A Preliminary Report On Edge-Verified Machine Learning (evML)

Abstract

This paper introduces evML, a protocol for secure distributed compute networks with negligible entry costs. It requires nodes to prove, using their secure enclave, that they're computing results within a trustworthy environment. They can't cheat unless the node operator performs costly hardware tampering. evML then uses spot-checks to catch nodes submitting false outputs. When caught, the enclave's unique identifier is blacklisted, making the attack cost a sunk investment. Therefore, honesty the only rational behavior. The worst-case analysis shows that honesty is optimal with a 5% computational overhead, assuming a hardware attack costs over $2000. Our results demonstrate that it is irrational within evML, though further empirical validation is warranted.

Keywords Game Theory · Trusted Execution Environments · Computation Integrity

This document is a preliminary report about evML, a novel cost-effective approach for verifying the correct execution of computations within distributed networks. This work was led by Arbion Halili in collaboration with the EXO Labs team. A revised, expanded copy of this document will be published in the near future. We offer our special thanks to the following people for their feedback and helpful discussions: Daniel Lubarov, Sreeram Kannan, Tarun Chitra, Danny Sursock and David Wong.

1 Motivation

In this report, we consider a distributed network of edge devices for Machine Learning workloads. The primary challenge we are considering is ensuring computation integrity and reliability. Without effective verification, participants may provide fraudulent results. However, the lack of central oversight in decentralized systems makes it harder to prevent fraud.

zkML ([1], [2]) uses zero-knowledge proofs to ensure computational correctness at a very high degree of security, but with significant computational costs. It is suitable for high-stakes applications like medical screening and financial modelling, but prohibitively expensive in other contexts. Optimistic Machine Learning (opML, [3], [4]) reduces overheads by using economic disincentives instead of cryptography, but its mixed-strategy Nash Equilibrium implies some fraud will go undetected unless each query is recomputed by an honest validator. Proof of Sampling (PoSP, [5]) protocols enforce honesty with penalties like slashing deposits, but their high staking cost discourages new nodes from joining, restricting scalability. Both OpML and PoSP lack privacy-by-default, making them unsuitable for user-submitted data.

evML was designed to bridge the gap between the existing solutions by offering a scalable approach with an excellent cost-accuracy tradeoff and rapid result finality. It promotes network growth by minimizing participation costs, using revokable hardware-bound keys instead of staking to mitigate Sybil attacks. evML uses probabilistic spot-checks to make fraud irrational, thus offering moderate-to-high correctness assurances, whilst minimizing redundant computation. evML also introduces a cost to snooping, thus offering resistance to mass surveillance.

2 Why evML is built on TEEs

Trusted Execution Environments (TEEs) are specialized areas within a processor designed to run sensitive code securely and in isolation. This isolation ensures that the code and data processed within the TEE remain confidential and are protected from tampering, even if the device's main operating system is compromised. TEEs achieve high performance [6] by running programs directly in their secure environment (enclave) without the need for cryptographic methods, which are computationally expensive.

evML is designed to enhance the trustworthiness of TEEs by continuously monitoring the network to ensure honest behavior. If dishonesty is detected, evML can permanently ban compromised devices, effectively **nullifying any investment made in compromising the device.** This allows us to enjoy the efficiency of TEE-based verifiable compute, but with better assurance that the reported results are correct. In Section 4, we model the choices of a rational node to show that our techniques significantly improve the viability of TEEs for secure outsourced computation.

2.1 Accountability via Revocation

Remote Attestation is a key feature of Trusted Execution Environments (TEEs) that allows a remote party to verify the authenticity of the results produced by the TEE. This is done by generating a cryptographic report, which is signed using a unique key that is securely embedded in the hardware, commonly at the time of manufacture. The remote party can then use this signed report to confirm that the TEE used to produce the results is trustworthy, and by extension, that the reported results are genuine and accurate.

Devices equipped with TEEs are issued endorsement keys, which are securely stored and used to generate these cryptographic reports. Our system employs dynamic group signature schemes to efficiently update the set of valid endorsement keys, as described in Bootle et al. (2020) [7].

