PRIVATON

Privacy Preserving Automaton for Proof of Computations

Bala Subramanyan
Research, Credora Inc
San Rafael, California, USA
Email: bala@credora.io

Abstract

Amid the landscape of confidential computing, where security and privacy reign supreme, PRIVATON emerges as a pioneering and practical solution to safeguard sensitive data and computations. A verifiable proof of computation model, with one of its variant built upon the dual sandbox strategy, PRIVATON combines Trusted Execution Environment (TEE) technologies with WebAssembly (WASM) runtime environments to establish an ecosystem for privacy-preserving computations. This approach involves fine grained modeling of computations as finite state automatons, guided by verifiable proofs that attest to their unerring execution.

PRIVATON is guided by the profound principles of "least privilege" and "intentional use." Through the former, each computation module's privileges are meticulously constrained, reducing vulnerability vectors. The latter ensures that privileges are allocated explicitly, fostering comprehension and security. This rigorous adherence minimizes privilege misuse and information leakage, bolstering the overall security posture.

At its core, PRIVATON's innovation lies in its comprehensive assurance of data privacy and security. State machine proofs not only attest to the absence of data leakage but also prevent unauthorized data transmission. By providing unassailable proof of computation integrity, PRIVATON shields against code misuse within the system. This proactive stance fortifies its mission to safeguard the sanctity of data, computations, and the privacy of all stakeholders.

Evidenced by its real-world application, PRIVATON has been validated within the cryptocurrency trading ecosystem, where it acts as a distributed and privacy-preserving credit oracle. Its implementation within Credora's landscape underlines its potential to transform data-centric paradigms, empowering stakeholders with an unshakeable confidence in data security. In a world where data privacy is paramount, PRIVATON stands as a guardian, epitomizing the convergence of technology, security, and trust.

Keywords: Confidential Computing, Trusted Execution Environment (TEE), WebAssembly (Wasm), Finite State Automata, Privacy-Preserving Computation, Verifiable Proofs, Dual Sandbox Strategy, Proof of Computations, Trusted Third Parties, Data Privacy.

Introduction

In the modern landscape of data-centric computing, ensuring the privacy and security of sensitive information has become a paramount concern. Among the three states of data security, data-in-use poses a unique challenge, as it necessitates decryption for processing, exposing it to potential breaches. In response to this critical issue, the concept of confidential computing has emerged, aiming to safeguard data while it is being utilized. Confidential Computing Consortium has united hardware vendors, cloud providers, and software developers to advance privacy-preserving computing, with hardware-assisted Trusted Execution Environment (TEE) as a prominent approach.

TEE offers the promise of secure execution environments where users can deploy their sensitive code and data, ensuring their protection even against malicious platform providers. However, real-world scenarios often involve third-party TEE providers, introducing additional layers of trust. This scenario raises concerns about whether sensitive data might be accessed illicitly, even by TEE service providers. Although existing efforts have explored the idea of securing applications against malicious TEE service providers, they fall short in providing formal verification and guarantees of data security.

In this backdrop, PRIVATON (PRIVacy-preserving automATON) emerges as a novel solution that tackles these challenges head-on. At Credora Inc., PRIVATON has been harnessed to deliver a provably private and neutral credit risk oracle for cryptocurrency trading firms. This endeavor involves a network of TEE nodes, establishing a decentralized system capable of performing computations on sensitive information in a completely private and predefined manner. The objective is to create a secure and private environment where data owners need not solely rely on trust.

In this paper, we present PRIVATON as a groundbreaking approach to verifiable privacy-preserving computations, underpinned by the concept of finite state automatons. By seamlessly integrating Trusted Execution Environment (TEE) technologies and WebAssembly (WASM) runtime environments, PRIVATON introduces a dual sandbox strategy. This innovative architecture has a profound impact on data privacy and computation integrity. PRIVATON empowers developers and users to define a controlled set of privileges via verifiable policies, facilitating explicit control over computation capabilities.

Throughout this paper, we delve into the inner workings of PRIVATON, highlighting its capability to provide a verifiable proof of computation model. By embracing dual sandbox environments and cryptographic techniques, PRIVATON ensures that computations are executed exactly as intended, with the added assurance of data confidentiality. We explore the model's versatility in accommodating diverse computational paradigms, including TEE-WASM integration. Additionally, we present a comprehensive evaluation of PRIVATON's security and performance aspects, demonstrating its efficacy in preserving data confidentiality and computational accuracy.

The subsequent sections delve into related research, enabling technologies, the proposed PRIVATON solution, real-world use cases, experimental evaluations, and concluding remarks, showcasing PRIVATON's transformative potential in securing sensitive data and computation.

Related Work

In the pursuit of preserving data privacy and ensuring computation integrity, various research and practical endeavors have paved the way for solutions that resonate with the goals of PRIVATON. This section explores notable related work in the realm of confidential computing and verifiable computation.

3.1 Trusted Execution Environments (TEE): TEEs have garnered substantial attention as a hardware-based approach to ensuring secure execution environments. Intel Software Guard Extensions (SGX) [5] and ARM TrustZone [6] are prominent TEE technologies that facilitate the creation of isolated enclaves for computations, shielding them from potential adversaries. However, the reliance on TEE service providers introduces challenges concerning trust and data security.

3.2 Zero-Knowledge Proofs (ZKPs): Zero-knowledge proofs have emerged as a powerful cryptographic technique for verifying computations without revealing sensitive input data. zk-SNARKs and zk-STARKs [8] stand out as examples of ZKPs. While these methods ensure privacy, they often come with substantial overhead and complexities related to setup and verification.

3.3 Finite State Automatons and State Machines: Finite state automatons have been employed in various domains to model processes and systems with distinct states and transitions. These models provide a structured approach to computation and facilitate formal analysis. State machines and automata-based frameworks have been used to reason about system behavior and correctness, providing a strong foundation for PRIVATON's state-based approach.

3.4 WebAssembly (WASM) Runtimes: WebAssembly has gained prominence as a versatile platform-independent runtime environment. It allows for the execution of programs written in multiple programming languages, enhancing portability and security. The integration of WASM runtimes within PRIVATON highlights its adaptability to various computational paradigms.

3.5 Confidential Computing Consortium (CCC): The CCC has brought together industry leaders to address challenges related to confidential computing. While the consortium focuses on TEE-based solutions, PRIVATON extends the conversation by incorporating the dual sandbox strategy and finite state automatons for verifiable computation.

3.6 Dual Sandbox Strategy and Verifiable Proofs: Prior research has explored the idea of securing applications against malicious TEE service providers by employing a dual sandbox strategy. PRIVATON extends this approach by introducing the concept of verifiable proofs of computation. These proofs guarantee the accuracy and integrity of computations and eliminate the need for blind trust in TEE service providers.

