arXiv:2503.15654v2 [cs.CR] 8 Jan 2026

ACURAST: DECENTRALIZED SERVERLESS CLOUD

0.3 - JAN 08, 2026, WHITE PAPER

CHRISTIAN KILLER, ALESSANDRO DE CARLI, PASCAL BRUN,
AMADEO VICTOR CHARLÉ, MIKE GODENZI, SIMON WEHRLI

ACURAST ASSOCIATION, BAARERSTRASSE 43, 6300 ZUG
⟨ FIRST NAME ⟩ @ACURAST.COM

Abstract. Centralized trust is ubiquitous in today's interconnected world, from computational resources to data storage and its underlying infrastructure. The monopolization of cloud computing resembles a feudalistic system, causing a loss of privacy and data ownership.

Cloud Computing and the Internet in general face widely recognized challenges, such as *(i)* the centralization of trust in auxiliary systems (*e.g.,* centralized cloud providers), *(ii)* the seamless and permissionless interoperability of fragmented ecosystems and *(iii)* the effectiveness, verifiability, and confidentiality of the computation. Acurast is a decentralized serverless cloud that addresses all these shortcomings, following the call for a global-scale cloud founded on the principles of the open-source movement.

In Acurast, a purpose-built orchestrator, a reputation engine, and an attestation service are enshrined in the consensus layer. Developers can off-load their computations and verify executions cryptographically. Furthermore, Acurast offers a modular execution layer, taking advantage of secure hardware and trusted execution environments, removing the trust required in third parties, and reducing them to cryptographic hardness assumptions. With this modular architecture, Acurast serves as a decentralized and serverless cloud, allowing confidential and verifiable compute backed by the hardware of security and performance mobile devices.

1. Introduction

The broad adoption of cloud computing has resulted in an Internet landscape that is strikingly similar to feudalism [1]. The dominance of a few monopolistic players created a scenario where trust in cloud operators is a requirement, not just an option. Technology critics even argued that the "Lords of the Cloud" are wreaking havoc on the world economy [2], causing loss of privacy, data ownership, and ease of mass surveillance [1]. The only way forward is not simply to *distribute* the Internet's architecture, but *decentralize* it at large. A suitable path forward is to develop a global-scale cloud founded on open source principles, therefore aligning with the public interest [1].

> Trust is our only option. In this system, we have no control over the security provided by our feudal lords. We don't know what sort of security methods they're using, or how they're configured.
>
> Bruce Schneier, 2012 [3]

The fundamental prerequisites for a global-scale cloud network founded on open-source principles are: *(i)* global-scale connectivity, *(ii)* widespread access to computation (*e.g.,* smartphones), and *(iii)* societal acceptance of permissionless systems and their adoption.

First, the rapid spread of smartphones has profoundly influenced the modern digital landscape. As of mid-2023, nearly 96 percent of the global digital population had accessed the Internet using a mobile device. Furthermore, improved efficiency are exemplified by ARM-based architectures. Reduced Instruction Set Computing (RISC) forms the foundation of the Advanced RISC Machine (ARM) architecture, providing superior energy efficiency compared to traditional x86 architectures.

The proliferation of smartphones has also influenced laptops and desktops to adopt ARM-based instruction set architectures, even expanding to commercial environments, where ARM chips are increasingly employed in cloud servers [4].

In parallel to these developments, society's understanding of cryptoeconomic systems has evolved. Cryptographic tools have been developed in an open source way, shaping society's willingness to deploy cryptoeconomic systems [5]. The rapid development and adoption of public blockchains has driven forward real-world use cases. These use cases demonstrate the long-term impact and relevance to key elements of society's fabric that create resilient and *unstoppable* systems [6].

A key property of decentralized protocols is the verifiability of the computation. In blockchain-based systems, the consensus layer ensures that state changes are verified and the consensus rules are enforced and followed. Relying on distributed consensus guarantees high security; however, in practice, it means that every participating node has to perform the computation, also.

The blockchain trilemma is a guiding principle in the design and implementation of blockchains. However, the trilemma ignores two crucial factors: the first is the effectiveness of the computation. Thus, the blockchain quadrilemma [7] further extends this notion, considering the ability *computational effectiveness*, *i.e.,* to run complex decentralized computations at affordable costs, which poses a fundamental problem in blockchains today.

Approaches addressing these issues often ignore crucial factors, such as the hardware layer *i.e.,* the *trustless* decentralization of its physical execution layers, leading to centralization of crucial auxiliary systems (*e.g.,* cross-chain bridges or sequencers). A recent study estimates that approximately $2 billion in cryptocurrency has been

Archived copy · cryp2nova.com

stolen in 13 different cross-chain bridge hacks, representing 69% of the total stolen funds alone in 2022 [8]. Ironically, many of these auxiliary systems are run on public clouds, and by now public cloud providers are even actively providing blockchain tools [9].

Furthermore, a major strength of blockchains can be considered as their main weakness: the public and transparent availability of all events for anyone to read [10]. Thus, the second extension of the blockchain trilemma is that confidentiality is required for certain computations, and the public nature of blockchains negatively impacts the relevant execution. Therefore, we extend the quadrilemma with a nuanced notion of *computational effectiveness and confidentiality*.

> In words from history, let us speak no more of faith in man, but bind him down from mischief by the chains of cryptography.
>
> Edward Snowden [11]

Acurast is a novel decentralized serverless cloud that addresses the shortcomings of today's computing infrastructure. With its novel zero trust approach, Acurast *trustlessly* provides a confidential and verifiable compute infrastructure. Acurast follows a modular separation of the consensus, execution, and application layer. The Acurast orchestrator is embedded in the consensus layer and contains a purpose-built reputation engine to ensure reliability and incentivize honest behavior. Furthermore, the consensus layer contains an Attestation service, ascertaining that the computation is end-to-end verifiable.

The foundation of the Acurast execution layer is made up of individual compute units (*i.e.,* smartphones). Smartphones are extremely energy efficient and yield better performance-per-watt compared to traditional server hardware [12]. Furthermore, ARM-based Systems-on-a-Chip (SoC) not only outperform traditional hardware, but also offer more secure hardware.

For instance, on Android-based smartphones, the Acurast Trusted Execution Environments (TEE) is bootstrapped by fully locking down the OS. Then, the TEE signs and publishes the public key on the Attestation service, allowing anyone to verify the authenticity of the TEE and hardware. The deployment of TEE reduces the trust assumptions to *(i)* cryptographic hardness assumptions and *(ii)* a single root of trust (*i.e.,* secure hardware with key attestation and an external coprocessor).

Acurast lays the foundation for a confidential, verifiable, and decentralized cloud. Potential use cases range from computation (*e.g.,* Federated Learning), or data blockchain infrastructure (*e.g.,* Zero Knowledge-as-a-Service, On-Chain Automation, Price feeds) to distributed trust in Federated Learning (FL) [13], native cross-chain DeFi capabilities, and privacy mixing [14, 15].

The remainder of this work is organized as follows. An end-to-end deployment is detailed in Section 2. The Acurast architecture is described in Section 3, which contains an overview of the high-level architecture and its different layers. The zero trust execution layer is described in Section 4. Section 5 discusses the consensus layer. Finally, Section 6 describes different areas of application of Acurast, and Section 8 summarizes.

2. End-to-End Zero Trust Deployment

Acurast introduces a paradigm shift in *verifiable and confidential* computation, advancing the way decentralized clouds are operated, and applications are developed and deployed. To emphasize the inner workings of Acurast, the following description follows a `deployment` from its prerequisite steps, its inception to the final completion (*cf.* Fig. 1).

(0) Processor Attestation This step is a prerequisite to bootstrapping the Acurast Trusted Execution Envirnoment (TEE) of the individual processors. As a first step, the processor is generating an asymmetric key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KGen}$, whereas the private key of that pair is remaining in the dedicated secure element (*i.e.,* coprocessor, depending on the terminology of the hardware SoC). For example, in novel Google phones, the Titan M2 chip's dedicated coprocessor is used [16], whereas on Snapdragon Systems-on-a-Chip (SoC), the Qualcomm Secure Execution Environment (QSEE) is used). To conclude this step, `pk` is signed and sent to the Attestation service, where the trust root of the respective TEE can be verified (*e.g.,* Google as a trust root for Titan M2)

(1) `Deployment` Registration: As a first step, developers define their `deployment` details. For example, what kind of Runtime should be used, *e.g.,* Node.js. Also, whether or not the application contains a Web3 destination *i.e.,* at what destination the `deployment` should be *settled*. In other words, on which protocol the `deployment` `output` should be persisted (*e.g.,* Bitcoin Mainnet), or whether the app is settled by calling a REST API (*e.g.,* web crawling). After that, the developer can select *ready-to-deploy* templates, which can be adapted to the developer's needs, or a custom `deployment` can be defined.