This allows for the swift **removal of compromised devices** from our network, called revocation. It is possible to revoke the keys used to produce the fraudulent cryptographic report without revealing the identity of the lying node. This enables the network participants to remain entirely anonymous. If a node's endorsement keys have been revoked, they cannot participate in the network anymore, thus the time and money spent on TEE tampering is wasted. This serves to deter TEE tampering.

When critical vulnerabilities are discovered for a hardware platform, all associated devices may be preemptively suspended from the network via the revocation mechanism. This ensures that platform-specific exploits don't affect the entire network, which features multiple independent TEE vendors.

2.2 Approximating TEE Behavior on Consumer Hardware

Although many modern edge devices come equipped with an on-board Trusted Execution Environment (TEE), popular operating systems like Android and iOS do not provide TEE-related APIs to application developers. As a result, we must create an approximation of TEE functionality that retains some of the hardware-backed security guarantees, even though absolute memory isolation cannot be ensured. We will refer to this approximation, which must include Remote Attestation capabilities, as a Secure Context (SC).

Play Integrity (Android, [8], [9]) and DeviceAttest (Apple, [10]) can be used to implement Secure Context functionality on mobile devices, leveraging the capabilities of Secure Boot provide correctness guarantees. These will be our primary examples, although similar capabilities have been implemented on non-ARM platforms using Intel SGX [11] and RISC-V's Keystone [12].

Secure Boot enables the device's hardware to verify the integrity of the operating system and other cryptographic components before loading them into memory. This process creates a chain of trust from hardware to software, ensuring the integrity of the device's execution environment.

If Secure Boot succeeds, Play Integrity and DeviceAttest can generate **hardware-backed device attestations**. These attestations ensure the integrity of the device's execution environment and confirm the specific app binary running on the device. This indicates that any network request which bears these proofs arises from the **correct execution** of this verified application.

The capacity to produce these hardware-backed proofs is guaranteed at the silicon-level, often by ARM TrustZone [9]. TrustZone was introduced as part of the ARMv6 architecture, which is ubiquitous among mobile devices. TrustZone establishes trust in devices using a device-specific key, often referred to as the 'hardware root of trust' or 'device root key'. This key is embedded into the device at time of manufacture, is unique to the device, and is persistent. It is verifiably linked to the central root of the trust, the manufacturer, using a cryptographic certificate chain.

Although TrustZone and Play Integrity are enough to implement a Secure Context on-device, the device root key used prevents devices from generating cryptographic reports anonymously and can only be revoked by the hardware manufacturer. Instead, we use Remote Attestation to authenticate each device that joins the network, then issue it a new set of endorsement keys. These keys can be used anonymously and get revoked by the network [7]. From this point, "endorsement keys" will refer to the network-issued keys, and the device root key will be called by its proper name.

The endorsement keys are held in secure storage by an evML-enabled app, letting it participate as an independent remote-attested Secure Context on the evML network. This app is responsible for all evML-related logic, such as performing ML computations and interfacing with other devices over the network. While the integrity of the app's execution is ensured by TrustZone, the app's source code must be vetted by the issuer of endorsement keys. This ensures that the app itself does not tamper with the outputs of the computation, nor leak any data. As discussed, the issuer of endorsement keys can check which app it is issuing to using PlayIntegrity.

Since the device root key is device-unique and known to the issuer of endorsement keys, they can ensure that multiple keys aren't issued to the same, possibly malicious, devices. The device root key is never exposed to other network participants.

Present in most edge devices, including smartphones and tablets, TrustZone and TrustZone-like facilities **eliminate the need for specialized hardware** or significant capital investment. This method reduces barriers to entry, ensuring that underutilized resources are put to work.

In the rest of this document, we will use the term TEE interchangeably with Secure Context (SC), since they serve the same purpose and may be used to draw identical conclusions.