3.7 Decentralized Computation and Privacy-Preserving Oracles: The blockchain domain has witnessed the emergence of decentralized computation platforms and privacy-preserving oracles. These platforms aim to enable secure and private execution of smart contracts and computations within a decentralized ecosystem. PRIVATON aligns with these objectives by offering a similar vision of secure computation in a decentralized setting.

In summary, the related work showcases a rich landscape of efforts dedicated to preserving data privacy, enhancing computation integrity, and securing sensitive information. PRIVATON distinguishes itself by introducing a novel approach that combines dual sandbox strategies, finite state automatons, verifiable proofs of computation, and the flexibility of WebAssembly runtimes, offering a comprehensive solution for privacy-preserving computations across diverse contexts.

Background

In the rapidly evolving landscape of modern computing, where the triumvirate of security, privacy, and data integrity stands as the cornerstone, new paradigms have emerged to address the complex challenges posed by remote processing and data utilization. This section delves into the multifaceted background that has culminated in the conception of PRIVATON—an innovative solution aimed at ushering in a new era of privacy-preserving computations.

4.1 Finite State Automata and State Machines: Establishing Computational Rigor

Finite State Automata (FSA) and state machines, foundational constructs within theoretical computer science, provide a formal framework to model sequential processes and transitions. A finite state machine is defined as a tuple $(Q, \Sigma, \delta, q_0, F)$, where:

- Q represents a finite set of states.
- Σ denotes a finite input alphabet.
- δ signifies the transition function, mapping (state, input) pairs to the next state: $\delta: Q \times \Sigma \rightarrow Q$.
- q_0 is the initial state.
- F constitutes the set of accepting states.

Mathematically, a finite state machine is represented as:

$$\mathbf{M=(Q,\Sigma,\delta,q_0,F)}$$

The transition function δ encapsulates the behavior of the state machine, dictating how it moves from one state to another upon receiving specific inputs. The behavior of a state machine can be summarized through its state transition diagram or a transition table.

In a state transition diagram, states are represented as nodes, while transitions are depicted as directed edges labeled with corresponding input symbols. This graphical representation provides an intuitive visualization of the state machine's behavior.

Theorem 1: Determinism of Finite State Machines
A finite state machine is deterministic if, for every state q and input symbol a, there exists exactly one transition δ(q, a) leading to another state. In other words:

$$|\delta(q,a)| = 1 \text{ for all } q \in Q, a \in \Sigma$$

Theorem 2: Equivalence of Finite State Machines
Two finite state machines $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ are equivalent if their languages $L(M_1)$ and $L(M_2)$ are the same, i.e.,

$$L(M_1) = L(M_2)$$

Where the language of a finite state machine M, denoted as L(M), comprises all strings that lead M from the initial state to an accepting state.

Theorem 3: Closure Properties of Finite State Machines
Finite state machines exhibit various closure properties, enabling operations like union, concatenation, and Kleene closure. Given two finite state machines M_1 and M_2:

1. Union: The union of M_1 and M_2, denoted as $M_1 \cup M_2$, recognizes the language union of $L(M_1)$ and $L(M_2)$.
2. Concatenation: The concatenation of M_1 and M_2, denoted as M_1M_2, recognizes the language concatenation of $L(M_1)$ and $L(M_2)$.
3. Kleene Closure: The Kleene closure of M_1, denoted as M_1^*, recognizes the language containing all possible concatenations of strings from $L(M_1)$.

These theorems underpin the rigorous mathematical foundation of finite state machines, enabling their formal analysis, composition, and manipulation. By integrating these mathematical constructs and principles, PRIVATON leverages the versatility of finite state automatons and state machine proofs to engender a robust platform for privacy-preserving computations, ensuring the integrity and security of the executed processes.

4.2 Trusted Execution Environments (TEE): Fortifying Security

Within this backdrop, the concept of Trusted Execution Environments (TEE) has emerged as a pivotal solution for enhancing computational security. TEEs offer isolated execution environments within a processor, commonly isolated from the host operating system. This isolation assures that the execution remains tamper-resistant and confidential, impervious even to privileged attackers. In essence, TEEs provide a bastion of security for computations, insulating them from external threats and vulnerabilities.

4.3 WebAssembly (WASM): Bridging Computational Boundaries

In parallel, the rise of WebAssembly (WASM) has transformed the way software is executed across different platforms. WASM is a binary instruction format that enables efficient and secure execution of code on various environments, fostering compatibility across diverse systems. Its sandboxed execution model guarantees isolation, preventing unauthorized access and enabling cross-platform deployment without compromising security.

4.4 Remote Attestation (RA): Trust in Untrusted Environments

Nevertheless, the utilization of TEEs also brings forth challenges, particularly in the context of verifying the integrity and authenticity of the executed code within these trusted enclaves. Remote Attestation (RA) emerged as a solution to this predicament, enabling a remote party to verify the integrity of the TEE's state and code execution. RA assures that the TEE operates within its expected parameters, safeguarding against potentially malicious or compromised environments.

4.5 The Quest for Privacy-Preserving Computations

Within this intricate milieu of concepts and technologies, PRIVATON crystallized as a visionary solution to the perennial dilemma of executing computations while ensuring utmost privacy and security. PRIVATON leverages a dual sandbox strategy, seamlessly combining the power of TEE technologies with WebAssembly (WASM) runtime environments. This strategic amalgamation establishes an unparalleled stronghold, where the sanctity of computations is guaranteed through meticulous modeling, verifiable proofs, and controlled execution.

The subsequent sections of this paper delve into the details of PRIVATON's architecture, its integration of finite state automatons and state machine proofs, the dual sandbox approach, and the comprehensive evaluation that underscores its efficacy. Through this exploration, PRIVATON emerges as a formidable contender in the realm of confidential computing, bridging theoretical foundations with cutting-edge technologies to pave the way for secure, private, and trustworthy computations in an ever-connected world.

Current Limitations

Within the landscape of confidential computing, the prevailing approaches, including those leveraging Intel SGX and WebAssembly sandboxing techniques, exhibit a range of limitations that underscore the need for more robust solutions. These limitations encompass various aspects of security, privacy, and operational integrity. In this section, we shed light on the critical limitations inherent in the current models and elucidate the challenges that the PRIVATON framework is designed to overcome.

- Enclave Malleability: Current confidential computing frameworks, such as those relying on Intel SGX, suffer from enclave malleability. Enclaves can be exited through different paths, including ECALLs, OCALLs, and AEX. This variability in exit paths can enable attackers to manipulate the internal state of the enclave, compromising its integrity and raising concerns about the security guarantees offered.

