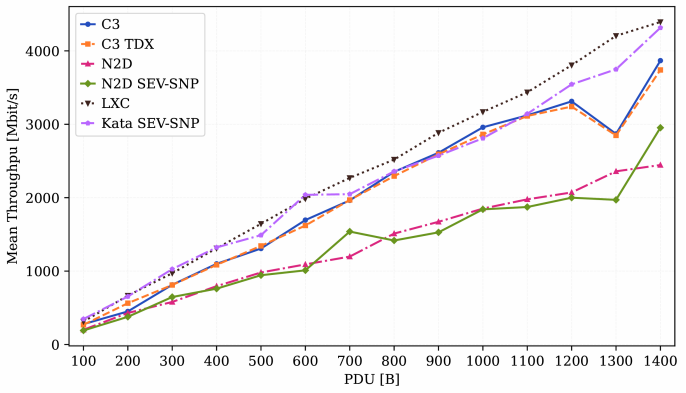
5.1.1 TEE Baseline. Based on Section 4.3, we motivated the focus on TEE baseline evaluation. Starting with CPU performance, Figure 5 shows the triad and matrix benchmark results. We conducted the experiments on C3 and N2D instances of GCP and locally on Kata and its baseline focusing on understanding the TEE overhead. We do not have Intel TDX locally, so we present only the AMD SEV-SNP results. The triad results in Figure 5a indicate worse performance for C3 machines, starting with vector dimensions larger than 2^{10}. Cloud deployments, in general, begin to perform worse than local deployments with vectors larger than 2^{20}. If we want to compare the local and cloud settings, we see roughly double the execution time for cloud deployments compared to local Kata containers for the vector sizes of 2^{27} and more. Overall, we see similar trends for versions with and without TEE, where TDX starts to have a worse performance for vectors size above 2^{24}.

The matrix benchmark results, introduced in Figure 5b, present similar findings to the triad results. We observe a zig-zag pattern for various matrix sizes between the Intel and AMD-based platforms, with Intel performing more stably. This pattern is more prominent for TEEs, particularly in the local deployment. Again, we observe roughly $1.5 - 2\times$ better performance for local deployments and larger matrix sizes. Unlike the triad benchmark, the performance of both cloud deployments converges to a similar point for matrix sizes above 1500. This hints at the better single-core performance of N2D machines but worse memory-bound scenarios. Overall, the TEE configurations perform slightly worse than their respective non-TEE baselines in some benchmarks but generally follow the same trends.

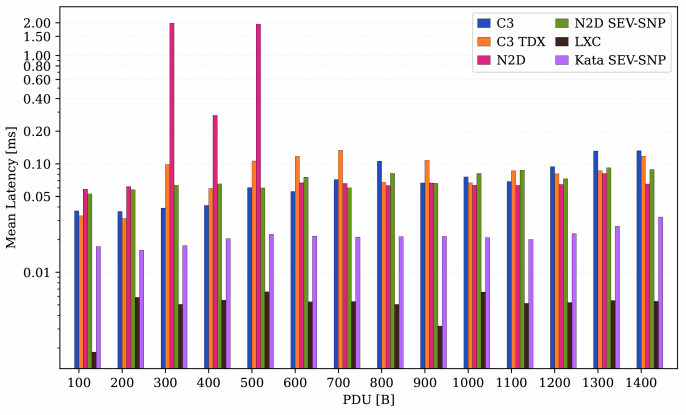
The second set of experiments focuses on network performance, as shown in Figure 6. In Figure 6a, we display throughput with varying Protocol Data Unit (PDU) sizes between 100 and 1400 B of UDP traffic. All results are stable, with more fluctuations for AMD SEV-SNP both locally and in the cloud. We assume this might be caused by the bridge solution connecting the physical interface and the Kata container running in the TEE. Besides, the software tooling for Kata is less mature based on our experience. The cloud TEEs show comparable performance impact, hinting at the maturity of the deployment stacks for the VM solutions. Completing the throughput analysis, we continue with latency in Figure 6b. We observe a general tendency of increased latency with larger payloads. The TDX-enabled C3 machine introduces higher latency for smaller packets than the baseline. However, for N2D machines, we observe the opposite. As expected, the local deployment latency is low due to the point-to-point connection and precise timestamping in HW. Unlike the bridge for Kata containers, LXC uses Single Root I/O Virtualization (SR-IOV), which introduces negligible delays [27].

The family of IO experiments focuses on the disk measurements, touching on latency and bandwidth for read and write operations, Figure 7. We rely on the FIO benchmarking tool [35]. Figure 7a presents the I/O throughput cross-available setups by configuring the FIO to randomly read a single 10 GiB file for 300 s with a block size of 128 KiB and queue depth of 32. First, the cloud deployments reach symmetric read and write throughput. This hints at the capped throughput by GCP for our setup and the benchmark. The standard C3 and N2D machines reach a maximum I/O throughput of 200 MiB s^{-1}. The N2D machine could achieve up to 800 MiB s^{-1}, but due to the benchmark's restriction to only one vCPU, we observe comparable performance to the C3 node. The local setup, in general, shows higher write speeds. However, we see a much worse performance for reads. For the case of TEEs, this is caused by ownership checks through nested page table walks and decryption of respective blocks. For native Kata containers, we observe much higher throughput and comparable read and write speeds.

The second metric is the latency of our system, Figure 7b; we configure the FIO to randomly read a single 10 GiB file for 300 s with a queue depth of 4. Across various configurations, the cloud deployments demonstrate consistent and comparable performance. For the local deployment, we see a worse performance for the TEE with Kata for writes but comparable for reads.