3 Overview of the evML Protocol

Validators register their TEEs with the Network Operator (NO), which uses mechanisms like Google Play Integrity or Apple DeviceAttest to authenticate the TEEs. The NO exists to manage the set of endorsement keys, banning compromised devices if necessary, and can be implemented in a trust-minimized manner using sMPC. The NO may be provided as a public good by an altruistic organization, by a network of fee-collecting nodes, or otherwise.

If the NO is satisfied with its authenticity checks, it will issue cryptographic endorsement keys to the TEEs over an encrypted channel, which will enable them to join in the network.

Users looking to offloading computation use a gossip protocol [13] to find validators. Validators, who can perform computations, prove their membership of the network using their endorsement keys and a user-provided nonce on-request. Users generate a query, consisting of a program hash and serialized inputs, to send to their chosen validator. The program hash specifies the ML model that the user wants to use. The inputs may be encrypted to the validator with a symmetric key established via a key agreement protocol, like ECDH.

Users send queries to their chosen validator, who will perform the ML tasks and submit the result with its computational trace, jointly signed using the validator's endorsement key. With probability p, a challenge protocol is triggered to verify the validator's output. This challenge protocol is described in the full paper, alongside a description of the computational traces it uses.

Although not every task is checked, the probability of being caught by the challenge mechanism and revoked is high enough that it is always irrational to misbehave, thus ensuring that it is irrational for nodes to compromise their TEE. Matching results reward all honest parties; discrepancies lead to the revocation of endorsement keys.

We define C as the cost of performing the requested computation, $W(x)$ as the profit from selling knowledge of the private input x, $U(x)$ as the additional profit for undetected lying by nodes, K as the cost of compromising a TEE, and S as the cost of replacing a revoked device.

3.1 Cheating and Privacy Are Related

In incentive-based distributed compute networks that lack privacy, nodes can see queries and estimate the payoffs for behaving maliciously on a per-query basis. If lying is profitable for a specific query, the node should lie. Since estimated payoffs vary by query, it may be rational to lie sometimes but not always. Setting the security parameters to make misbehavior irrational across all cases is prohibitively expensive, since we must assume the worst-case payoffs to deter all potential fraud.

With a TEE, snooping on queries requires the node operator to compromise the TEE at a cost, as does cheating. The decision to compromise the TEE can only be rationally made using

the expected profit from dishonest activities $W(x) + U(x)$, since the actual profits from lying or snooping are not known in advance. This expectation is determined by the network-wide distribution of $W(x)$ and $U(x)$, thus the rationality of cheating is not solely determined on a per-query basis.

evML's use of TEEs makes it possible to parameterize the protocol such that misbehavior is always irrational, regardless of the any query-specific factors. This reduces the likelihood of snooping and dishonest behavior, and since TEEs are private by default, it makes mass-surveillance implausible.

3.2 Transmissible Proofs

While evML ensures honest outputs for users following the protocol, it does not guarantee users will follow the protocol correctly. A user may select any validator, including their own malicious nodes, to generate an output. Therefore, if a user wants to prove their query result was generated correctly to a third party, simply relaying the outputs and the associated data is insufficient, as the third party cannot verify the validator was chosen randomly. This issue can be resolved by delegating validator selection to a mutually trusted entity, such as a smart contract on a trustless network, ensuring that the random nature of the choice is provable. This feature is optional and outside of this paper's scope.

Our use of group signatures as endorsement keys allows the authentication of work done by nodes over the network to occur without revealing the identity of any participants. When combined with network-level anonymity technologies, such as Dandelion++ ([14], [15]), this prevents attacks on the infrastructure that involve targeting specific compute nodes or tracking the kinds of work they do.

3.3 Enforcing Spot-Checks

Network operators must cover the cost of triggering the challenge protocol on behalf of the users, or handle this duty. This prevents users from skipping the verification steps to save money, thus ensuring that validator's work is checked frequently and keeping threat of penalties for lying credible.