- Knowledge Extraction Vulnerability: Zero-Knowledge Proof-of-Knowledge (ZKPK) protocols, while promising for ensuring data privacy, introduce a potential knowledge extraction vulnerability. Within SGX enclaves, AEX can interrupt execution, potentially allowing adversaries to force enclave execution paths that inadvertently reveal secret keys. This vulnerability poses a challenge to maintaining the confidentiality of sensitive information.
- Remote Attestation Security Vulnerabilities: The process of remote attestation, vital for verifying the integrity of enclave execution, faces security vulnerabilities such as Man-In-The-Middle (MITM) attacks. These attacks encompass replay attacks, where malicious entities replay old measurement values and TPM Quotes to impersonate a valid platform, tampering with measurements and TPM Quotes, and masquerading, where attackers present measurement and TPM Quote data from another legitimate system.
- Type Confusion and Module Integrity in WebAssembly: WebAssembly's utilization of SharedArrayBuffer introduces vulnerabilities, including type confusion, in multi-threaded environments. This can result in classic type-confusion exploit chains that potentially compromise security. Additionally, WebAssembly implementations have faced issues related to module integrity, including vulnerabilities like use-after-free, double-free, and buffer overflows, leading to data leakage and potential exploitation.
- Limitations of zkVM: Zero-knowledge virtual machines (zkVM) offer a paradigm shift in preserving data privacy and integrity; however, they are not immune to certain limitations and challenges:
    - Trusted Setup Complexity: zkVM implementations often require a trusted setup phase, where initial parameters are generated to ensure the system's security. This setup phase can be complex and time-consuming, potentially raising concerns about the security of the initial parameters and the overall reliability of the system.
    - Performance Overhead: While zkVMs provide strong privacy guarantees, they can introduce computational overhead due to the complex zero-knowledge proof generation and verification processes. This overhead can impact the system's responsiveness and efficiency, especially in real-time or high-throughput scenarios.
    - Proof Size and Verification Complexity: Zero-knowledge proofs generated by zkVMs can be substantial in size, leading to increased communication and storage requirements. The verification of these proofs can also be computationally intensive, posing challenges in scenarios where rapid verification is essential.
    - Trade-off between Proof Efficiency and Verification Complexity: Striking a balance between efficient proof generation and manageable verification complexity is a challenging endeavor. While optimizing proof size and generation time can improve performance, it may also increase the complexity of verifying these proofs, potentially hampering scalability.
    - Application Integration Challenges: Integrating zkVMs into existing applications or systems can be intricate. Developers need to adapt their code to work within the constraints and requirements of zkVMs, potentially leading to code modifications and challenges in maintaining compatibility.
    - Dynamic State Changes: zkVMs may face challenges in handling dynamic state changes efficiently. Ensuring that proofs remain valid and accurate when state transitions occur

dynamically within a computation can be complex, particularly in scenarios with frequent updates or modifications.

- Trust in Setup Parameters: The initial trusted setup parameters used in zkVM implementations are critical for the security of the entire system. Ensuring the trustworthiness of these parameters is paramount, as any compromise or manipulation could compromise the privacy guarantees of the zkVM.

The limitations highlighted above underscore the pressing need for an innovative solution that addresses the gaps and vulnerabilities within the current models. PRIVATON, with its verifiable proof of computation paradigm, stands as a promising approach to overcome these challenges. In the subsequent sections, we delve into the construction, principles, and capabilities of PRIVATON, presenting a comprehensive solution that aspires to redefine the landscape of confidential computing.

PRIVATON - The Proposed solution

In this section, we introduce PRIVATON, a cutting-edge solution designed to address the critical challenges of privacy-preserving computations in the realm of confidential computing. PRIVATON is a verifiable proof of computation model that leverages state-of-the-art technologies, including Trusted Execution Environments (TEEs) and WebAssembly (WASM) runtime environments, to establish a secure and trustworthy platform for executing computations while safeguarding sensitive data.

6.1 Finite State Automaton Modeling

PRIVATON introduces a novel approach to modeling computations using finite state automatons. Each computation is broken down into a series of states, each with specific privileges, permissions, and capabilities. This approach adheres to the "principle of least privilege" and the "principle of intentional use," minimizing the privileges accessible to each computation module and explicitly allocating privileges to prevent arbitrary allocations.

6.2 Verifiable Proof of Computation

One of the most groundbreaking features of PRIVATON is its ability to generate verifiable proofs of computation. Each computation's execution is accompanied by a verifiable proof that attests to the fact that it was executed exactly as specified within the given execution runtime. This proof ensures that no unauthorized data leakage or transmission occurs during the computation process, thus preserving the privacy and security of sensitive information.

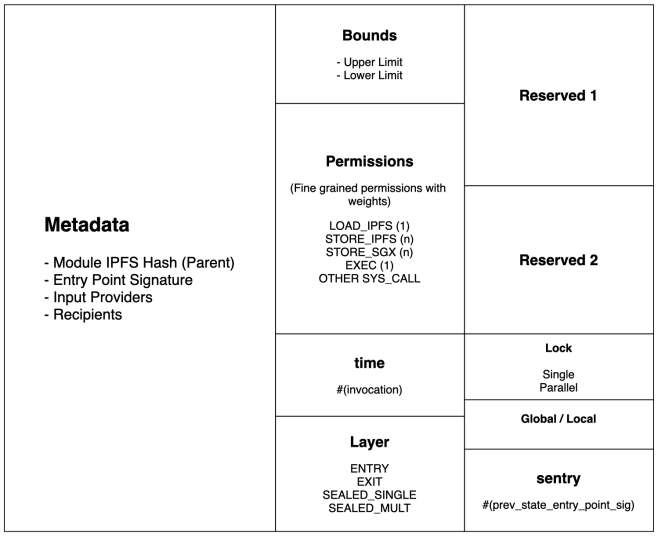
6.3 State Capability Model

PRIVATON implements a robust state capability model that defines the boundaries, permissions, and capabilities associated with each computation state. This model is crucial for enforcing security constraints and maintaining controlled access to computational resources. The state capability model encompasses attributes such as module IPFS hash, entry point signature, input providers, recipients, bounds, permissions, layers, sentry values, and lock attributes.

6.4 Verifiable Policy

Verifiable Policies (VPs) are integral components of the PRIVATON solution, designed to ensure secure and controlled execution of computations within the system. These capabilities, infused with the principles of fine-grained control and explicit permissions, form a robust foundation for secure and verifiable computations.

A Verifiable Policy (VP) is a structured representation that outlines the permissible behaviors of a computation module within a given computational environment. It defines the set of allowable actions and interactions that the module can perform, along with the conditions under which these actions are permitted. The primary purpose of a VP is to ensure that the execution of a computation adheres to a predefined set of rules, thereby enabling verification that the computation has been carried out as intended. This section delves into each capability, elucidating their significance and role within the VP state model.



State Capability Model

Capability Model Attributes

- Module IPFS Hash (Parent): A cryptographic hash of the parent module's IPFS (InterPlanetary File System) address establishes a secure linkage between the module and its origin.