Depending on fidelity of integration of the destination ecosystem with Acurast, the pre-payments for gas fees and rewards are settled in the native currency the developer prefers (*e.g.,* native `TEZ` for Tezos or `ETH` for Ethereum) or in native Acurast `ACU` token.

Next, the developer must state on which processors the `deployment` should be executed, either *(a)* on personal processors, or *(b)* on selected, known processors (*e.g.,* known trusted entities), or *(c)* on public processors. For *(a)*, a processor reward is not a hard requirement, since it is a permissioned setting. For *(b)* a reward is optional, and for *(c)* the liquid matching engine and the Acurast orchestrator will match appropriate processor resources with developers' `deployments`.

In addition, more details of the `deployment` need to be declared, such as *scheduling* parameters, including the start time, end time, the interval between executions, as well as the duration in milliseconds and the maximum start delay in milliseconds. Furthermore, specific resource management parameters, such as memory usage, network requests, and storage requirements of the `deployment` need to be declared. Finally, the reward for the execution of the deployment should be declared, as well as the minimum reputation (only applies to *(c)* public processors). Then the `deployment` will be persisted on the Acurast Consensus Layer and reaches `OPEN` state (*cf.* Fig. 2).

Archived copy · cryp2nova.com

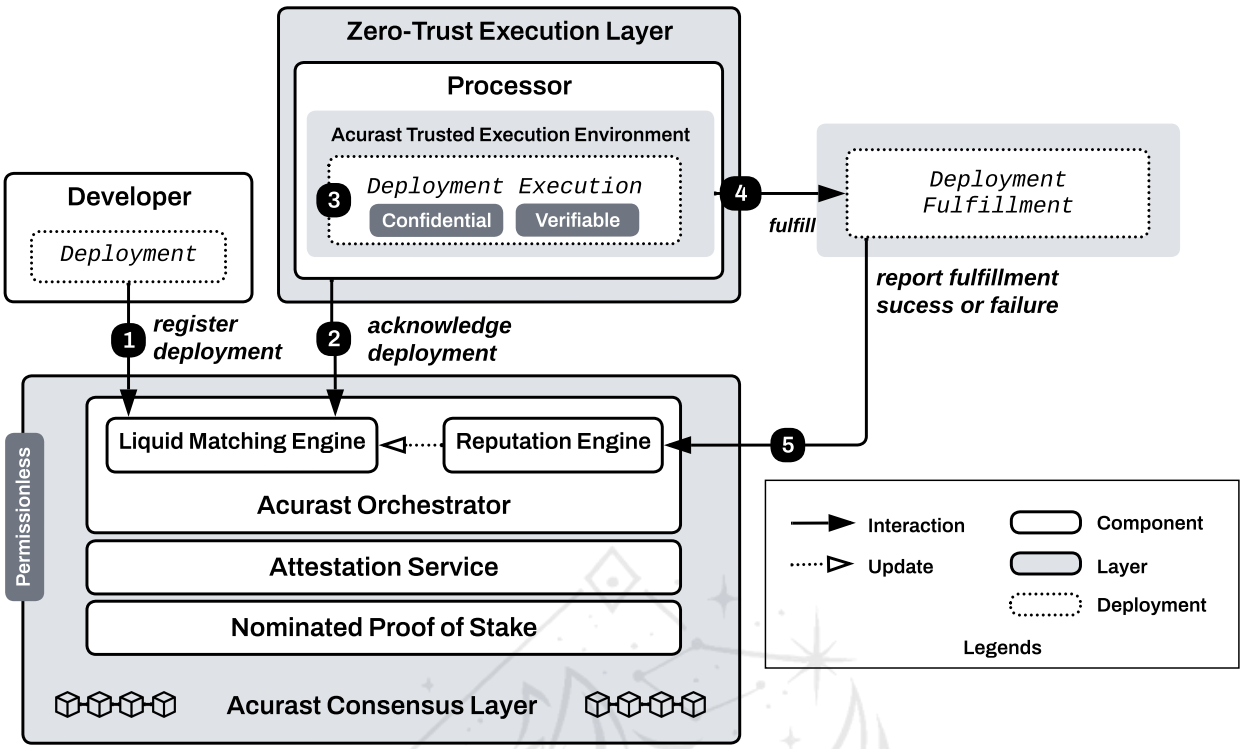


FIGURE 1. End-to-End Zero Trust Deployment Execution

(2) `Deployment` Acknowledgment: Second, the processor acknowledges the `deployment` and fetches the details from the Acurast network. Depending on the fulfillment definition of the respective `deployment`, the Merkle root of the `deployment` with proof of assignment is persisted on the target destination (*e.g.,* on a different target chain). Now the `deployment` reaches the `MATCHED` state, and no other processors will attempt to acknowledge it.

A prerequisite for assigning the deployment to the processor is that the processor can execute the `deployment` in full, following the *all-or-nothing* principle (*cf.* Sec. 5.4 for more details). Since `deployments` can have different scheduling configurations (*e.g.,* on demand, every minute, etc.). Therefore, if the processor acknowledges that all slots can be adhered to, the `deployment` reaches the `ASSIGNED` state.

(3) `Deployment` Execution: Next, the `deployment_script` is executed in the processor runtime. In the illustrated example of Fig. 1, the execution is performed inside of the Acurast Trusted Execution Environment (TEE) (*cf.* Sec. 4.1), *i.e.,* because *confidentiality* is ascertained by secure hardware *e.g.,* an isolated and external coprocessor (*e.g.,* Google's Titan Chip [16]).

(4) `Deployment` Fulfillment: Once the `deployment` execution is completed, the output is delivered to the declared destination, which could be a REST-API, another AI model, or even another Web3 system (*e.g.,* Tezos, Ethereum). In case of a cross-chain transaction, the processor settles the gas fees on the destination chain, since the developer has locked the necessary reward and gas fee amount up front when registering the `deployment`.

(5) `Deployment` Reporting: After completion, the processor reports back to the Acurast Consensus Layer, more specifically to the reputation engine. If fulfillment was successful, the report contains a transaction hash of the target chain containing the fulfillment transaction. In case of failure, the report contains error messages. Finally, the `deployment` is now in `DONE` state.

To ensure the reliability of the Acurast protocol, the reputation engine is continuously fed with reliability metrics, for instance, right after `deployment` completion or failure. Furthermore, the attestation service is updated if a certificate is revoked (*e.g.,* a certain hardware module is compromised or the attestation expired).

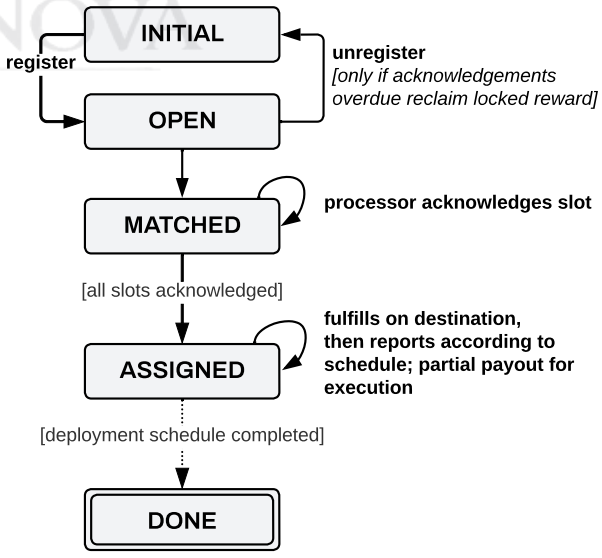


FIGURE 2. States of a deployment.