To prevent malicious users from abusing the challenge mechanism and draining the network's security budget, network operators should only cover challenge costs if it can be proven that the challenge was triggered randomly at the specified frequency. This can be ensured by using the hash of the non-malleable communication transcript between the user and validator, which is unique to each round of computation, as a random oracle [16]. When the hash falls within a specified range, corresponding to a random probability p, the network operator can trigger the challenge mechanism on behalf of the user. The transcript contains sufficient information to recompute the query and includes group signatures from the validator, providing enough data to trigger the challenge mechanism and revoke the validator's endorsement keys if cheating is detected. The network operator should ignore previously seen hashes to prevent replay attacks.

This solution still permit users to perform additional validation by initiating the challenge mechanism at a cost, which may be desirable if their query is especially sensitive.

If the inputs in the transcript are encrypted, as per Section 3, then user must recover the associated symmetric key and transmit it to the challenge mechanism. The mechanism will decrypt the contents of the transcript needed to re-compute the query, for comparison with the first validator's results.

4 Mathematical Description of evML

The setup of the game involves a set of users who participate in the protocol. The game progresses through a series of steps (t), which can be either finite or infinite in number.

Players initially engage in the game **non-maliciously** using a **Type A** strategy. However, they have the option to compromise the Trusted Execution Environment (TEE) and switch to a **malicious Type B** strategy, which incurs an **initial cost K**.

During each step of the game, a query x is submitted to the network, which can be considered as the outcome of a random variable X representing the distribution of all the computations submitted to the network. A validator v from the set V is randomly selected to perform an action based on their chosen strategy.

There is a **probability p** that the **challenge mechanism will be triggered** during this step, the purpose of which is to **detect and punish dishonest users.**

4.1 Types of Player

4.1.1 Type A Users

Type A users do not tamper with their TEEs, and thus can not misbehave. They receive the reward R for submitting the output of each computation, generated at a cost of C. They do not know the plaintext query, x, thus they can not sell user data to extract additional revenue.

With a probability of p, their correctness of their computation will be checked via the challenge mechanism. Given that work is checked, there is a further probability q_h that the honest user is punished by mistake, leading to the payoff $-C$ and the revocation of their endorsement keys. In practice, q_h will be close to zero due to our challenge mechanism design, which is detailed later.

4.1.2 Type B Users

Type B users, having broken the TEE at a cost of K, know the value of x and can choose between two actions. They can either act **honestly (Type B1)**, receiving the reward $R - C$ and an additional profit $W(x)$ from knowing x, or they can act **dishonestly (Type B2)**, misreporting the output of the computation over x to extract a further profit of $U(x)$.

When the challenge mechanism is triggered for dishonest agents, there is a further **probability q_d that they will be caught** and incur a cost S instead of receiving the reward $R - C$, which is the cost of replacing the device. The likelihood of being caught when dishonest, represented by pq_d, depends on the design of the challenge mechanism, especially its **resistance to collusion**. Similarly, the probability q_h represents the likelihood of a malicious, but not dishonest, actor being punished by mistake via revocation of their endorsement keys and having to pay S.

If caught, they must restart the game and decide again whether to play as Type A or Type B, incurring the cost K again if they choose Type B. If not caught, with probability $1 - pq_d$, they receive an additional profit $U(x)$ and continue playing as Type B.

4.1.3 States and Actions

The states in the Markov model include S_A for non-malicious actors (Type A), S_B for malicious actors of Type B1 or B2, and S_{restart} for actors who have been kicked out.

Action a_A represents the choice to continue as type A, whilst a_{B1} represents the choice to play Type B1 and a_{B2} represents the choice to play Type B2.

4.2 Markov Model

The Markov Model is explained graphically in Appendix B, which may aid your intuition regarding how evML is constructed. We strongly recommend that you view the appendix.