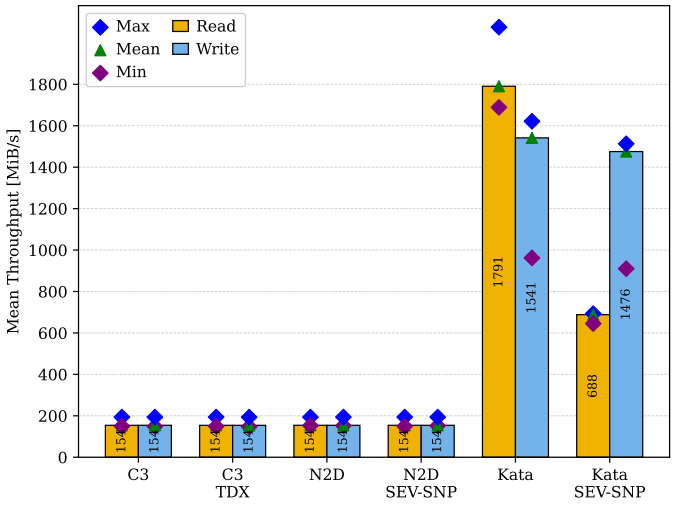
(a) Network throughput with varying PDU sizes



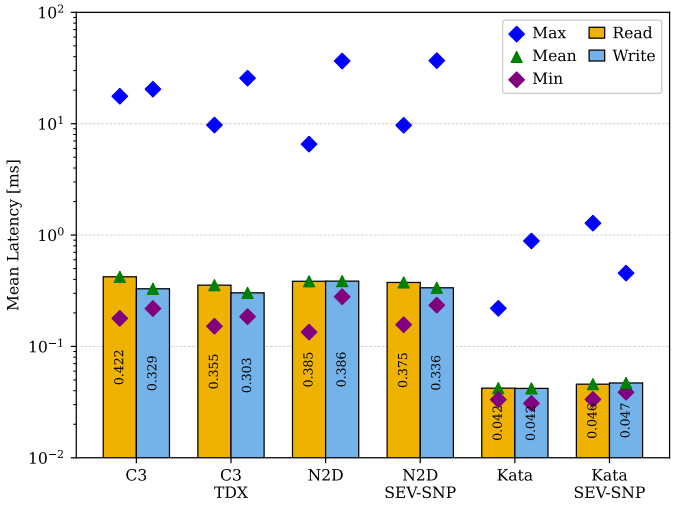
(b) Network mean latency with varying PDU

Fig. 6. Comparison of network throughput and latency

5.1.2 Microbenchmarking. Contributing to the Middleware layer, we micro-benchmark the FROST scheme in local deployment and cloud as shown in Figure 8. The benchmark ran without any communication stack, including only the plain function operation. We present the same experiment execution locally (Section 5.1.1) and in the cloud (Figure 8b). Each figure is divided by the individual operations of the FROST scheme. The operations from *Participant (Part.) creation* to *Finish* contribute to DKG. Partial (Part.) signature creation and signature (Sign.) aggregation (Agg.) are part of the online *Signing* process. Lastly, signature verification (Ver.) is the final step of verifying the aggregated signature. The most expensive operation is the long-term (LT) verification of shares, which scales with a complexity of $O(n^2 \cdot t)$ and serves to verify individual partial signatures before aggregation. Their generation takes around 2 ms. This is especially prominent for higher values of n, where for the largest $n = 40$, we see 2.9 s of computation is the most significant factor. We ran this experiment in addition to the Figure 8. For signing, we observe a negligible cost for creating a partial signature. On the other hand, partial signature aggregation is costlier. The aggregation must be performed on at least one of the peers to obtain a complete signature.



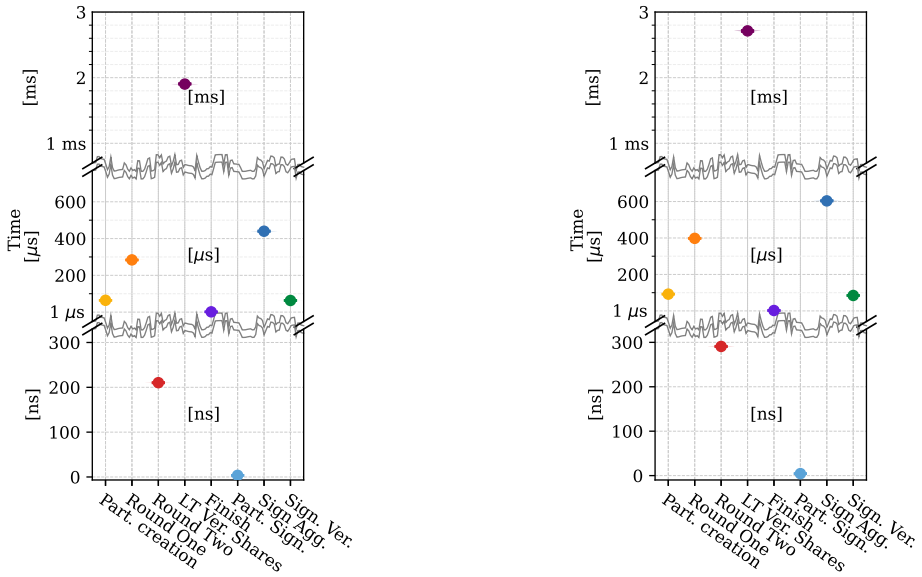
(a) Read and write throughput



(b) Read and write latency

Fig. 7. Comparison of disk I/O throughput and latency

Across the matched pairs, confidential execution generally follows the trend of its corresponding baseline and introduces limited overhead or additional variance for the workloads studied. The magnitude and direction depend on the benchmark and platform. The larger differences between local and cloud results cannot be attributed to the TEE alone because the configurations also differ in processor, resource allocation, virtualization, network, operating system, and timing method. We therefore report local-versus-cloud results as end-to-end configuration comparisons and reserve the term “ TEE overhead” for matched pairs.



(a) Local - Kata + SEV-SNP

(b) Cloud - C3 TDX

Fig. 8. FROST microbenchmarks for $n = 5$ and $t = 3$ in local Kata container and runC container in GCP

5.2 TSS Baseline and RSS Stack

Having established the infrastructure-level baselines and platform-specific effects, we next examine the end-to-end latency of DKG, preprocessing, and signing for the selected schemes. These steps are relevant for **Step 3** in Figure 3. We run our experiments in GCP for signing operation of FROST and BLS and compare the results for local and cloud deployments. We consider signing to be more relevant than DKG due to the frequency of its usage.

5.2.1 Key Generation. As shown in Table 3, all schemes provide comparable complexities and rely on similar primitives, with the main difference being the auxiliary data. Therefore, we consider DKG of Schnorr and GG20 ECDSA, shown in Figure 9. Figure 9 shows the results for selected values of n between $n = 5$ and $n = 40$ and $t \in 2 * 2^i, i \in [0, 4]$, up to $t = n - 1$. We compare LXC and Kata in the local deployment. These values of n provide sufficient differences to observe relevant patterns. For FROST, the largest evaluated configuration reaches an end-to-end DKG latency of approximately 4.2 s. Increasing n from 20 to 40 produces an approximately fourfold increase for comparable thresholds, consistent with the quadratic component identified for this implementation. The comparison with GG20 is a full-stack comparison rather than a cryptographic-primitive-only comparison: GG20 uses reliable echo broadcast and computes additional zero-knowledge range proofs, whereas the FROST implementation uses authenticated point-to-point channels and different auxiliary data. The observed gap therefore reflects both protocol construction and communication-stack choices.