Archived copy · cryp2nova.com

3. Acurast Architecture

Acurast separates the consensus, execution, and application layer (*cf.* Fig. 3). The consensus layer is the permissionless foundation of Acurast and is built on an adaptation of the Nominated Proof-of-Stake (NPoS) algorithm [17] (*cf.* Sec. 5). The Acurast Orchestrator is a vital component of the consensus layer, matching consumer's `deployments` to processors, as previously outlined in the End-to-End flow (*cf.* Sec. 2).

The second core part of the consensus layer is the reputation engine (*cf.* Sec. 5.5), which assures that the reputation scores of processors are correctly updated and incentivize honest behavior. A third key element is the attestation service which is responsible for the cryptographically verifiable attestations of the Acurast TEEs. The service is responsible for storing signed public keys, or revocations thereof.

The Zero Trust Execution Layer (*cf.* Sec. 4) has major components. The first is composed of the execution environment, namely the Acurast TEE, *cf.* Sec. 4.1). The second key component are the Runtimes (*cf.* Sec. 4.3), which contains multiple modules that enable native interaction with different ecosystems.

The third layer is the application layer, where applications composed of deployments are run (*cf.* Sec. 6). Acurast infuses the development of a wide range of use cases that were previously not possible to implement in a permissionless and confidential manner. *E.g.,* the trustless execution of market intelligence tasks (*i.e.,* web crawling), or the training and inference of an Artificial Intelligence (AI) models.

In summary, Acurast's Zero Trust architecture transforms the way applications are designed and deployed. The modular nature allows for a truly decentralized, verifiable, and confidential compute platform, without introducing new trusted entities.

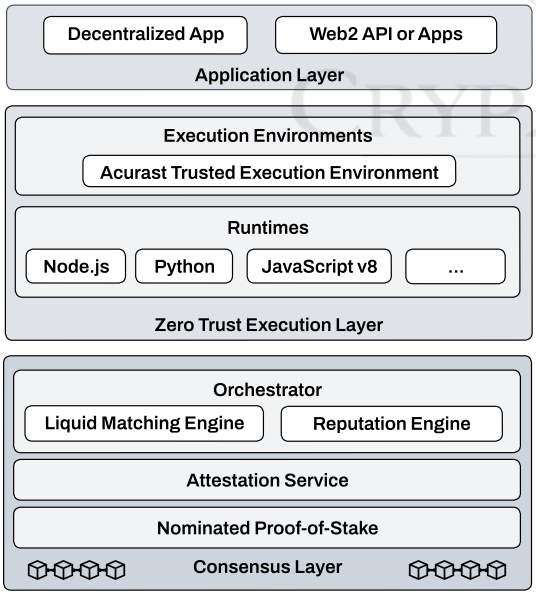


Figure 3. Acurast Architecture

4. Zero Trust Execution Layer

Acurast's execution layer is modular, allowing the flexible selection of runtimes according to the requirements of the use-case and the `deployment`, respectively. Decoupling the execution layer from the consensus and application layer allows the long-term evolution of the execution environment and its runtimes, avoiding dependency lock-ins. In addition, it ensures the highest level of service and confidentiality because security models can evolve iteratively with upgrades as novel threats emerge or new requirements arise.

Acurast offers native and straightforward bootstrapping of permissioned consortia. Depending on the requirements, either *(a)* consumers can directly leverage the Acurast orchestrator to select from a public pool of processors, or *(b)* or use dedicated processors (*e.g.,* from trusted entities, or use consumer-supported self-service processors). This composability allows consumers to customize access control and define individual trust models depending on the `deployments` that are executed.

The following sections outline the details of Acurast TEE (*cf.* Sec. 4.1), then detailing the Acurast Runtimes (*cf.* Sec. 4.3). Finally, verifiable computation and rationales on using smartphones are outlined.

4.1. Acurast Trusted Execution Environment.

The Acurast Trusted Execution Environment (ATEE) is a generic approach to achieve a confidential execution layer while assuming a timely threat model, thus ensuring the highest possible level of security. The security guarantees achieved by secure hardware are generally highly divergent, from virtual processors to on-SoC processors, and finally, to the current bleeding edge of an *external coprocessor*, which is a physically separated and independent chip, dedicated to only security-sensitive operations [16].

Smartphones are among the most complex cases when it comes to information security. Their computing power has grown to the point of being almost indistinguishable from computers, they store the most valuable personal data and are used to carry out security-sensitive activities, which make them extremely attractive targets for attackers. With such a wide-ranging threat model and the fact that the vast computing base of a modern OS cannot be fully trusted, vendors have begun to use hardware to improve the security of their systems [16].

Thus, a highly secured ATEE instantiation is based on coprocessors provided by the Google Titan chip [18]. The Titan chip has not been compromised, unlike most secure hardware platforms. Furthermore the Titan M2 is a standalone processor with separate flash memory and minimal OS (often called microkernel). Crucially, these so-called coprocessors do not share memory or cache. Although high-reward bug bounties [19] and the highest zero-day vulnerability payouts [20] do not *guarantee* security, they are a solid indication of the security level achieved.

Usually, TEEs are created by integrating protection mechanisms directly into the processor or using dedicated external secure elements. However, both approaches only cover a narrow threat model, resulting in very limited security guarantees. For instance, enclaves nested in the application processor provide weak isolation and weak protection against side-channel attacks. Regardless of the approach used, TEEs often lack the ability to establish

Archived copy · cryp2nova.com

secure communication with peripherals, and most operating systems run inside TEEs do not provide state-of-the-art defense strategies, making them vulnerable to various attacks. Arguably, TEEs, such as Intel SGX [21, 22] or ARM TrustZone [23], implemented on the main application processor, are insecure, *particularly* when considering side-channel attacks. For that reason, the security level of the ATEE is also dependent on the underlying hardware.

4.2. **Verifiable Computation.** A key property of decentralized computation protocols is the verifiability of the said computation. In blockchain-based systems, the consensus layer ensures that state changes are verified and the consensus rules are enforced and followed. Relying on distributed consensus guarantees high security; however, in practice, it means that every participating node has to perform the computation, also. For that reason, blockchains are impractical for use cases normally running on centralized clouds.

4.2.1. *ZK-based Verifiable Compute.* A potential approach is presented by Zero-Knowledge Proof (ZKP) runtimes, as these allow for arbitrary computation (*e.g.,* Mina [24]). While the verification of ZKPs is a low-effort computational task, ZKP generation is an intensive workload. Although ZKPs are becoming more efficient with time, the overhead of proof generation is most likely to remain in the foreseeable future [25]. However, the complexity of ZKP implementations and its rapid evolution may introduce various vulnerabilities, undermining its very properties that it aims to introduce [26].

4.2.2. *Secure Hardware-based Verifiable Compute.* Another approach to verifiable computation is the use of secure hardware. Acurast implements this approach in its ATEE (*cf.* Sec.4.1, where secure hardware is leveraged to create an environment able to prove its integrity and allow for efficient and arbitrary verifiable computation.

4.3. **Runtimes.** The Acurast execution layer natively offers multiple native `runtimes` that are continuously extended. `Runtimes` are APIs that are exposed to different apps running as `deployments`. For instance, runtimes include Node.js, JavaScript v8, and Python. Runtimes will be extended continuously.

5. Permisionless Consensus Layer

The Permissionless Consensus layer forms the base of the Acurast protocol and is based on a variant of the Nominated Proof-of-Stake (NPoS) algorithm [17]. Unlike traditional Proof-of-Stake (PoS) networks, there are *validators* and *nominators* in NPoS. Block validators verify transactions to be included in the next block, similar to traditional PoS block validators. The key difference is that instead of being randomly chosen, the *validator* nodes are *nominated* by another node.

In Acurast's NPoS, an unlimited amount of token holders can participate as *nominators*, backing a limited set of *validators* with their stake. Having a limited set of *validators* assures the long-term scalability of the consensus, allowing the increase of the maximum threshold through governance decisions [27]. An unlimited set of *nominators* assure that higher value is at stake, assuring a high level of security. Due to its upgradable runtimes, consensus parameters are configurable by governance decisions, *e.g.,* The maximum number of *validators*, and the minimum amount of stake for *validators*.