4.2.1 Payoff Matrix

State × Action	Result State	Probability	Reward
S_A, a_A	S_A	$1 - pq_h$	$R - C$
S_A, a_A	S_r	pq_h	$-C$
S_A, a_{B1}	S_B	$1 - pq_h$	$-K + R - C + W(x)$
S_A, a_{B1}	S_r	pq_h	$-K - C + W(x)$
S_A, a_{B2}	S_B	$1 - pq_d$	$-K + R - C_1 + U(x) + W(x)$
S_A, a_{B2}	S_r	pq_d	$-K - C_1 + W(x)$
S_{B1}, a_{B1}	S_B	$1 - pq_h$	$R - C + W(x)$
S_{B1}, a_{B1}	S_r	pq_h	$-C + W(x)$
S_{B1}, a_{B2}	S_B	$1 - pq_d$	$R - C_1 + U(x) + W(x)$
S_{B1}, a_{B2}	S_r	pq_d	$-C_1 + W(x)$
S_{B2}, a_{B1}	S_B	$1 - pq_h$	$R - C + W(x)$
S_{B2}, a_{B1}	S_r	pq_h	$-C + W(x)$
S_{B2}, a_{B2}	S_B	$1 - pq_d$	$R - C_1 + U(x) + W(x)$
S_{B2}, a_{B2}	S_r	pq_d	$-C_1 + W(x)$
S_r, a_A	S_A	$1 - pq_h$	$-S$
S_r, a_A	S_r	pq_h	$-S$

In Table 1, C_1 represents the cost of generating a lie, which is unique to the Type B2 strategy. The remainder of the table is consistent with the Markov Graph and associated explanations.

4.3 Summary of Parameter Relationships

The optimal policy depends on the distribution of x, values of $W(x)$ and $U(x)$, costs K and S, and probabilities pq_h and pq_d of being caught. To set Type A as the optimal policy, the reward R must be high, the cost K of breaking the TEE must be substantial, and the penalty S must be significant. The probability pq_d of being caught should be high to deter dishonesty, whilst the likelihood of erroneous punishment q_h and the expected profit $\mathbb{E}(U(x) + W(x))$ from dishonesty should be low relative to R, K, and S.

5 Closing remarks

The current report provides preliminary information about evML and its core components. Although the Challenge Mechanism is an important aspect of evML, it is not discussed in this document. There may be multiple useful challenge mechanisms, including m-of-n sampling and ZKP-based approaches. A m-of-n challenge mechanism has been evaluated, the model and description of which is provided in a Github repository alongside this document. The design of this mechanism is subject to change.

Our initial results indicate that Type A, the non-malicious strategy, is the optimal policy for validators. The high cost of breaking the TEE (K), combined with the penalty for being caught (S), and the reward for honest behavior (R), make honesty the most optimal strategy.

The probability of being caught (pq) was sufficient to deter dishonesty. Our estimations for S, K, and W are discussed in the documentation on Github, which seeks to provide a comprehensive account of the economic incentives and costs associated with the evML network. Further research is warranted to refine these parameters and better reflect the features and capabilities of popular edge hardware. This will ensure the robustness and scalability of the evML network in real-world applications. However, more rigerious experiments and different challenging mechanisms are being tested.

Future work will contain detailed description of the experimental results as well as systematic comparisons with other methods.

Bibliography

[1] D. Camuto, "Honey I SNARKED the GPT." [Online]. Available: https://hackmd.io/mGwARMgvSeq2nGvQWLL2Ww

[2] S. Lee, H. Ko, J. Kim, and H. Oh, "vCNN: Verifiable Convolutional Neural Network based on zk-SNARKs." [Online]. Available: https://eprint.iacr.org/2020/584

[3] C. So, K. Conway, X. Yu, S. Yao, and K. Wong, "opp/ai: Optimistic Privacy-Preserving AI on Blockchain." 2024.

[4] K. Conway, C. So, X. Yu, and K. Wong, "opML: Optimistic Machine Learning on Blockchain." 2024.

[5] Y. Zhang and S. Wang, "Proof of Sampling: A Nash Equilibrium-Secured Verification Protocol for Decentralized Systems." 2024.

[6] X. Li, B. Zhao, G. Yang, T. Xiang, J. Weng, and R. H. Deng, "A Survey of Secure Computation Using Trusted Execution Environments." 2023.