We see that GG20 is slower for smaller values of t than FROST's DKG. FROST's delay increase is caused by the expensive computation of the long-term verification keys. Even though they both use similar cryptographic primitives for the DKG, the auxiliary data and their cost significantly differ. Such factors are only partially visible in the Table 3 and

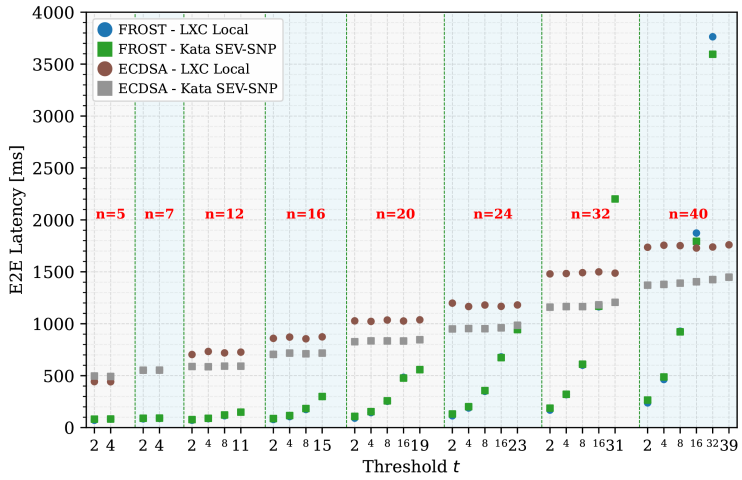


Fig. 9. *DKG*: FROST and GG20 comparison of DKG times in local LXC and Kata deployments with a subset of n values

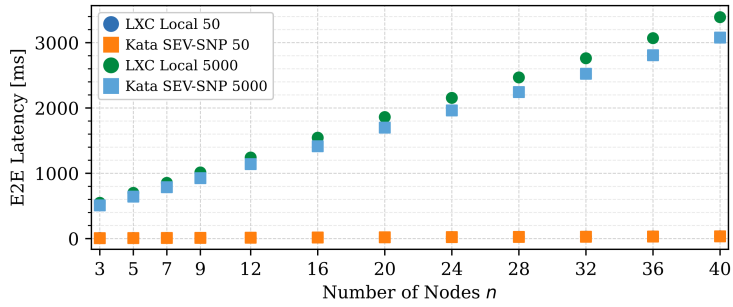
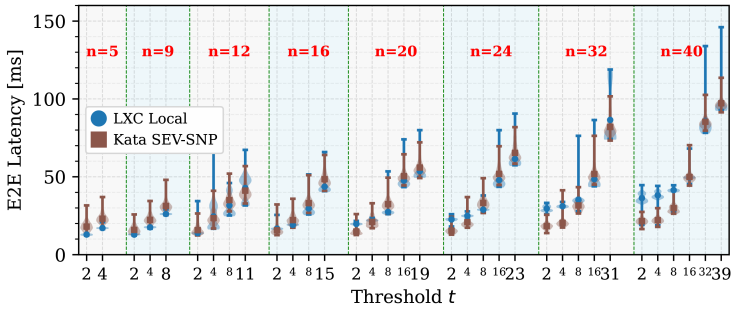


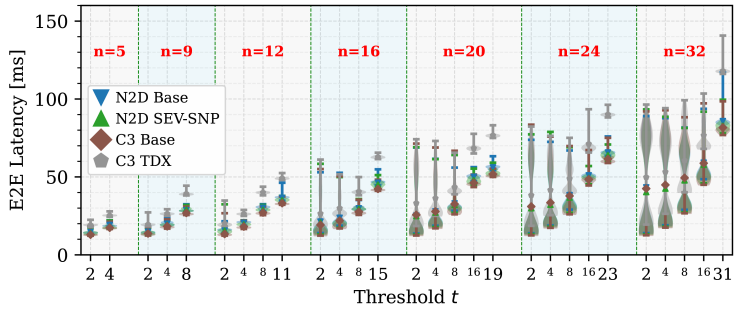
Fig. 10. *Preprocess*: FROST latency in local LXC and Kata deployments for increasing n and fixed $t = 2$

would require a per-primitive view and corresponding computation. Similarly, ECDSA has a dependency on t as well, which holds only for a few computational steps, namely the ones that operate on Feldman's VSS, which is relatively inexpensive. Notably, for GG20, we use the reliable echo broadcast, unlike plain secure channel for FROST. The primary auxiliary data are the Zero Knowledge Proof (ZKP) range proofs, which add a fixed overhead. As with FROST, Kata does not introduce a measurable additional overhead in this experiment. Unlike FROST, the varying values of t for the respective n do not significantly impact the E2E latency. This hints that the t used plays a minor role.

5.2.2 Preprocess. For non-deterministic schemes, nonce generation is crucial. We focus on FROST with its two communication rounds as shown in Figure 10. For each experiment, we generate 50 and 5000 nonces at once in a batch. Figure 10 shows the comparison of Kata with TEE and LXC for varying numbers of nodes and batch size of nonces, but a fixed threshold $t = 2$. For $n = 40$ and generation of 50 nonces, the total time is around 35 ms, roughly corresponding to 0.7 ms per nonce. This is comparable to the 100x increase to 5000 nonces, taking around 3500 ms. We observe a linear increase with a corresponding value of n coming from the $O(n)$ for the *Preprocessing* phase. TEEs in this context can also help preserve the nonces' security, as they should not be compromised before they are used.



(a) BLS local LXC and Kata SEV-SNP



(b) BLS Cloud, AMD Baseline, AMD SEV-SNP, Intel TDX

Fig. 11. *Signing*: Subset latency of n; BLS in local (max $n = 40$) and cloud (max $n = 32$), payload size 32 B

5.2.3 Signature Generation. We compare BLS and FROST *Signing* deployed locally and in the cloud. We fix the message to be signed to 32 B. In Figure 11, we start with threshold BLS and present the local and cloud deployment with varying t and n. Figure 11a presents BLS, where we observe a significant increase with larger threshold values. This is especially interesting as even for higher values of n, t is still a determining delay factor. For the local deployment, Kata performs better, especially for higher values n. This suggests the `netem` impacts the overall E2E latency too much. Since the LXC latency is low based on Figure 6b, we introduce `netem` [53] latency on the containers on node-1. The latency fluctuates as 0.03 ms±0.01 ms. This helps in scenarios where $n \leq 10$ as the containers are running on the same node. We see a linear increase for larger values of t for fixed values of n and a linear increase of n. Even though aggregation is $O(t^2)$, it is less prominent, hinting it is an inexpensive operation.