The Acurast NPoS system heavily leverages nominators to ensure network integrity. Nominators and validators have multiple aligned incentives. Nominators hold a financial stake in the system, which means that they could suffer a loss if a validator acts maliciously. In addition, nominators are financially rewarded for selecting a reliable and high-performance validator. Both nominators and validators have reputational stakes, with the credibility of nominators affected by their validator choices. Finally, the limited number of validator slots in NPoS structures creates a competitive environment, pushing nominators to select the most efficient validators, but also democratizes the process, making voting power essential.

NPoS has proven to be an efficient way to achieve high levels of security, scalability, and decentralization over time. Congruent to [17], nominators share the rewards, or eventual slashings [27], with the validators they nominated on a *per-staked-`ACU`* basis.

5.1. **Attestation Service.** The Attestation service is an integral part of the Acurast security mechanisms. Acurast uses secure hardware (*cf.* Sec. 4.1) to attest its processors hardware and the executable in a fully remote fashion, *i.e.,* a remote attestation is requested, which cryptographically links the binary hash of the executable e (*i.e.,* app `deployment`) with the secret key sk_p, remaining securely in the ATEE of the respective processor p.

The respective key pair $(\mathsf{pk}_p, \mathsf{sk}_p)$ can only be accessed by that specific executable e, if e is tampered with or deleted, and the access to sk is lost. The chain of trust is cryptographically verifiable from the executable e to the original hardware manufacturer, allowing to form a verifiable execution environment inside the executable, which can further specify the preconditions for *e.g.,* allowing the executable to sign payloads.

For example, a specific NodeJS `deployment` d receives its individual key pair $(\mathsf{sk}_d, \mathsf{pk}_d)$ linked to the code of the `deployment` itself. So, if the NodeJS code is changed, access to the key pair $(\mathsf{pk}_d, \mathsf{sk}_d)$ is lost forever, and only the said deployment can effectively sign payloads with that sk. This allows an external party to deduce that if a specific cryptographic signature is linked to a remotely attested key pair, this signature could only be a result of the execution of that specific set of instructions, offering an end-to-end verifiable execution environment.

In the Attestation Service, devices advertise their compute resources along with their attestations. Only devices with valid attestations can then be matched with deployments, giving developers the security guarantees of an untampered execution environment. Furthermore, the attestations also include valuable information of the specific model of the hardware security module as described above, effectively allowing the developer to choose the level of security they require for their `deployment`.

5.2. **Acurast Orchestrator.** The Acurast orchestrator is a centerpiece of the consensus layer, combining the orchestration (*i.e.,* the scheduling of `deployments` and enabling the liquid matching) of the Processor's computational resources and developers. The orchestrator plays an essential role in the definition, agreement, and enforcement of value exchange.

Archived copy · cryp2nova.com

The orchestrator is where the liquid matching engine (*cf.* Sec. 5.4) pairs the advertised processor resources with the defined requirements. The orchestrator natively supports various price-finding mechanisms (*e.g.,* auctions and advertisements), making DevEx highly accessible and seamless.

Every agreement between processor and developer is specified in an entity called `deployment`. The `deployment` specifies *(i)* a set of instructions that are executed on the processor, *(ii)* its scheduling parameters, and *(iii)* settlement configuration (*i.e.,* where the output is further processed or persisted), and *(iv)* finally, the pricing.

5.3. **Compute Costs and Rewards.** When scheduling a deployment, developers define the compute cost for execution in native ACU tokens. This cost is fixed at submission time, enabling deterministic financial planning and accurate budgeting of computational expenses throughout the deployment's duration.

Critically, all compute costs are paid as gas (transaction) fees. There is no direct reward flow from developers to processors. Instead, processors earn rewards through two primary mechanisms that align their incentives with overall network health.

5.3.1. *Base Benchmark Rewards.* Providers receive rewards based on the benchmarks of their participating devices, paid from Acurast's token inflation. A total of 10% of inflation is distributed per epoch (approximately every 1.5 hours) among all compute providers, independent of their participation in staked compute. This baseline structure ensures all processors contributing valid resources receive compensation, regardless of deployment selection during that epoch.

These rewards are split across the four benchmark metric pools, where each device competes based on its specifications. Higher-quality hardware receives proportionally greater rewards, incentivizing providers to contribute performant devices. *C.f.,* Table 1 in Subsection 7.1.

5.3.2. *Staking Rewards.* Compute providers participating in staked compute earn additional rewards by committing compute power and staking tokens. The majority (70%) of Acurast's inflation is distributed as staking rewards every epoch, allocated based on hardware performance, stake size, and commitment duration.

The staking mechanism also serves as a security measure. Processors with staked tokens have financial incentive to behave honestly. Malicious or negligent behavior can result in slashing penalties, creating strong disincentives against attacks or unreliable service.

5.3.3. *Deployment Execution Bonus.* When a processor executes a deployment during an epoch, it receives a bonus weight on its benchmark metrics. This bonus increases the processor's scoring, and therefore its share of rewards, in both the Staked Compute Pool and the Compute Pool for that epoch.

This mechanism ensures processors are incentivized to actively execute deployments rather than passively holding stake. The bonus scales with deployment complexity, appropriately compensating processors handling more demanding workloads.

5.3.4. *Inflation Distribution.* Acurast's annual token inflation is structured to support various stakeholders contributing to network operation:

- **70% → Staked Compute Pool:** Rewards processors providing reliable computational resources.
- **15% → Treasury:** Funds ongoing development, ecosystem grants, and partnerships through governance-directed allocation.
- **10% → Base Benchmark Rewards:** Compensates all providers based on hardware capabilities, regardless of current deployment activity.
- **5% → Block Producers (Collators):** Compensates validators maintaining blockchain infrastructure and producing blocks.

This distribution model aligns incentives across all network participants while ensuring sustainable growth of the compute ecosystem.

5.4. **Scheduling Engine for Liquid Matching.** Acurast developers can retrieve data (*i.e.,* use Acurast as an Oracle framework) and relay the retrieved information to Smart Contracts (or other Web2 targets). Although developers could register an individual `deployment` for each required query, Acurast provides developers with a Scheduling Engine that guarantees a certain level of service.

Given multiple scheduled executions, denoted $\text{EXECUTION}_{i,k}$, as one $\texttt{deployment}_i$, this allows Acurast to plan ahead and ensure that all executions, repeated with interval_i throughout a schedule, fit the processor's capacity (*all-or-nothing* principle). Furthermore, to track the level of service provided by processors and to specify Service Level Agreements (SLA) with the developer at the time of matching `deployments` with processors by feeding the reputation system with reliability metrics.

Additional challenges are imposed on the protocol to achieve reliable scheduling, aligning incentives of all stakeholders correctly so that developers are ensured a high level of service from processors, which would damage their reputation if scheduled executions are not fulfilled. Lastly, processors can obtain commitments to predictable earnings, while the pricing for all executions is agreed on before the start of a `deployment`.

To perform more `deployments` with flexible schedules that suit the processor capacity, Acurast introduces *planning flexibility* for each $\texttt{deployment}_i$, denoted by $\texttt{max_start_delay}_i$. The matchmaker chooses the actual delay and is denoted by $\texttt{start_delay}_i$. Figure 4 sketches the relationship between all parameters of $\texttt{deployment}_i$ and the period within which the fulfillment and subsequent reporting to Acurast are accepted. Outside the acceptance period for $\texttt{report}_{i,1}$ a processor submission is not accepted, leading to a *malus* in the reputation system.

5.5. **Reputation Engine.** Traditionally, reputation systems are used to gauge the trustworthiness of system participants. In Acurast, the trustworthiness and reliability of processors is crucial for the network's execution layer. Although Acurast's zero trust architecture significantly reduces the required trust assumptions, it is still a crucial metric *i.e.,* reputation scores in Acurast indicate the processor's capability to provide the execution layer infrastructure reliably.