[7] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, and J. Groth, "Foundations of Fully Dynamic Group Signatures," *Journal of Cryptology*, vol. 33, no. 4, pp. 1822–1870, Oct. 2020, doi: 10.1007/s00145-020-09357-w.

[8] G. (Android Developers), "Documentation for Android Key Attestation." [Online]. Available: https://developer.android.com/privacy-and-security/security-key-attestation

[9] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "TrustZone Explained: Architectural Features and Use Cases," in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, 2016, pp. 445–451. doi: 10.1109/CIC.2016.065.

[10] Apple, "DeviceCheck."

[11] V. Costan and S. Devadas, "Intel SGX Explained." [Online]. Available: https://eprint.iacr.org/2016/086

[12] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: an open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, in EuroSys '20. Heraklion, Greece: Association for Computing Machinery, 2020. doi: 10.1145/3342195.3387532.

[13] T. Lu, B. Fang, X. Cheng, and Y. Sun, "Peer Discovery in Peer-to-Peer Anonymity Networks," in *2006 IEEE International Conference on Networking, Sensing and Control*, 2006, pp. 131–136. doi: 10.1109/ICNSC.2006.1673130.

[14] G. Fanti *et al.*, "Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees." 2018.

[15] ErCiccione, "Another privacy-enhancing technology added to Monero: Dandelion++." 2020.

[16] N. Koblitz and A. J. Menezes, "The random oracle model: a twenty-year retrospective," *Designs, Codes and Cryptography*, vol. 77, no. 2, pp. 587–610, Dec. 2015, doi: 10.1007/s10623-015-0094-2.

A Markov Analysis in Python

```
import numpy as np
import mdptoolbox

# Defining Parameters
p = 0.05        # Probability of triggering the challenge mechanism
q_d = 1         # Probability of true positives
q_h = 0         # Probability of false positives
R = 0.5         # Reward for completing the computation
C = 0.45        # Cost for completing the computation
C_1 = 0.45      # Cost for just decrypting the data
discount = 0.96 # Discount factor
K = 1000        # Cost of breaking the TEE (Trusted Execution Environment)
S = 100         # Cost of replacing the device
W = 1           # Reward of knowing private data
U = 1           # Reward of altering the data

# States are enumerated as follows:
# Type A: 0
# Type B1: 1
# Type B2: 2
# Restart: 3

# Defining the Transition Model
# Dimensions: (number of actions, number of states, number of next states)
transition_model = np.zeros((3, 4, 4))

# Action 1 (a_A)
transition_model[0, :, :] = np.array([
    [1 - p*q_h, 0, 0, p*q_h],  # From state 0 to state 0
    [0, 1, 0, 0],  # From state 1 to state 1
    [0, 0, 1, 0],  # From state 2 to state 2
    [1 - p*q_h, 0, 0, p*q_h]   # From state 3 to state 0 (restart)
])

# Action 2 (a_B1)
transition_model[1, :, :] = np.array([
    [0, 1 - p*q_h, 0, p*q_h],  # From state 0 to state 1
    [0, 1 - p*q_h, 0, p*q_h],  # From state 1 to state 1
    [0, 1 - p*q_h, 0, p*q_h],  # From state 2 to state 1
    [0, 0, 0, 1]   # From state 3 to state 3 (restart)
])

# Action 3 (a_B2)
transition_model[2, :, :] = np.array([
    [0, 0, 1 - p*q_d, p*q_d],  # From state 0 to state 2 or 3
    [0, 0, 1 - p*q_d, p*q_d],  # From state 1 to state 2 or 3
    [0, 0, 1 - p*q_d, p*q_d],  # From state 2 to state 2 or 3
    [0, 0, 0, 1]           # From state 3 to state 3 (restart)
])
```

```
# Defining the Reward Model
# Dimensions: (number of actions, number of states, number of next states)
reward_model = np.zeros((3, 4, 4))

# Action 1 (a_A)
reward_model[0, :, :] = np.array([
    [R - C, 0, 0, -C],  # Reward
    [0, 0, 0, 0],      # No reward for transitioning from state 1
    [0, 0, 0, 0],      # No reward for transitioning from state 2
    [-S, 0, 0, -S]      # Cost for restarting from state 3
])

# Action 2 (a_B1)
reward_model[1, :, :] = np.array([
    [0, -K + R - C + W, 0, -K - C + W],
    [0, R - C + W, 0, - C + W],
    [0, R - C + W, 0, - C + W],
    [0, 0, 0, 0]
])

# Action 3 (a_B2)
reward_model[2, :, :] = np.array([
    [0, 0, -K + R - C_1 + W + U, -K - C_1 + W],
    [0, 0, R - C_1 + W + U, -C_1 + W],
    [0, 0, R - C_1 + W + U, -C_1 + W],
    [0, 0, 0, 0]
])

# Applying Value Iteration
pi = mdptoolbox.mdp.PolicyIteration(transition_model, reward_model, discount, 1000000)
pi.run()

# Outputting the optimal policy and value function
print(pi.policy)  # Optimal policy for each state
print(pi.V)       # Value function for each state
```