- Entry Point Signature: The unique cryptographic signature of the entry point function enhances identification and access control.
- Input Providers: A comprehensive list of functions that serve as data input providers ensures that only trusted sources feed information into computations.
- Recipients: Trusted recipients are explicitly defined to receive the output of computations, preventing unintended data leakage.
- Bounds: Specify the maximum permissible size of the code and memory, preventing potential exploits and resource abuse.
- Fine-Grained Permissions: Permissions are tailored to each function's state requirements, offering distinct capabilities like loading data from IPFS, storing to IPFS, accessing SGX memory, executing, and other system calls granularly for each state.
- Layer: The layer attribute encapsulates the specific execution phase, encompassing ENTRY, EXIT, SEALED_SINGLE, and SEALED_MULT stages. This attribute governs the contextual access rights for each function.
- Sentry: The sentry attribute ensures secure invocation by including the previous state's entry point signature. This safeguard guarantees that only authorized invocations are accepted.
- Lock: Lock attributes categorize execution states as Single or Parallel, dictating whether the computation is confined to a single-threaded or multithreaded execution environment.

These policies are constructed based on the information partially extracted from the WebAssembly (WASM) debug dump using the Debugging with Attributed Record Formats (DWARF) conventions. These attributes collectively represent the sequence of operations and steps that a computation module can perform.

The DWARF conventions serve as a standardized format for encoding debugging information, including the structure of stack frames in the source code. When a computation module is compiled into WASM code, the associated DWARF debug dump contains metadata that describes the composition of the module's stack frames. The script provided utilizes DWARF conventions to accurately extract this stack frame information.

At a high level, the process of constructing VPs using DWARF conventions involves the following steps:

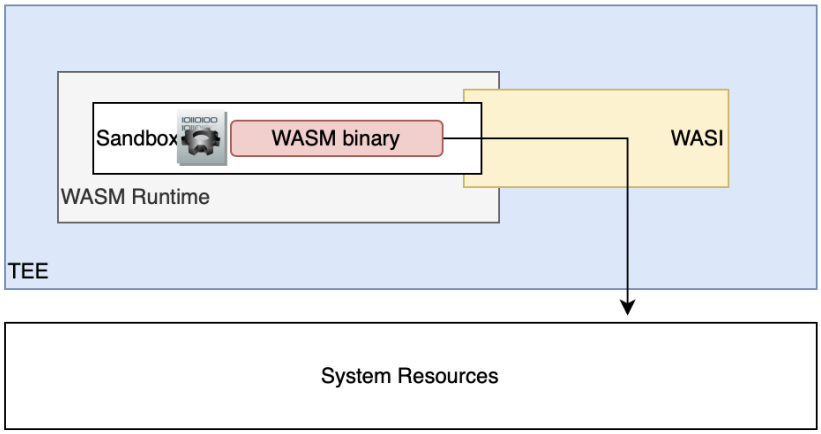
- Detection of Tagged Entities: The script scans through the debug dump to detect tagged entities that signify the beginning of stack frame information.
- Attribute Extraction: Relevant attributes within the tagged entities are extracted. These attributes include information about origin, linkage name, and program counter.
- Information Sequencing: Extracted attributes are processed in a sequence that adheres to DWARF conventions. This sequence captures essential details about each stack frame.

These policies serve as blueprints for controlling the behavior of computations, ensuring security, privacy, and compliance within the PRIVATON framework. The integration of these attributes within the VP state model facilitates fine-tuned control, safeguarding against data leaks, unauthorized access, and resource exploitation. The inclusion of cryptographic hashes, cryptographic signatures, and explicit recipient definitions bolsters data

integrity and privacy. The allocation of precise permissions and their associated weights ensures responsible resource utilization. The layer, sentry, and lock attributes enhance contextual security and control, enabling computations to be securely orchestrated.

6.5 Support for TEE-WASM Variants

PRIVATON's innovative approach leverages the power of Trusted Execution Environment (TEE) technologies with the versatility of WebAssembly (WASM) runtime environments in a unified dual sandbox strategy. This integration also caters to both TEE-backed and standalone WASM environments, ensuring robust privacy-preserving computations in various execution scenarios.



6.5.1 TEE-WASM Synergy

In the TEE-WASM variant, PRIVATON capitalizes on the secure execution environment provided by TEEs, such as Intel Software Guard Extensions (SGX). TEEs establish isolated enclaves where computations can be executed with utmost confidentiality and integrity. Within this TEE-backed environment, PRIVATON enforces its dual sandbox strategy, compartmentalizing computations and their associated resources to prevent unauthorized access and data leakage.

6.5.2 WASM Sandbox Adaptation

In the standalone WASM variant, PRIVATON adapts its dual sandbox strategy to the native features of WASM runtime environments. WASM sandboxes enable secure execution of code, but they lack the inherent isolation of TEEs. PRIVATON's approach introduces additional layers of isolation by defining boundaries for computations, managing permissions, and enforcing verifiable policy state capabilities within the WASM context. This ensures that even in a non-TEE environment, computations are shielded against external threats and unauthorized data exposure.

6.5.3 Unified Privacy-Preserving Execution

The integration of TEE and WASM technologies in PRIVATON's dual sandbox strategy offers a unified platform for executing privacy-preserving computations. Regardless of the chosen variant, the underlying principles remain consistent: computations are isolated, privacy is preserved, and verifiable proofs of computation are generated. This adaptability empowers developers to choose the execution environment that best aligns with their requirements, while maintaining the same high standards of security and privacy.

6.6 Enabling Decentralized Computational Platforms

PRIVATON revolutionizes decentralized computational platforms by providing an unprecedented level of security and confidentiality. The integration of finite state automatons, verifiable proofs of computation, and state capability models empowers PRIVATON to drive next-generation computing paradigms, safeguarding sensitive data and enabling trustless and secure interactions.

Our Construction

PRIVATON's operational mechanisms intricately weave together its state model, verifiable policy state capabilities, and the generation of verifiable proofs of computation. This section elucidates the core concepts behind PRIVATON's operational mechanisms, providing an in-depth understanding of how it achieves secure and privacy-preserving computations.

In this section, we present the PRIVATON model of computation. We models a computation X into a deterministic finite state automaton (DFSA),
mathematically as a six-tuple $(K,\Sigma,\delta,s,A,\boldsymbol{\Theta})$ where:

- K is a finite set of computation states of the dual sandbox strategy
- Σ is the computation's inputs provisioned by the participating principals defined in the associated verifiable public policy.
- $s \subset K$ is the start state, in most cases representing the host with an uninitialised dual sandbox
- $A \subseteq K$, is the set of accepting states of the computation
- δ is the state transition function such the $K \times \Sigma \rightarrow K$
- $\boldsymbol{\Theta}$ is the computational output

A *batch (**β**)* is a finite sequence, possibly empty, of principals Σ, part of X-Margin's pool of providers for a given computation. Given any pool, the smallest batch that can be formed from is an empty batch, which we will write as ε. A configuration of a computation X is an element of $K \times \boldsymbol{\beta}$, that essentially captures the current state and the input that is left to be received.