Archived copy · cryp2nova.com

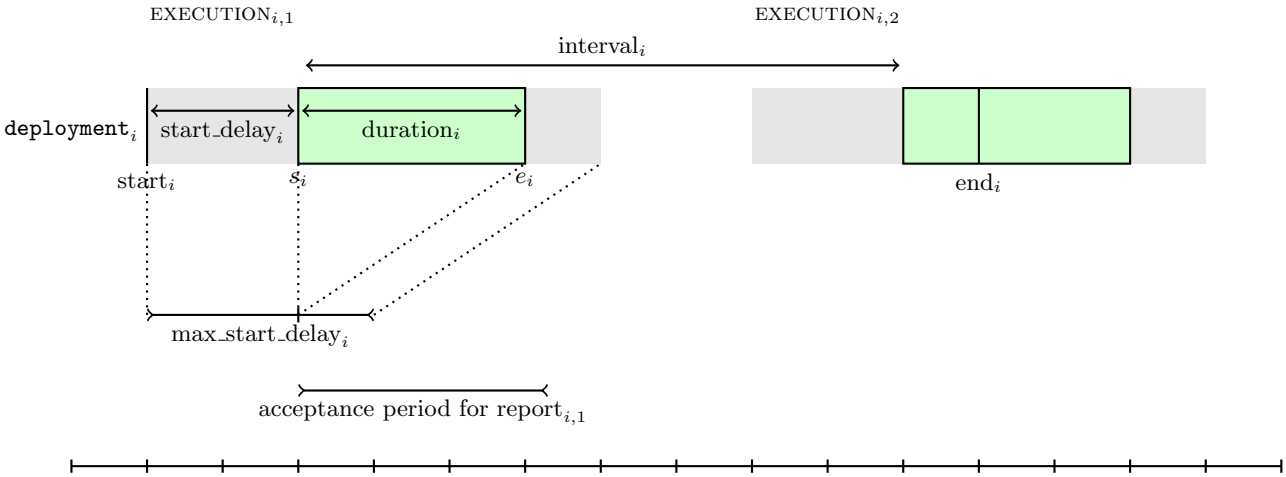


FIGURE 4. `deployment` Execution Scheduling Parameters

The binary input of the reputation engine is automatically obtained from the system parameters [28]. More specifically, a processor's reputation is determined by both successful and unsuccessful `deployment` fulfillment. Its statistical foundation is rooted in the beta Probability Density Function (PDF) (*cf.* Fig. 5), commonly used to represent the *a posteriori* probabilities of binary events. The beta PDF can be expressed in terms of the gamma function Γ as follows [29]:

$$f(p|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1} \quad \text{where } 0 \le p \le 1,\ \alpha > 0,\ \beta > 0 \tag{1}$$

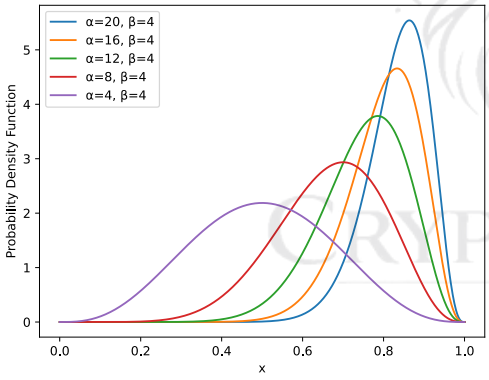


FIGURE 5. Beta Probability Density Functions

Figure 5 displays PDFs of five different beta distributions with varying shape parameters α and β. Given the setting of binary events $\{A, B\}$ Parameters r and s describe the observed frequency of occurrence of A and B, respectively. Thus, α and β can be set as follows to model the PDF as a function of earlier observations:

$$\left.\begin{aligned}\alpha &= r+1\\ \beta &= s+1\end{aligned}\right\} r, s \ge 0 \tag{2}$$

Let f_T represent the total number of favorable ratings observed given to entity T. Similarly, g_T is defined as the total number of ratings given to the same entity [29]. With the knowledge that the probability expectation value of the beta distribution is given as $\mathbb{E}(p) = \frac{\alpha}{\alpha+\beta}$ and applying the definitions of Eq. 2, thus the probability expectation value of our reputation function is defined as

$$Rep(f_T, g_T) = E(f(p|f_T, g_T))) = \frac{f_T+1}{f_T+g_T+2} \tag{3}$$

To account for context awareness, a weighting factor w is introduced, which includes the *transactional value* in the reputation model, thus preventing *value imbalance* attacks [30]. In a reputation update, the weight of the underlying DEPLOYMENT is determined as the ratio between the DEPLOYMENT reward and the difference between the average `deployment` reward (concerning the asset in which the reward is specified; excluding the reward of the `deployment` for which the reputation update is invoked) and the `deployment` reward. Let n denote the number of assigned `deployments`. Then the weighting factor (w_i) for the reputation update for the *ith* assigned `deployment` is defined as

$$w_i = \begin{cases} 1, & \text{if } n = 1 \\ \dfrac{\phi_i}{\frac{1}{n-1}\sum\limits_{\substack{j=1 \\ j\neq i}}^{n} \phi_j + \phi_i}, & \text{if } n > 1 \end{cases} \tag{4}$$

By alternating honest and malicious behavior, an agent can strategically manipulate a reputation system [31]. To counteract this threat event, a discount factor λ [29] is introduced to accurately represent the current reliability levels of the processors in real-time. A λ value of 0 implies that only the last reputation update is considered in the reputation scores, while a λ value of 1 does not have any discounting effects. In the presence of weights and λ, the order of reputation updates is crucial to calculate reputation scores. The upper summation limits refer to the most recent reputation update, and the lower limits refer to the oldest reputation update. Let r_T denote the discounted and weighted favorable reputation updates. Similarly, let s_T be the discounted and weighted total reputation updates.

Archived copy · cryp2nova.com

We can then express r_T and s_T as follows for $0 < \lambda < 1$:

$$r_T = \sum_{i=1}^{n} r_{T,i} w_i \lambda^{n-i}$$
$$s_T = \sum_{i=1}^{n} s_{T,i} w_i \lambda^{n-i} \tag{5}$$

Due to discounting of reputation updates, the maximum possible reputation score is less than in the naive case of the reputation function (*cf.* Equation 3). Given the definition of r_T *cf.* Eq.5, the knowledge that $w_i < 1$ with the fact that the geometric sum $\sum_{i=1}^{\infty} \lambda^i$ can be written as $\frac{1}{1-\lambda}$ for $\lambda < 1$, it follows that

$$\max_{\{r_T\}} = \frac{1}{1-\lambda} \tag{6}$$

Let μ be defined as the maximum value of the reputation function, given r_T, s_T, λ and setting $w = 1$. It should be noted that in practice, w is less than 1 and, on average, 0.5. Using the result of equation 6, we can express μ as

$$\mu = \frac{\frac{1}{1-\lambda} + 1}{\frac{1}{1-\lambda} + 2} < 1 \quad \forall \lambda \in [0, 1) \tag{7}$$

Scaling reputation scores using μ, such that possible reputation scores are $\in (0, 1)$. Equations 3, 5 and 7 are combined to provide the following final version of our reputation formula:

$$Rep(r_T, s_T) = \frac{1}{\mu} \frac{\sum_{i=1}^{n} r_{T,i} w_i \lambda^{n-i} + 1}{\sum_{i=1}^{n} r_{T,i} w_i \lambda^{n-i} + \sum_{i=1}^{n} s_{T,i} w_i \lambda^{n-i} + 2} \tag{8}$$

Under the assumption that arithmetic operations are executed in constant time and that the values r_T and s_T from Equation 4 are stored, the reputation algorithm has a time complexity of $\mathcal{O}(1)$. Values `r` and `s` are persisted in the Substrate runtime storage for each processor. As such, the reputation scores are calculated from `r` and `s`, but are not stored separately. A developer can define the minimum reputation that a processor must have to be eligible to be assigned the specified `deployment` when registering a `deployment` on the market.

5.5.1. *Evaluation.* The Acurast reputation system has been evaluated in [28] using agent-based modeling (ABM) methods. The behavior of processors and developers was specified, and Acurast orchestrator interactions were captured in our model. Three distinct categories of processors were introduced: low, medium and high success rate groups with respect to their probability of fulfilling a standardized `deployment`. Evaluations followed a 200-step model of 300 processors and 900 consumers for ten iterations with values $p = 0.8$, $p = 0.9$ and $p = 0.99$.