B Markov Model, Graphically Explained

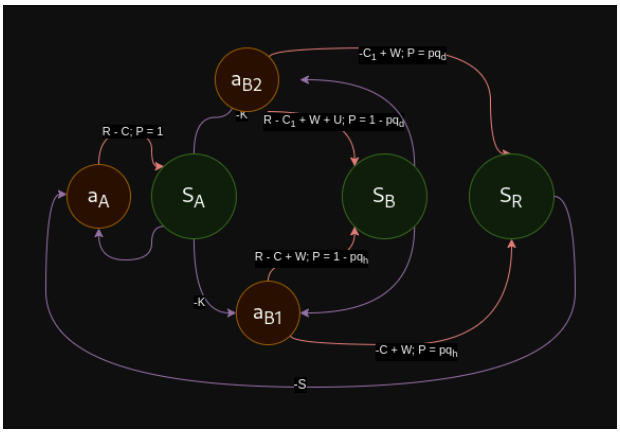

Figure 1: This diagram illustrates the actions available to each validator (orange), the possible states (green), and the associated payoffs. Validators start at S_A and choose actions to maximize their total payoff. The payoffs for each action depend on the current state.

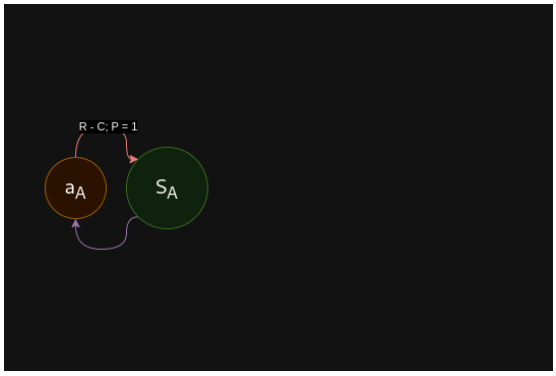

Figure 2: Validators are non-malicious by default, and can participate in the network for the reward $R - C$.

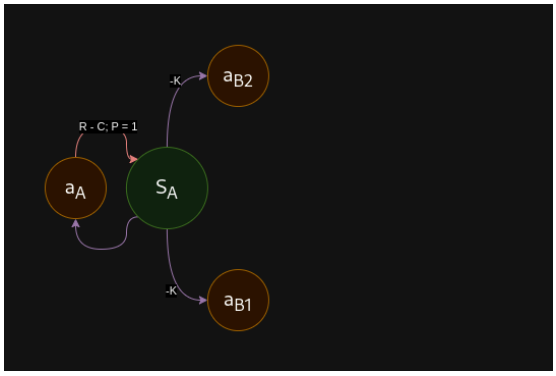

Figure 3: If a validator wishes to change from non-malicious to malicious, they must pay the one-time cost K.