Running similar experiments in the cloud, as shown in Figure 11b, we observe a comparable tendency for increased values of t. Unlike a local deployment, we are limited to $n = 32$ in a single region. The baseline evaluation in the cloud covers both Intel and AMD CPUs and their respective TEEs. Since we observe comparable performance between the base AMD and Intel platforms, we choose AMD for the next baseline evaluation. The N2D instances impose fewer limitations and costs compared to C3. We also provide evaluations with their respective TEEs.

In the cloud BLS experiment, C3-TDX exhibits the highest latency and variance among the evaluated configurations. This observation is specific to the C3/C3-TDX configuration and should not be interpreted as a general Intel-versus-AMD or TDX-versus-SEV-SNP conclusion, because the instance families also differ in processor and system configuration.

Within matched pairs, confidential execution adds less latency than the protocol-level increase caused by larger values of t, although the cloud measurements show additional variance.

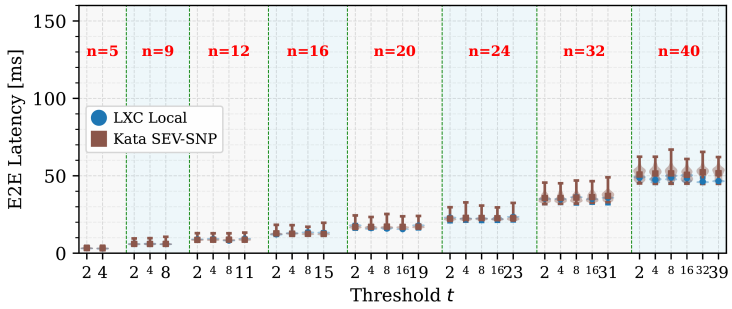
In our FROST implementation, signing latency depends primarily on the selected group size n, while changes in t have a small effect because partial-signature aggregation is inexpensive at the evaluated scales. The observed dependence on n includes full signer/nonce-list handling and fixed selected-set behavior and should not be interpreted as a universal complexity bound for every FROST implementation. The results show that even though the complexities in Table 3 provide insights, it is also important to consider how expensive the corresponding operation is when implemented. We see a closer to quadratic tendency following the n and fixed t. For instance, for $(n = 20, t = 2)$ and $(n = 40, t = 2)$, we see roughly 3.2× increase from 17.5 ms to 55 ms. Nevertheless, the increase is slower than anticipated due to the operations being rather cheap. This is most likely due to the underlying operations, i.e., hashing and network transmission, being comparatively cheap in our setup. The cost for the summation is also supported by Figure 8, where the cost of generating a partial signature is negligible. Comparing the local and cloud deployments, we see comparable performance for the cloud. As identified, FROST is more communication-intensive than computation-intensive, which fits better in the cloud environment. We could reach $n = 40$ for the cloud deployment, hinting at FROST being less computationally expensive than BLS.

Overall, we can see that BLS is roughly twice as expensive for the same n and t. This validates the fact that pairing-based cryptography requires more resources. We observe from the data and fluctuations that BLS is more computationally expensive and FROST is more communication-heavy. The cloud performs worse for computationally heavy tasks, but network performance does not pose a significant bottleneck. The observed trends are consistent with the implementation-level analysis in Table 3: BLS latency is driven primarily by t, whereas FROST latency is driven primarily by n in the evaluated configuration.

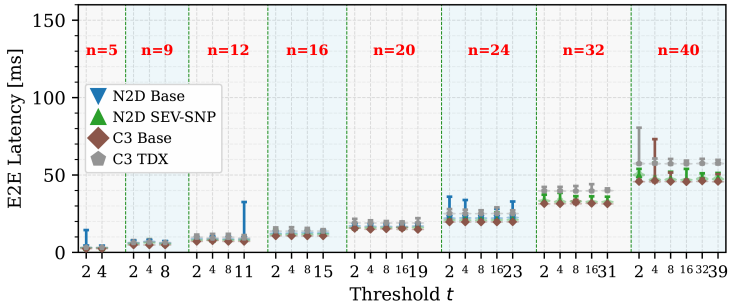
5.3 Applicability of our Findings

As an outcome of our evaluation, we see that DKG poses a challenge to large-scale deployments. Recent work [29] promises improvement in scalability. It would be interesting to see their performance combined with the generation of auxiliary keying material needed for the respective TSS e.g., FROST, as shown in Figure 9. Nevertheless, the improved scalability does not hinder our findings as these schemes can be integrated into RSS and evaluated using our methodology and steps. The new schemes might perform better, increasing the system's usability but not affecting our feasibility study of TSS and TEE integration. For the *Signing*, we observe that BLS generally performs worse than FROST in the online phase, linearly increasing the delays based on the t. Even when combining the offline and online phases, we observe better performance for FROST. However, this might differ if the network conditions worsen, as multiple communication rounds are needed and the factor of message and message size can become more prominent. For that, latency among the threshold t nodes plays a significant role. FROST is closer to quadratic increase with varying n. Instead of being fully quadratic, the operations involved are rather inexpensive on modern systems, resulting in less increase, at least in the current setting. Unlike BLS, which is mainly driven by t, FROST increases the base E2E latency instead based on n, which offers fewer design considerations for security and performance trade-off. Even though FROST is not robust, it should not affect the performance, and the baseline findings should apply to newer schemes, e.g., ROAST [77], that extend FROST.

We do not measure saturated throughput, concurrent clients, batching, or pipelining. Taking the reciprocal of the measured end-to-end latency gives only a single-client sequential signing-rate estimate. For the local configurations,



(a) FROST local LXC and Kata SEV-SNP



(b) FROST Cloud, AMD Baseline, AMD SEV-SNP, Intel TDX

Fig. 12. *Signing*: Subset latency of n; FROST in local and cloud, payload size 32 B

these estimates range from approximately 10–50 signatures/s for threshold BLS and 20–100 signatures/s for FROST across the reported parameter settings. They should not be interpreted as measured system capacity under load.