Henceforth, we refer to this model of computation within the dual sandbox environment as the PRIVATON (X_p).

Definition 1: (The PRIVATON Computation)

As the transition function δ, defines the operations of PRIVATON (X_P) one step at a time along with the Verifiable public policy (V_P) to define the sequence of configurations that the given computation (X_P) will enter. We start by defining the relation "yields-in-one-step" ($|\text{-}_X$), wherein it relates configuration[1] to configuration[2] when X_P can move from the former to the latter in a single step. Mathematically,

$$(q_1, c) \mid\text{-}_X (q_2, \Theta) \text{ if } ((q_1, c), q_2) \subset \delta$$

where c is any element of Σ

So now the relation "yields" ($|\text{-}_X^*$) can be defined as reflexive, transitive closure of $|\text{-}_X$ i.e. a configuration C_1 yields configuration C_2 if X_P can go from C_1 to C_2 in zero or more steps. This can be written as:

$$C_1 \mid\text{-}_X^* C_2$$

The PRIVATON (X_P) computation can now be defined as a finite sequence of configurations C_0, C_1 ... C_n for n ≥ 0 such that:

- C_0 is the initial configuration
- C_n is of the form of (q,ε), for some state q ⊂ K,
- $C_0 \mid\text{-}_X C_1 \mid\text{-}_X C_2 \mid\text{-}_X \ldots \mid\text{-}_X C_n$

Given **β** a batch of Σ for a given computation, we define the following:

- X_P accepts **β** if (s,**β**) $|\text{-}_X^*$ (q,**Θ**) for some q ⊂ A. Any configuration (q, **Θ**), for some q ⊂ A, is called an accepting configuration of X_P.
- X_P rejects **β** if (s, w) $|\text{-}_X^*$ (q, **Θ**), for some q ⊄ A. Any configuration (q, **Θ**), for some q ⊄ A, is called an rejecting configuration of X_P.

Definition 2: (PRIVATON Halt)

Every PRIVATON (X_P) on input **β**, halts after |**β**|+4 steps.

On input **β** PRIVATON (X_P) executes computations $C_0 \mid\text{-}_X C_1 \mid\text{-}_X C_2 \mid\text{-}_X \ldots \mid\text{-}_X C_n$ where C_0 is the initial configuration, and C_n is of the form (q,**Θ**) for some state q ⊂ K. C_n is either an accepting or a rejecting configuration, so X_P will halt when it reaches C_n. Each step in the X_P computation comprises of

- A empty dual sandbox initialisation on the host as a fixed initial configuration C_1
  $C_0 \mid\text{-}_X C_1$
- A relevant verifiable public policy import into the freshly initialized sandbox configuration state
  $C_1 \mid\text{-}_X C_2$
- Next |**β**| steps in the computation that consumes one input from each of the principals
  $C_2 \mid\text{-}_X C_3 \mid\text{-}_X \ldots \mid\text{-}_X C_{|\beta|+2}$
- A ready to execute (r) state after all the |**β**| inputs to the computations have been provisioned
  $C_{|\beta|+2} \mid\text{-}_X C_r$ where r ⊆ A
- A final teardown state (C_n ⊆ A) that clears the sandbox environment.
  In most of the scenarios $C_n == C_0$ i.e. will revert back to its original state prior to the computation

So n = $|\boldsymbol{\beta}|+4$. Thus X_p will halt after $|\boldsymbol{\beta}|+4$ steps.

7.1.1 Verifiable Policy State Capabilities

PRIVATON's operational mechanisms are fortified by the Verifiable Policy State Capabilities. These capabilities define the rules, permissions, and bounds within which computations operate. The verifiable public policy associated with each computation encapsulates attributes such as metadata, permissions, and layers. It ensures computations are confined to designated states, limiting access to specified operations and data.

The PRIVATON proof is composed of the following fields:

```
{
        computation seed : bigint,
        proof : HASH,
        state proofs :
        {
                state commitments : HASH,
                data commitments : HASH,
                meta commitments : HASH
        }
}
```

- **computation seed**: to introduce a randomness to the given proof computation
- **proof**: unique fingerprint of the given computation derived from the state proofs
- **state proofs**: granular proofs of the states transitioned for the given computation each comprising the below commitments.
    - **state commitments**: hash of the current state's [function signatures, incoming state inputs, (optional) auxiliary data structures referenced or computed within that are not part of state outputs]
    - **data commitments**: hash of current state's output(s) which would serve as the input for the next state or as the final computation output
    - **meta commitments**: hash[state commitments (state(i)), meta commitments (state (i-1))] that tracks the chained state transition sequence

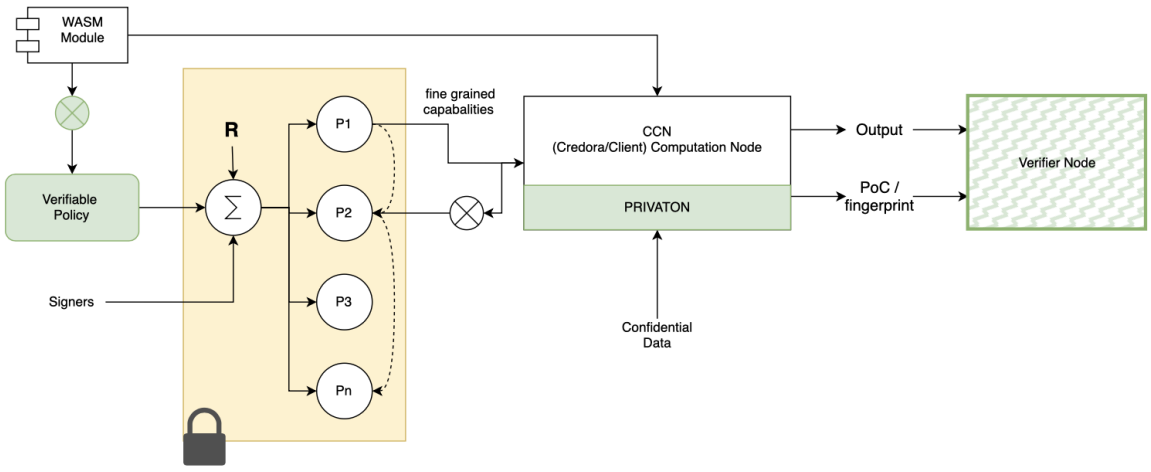
7.1.2 Generation of Proofs of Computation

A cornerstone of PRIVATON's operational mechanisms is the generation of verifiable proofs of computation. As a computation unfolds within the dual sandbox strategy, PRIVATON meticulously records each step. These steps culminate in a cryptographic proof that validates the computation's precise execution as per the defined state model and verifiable public policy. These verifiable proofs enhance trust and accountability in the computation's outcome.