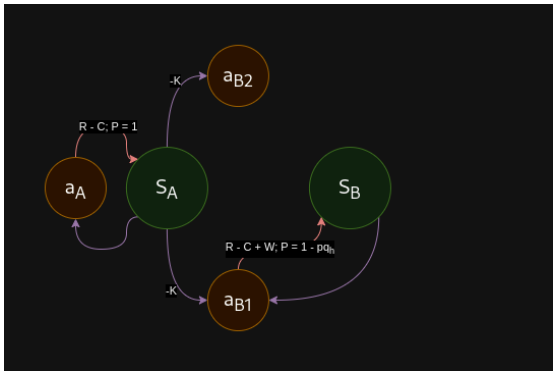

Figure 4: If they chose malicious strategy B1, they may payoff $R - C + W$ with the probability $1 - pq_h$.

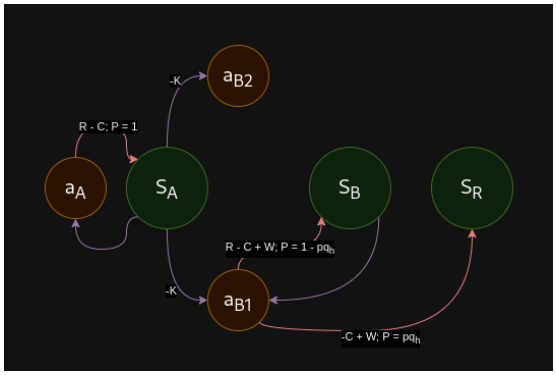

Figure 5: With probability pq_h, their payoff is $-C_1$: they are banned from the network and can't continue without paying S.

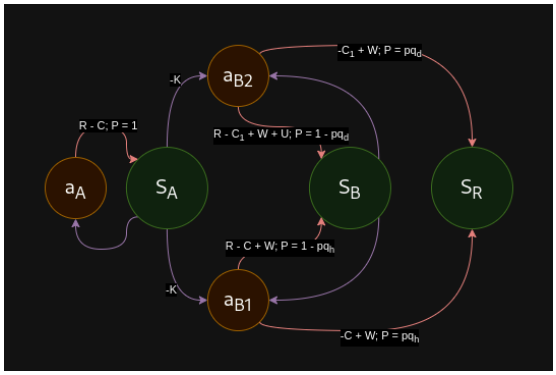

Figure 6: If they chose malicious strategy B2, they may attain payoff $R - C_1 + W + U$, but only with the probability $1 - pq_d$

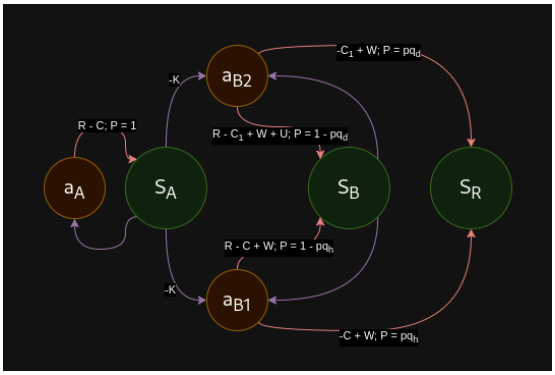

Figure 7: With probability pq_d, their payoff is $-C_1$: they are banned from the network and can't continue without paying S.

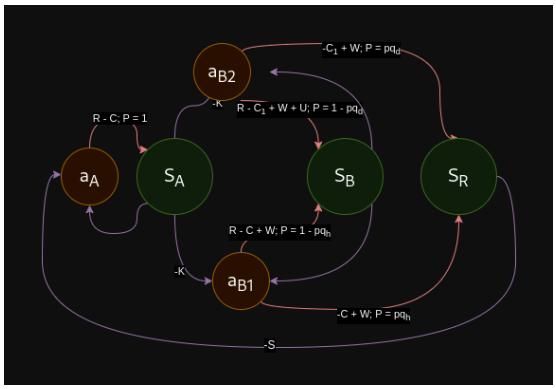

Figure 8: If a banned validator wishes to restart, this will require the operator to replace the device at cost S.