6 Related Work

We identify as related work publications evaluating TEEs, an overview of threshold schemes, and combination TEE and threshold cryptography. First, the previous works evaluate Intel SGX, AMD SEV-SNP, and other TEEs, but none of them provides a detailed performance evaluation of Intel TDX as we do [58, 64, 68]. For the threshold schemes, we see recent survey papers [2, 81]. The authors of [2] focus on threshold ECDSA and its building blocks. In [81], authors cover other threshold scheme families on a higher level and discuss post-quantum threshold cryptography schemes. General purpose Multi-Party Computation (MPC) is discussed along with TEEs in the hybrid setting [86] for the anonymous database reads. Their approach primarily focuses on lowering the trust assumptions of MPC rather than a defense-in-depth. They only consider Intel SGX TEE. Thetacrypt provides a distributed service and controlled testbed for integrating and comparing multiple threshold-cryptographic schemes, emphasizing that microbenchmarks alone omit distributed-system effects [5]. Haller et al. present SpliTEE and analyze both performance and security trade-offs of TEE-assisted threshold cryptography [51]. This work is the closest overlap with RSS. Our scope is complementary: RSS compares three threshold-signature families in a full-stack experiment framework and studies VM-based Intel TDX and AMD SEV-SNP across local and public-cloud configurations. We therefore frame our novelty in terms of this

cross-platform and cross-scheme evaluation rather than claiming to be the first work to combine TEEs and threshold cryptography.

7 Limitations

Heterogeneous configurations. The local and cloud environments differ in processor model, vCPU allocation, operating-system version, virtualization stack, network, and clock synchronization. Consequently, local-versus-cloud and Intel-versus-AMD comparisons describe complete configurations rather than isolated causal factors. We isolate confidential-computing overhead only through matched pairs where the underlying instance family and software configuration are otherwise held fixed. The local end-to-end TSS comparison between LXC and Kata SEV-SNP combines virtualization and TEE effects.

Logical scale versus failure-domain scale. The experiments use up to 40 logical threshold participants across four physical hosts or four cloud VMs. Participants colocated in one VM share its guest kernel, TEE instance, and administrative failure domain. The experiments therefore measure protocol scaling to 40 participants, but they do not provide the same security interpretation as 40 independently administered and attested machines.

Security lifecycle and faults. The evaluation focuses on cryptographic execution inside confidential guests. It does not benchmark a complete attestation-bound enrollment and provisioning protocol, sealed persistent storage, freshness and rollback protection, revocation, recovery, or software update. It also excludes crashes, malformed partial signatures, malicious coordinators, adversarial network schedules, and common-mode TEE compromise. The results establish benign-operation performance feasibility rather than empirical Byzantine robustness.

Implementation and workload scope. The results depend on the selected Rust libraries, our communication extensions, and their particular algorithms for signer selection, list handling, reliable broadcast, and interpolation. The complexity discussion therefore includes implementation-level choices. Signing uses a fixed 32-byte payload, cloud experiments are largely confined to one region, and we measure end-to-end latency rather than concurrent or saturated throughput. Other libraries, larger messages, wide-area networks, batching, and parallel clients may produce different trade-offs.

8 Conclusion and Future Work

We present RSS, a full-stack evaluation of threshold-signature execution inside VM-based TEEs. The implementation integrates GG20 threshold ECDSA, FROST, and threshold BLS into an extended EnGINE framework and evaluates DKG, preprocessing, and signing in local and Google Cloud configurations using AMD SEV-SNP and Intel TDX. The experiments scale to 40 logical participants allocated across four physical hosts or CVMs.

The results show that protocol and deployment choices dominate much of the observed end-to-end cost. In matched-platform comparisons, confidential execution generally introduces limited overhead or additional variance. DKG is the principal scaling bottleneck. Online signing remains in the order of tens of milliseconds in the evaluated configurations, with threshold BLS approximately twice as slow as FROST for comparable values of n and t. These findings support the performance feasibility of executing threshold-signature workloads in current VM-based TEEs under benign-operation assumptions.

The results do not establish protection against every malicious node operator or empirical robustness under Byzantine faults. A production RSS deployment must place shares across independent failure domains and add attestation-bound enrollment, persistent-state freshness, request authorization, revocation, recovery, and monitoring. Future work should

evaluate these lifecycle mechanisms, inject crash and Byzantine faults, measure concurrent throughput and wide-area deployments, and investigate whether TEE-assisted designs can safely reduce the communication or trust assumptions of DKG without introducing a common-mode compromise path.

References

[1] AMD. n.d.. AMDSEV: Secure Encrypted Virtualization, snp-latest Branch. https://github.com/AMDESE/AMDSEV/tree/snp-latest Accessed 2024-05-10.

[2] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. 2020. A Survey of ECDSA Threshold Signing. Cryptology ePrint Archive, Report 2020/1390. https://eprint.iacr.org/2020/1390

[3] L. Baird, S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang. 2023. Threshold Signatures in the Multiverse. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, Los Alamitos, CA, USA, 2057–2073. doi:10.1109/SP46215.2023.00118

[4] BankInfoSecurity. n.d.. Crypto Hackers Exploit Ronin Network for $615 Million. https://www.bankinfosecurity.com/crypto-hackers-exploit-ronin-network-for-615-million-a-18810 Accessed 2024-05-10.

[5] Mariarosaria Barbaraci, Noah Schmid, Orestis Alpos, Michael Senn, and Christian Cachin. 2025. Thetacrypt: A Distributed Service for Threshold Cryptography. doi:10.48550/arXiv.2502.03247 arXiv:2502.03247 [cs.CR]

[6] John Bebel and Dev Ojha. 2022. Ferveo: Threshold Decryption for Mempool Privacy in BFT Networks. Cryptology ePrint Archive, Paper 2022/898. https://eprint.iacr.org/2022/898

[7] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. 2024. SPRINT: High-Throughput Robust Distributed Schnorr Signatures. In *Advances in Cryptology – EUROCRYPT 2024, Part V (Lecture Notes in Computer Science, Vol. 14655)*. Springer, Cham, Switzerland, 62–91. doi:10.1007/978-3-031-58740-5_3

[8] F. Benhamouda, S. Halevi, H. Krawczyk, A. Miao, and T. Rabin. 2022. Threshold Cryptography as a Service (in the Multiserver and YOSO Models). In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 323–336. doi:10.1145/3548606.3559397

[9] Dan Boneh, John Bradley, Sergey Gorbunov, Riad S. Wahby, Hoeteck Wee, Christopher A. Wood, and Zhenfei Zhang. 2025. *BLS Signatures*. Internet-Draft draft-irtf-cfrg-bls-signature-06. Internet Research Task Force. https://datatracker.ietf.org/doc/draft-irtf-cfrg-bls-signature/06/ Expired Internet-Draft; latest revision 2025-11-02.