Initial orchestrator simulations were performed in the absence of the reputation system. Assuming a standardized `deployment` with both agent types implementing a pricing strategy, the average `deployment` reward was 101.30 units. Although reputation scores were not considered in this scenario, reputation updates were tracked, resulting in Figure 7. The visible difference in rewards is caused by the varying success rates of processor groups and the fact that rewards are only paid for successful fulfillment.

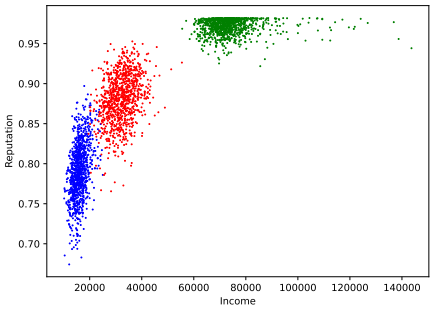


FIGURE 6. Simulation results obtained considering the reputation system in matching the reputation with the processor rewards. Outcomes of 10 iterations of a 200-step model with 3000 processor data points

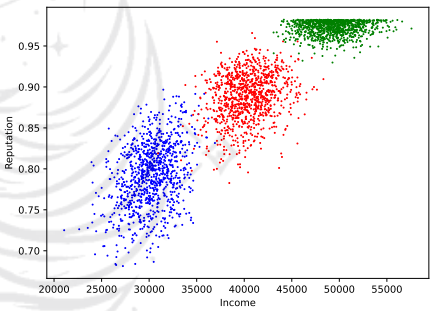


FIGURE 7. Simulation results obtained without the reputation system; reputation vs. processor rewards. Outcomes of 10 iterations of a 200-step model with 3000 processor data points

The effect of the reputation engine on the matching is considered in another simulation scenario. Developers specify a minimum reputation in their `deployment` registrations with $p = 0.5$, with the threshold value randomly drawn from a uniform distribution [0.8, 1].

Figure 6 visualizes the results. Evidently, there is a change in `deployment` allocations with respect to the processor groups, in contrast to the previous scenario. Due to the fact that high-success processors are allocated a larger proportion of total assigned DEPLOYMENTS, the average income for this group increases by 51.77%. However, this occurs at the expense of the other groups, with the medium-success group booking 18.84% less income while the low-success processor's income is diminished by 46%.

The Gini coefficient with respect to processor rewards is closely related to the percentage of `deployments` assigned to low-, medium-, and high-success processors. An important finding is that the presence of the reputation system significantly lowers the observed failure rate of allocated `deployments` by 31.37%, an effect that becomes even more pronounced if a higher percentage of developers demand a minimum reputation.

Archived copy · cryp2nova.com

6. Application Layer and Use Cases

In the Internet of today, almost every application relies heavily on auxiliary systems. Whether external APIs are used for authentication, basic infrastructure (hosting), data availability, reliance, which can benefit from extending or replacing services and core elements with trustless `deployments`, essentially eliminating a host of threat events. The possibilities with Acurast are near-endless since today's centralized Internet is heavily centralized both logically and in terms of trust anchors.

Acurast's execution layer transforms the way applications are designed and deployed, achieving an unparalleled Developer Experience (DevEx) by offering the Acurast Console [32] to developers, where a self-service model allows developers to integrate and develop their applications. Certain use cases can benefit tremendously from certain properties of Acurast, *e.g.,* geological distribution, confidentiality, verifiability.

The following subsections outline potential use-cases that can be deployed through Acurast.

6.1. **Market Intelligence.** Market intelligence involves collecting and analyzing data on a market, allowing businesses to make informed strategic decisions and stay competitive. Of all the Internet traffic in 2022, 47.4% was automated traffic, also commonly referred to bots [33]. Of that automated traffic, 30.2% were bad bots, while good bots are also on the rise, accounting for 17.3%. The percentage of human traffic continues its downward trend, from 57.7% in 2021 to 52.6% in 2022. Bots, in that context do not refer to volumetric Distributed Denial-of-Service attacks, but the bot activity on layer 7 of the OSI model. In general, good bots are important for various business models and applications (*e.g.,* market intelligence), since they are scraping data and feed models for decision making or business logic directly.

With Acurast, the scraping infrastructure can be fully decentralized logically and physically, leveraging the network of processor resources that confidentially execute these tasks, without leaking any data about the querying party. For example, when intelligence is gathered (*e.g.,* for investment or merger decisions), a large amount of data must be scraped confidentially.

6.2. **Artifical Intelligence.** The recent surge of Artificial Intelligence (AI) applications has led to increased research and development in these areas. Although the potential of these technologies is vast, the risks associated with the centralized deployment and privacy of data are crucial to evaluate carefully. In Acurast, the *Singularity* module allows the execution of Artificial Intelligence (AI) in a decentralized and confidential fashion. *E.g.,* Acurast enables Large Language Models (LLM) to be executed in a federated, privacy-preserving, and trustless way [34], *i.e.,* Distributed Llama [35] Another example is Distributed LLama.

6.3. **IoT.** In general terms, the Internet of Things (IoT) refers to interconnected computing devices that form a network and monitor environmental variables (*e.g.,* health caren [36]). Often, due to its limited resources, heterogeneity, and lack of computing power, IoT faces many security and privacy challenges. Data is transferred between IoT devices without human intervention, making zero trust an essential aspect in network and trust management. Depending on the Acurast processor used, built-in Bluetooth modules or WiFi direct connections can be used to collect metrics or data and confidentially process the data.

6.4. **Zero-Knowledge Proof Applications.** Zero-Knowledge Proof protocols find vast application in blockchains [37]. Potential use cases range from anonymous voting systems [38, 39], to secure and privacy-preserving digital assets exchange, secure remote biometric authentication, or Proof-of-Reserves [40].

Acurast can be leveraged in multiple areas of ZKP applications, for instance, to offload high intensity computation in a zero trust manner [41], or to form sub-consortia of processors that can, for instance, mix and generate proofs that the mixing has been performed correctly.

6.4.1. *Privacy-Preserving Mixing.* With ZKPs, privacy mixing can occur in a way that allows transactions to be validated without exposing the details of those transactions. However, the mixing is not limited to Web3 transactions and can be extended to other data sensitive to privacy (*e.g.,* metadata of internet traffic or files metadata).

6.5. **Secure Multi-Party Computation.** Secure Multi-Party Computation (SMPC) is a cryptographic primitive that enables distributed parties to conduct joint computations without revealing their own private inputs and outputs to the computation [42]. For instance, doctors may query a database containing private information, or banks may invest in a fund that must satisfy both banks private constraints. Usually, one trusted entity must know the inputs from all the participants, however, if no Trusted Third Party (TTP) is available or suitable, privacy concerns are evident [42]. With Acurast, a processors can be selected for SMPC algorithms to execute *e.g.,* a permissionless poker game [43].

6.6. **Blockchain Infrastructure.** Blockchain networks' rising adoption and complexity of blockchain networks has led to an increasing need for a reliable blockchain infrastructure. Novel incentive structures (*e.g.,* slashing in PoS) have intensified this further. It is crucial that this infrastructure is neither logically nor physically centralized because it would introduce new trust assumptions that undermine the permissionless nature of blockchains. For these services, Acurast can serve as a decentralized, serverless backend.

6.6.1. *Incorruptible Sequencer.* A huge issue in public blockchains is Blockchain Extractable Value (BEV), and Miner Extractable Value (MEV), where DeFi users are at risk of being attacked [44] (*e.g.,* frontrunning and sandwich attacks [45]). With Acurast, processors can serve as Sequencing-as-a-Service consortia [46], ensuring that the order of transactions is deterministic and immune to external influence.

6.6.2. *Beyond Oracles: Serverless Applications.* Oracles and on-chain automation are key ingredients of blockchain infrastructure. While oracles enable external data to be imported into the blockchain, oracles mainly deal with data retrieval and validation, ensuring that accurate and

Archived copy · cryp2nova.com

reliable data is fed into smart contracts. However, on-chain automation has a broader scope, encompassing automated liquidity provision, periodic settlements, debt restructuring, yield harvesting, and much more. The emphasis here is on action and execution based on specific conditions.

6.6.3. *Native Cross-Chain DeFi.* Native cross-chain DeFi capabilities have been developed to allow seamless interactions and transactions between blockchains, creating a more inclusive and expansive financial ecosystem. Applying Account Abstraction allows for the design of accounts that can interact and integrate across various platforms and protocols, simplifying user experiences, and opening the door for innovative use cases.