7.1.3 Integration of Dual Sandbox Strategy

PRIVATON's operational mechanisms are seamlessly integrated with the dual sandbox strategy. Whether within a TEE-WASM enclave or a standalone WASM runtime, computations are executed within confined environments. In TEE-backed enclaves, the TEE enforces isolation and attestation, while in standalone runtimes, virtual sandboxes ensure strict access controls. This integration guarantees secure execution and privacy preservation, regardless of the execution context.

By unifying the state model, verifiable policy capabilities, proof generation, and the dual sandbox strategy, PRIVATON's operational mechanisms provide a comprehensive framework for secure and private computations. This holistic approach confronts the challenges of confidential computing, safeguarding sensitive data and computations against a backdrop of varying execution contexts.



PRIVATON Computation Execution Framework (CEF)

The PRIVATON Computation Execution Framework (CEF) encapsulates the operational intricacies of the PRIVATON model, orchestrating the execution of privacy-preserving computations through a systematic approach. These computations, modeled as finite-state units, are subject to the verification and consensus among multiple PRIVATONS, ensuring the correctness and agreement on their execution.

8.1 The Essence of PRIVATON Consensus

In the context of decentralized systems, consensus forms the cornerstone for fault-tolerant operations. For PRIVATONS, consensus involves harmonizing the execution sequence, intermediate values, and final outputs of a given computation. To illustrate this concept, consider the following notations:

- f(x): The unit of computation to be performed.
- P: A PRIVATON capable of executing f(x).
- N: The number of PRIVATONS performing the same computation f(x), where N>1
- S0 ..K: Represents the states of a PRIVATON P.

- K: Denotes the number of states required for the successful execution of f(x) by a PRIVATON P.

The essence of PRIVATON consensus can be captured through the following mathematical formulation:

$$\mathbf{P(f(x))=\Sigma i=0\ K\ (\Phi(S_i,\ \Lambda i)+\delta(S_i\ ,\ S_i+1))}$$

Here:

- Λi : Signifies the input dependencies of state Si concerning computation f(x).
- δ: Represents the state transition function from Si to Si+1
- Φ: Stands for the computational output of state Si
- SK+1 = S0, ensuring cyclic transition from the last state back to the initial state.

8.2 Verifiable Consensus

A core tenet of PRIVATON consensus is the verifiable nature of computations across PRIVATONS. To ascertain the validity of computations and to achieve consensus, the following conditions must hold:

- Identical State Commitments: The cryptographic commitments Ci of states Si across PRIVATONS should be identical for a given computation f(x). This ensures uniformity and prevents tampering.
- Threshold-Based Execution Time: The computation execution times ti for state Si across PRIVATONS should not deviate significantly from the average threshold time across all PRIVATONS. This guards against potential deviations and irregularities.

8.3 Defining Fault Tolerance

In the context of PRIMATON consensus, fault tolerance refers to the system's resilience in the face of colluding or malfunctioning PRIMATONS. Specifically, we denote T as the maximum number of faulty PRIVATONS that the system can withstand without compromising the consensus process.

Definition 3: Non-Faulty PRIVATON Computation

For a given computation f(x) and a PRIVATON Pn, the computation is deemed non-faulty if the following criteria are met:

1. The cryptographic commitments Ck of each state Sk are identical across all PRIVATONS, n=1,...,N and k=0,...,K.
2. The execution time t_i of state Si for PRIVATON Pn does not deviate unusually from the average threshold time across all PRIVATONS, n=1,...,N and i=0,...,K.

8.4 Ensuring PRIMATON Halt and State Transitions

PRIVATON consensus encompasses the mechanism to ensure that non-faulty PRIVATONS halt in a synchronized manner and that state transitions occur securely. In mathematical terms:

$$\mathbf{P_n \,|S_k\,(commitment)=C_k \quad \forall\,(k=0,...,K)\ and\ (n=1,...,N)}$$

For state transitions, the following condition is established:

$$\begin{cases} \delta(S_i\,,\,S_i{+}1), & \text{if } T \le t \\ \text{Halt}, & \text{if } T > t \end{cases}$$

Where T represents the maximum permissible number of faulty PRIVATONS, n=1,...,N, and i=0,...,K.

In summary, PRIVATON consensus ensures the verifiable agreement among PRIVATONS regarding the execution sequence, intermediate outputs, and final results of a computation. Through the establishment of fault tolerance, identical state commitments, and synchronized state transitions, the PRIVATON Computation Execu{tion Framework guarantees the integrity and correctness of computations in a privacy-enhanced and decentralized environment.