[10] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact Multi-Signatures for Smaller Blockchains. In *Advances in Cryptology – ASIACRYPT 2018, Part II (Lecture Notes in Computer Science, Vol. 11273)*. Springer, Cham, Switzerland, 435–464. doi:10.1007/978-3-030-03329-3_15

[11] Dan Boneh, Rosario Gennaro, and Steven Goldfeder. 2019. Using Level-1 Homomorphic Encryption to Improve Threshold DSA Signatures for Bitcoin Wallet Security. In *Progress in Cryptology – LATINCRYPT 2017 (Lecture Notes in Computer Science, Vol. 11368)*. Springer, Cham, Switzerland, 352–377. doi:10.1007/978-3-030-25283-0_19

[12] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *Advances in Cryptology – ASIACRYPT 2001 (Lecture Notes in Computer Science, Vol. 2248)*. Springer, Cham, Switzerland, 514–532. doi:10.1007/3-540-45682-1_30

[13] Marcin Bosk*, Filip Rezabek*, Kilian Holzinger, Angela G. Marino, Abdoul Aziz Kane, Francisco Fons, Jörg Ott, and Georg Carle. 2022. Methodology and Infrastructure for TSN-Based Reproducible Network Experiments. *IEEE Access* 10 (2022), 109203–109239. doi:10.1109/ACCESS.2022.3211969

[14] G. Bracha. 1987. Asynchronous Byzantine Agreement Protocols. *Information and Computation* 75, 2 (1987), 130–143. doi:10.1016/0890-5401(87)90054-X

[15] Luís T. A. N. Brandão and Rene Peralta. 2026. *NIST First Call for Multi-Party Threshold Schemes*. NIST Interagency/Internal Report 8214C. National Institute of Standards and Technology. doi:10.6028/NIST.IR.8214C

[16] C. Cachin, K. Kursawe, and V. Shoup. 2000. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing (PODC '00)*. Association for Computing Machinery, New York, NY, USA, 123–132.

[17] G. Caldarelli. 2020. Understanding the Blockchain Oracle Problem: A Call for Action. *Information* 11, 11 (2020), 509. doi:10.3390/info11110509

[18] G. Caldarelli. 2022. Overview of Blockchain Oracle Research. *Future Internet* 14, 6 (2022), 175. doi:10.3390/fi14060175

[19] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. 2020. UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, New York, NY, USA, 1769–1787. doi:10.1145/3372297.3423367

[20] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. 2023. YOLO YOSO: Fast and Simple Encryption and Secret Sharing in the YOSO Model. In *Advances in Cryptology – ASIACRYPT 2022, Part I (Lecture Notes in Computer Science, Vol. 13791)*, Shweta Agrawal and Dongdai Lin (Eds.). Springer, Cham, 651–680. doi:10.1007/978-3-031-22963-3_22

[21] Andrea Cerulli, Armando Connolly, Gregory Neven, Friederike-Sophie Preiss, and Victor Shoup. 2023. vetKeys: How a Blockchain Can Keep Many Secrets. Cryptology ePrint Archive, Paper 2023/616. https://eprint.iacr.org/2023/616

[22] Jing Chen, Sergey Gorbunov, Silvio Micali, and Georgios Vlachos. 2018. Algorand Agreement: Super Fast and Partition Resilient Byzantine Agreement. Cryptology ePrint Archive, Paper 2018/377. https://eprint.iacr.org/2018/377

[23] Hao Cheng, Yuan Lu, Zhenliang Lu, Qiang Tang, Yuxuan Zhang, and Zhenfeng Zhang. 2024. JUMBO: Fully Asynchronous BFT Consensus Made Truly Scalable. arXiv:2403.11238. https://arxiv.org/abs/2403.11238

[24] chrony project. n.d.. chrony: Introduction. https://chrony-project.org/ Accessed 2024-05-12.

[25] Cointelegraph. n.d.. North Korea's Lazarus Group Masterminded $100M Harmony Hack: FBI Confirms. https://cointelegraph.com/news/north-korea-s-lazarus-group-masterminded-100m-harmony-hack-fbi-confirms Accessed 2024-05-10.

[26] Elizabeth Crites, Chelsea Komlo, and Mary Maller. 2021. How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures. Cryptology ePrint Archive, Paper 2021/1375. https://eprint.iacr.org/2021/1375

[27] Alexander Daichendt, Florian Wiedner, Jonas Andre, and Georg Carle. 2024. Applicability of Hardware-Supported Containers in Low-Latency Networking. In *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, Prague, Czech Republic, 8 pages. doi:10.23919/CNSM62983.2024.10814577

[28] A. Dalskov, C. Orlandi, M. Keller, K. Shrishak, and H. Shulman. 2020. Securing DNSSEC Keys via Threshold ECDSA from Generic MPC. In *Computer Security - ESORICS 2020 (LNCS Sublibrary. SL4 -Security and Cryptology)*, Liqun Chen (Ed.). Springer, Cham, Switzerland, 654–673.

[29] Sourav Das, Zhuolun Xiang, Alin Tomescu, Alexander Spiegelman, Benny Pinkas, and Ling Ren. 2023. Verifiable Secret Sharing Simplified. Cryptology ePrint Archive, Paper 2023/1196. https://eprint.iacr.org/2023/1196

[30] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical Asynchronous Distributed Key Generation. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, Piscataway, NJ, USA, 2518–2534. doi:10.1109/SP46214.2022.9833584

[31] Yvo Desmedt. 1988. Society and Group Oriented Cryptography: A New Concept. In *Advances in Cryptology – CRYPTO '87 (Lecture Notes in Computer Science, Vol. 293)*, Carl Pomerance (Ed.). Springer, Berlin, Heidelberg, 120–127. doi:10.1007/3-540-48184-2_8

[32] Yvo Desmedt and Yair Frankel. 1990. Threshold Cryptosystems. In *Advances in Cryptology – CRYPTO 1989 (Lecture Notes in Computer Science, Vol. 435)*. Springer, Cham, Switzerland, 307–315. doi:10.1007/0-387-34805-0_28

[33] Sisi Duan, Michael K. Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT Made Practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. Association for Computing Machinery, New York, NY, USA, 2028–2041. doi:10.1145/3243734.3243812

[34] ESnet. n.d.. iperf3 Documentation. https://software.es.net/iperf/ Accessed 2026-08-12.

[35] fio project. n.d.. fio 3.36 Documentation. https://fio.readthedocs.io/en/latest/fio_doc.html Accessed 2024-10-17.