6.6.4. *Data Availability as-a-Service.* (DAaaS) provides decentralized storage solutions to ensure data remain accessible and intact, fortifying the robustness of the entire decentralized ecosystem.

7. Tokenomics

The total supply of both `ACU` and `cACU` (Canary Acurast Tokens) is 1 billion (1.000.000.000 tokens) subject to change based on the inflation and burn rate of the token. Details on token distribution and the Token Generation Events (TGE) are documented in [27, 47]. As a decentralized NPoS network, the protocol requires participants to stake (and nominate) to ensure that the protocol achieves its security and finality guarantees. Stakers are incentivized to participate by being rewarded with staking rewards that are derived from the inflation of the Acurast protocol. It is crucial to note that `ACU` tokens (`cACU` respectively) do not confer or represent any direct or indirect equity stake of the Acurast Association and/or the project or any other ownership right, share or equivalent right, voting right, or any other right, contractual or otherwise, to receive future profit shares, intellectual property rights, or any other form of participation.

7.1. **Staked Compute.** The continuity of service and the stability of compute supply are pivotal in long-term operations. Both the compute supply and demand sides require precise incentive alignment. For this reason, Acurast formulates a novel concept called *staked compute*, allowing managers of compute resources to specify how long their compute will be available on the network. In addition, the manager assigns a certain stake to that commitment, ensuring a strong incentive for managers to adhere to their commitment without requiring a central authority. If the manager breaches their commitment, their stake will be reduced, with the penalty in relation to the duration of the breach.

7.1.1. *Benchmark Metrics.* In order to make compute measurable, Acurast introduces four standardized *benchmark metrics* that quantify a device's computational capacity. Each metric is assigned a weight reflecting its importance to the network's computational requirements (*cf.* Tab. 1)

Metric	Weight
CPU Single Core	0.2307
CPU Multi Core	0.2307
RAM	0.4615
Storage	0.0769

Table 1. Benchmark metric weights

These weights define how staking rewards are distributed across four separate *benchmark metric pools*. For each unit of reward emitted, the distribution follows the weights above, with RAM being the most heavily weighted due to its critical role in application performance.

7.1.2. *Current and Committed Compute.* The *current compute* represents the measured compute power across all of a provider's devices for a given epoch. These measurements are conducted automatically during each device heartbeat (every 30 minutes) and reported on-chain.

A compute provider commits to maintaining a percentage of their current compute over the lifetime of their stake. The *committed compute* is capped at 80% of the measured current compute to prevent early slashing due to natural fluctuations. For each benchmark metric p, we denote the committed compute as $m_c^{(p)}$.

7.1.3. *Staking Weight.* The staking weight determines a participant's share of the reward pool. Let s denote stake, τ denote cooldown period, and τ_{max} denote the maximum cooldown (approximately 3.68 years on Mainnet). For committer c:

$$W_c = s_c \times \frac{\tau_c}{\tau_{max}} \tag{9}$$

The aggregate weight of delegators to committer c:

$$W_D = \sum_d s_d \times \frac{\tau_d}{\tau_{max}} \tag{10}$$

7.1.4. *Stake Limits.* To prevent disproportionate stake allocation relative to compute capacity, Acurast introduces a *target weight per compute* for each benchmark metric pool p. Let S denote total supply and $M^{(p)}$ denote total benchmarked metric:

$$T^{(p)} = \frac{0.8 \times S}{M^{(p)}} \tag{11}$$

This parameter expresses the ideal staking rate given the current compute offered across the network.

7.1.5. *Reward Distribution.* Rewards are calculated independently for each benchmark metric pool. The committer's score for pool p:

$$\theta_c^{(p)} = \min\left(W_c + W_D,\ T^{(p)} \times m_c^{(p)}\right) \tag{12}$$

Let $R^{(p)}$ denote total rewards for pool p. The reward share for committer c:

$$r_c^{(p)} = R^{(p)} \times \frac{\theta_c^{(p)}}{\sum_c \theta_c^{(p)}} \tag{13}$$

Archived copy · cryp2nova.com

7.1.6. *Stake Delegations.* Every compute manager can stake and accept stake delegations. The total *commitment stake* consists of the manager's own stake plus all delegated stakes. To avoid centralization, Acurast enforces two limits:

(1) The committer must provide at least $\frac{1}{10}$ of the total commitment stake (maximum 1:9 ratio between own and delegated stake).
(2) Total commitment stake cannot exceed $T^{(p)} \times m_c^{(p)}$ for any pool p.

Committers specify a *delegation fee* (ϕ) upon stake creation, applied when rewards are distributed. The fee can be decreased but never increased. For delegator d with weight W_d, let r_c denote total committer rewards and ψ_d denote any slashing. The delegator reward:

$$r_d = \frac{W_d \cdot (1-\phi) \cdot r_c}{W_c + W_D} - \psi_d \tag{14}$$

7.1.7. *Cooldown Period.* When a committer or delegator initiates unstaking, a cooldown countdown begins. On Mainnet, cooldown periods range from 28 days to approximately 3.68 years. During cooldown:

- Reward weights are reduced to 50%
- Slashing penalties remain at 100%
- Full committed compute must be maintained

This mechanism protects the network from sudden losses of compute capacity.

7.1.8. *Slashing.* When current compute falls below committed compute in any metric during an epoch, slashing applies. Let $\rho = 0.003424657534\%$ be the maximum slash rate per epoch, $w^{(p)}$ the metric weight, and $\delta^{(p)}$ the shortfall ratio:

$$\delta^{(p)} = \frac{m^{(p)}_{committed} - m^{(p)}_{current}}{m^{(p)}_{committed}} \tag{15}$$

$$\psi^{(p)} = s \times \rho \times w^{(p)} \times \delta^{(p)} \tag{16}$$

The total slash across all metrics cannot exceed $\rho \times s$ per epoch. Of slashed tokens, 10% goes to the slasher; 90% is burned. Delegators share slashing risk proportionally but may redelegate to a different committer at any time. Any updates to the Staked Compute mechanism will be documented in the official documentation [47].

8. Summary

Acurast is a novel decentralized serverless global-scale cloud built on open source. The execution layer is formed by a distributed backbone of mobile devices. With superior hardware and performance per watt, ARM-based SoCs offer a better alternative to centralized clouds. Acurast follows a modular architecture that separates the consensus, execution, and application layer.

In the consensus layer, the orchestrator contains a liquid matching engine, which ensures that the supply (*i.e.,* processors) and the demand for computational resources (*i.e.,* consumers) match quickly. Furthermore, a custom-made reputation engine ensures reliability and incentivizes the genuine behavior of processors. Finally, the attestation service ensures that processors provide the Acurast TEE in a fully end-to-end verifiable manner.

Acurast's execution layer offers the Acurast Trusted Execution Environment (ATEE), ascertaining verifiable and confidential compute, drastically reducing trust assumptions. Furthermore, various runtimes offer multiple native `runtimes` that are continuously extended, *e.g.,* Node.js, JavaScript v8, and Python.

In summary, Acurast processors offer a flexible solution to run `deployments` tailored to application-specific requirements. Acurast presents a paradigm shift of verifiable compute, where trustless transfers across ecosystems are enabled through robust decentralized computational infrastructure, combining the benefits of confidentiality and censorship-resistance. The applicability of Acurast is broad, spanning from blockchain-specific utilities to distributed trust mechanisms in federated learning, and confidential computing.

Acknowledgements

The authors thank Rodrigo Quelhas, Matthias Tarasiewicz, Maximilian Fries, Ralph Pichler, Jake Blew, and Jan von der Assen for their input.