Implementation Details

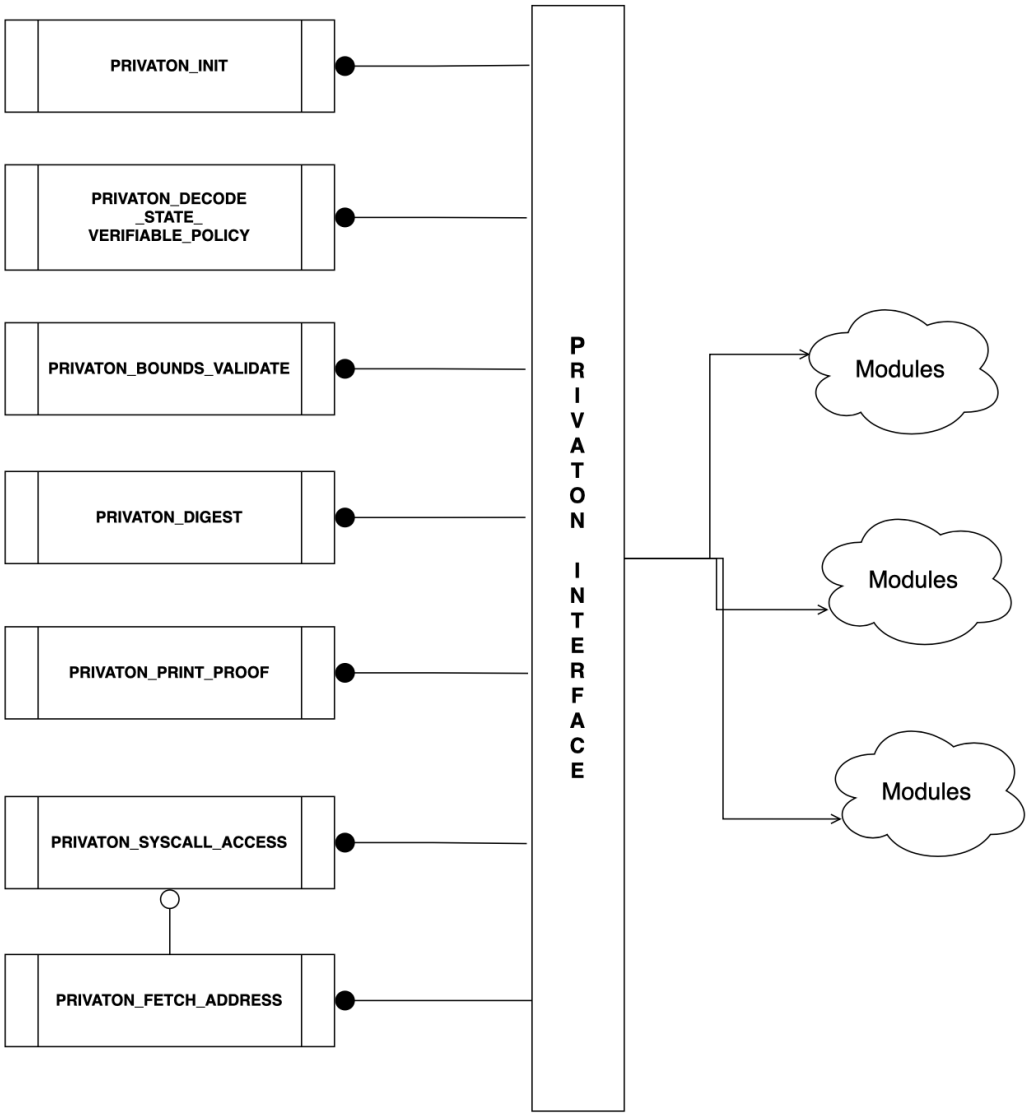
9.1 Data Structures

- json proof_document: This JSON object serves as the central repository for storing proof-related information. This data structure enables the framework to maintain a record of executed computations and their associated commitments.
- policies_cache and stack_frames_cache: These maps are used to cache state capabilities. policies_cache holds a map of computation modules to their respective state capabilities, while stack_frames_cache stores state capabilities for individual stack frames. These caches provide efficient access to state capabilities, aiding in validation and proof generation.

9.2 Core Functions

- **privaton_init**: This function initializes the framework, generating a computation salt. The salt is used to introduce randomness and security to commitment calculations. It generates a random seed and computes a commitment of the seed. The commitment and seed are then stored in the proof_document.
- **privaton_bounds_validate**: This function validates execution bounds and permissions for a given computation state. It checks if the current state's capabilities match the function's metadata and execution bounds. If skip_bounds is enabled, only the checkpoint disabling is verified.
- **privaton_decode_verifiable_policy**: This function decodes a verifiable policy JSON. It parses the JSON and extracts state capabilities, committing the policy for verification purposes. It also populates stack_frames_cache with state capabilities.

- **privaton_syscall_write_access** and privaton_syscall_read_access: These functions check permissions for syscall access, specifically write and read operations. They verify whether the current state has the required permissions based on its capabilities.
- **privaton_syscall_access**: This function checks general syscall access, combining the checks for read and write access. It uses the aforementioned functions to verify the permissions for the current state.
- **privaton_fetch_address**: This function fetches an address location based on permissions and a given key. It verifies the permissions and uses the key to retrieve the address from the capability's file address map.
- **privaton_digest**: This function generates and records digests for verification purposes. It calculates digests based on provided inputs, such as computation salt, function data, and other arguments. The function computes hash digests and constructs proof entries in the proof_document.



- **privaton_get_next_state** and **privaton_get_nth_previous_state**: These functions retrieve the next or nth previous computation states based on the current state. They utilize the state capabilities and sequencing information stored in policies_cache.
- **proof_builder**: This function constructs and appends proof information to the proof_document. It calculates hash digests for various parameters and commitments and updates the proof_document with relevant information. It includes function commitments, data commitments, state commitments, and more.
- **privaton_print_proof**: This function prints the proof information stored in the proof_document. It displays the JSON structure containing computation proofs, including function commitments, data commitments, and state commitments.
- **privaton_proof_matrix**: This function generates a QR code containing the proof information. It encodes the proof document using QR code generation libraries and displays a visual representation of the proof.

The PRIVATON framework's implementation covers a range of functionalities, from commitment calculations and state validation to access control and proof generation. The modular design and efficient data structures enable the framework to facilitate privacy-preserving computations while maintaining a clear record of executed computations for verification.

Transient Speculative Execution Attacks and PRIVATON

In the intricate landscape of modern computing, the emergence of transient speculative execution attacks poses a significant threat. These attacks exploit the inherent optimizations in hardware or compiler/runtime systems, executing predicted branches or states before they are conclusively resolved. Within the realm of PRIVATON, such scenarios could lead to the execution of unintended states, creating vulnerabilities that malicious actors could exploit.

Transient speculative execution attacks encompass two distinct categories: speculative execution attacks and transient execution attacks. Speculative execution attacks involve injecting false predicted states to manipulate module behavior. This could entail revealing sensitive information or altering execution timing. Transient execution attacks, exemplified by the infamous Spectre vulnerabilities, enable attackers to manipulate speculations to access cached data or affect execution timings, potentially undermining system security.

To counter these evolving threats, PRIVATON employs a multi-faceted defense strategy. First, at the software level, PRIVATON enforces strict sequencing of state code execution. It ensures that a given state's metadata validation must be completed before its execution, and subsequent states are executed only after the current state's execution and validation. This sequencing minimizes the window for transient code execution exploits, reducing the chances of speculation-driven attacks. Furthermore, the deterministic execution flow, guided by verifiable policy definitions, prevents out-of-order state executions and curtails speculation-driven deviations.

However, it is important to note that PRIVATON's approach does not eliminate speculative execution itself, as it's a hardware-level property. Instead, it focuses on controlling the timing and sequencing of states to mitigate the risks. The predefined state execution sequence and mandatory metadata validations signal speculative mispredictions and enforce policy adherence. While this strategy addresses transient execution attack windows, it doesn't entirely handle speculative execution attacks at the hardware level.

Additionally, to enhance mitigation strategies, PRIVATON explores the concept of introducing noise in the form of random and non-relevant states within its finite state execution. This concept, inspired by Google's retpoline, disrupts hardware's ability to learn patterns and preemptively counters speculative attacks by introducing randomness into the execution patterns.

In conclusion, PRIVATON's approach to tackling transient speculative execution attacks rests on meticulously orchestrating state sequencing, deterministic execution flows, and the introduction of noise. By synergizing these tactics, PRIVATON fortifies its defense against these intricate attacks, thereby reinforcing the resilience, security, and integrity of its computational ecosystem.

Conclusion

In the rapidly evolving landscape of confidential computing, the PRIVATON Proof of Computation framework emerges as a groundbreaking solution that addresses critical challenges surrounding security, privacy, and integrity. With its verifiable proof generation, state capability model, and dual sandbox strategy, PRIVATON sets a new standard for executing privacy-preserving computations.