[36] frost-dalek contributors. n.d.. frost-dalek: A Rust implementation of FROST: Flexible Round-Optimised Schnorr Threshold signatures using the Ristretto group. https://github.com/isislovecruft/frost-dalek Accessed 2024-10-15.

[37] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong. 2020. Fully Distributed Verifiable Random Functions and their Application to Decentralised Random Beacons. Cryptology ePrint Archive, Report 2020/096.

[38] Rosario Gennaro and Steven Goldfeder. 2018. Fast Multiparty Threshold ECDSA with Fast Trustless Setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM, Association for Computing Machinery, New York, NY, USA, 1179–1194. doi:10.1145/3243734.3243859

[39] Rosario Gennaro and Steven Goldfeder. 2020. One Round Threshold ECDSA with Identifiable Abort. Cryptology ePrint Archive, Paper 2020/540. https://eprint.iacr.org/2020/540

[40] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. 2016. Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. In *Applied Cryptography and Network Security – ACNS 2016 (Lecture Notes in Computer Science, Vol. 9696)*. Springer, Cham, Switzerland, 156–174. doi:10.1007/978-3-319-39555-5_9

[41] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. 2003. Secure Applications of Pedersen's Distributed Key Generation Protocol. In *Topics in cryptology, CT-RSA 2003 (Lecture Notes in Computer Science)*, Marc Joye (Ed.). Springer-Verlag, Berlin and New York, 373–390.

[42] Rosario Gennaro, Tal Rabin, Stanisław Jarecki, and Hugo Krawczyk. 2000. Robust and Efficient Sharing of RSA Functions. *Journal of Cryptology* 13, 2 (2000), 273–300. doi:10.1007/s001459910011

[43] C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakoubov. 2021. YOSO: You Only Speak Once. In *Advances in Cryptology – CRYPTO 2021, Part II (Lecture Notes in Computer Science, Vol. 12826)*, Tal Malkin and Chris Peikert (Eds.). Springer International Publishing, Cham, 64–93. doi:10.1007/978-3-030-84245-1_3

[44] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 51–68. doi:10.1145/3132747.3132757

[45] Google Cloud. n.d.. Confidential VM Supported Configurations. https://cloud.google.com/confidential-computing/confidential-vm/docs/supported-configurations Accessed 2024-05-21.

[46] Google Cloud. n.d.. Create VMs with Multiple Network Interfaces. https://cloud.google.com/vpc/docs/create-use-multiple-interfaces Accessed 2024-05-12.

[47] Vincent Gramoli, Rachid Guerraoui, Andrei Lebedev, Christopher Natoli, and Gauthier Voron. 2023. Diablo: A Benchmark Suite for Blockchains. In *Proceedings of the Eighteenth European Conference on Computer Systems (EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 540–556. doi:10.1145/3552326.3567482

[48] Jens Groth and Victor Shoup. 2022. Design and Analysis of a Distributed ECDSA Signing Service. Cryptology ePrint Archive, Paper 2022/506. https://eprint.iacr.org/2022/506

[49] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. SBFT: A Scalable and Decentralized Trust Infrastructure. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, Piscataway, NJ, USA, 568–580. doi:10.1109/DSN.2019.00063

[50] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. 2020. Dumbo: Faster Asynchronous BFT Protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (ACM Digital Library)*, Jay Ligatti (Ed.). Association for Computing Machinery, New York,NY,United States, 803–818. doi:10.1145/3372297.3417262

[51] Marius Haller, Marc Leinweber, Tilo Spannagel, Markus Raiber, and Hannes Hartenstein. 2026. Performance and Security of TEE-Based Threshold Cryptography. In *Proceedings of the 9th Workshop on System Software for Trusted Execution (SysTEX '26)*. Association for Computing Machinery, New York, NY, USA, 42–49. doi:10.1145/3805690.3805725

[52] T. Hanke, M. Movahedi, and D. Williams. 2018. DFINITY Technology Overview Series: Consensus System. https://internetcomputer.org/whitepaper.pdf

[53] Stephen Hemminger. n.d.. tc-netem(8) Linux Manual Page. https://man7.org/linux/man-pages/man8/tc-netem.8.html Accessed 2024-05-18.

[54] Intel. n.d.. tdx-tools: Cloud Stack and Solutions for Intel TDX. https://github.com/intel/tdx-tools Accessed 2024-05-10.

[55] Aniket Kate, Yizhou Huang, and Ian Goldberg. 2012. Distributed Key Generation in the Wild. Cryptology ePrint Archive, Paper 2012/377. https://eprint.iacr.org/2012/377

[56] Chelsea Komlo and Ian Goldberg. 2021. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. In *Selected Areas in Cryptography – SAC 2020 (Lecture Notes in Computer Science, Vol. 12804)*. Springer, Cham, Switzerland, 34–65. doi:10.1007/978-3-030-81652-0_2

[57] Chelsea Komlo and Ian Goldberg. n.d.. FROST Reference Implementation. https://git.uwaterloo.ca/ckomlo/frost/ Accessed 2024-05-18.

[58] Rujia Li, Qin Wang, Qi Wang, David Galindo, and Mark Ryan. 2022. SoK: TEE-Assisted Confidential Smart Contracts. arXiv:2203.08548. https://arxiv.org/abs/2203.08548

[59] Y. Lindell. 2017. Fast Secure Two-Party ECDSA Signing. In *Advances in Cryptology – CRYPTO 2017, Part II (Lecture Notes in Computer Science, Vol. 10402)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, Cham, Switzerland, 613–644. doi:10.1007/978-3-319-63715-0_21

[60] Yehuda Lindell. 2024. Simple Three-Round Multiparty Schnorr Signing with Full Simulatability. *IACR Communications in Cryptology* 1, 1, Article 25 (2024), 24 pages. doi:10.62056/a36c0l5vt

[61] Yehuda Lindell and Ariel Nof. 2018. Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. Association for Computing Machinery, New York, NY, USA, 1837–1854. doi:10.1145/3243734.3243788

[62] Vasilios Mavroudis, Andrea Cerulli, Petr Švenda, Dan Cvrček, Dušan Klinec, and George Danezis. 2017. A Touch of Evil: High-Assurance Cryptographic Hardware from Untrusted Components. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA) *(CCS '17)*. ACM, New York, NY, USA, 1583–1600. doi:10.1145/3133956.3133961

[63] John D. McCalpin. 1995. Memory Bandwidth and Machine Balance in Current High Performance Computers. IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter. pp. 19–25.