References

[1] J. Feigenbaum and J. Koenig, "On the Feasibility of a Technological Response to the Surveillance Morass," vol. 8809, 03 2014, pp. 239–252.
[2] J. Lanier, *Who Owns the Future?* New York: Simon & Schuster, 2014.
[3] B. Schneier, "When It Comes to Security, We're Back to Feudalism," Cryptology ePrint Archive, Paper 2021/979, 2012. [Online]. Available: https://www.schneier.com/essay-406.html
[4] T. N. Rahman, N. Khan, and Z. I. Zaman, "Redefining computing: Rise of arm from consumer to cloud for energy efficiency," *World Journal of Advanced Research and Reviews*, vol. 21, no. 1, p. 817–835, Jan. 2024. [Online]. Available: http://dx.doi.org/10.30574/wjarr.2024.21.1.0017
[5] M. Tarasiewicz and A. Newman, "Chapter 10 - cryptocurrencies as distributed community experiments," in *Handbook of Digital Currency*, D. Lee Kuo Chuen, Ed. San Diego: Academic Press, 2015, pp. 201–222. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128021170000102
[6] G. Wood, "Join-accumulate machine: A semi-coherent scalable trustless vm," Draft 0.2.0 Tokyo, 2024.
[7] F. Mogavero, I. Visconti, A. Vitaletti, and M. Zecchini, "The blockchain quadrilemma: When also computational effectiveness matters," in *2021 IEEE Symposium on Computers and Communications (ISCC)*, 2021, pp. 1–6.
[8] Chainalysis Team, "Vulnerabilities in cross-chain bridge protocols emerge as top security risk," https://blog.chainalysis.com/reports/cross-chain-bridge-hacks-2022/, August, 2022.
[9] B. Betz, "Amazon expands web3 reach with cloud tools that help blockchain developers," https://www.coindesk.com/business/2023/07/27/amazon-expands-managed-blockchain-services/, 2023.
[10] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "ZEXE: Enabling Decentralized Private Computation," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 947–964.
[11] G. Greenwald, *No Place to Hide: Edward Snowden, the NSA, and the U.S. Surveillance State*. Metropolitan Books, 2014.
[12] J. von der Assen, C. Killer, A. D. Carli, and B. Stiller, "Performance analysis of decentralized physical infrastructure networks and centralized clouds," 2024. [Online]. Available: https://arxiv.org/abs/2404.08306
[13] S. K. Lo, Y. Liu, G. Yu, Q. Lu, X. Xu, and L. Zhu, "Distributed Trust Through the Lens of Software Architecture," 2023.
[14] C. Diaz, H. Halpin, and A. Kiayias, "The Nym Network: The Next Generation of Privacy Infrastructure," 2021, version 1.0.
[15] M. Theis, C. Douglas, and V. Srivatsa, "Mixclaves: Enclave-Based Mixnets," 2022, cS262a Fall 2022, Advanced Topics in Computer Systems, UC Berkeley.
[16] P. T. Maxime Rossi Bellom, Damiano Melotti, "2021: A Titan M Odyssey," 2021.

Archived copy · cryp2nova.com

[17] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. K. Alper, X. Luo, F. Shirazi, A. Stewart, and G. Wood, "Overview of Polkadot and its Design Considerations," 2020.

[18] C. Wankhede, "What is the Titan M2 security chip in Google's Pixel phones?" https://www.androidauthority.com/titan-m2-google-3261547/, Jan 2023.

[19] J. Reed, "Google's bug bounty hits $12 million: What about the risks?" https://securityintelligence.com/news/googles-bug-bounty-hits-12-million-what-about-the-risks-2/, May 2023.

[20] ZERODIUM, "ZERODIUM Payouts for Mobiles," https://zerodium.com/program.html.

[21] S. van Schaik, A. Kwong, and D. Genkin, "SGAxe: How SGX Fails in Practice," 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:220248073

[22] S. van Schaik, A. Seto, T. Yurek, A. Batori, B. AlBassam, C. Garman, D. Genkin, A. Miller, E. Ronen, and Y. Yarom, "SoK: SGX. Fail: How stuff get eXposed," 2022.

[23] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A Comprehensive Survey," *ACM Comput. Surv.*, vol. 51, no. 6, jan 2019. [Online]. Available: https://doi.org/10.1145/3291047

[24] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro, "Mina: Decentralized Cryptocurrency at Scale," 2020.

[25] N. Franzese, J. Katz, S. Lu, R. Ostrovsky, X. Wang, and C. Weng, "Constant-overhead zero-knowledge for RAM programs," Cryptology ePrint Archive, Paper 2021/979, 2021, https://eprint.iacr.org/2021/979. [Online]. Available: https://eprint.iacr.org/2021/979

[26] X. Tang, L. Shi, X. Wang, K. Charbonnet, S. Tang, and S. Sun, "Zero-knowledge proof vulnerability analysis and security auditing," Cryptology ePrint Archive, Paper 2024/514, 2024, https://eprint.iacr.org/2024/514. [Online]. Available: https://eprint.iacr.org/2024/514

[27] Acurast Association, "Acurast Tokenomics," 2023.

[28] A. V. Charle, "Decentralized Reputation in Blockchain Oracle," Master's thesis, 2023.

[29] A. Jøsang and R. Ismail, "The Beta Reputation System," in *Proceedings of the 15th bled electronic commerce conference*, vol. 5. Citeseer, 2002, pp. 2502–2511.

[30] Y. Yao, S. Ruohomaa, and F. Xu, "Addressing Common Vulnerabilities of Reputation Systems for Electronic Commerce," *Journal of theoretical and applied electronic commerce research*, vol. 7, no. 1, pp. 1–20, 2012.

[31] M. Srivatsa, L. Xiong, and L. Liu, "TrustGuard: countering vulnerabilities in reputation management for decentralized overlay networks," in *Proceedings of the 14th international conference on World Wide Web*, 2005, pp. 422–431.

[32] Acurast Association, "Acurast Console," https://console.acurast.com/, 2023.

[33] Imperva, "Imperva 2023 Bad Bot Report," https://www.imperva.com/resources/resource-library/reports/2023-imperva-bad-bot-report/, May, 2023.

[34] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated Learning for Internet of Things: A Comprehensive Survey," *IEEE Communications Surveys and Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.

[35] b4rtaz (Bart Tadych), "Distributed Llama," 2021. [Online]. Available: https://github.com/b4rtaz/distributed-llama

[36] S. B. Baker, W. Xiang, and I. Atkinson, "Internet of things for smart healthcare: Technologies, challenges, and opportunities," *IEEE Access*, vol. 5, pp. 26 521–26 544, 2017.

[37] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie, and X. Peng, "A survey on zero-knowledge proof in blockchain," *IEEE Network*, vol. 35, no. 4, pp. 198–205, 2021.

[38] C. Killer, M. Eck, B. Rodrigues, J. von der Assen, R. Staubli, and B. Stiller, "ProvotuMN: Decentralized, Mix-Net-based, and Receipt-free Voting System," in *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2022, pp. 1–9.

[39] C. Killer, B. Rodrigues, E. J. Scheid, M. F. Franco, and B. Stiller, "Blockchain-based Voting Considered Harmful?" *IEEE Transactions on Network and Service Management*, pp. 1–1, 2022.

[40] A. Dutta and S. Vijayakumaran, "Mprove: A proof of reserves protocol for monero exchanges," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2019, pp. 330–339.

[41] N. Ni and Y. Zhu, "Enabling Zero Knowledge Proof by Accelerating zk-SNARK Kernels on GPU," *Journal of Parallel and Distributed Computing*, vol. 173, pp. 20–31, 2023.

[42] W. Du and M. J. Atallah, "Secure Multi-Party Computation Problems and Their Applications: A Review and Open Problems," in *Proceedings of the 2001 Workshop on New Security Paradigms*, ser. NSPW '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 13–22. [Online]. Available: https://doi.org/10.1145/508171.508174

[43] R. Kumaresan, T. Moran, and I. Bentov, "How to use bitcoin to play decentralized poker," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 195–206. [Online]. Available: https://doi.org/10.1145/2810103.2813712

[44] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, K. Qin, R. Wattenhofer, D. Song, and A. Gervais, "SoK: Decentralized Finance (DeFi) Attacks," 2023.

[45] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges," 2019.

[46] 0xTariz (Tariz), "Shared Sequencer for MEV protection and Profitable Marketplace," 2023. [Online]. Available: https://rebrand.ly/sharedsequencer

[47] Acurast Association, "Acurast Docs," https://docs.acurast.com, 2023.

All links above were last accessed on January 8th, 2026.

CRYP2NOVA

Archived copy · cryp2nova.com