PRIVATON's strength lies in its ability to construct a meticulous blueprint for computations through finite state automatons, encapsulating privileges, permissions, and capabilities. This approach adheres to principles of least privilege and intentional use, safeguarding against arbitrary allocations and unauthorized data exposure. The state capability model enforces security constraints, granting controlled access to resources and preventing data leaks.

At the heart of PRIVATON's innovation is its verifiable policy concept. By outlining permissible behaviors and interactions, these policies ensure computations adhere to predefined rules, verified through cryptographic commitments and validation mechanisms. The generation of verifiable proofs of computation enhances accountability and trust, providing a transparent record of each step executed within the framework.

PRIVATON's dual sandbox strategy further fortifies its operational mechanisms. Whether leveraged within a TEE-backed enclave or a standalone WASM runtime, PRIVATON maintains the same standards of security, privacy, and correctness. This adaptability empowers developers to choose the execution environment that aligns with their needs, without compromising on the core principles of the framework.

Incorporating fault tolerance, consensus mechanisms, and synchronization across PRIVATONS, the PRIVATON Computation Execution Framework ensures the verifiable agreement on execution sequences,

intermediate values, and final outcomes. It establishes a foundation for trustless and secure interactions within decentralized computational platforms. As with any advanced technology, PRIVATON is not without limitations. Addressing challenges such as complex policy management, performance overhead, and trust in setup parameters requires ongoing research and development. However, these limitations underscore the need for continuous innovation and improvement in the field of confidential computing.

In conclusion, PRIVATON sets a new trajectory for confidential computing by introducing a verifiable proof of computation paradigm. Its holistic approach, encompassing state modeling, policy enforcement, and secure execution, reshapes the way we approach secure computations. PRIVATON's integration of advanced cryptographic techniques, decentralized consensus, and efficient data structures opens doors to a future where privacy and security are paramount in the digital realm.

PRIVATON 2.0 - Advancing PRIVATON

A central thrust of PRIVATON 2.0 is the introduction of Hierarchical Policy Management. The concept of a Policy of Policies envisions a hierarchical structure where individual computations are governed by their own specific policies, while higher-level policies define how these computations are orchestrated within a broader execution pipeline. This allows for a modular and flexible approach to policy management, enabling developers to define complex workflows while maintaining fine-grained control over each computation's behavior.

The architecture of a Merkle tree being leveraged to achieve this goal. Each leaf node of the tree represents an individual computation, carrying its own policy attributes. These policies can include permissions, access controls, bounds, and more. The internal nodes of the tree represent higher-level policies that define how computations are combined and executed.

The root of the Merkle tree contains the hash of the entire execution pipeline's policy, ensuring that the entire process remains secure and tamper-proof. By specifying a path through the tree, one can load and execute a specific computation along with its associated policies, ensuring that the execution adheres to the predefined rules and constraints.

This ongoing improvement not only enhances the PRIVATON framework's versatility but also simplifies policy management for complex scenarios. This can compose computations into larger workflows with ease, and the Merkle tree structure guarantees the integrity of policies and computations throughout the execution process.This approach provides the groundwork for a more comprehensive, adaptable, and secure framework that caters to the diverse needs of modern computational environments.

12.1 IPFS Integration: Transforming Computation Dynamics

In PRIVATON 2.0, dynamic computation takes center stage with the strategic integration of the InterPlanetary File System (IPFS). This integration is a symphony of responsiveness and adaptability, where wasm modules are referenced by Content Identifiers (CIDs) hosted on IPFS nodes. These CIDs become gateways to authorized modules, allowing PRIVATON's runtime environment to seamlessly load modules in real-time during

execution. The efficacy of this dynamic loading is fortified by stringent hierarchical policies, acting as sentinels to validate module authenticity. These policies prevent unauthorized loading of modules lacking permissions and meticulously track changes in the path. The result is a computational landscape that is no longer static but perpetually dynamic, adaptable, and efficient.

12.2 Hierarchical Policy Impacts and Workflow Adaptation

A profound enhancement in PRIVATON 2.0 is the ability to track the cascading effects of policy changes. Imagine a scenario where altering a leaf node policy reverberates through the hierarchy, reshaping the entire workflow. PRIVATON 2.0's hierarchical structure not only tracks these changes but also triggers a ripple effect— modules to load, shifts in proof production, adjustments in backdated entries, and more.

This adaptive capability is a cornerstone of PRIVATON 2.0's evolution. As the policy tree adapts, the framework not only ensures modules are dynamically loaded based on new policies but also recalibrates proofs, traces historical changes, and navigates seamlessly between past and present states. The result is an ecosystem that not only reacts to policy changes but orchestrates an entire symphony of computational adaptability.

12.3 Guardrails with eBPF-Based Explicit Syscall Permissions

The ongoing development of PRIVATON includes a significant enhancement: the integration of eBPF-filters & explicit syscall permissions. Leveraging extended Berkeley Packet Filters (eBPF), PRIVATON's runtime gains the capability to intercept and scrutinize system calls in real-time. This ensures that only authorized interactions occur between computations and the underlying system, reinforcing PRIVATON's security framework. The incorporation of eBPF-based explicit syscall permissions not only minimizes potential attack vectors but also adds a resilient layer of protection against security vulnerabilities. Notably, the detailed traces of these system call interactions are included in the generated proofs, enhancing transparency and accountability in the computation process.

Furthermore, the comprehensive proof generation process of PRIVATON doesn't merely validate the execution of computations but also meticulously reflects the eBPF traces of system call interactions. This level of transparency not only enhances the system's accountability but also ensures that any suspicious or unauthorized activities are promptly identified and addressed.

REFERENCES

1. C. C. Consortium, https://confidentialcomputing.io/ (2022).
2. D. Goltzsche, M. Nieke, T. Knauth, R. Kapitza, Acctee: A webassembly-based two-way sandbox for trusted resource accounting, in: Proceedings of the 20th International Middleware Conference, Middleware '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 123–135. doi:10.1145/3361525.3361541. URL https://doi.org/10.1145/3361525.3361541

3. J. Me´ne´trey, M. Pasin, P. Felber, V. Schiavoni, Twine: An embedded trusted runtime for webassembly, in: 2021 IEEE 37th International Conference on Data Engineering (ICDE), 2021, pp. 205–216. doi:10.1109/ICDE51399.2021.00025.
4. M. Nieke, L. Almstedt, R. Kapitza, Edgedancer: Secure mobile webassembly services on the edge, in: Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking, EdgeSys '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 13–18. doi:10.1145/3434770.3459731. URL https://doi.org/10.1145/3434770.3459731
5. Y. Ma, Q. Zhang, S. Zhao, G. Wang, X. Li, Z. Shi, Formal verification of memory isolation for the trustzone-based tee, in: 2020 27th Asia-Pacific Software Engineering Conference (APSEC), 2020, pp. 149–158. doi:10.1109/APSEC51365.2020.00023.