[64] Jérémy Ménétrey, Christian Göttel, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. 2022. An Exploratory Study of Attestation Mechanisms for Trusted Execution Environments. arXiv:2204.06790. https://arxiv.org/abs/2204.06790

[65] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. 2016. The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 31–42. doi:10.1145/2976749.2978399

[66] Bakhytzhan Nasrulin, Martijn de Vos, Georgy Ishmaev, and Johan Pouwelse. 2022. Gromit: Benchmarking the Performance and Scalability of Blockchain Systems. In *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE, Piscataway, NJ, USA, 56–63.

[67] K. Nelaturu, H. Du, and D.-P. Le. 2022. A Review of Blockchain in Fintech: Taxonomy, Challenges, and Future Directions. *Cryptography* 6, 2 (2022), 18. doi:10.3390/cryptography6020018

[68] Arttu Paju, Muhammad Owais Javed, Jari Nurmi, Juha Savimäki, Brian McGillion, and Billy Bob Brumley. 2023. SoK: A Systematic Review of TEE Usage for Developing Trusted Applications. In *Proceedings of the 18th International Conference on Availability, Reliability and Security (ARES '23)*. Association for Computing Machinery, New York, NY, USA, Article 34, 15 pages. doi:10.1145/3600160.3600169

[69] Trevor Perrin. n.d.. The Noise Protocol Framework. https://noiseprotocol.org/noise.html Accessed 2024-05-12.

[70] Michaella Pettit. 2021. Efficient Threshold-Optimal ECDSA. In *Cryptology and Network Security – CANS 2021 (Lecture Notes in Computer Science, Vol. 13099)*. Springer, Cham, Switzerland, 116–135. doi:10.1007/978-3-030-92548-2_7

[71] B. Pillai, K. Biswas, Z. Hou, and V. Muthukkumarasamy. 2022. Cross-Blockchain Technology: Integration Framework and Security Assumptions. *IEEE Access* 10 (2022), 41239–41259. doi:10.1109/ACCESS.2022.3167172

[72] Red Hat. n.d.. Ansible Automation Platform. https://www.ansible.com/ Accessed 2024-05-16.

[73] Filip Rezabek, Marcin Bosk, and Thomas Paul. 2022. EnGINE Experimentation Framework. https://github.com/rezabfil-sec/engine-framework Accessed 2024-05-18.

[74] Filip Rezabek, Marcin Bosk, Thomas Paul, Kilian Holzinger, Sebastian Gallenmüller, Angela Gonzalez, Abdoul Kane, Francesc Fons, Haigang Zhang, Georg Carle, and Jörg Ott. 2021. EnGINE: Developing a Flexible Research Infrastructure for Reliable and Scalable Intra-Vehicular TSN Networks. In *2021 17th International Conference on Network and Service Management (CNSM)*. IEEE, Izmir, Turkey, 530–536. doi:10.23919/CNSM52442.2021.9615529

[75] Filip Rezabek, Marcin Bosk, Thomas Paul, Kilian Holzinger, Sebastian Gallenmüller, Angela Gonzalez, Abdoul Aziz Kane, Francisco Fons, Haigang Zhang, Georg Carle, and Jörg Ott. 2022. EnGINE: Flexible Research Infrastructure for Reliable and Scalable Time Sensitive Networks. *Journal of Network and Systems Management* 30, 4 (08 Sept. 2022), 74. doi:10.1007/s10922-022-09686-0

[76] Filip Rezabek, Kilian Glas, Richard von Seck, Achraf Aroua, Tizian Leonhardt, and Georg Carle. 2023. Multilayer Environment and Toolchain for Holistic Network Design and Analysis. arXiv:2310.16190. https://arxiv.org/abs/2310.16190

[77] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. 2022. ROAST: Robust Asynchronous Schnorr Threshold Signatures. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 2551–2564.

[78] M. U. Sardar, S. Musaev, and C. Fetzer. 2021. Demystifying Attestation in Intel Trust Domain Extensions via Formal Verification. *IEEE Access* 9 (2021), 83067–83079. doi:10.1109/ACCESS.2021.3087421

[79] Claus-Peter Schnorr. 1990. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology – CRYPTO 1989 (Lecture Notes in Computer Science, Vol. 435)*. Springer, Cham, Switzerland, 239–252. doi:10.1007/0-387-34805-0_22

[80] Secret Network. n.d.. Secret Network: A Privacy-Preserving Secret Contract and Decentralized Application Platform. https://whitepaper.io/document/733/secret-whitepaper Accessed 2024-05-29.

[81] Keyvan Sedghighadikolaei and Attila Altay Yavuz. 2023. A Comprehensive Survey of Threshold Digital Signatures: NIST Standards, Post-Quantum Cryptography, Exotic Techniques, and Real-World Applications. arXiv:2311.05514. https://arxiv.org/abs/2311.05514

[82] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (nov 1979), 612–613. doi:10.1145/359168.359176

[83] K. Shrishak and H. Shulman. 2021. Privacy Preserving and Resilient RPKI. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. IEEE, Piscataway, NJ, USA, 1–10. doi:10.1109/INFOCOM42981.2021.9488759

[84] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. 2020. Towards Scalable Threshold Cryptosystems. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 877–893. doi:10.1109/SP40000.2020.00059

[85] Newton C. Will and Carlos A. Maziero. 2023. Intel Software Guard Extensions Applications: A Survey. *Comput. Surveys* 55, 14s, Article 322 (jul 2023), 38 pages. doi:10.1145/3593021

[86] P. Wu, J. Ning, J. Shen, H. Wang, and E.-C. Chang. 2022. Hybrid Trust Multi-Party Computation with Trusted Execution Environments. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2022)*. Internet Society, Reston, VA, USA, 18 pages. doi:10.14722/ndss.2022.24173

[87] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (Toronto ON, Canada) *(PODC '19)*. Association for Computing Machinery, New York, NY, USA, 347–356. doi:10.1145/3293611.3331591

[88] Anastasiya Yudina. 2022. 2022 Solana Hacks Explained: Wormhole. https://ackee.xyz/blog/2022-solana-hacks-explained-wormhole/ Accessed 2024-05-10.

[89] ZenGo-X. n.d.. multi-party-bls: Threshold BLS Signatures in Rust. https://github.com/ZenGo-X/multi-party-bls Accessed 2024-10-04.

[90] ZenGo-X. n.d.. multi-party-ecdsa: Rust Implementation of $\{t, n\}$-Threshold ECDSA. https://github.com/ZenGo-X/multi-party-ecdsa Accessed 2024-05-18.