REM: Resource-Efficient Mining for Blockchains

Fan Zhang[⋆,§]
fanz@cs.cornell.edu

Ittay Eyal[⋆,§]
ittay.eyal@cornell.edu

Robert Escriva[⋆]
escriva@cs.cornell.edu

Ari Juels[†,§]
juels@cornell.edu

Robbert van Renesse[⋆,§]
rvr@cs.cornell.edu

[⋆]Cornell University [†]Cornell Tech, Jacobs Institute
[§]Initiative for CryptoCurrencies & Contracts

Abstract

Blockchains show promise as potential infrastructure for financial transaction systems. The security of blockchains today, however, relies critically on Proof-of-Work (PoW), which forces participants to waste computational resources.

We present REM (Resource-Efficient Mining), a new blockchain mining framework that uses *trusted hardware* (Intel SGX). REM achieves security guarantees similar to PoW, but leverages the partially decentralized trust model inherent in SGX to achieve a fraction of the waste of PoW. Its key idea, *Proof-of-Useful-Work* (*PoUW*), involves miners providing trustworthy reporting on CPU cycles they devote to *inherently useful workloads*. REM flexibly allows any entity to create a useful workload. REM ensures the trustworthiness of these workloads by means of a novel scheme of *hierarchical attestations* that may be of independent interest.

To address the risk of compromised SGX CPUs, we develop a statistics-based formal security framework, also relevant to other trusted-hardware-based approaches such as Intel's Proof of Elapsed Time (PoET). We show through economic analysis that REM achieves less waste than PoET and variant schemes.

We implement REM and, as an example application, swap it into the consensus layer of Bitcoin core. The result is the first full implementation of an SGX-based blockchain. We experiment with four example applications as useful workloads for our implementation of REM, and report a computational overhead of $5-15\%$.

1 Introduction

Despite their imperfections [21, 31, 33, 61, 66], blockchains [34, 60, 62] have attracted the interest of the financial and technology industries [11, 20, 30, 41, 64, 69] as a way to build a transaction systems with distributed trust. One fundamental impediments to the widespread adoption of decentralized or "permissionless" blockchains is that Proofs-of-Work (PoWs) in blockchains are wasteful.

PoWs are nonetheless the most robust solution today to two fundamental problems in decentralized cryptocurrency design: How to select consensus leaders and how to apportion rewards fairly among participants. A participant in a PoW system, known as a *miner*, can only lead consensus rounds in proportion to the amount of computation she invests in the system. This prevents an attacker from gaining majority power by cheaply masquerading as multiple machines. The cost, however, is the above-mentioned waste. PoWs serve no useful purpose beyond consensus and incur huge monetary and environmental costs. Today the Bitcoin network uses more electricity than produced by a nuclear reactor, and is projected to consume as much as Denmark by 2020 [25].

We propose a solution to the problem of such waste in a novel block-mining system called REM. Nodes using REM replace PoW's wasted effort with useful effort of a form that we call *Proof of Useful Work* (PoUW). In a PoUW system, users can utilize their CPUs for any desired workload, and can simultaneously contribute their work towards securing a blockchain.

There have been several attempts to construct cryptocurrencies that recycle PoW by creating a resource useful for an external goal, but they have serious limitations. Existing schemes rely on esoteric resources [49], have low recycling rates [58], or are centralized [36]. Other consensus approaches, e.g., BFT or Proof of Stake, are in principle waste-free, but restrict consensus participation or have notable security limitations.

Intel recently introduced a new approach [41] to eliminating waste in distributed consensus protocols that relies instead on trusted hardware, specifically a new instruction set architecture extension in Intel CPUs called *Software Guard Extensions (SGX)*. SGX permits the execution of trustworthy code in an isolated, tamper-free environment, and can prove remotely that outputs represent the result of such execution. Leveraging this capability, Intel's proposed Proof of Elapsed Time (PoET) is an innovative system with an elegant and simple underlying idea. A miner runs a trustworthy piece of code that idles for a randomly determined interval of time. The miner with the first code to awake leads the consensus round and receives a reward. PoET thus promises energy-waste-free decentralized consensus with security predicated on the tamper-proof features of SGX. PoET operates in a *partially-decentralized model*, involving limited involvement of an authority (Intel), as we explain below.

Unfortunately, despite its promise, as we show in this paper, PoET presents two notable technical challenges. First, in the basic version of PoET, an attacker that can corrupt a single SGX-enabled node can win every consensus round and break the system completely. We call this the *broken chip* problem. Second, miners in PoET have a financial incentive to power mining rigs with cheap, outmoded SGX-enabled CPUs used solely for mining. The result is exactly the waste that PoET seeks to avoid. We call this the *stale chip* problem.

REM addresses both the stale and broken chip problems. Like PoET, REM operates in a partially decentralized model: It relies on SGX to prove that miners are generating valid PoUWs. REM, however, avoids PoET's stale chip problem by substituting PoUWs for idle CPU time, disincentivizing the use of outmoded chips for mining. Miners in a PoUW system are thus entities that use or outsource SGX CPUs for computationally intensive workloads, such as scientific experiments, pharmaceutical discovery, etc. All miners can concurrently mine for a blockchain while REM gives them the flexibility to use their CPUs for *any desired workload*.

We present a detailed financial analysis to show that PoUW successfully addresses the stale chip problem. We provide a taxonomy of different schemes, including PoW, PoET, novel PoET variants, and PoUW. We analyze these schemes in a model where agents choose how to invest capital and operational funds in mining and how much of such investment to make. We show that the PoUW in REM not only avoids the stale chip problem, but yields the smallest overall amount of mining waste. Moreover, we describe how small changes to the SGX feature set could enable even more efficient solutions.

Unlike PoET, REM addresses the broken chip problem. Otherwise, compromised SGX-enabled CPUs would allow an attacker to generate PoUWs at will, and both unfairly accrete revenue and disrupt the security of the blockchain [24, 70, 73]. Intel has sought to address the broken chip problem in PoET using a statistical-testing approach, but published details are lacking, as appears to be a rigorous analytic framework. For REM, we set forth a rigorous statistical testing framework for mitigating the damage of broken chips, provide analytic security bounds, and empirically assess its performance given the volatility of mining populations in real-world cryptocurrencies. Our results also apply to PoET.

A further challenge arises in REM due to the feature that miners may choose their own PoUWs workloads. It is necessary to ensure that miner-specified mining applications running in SGX accurately report their computational effort. Unfortunately SGX lacks secure access to performance counters. REM thus includes a *hierarchical attestation* mechanism that uses SGX to attest to compilation of workloads with valid instrumentation. Our techniques, which combine static and dynamic program analysis techniques, are of independent interest.

We have implemented a complete version of REM, encompassing the toolchain that instruments tasks to produce PoUWs, compliance checking code, and a REM blockchain client. As an example use, we swap REM in for the PoW in Bitcoin core. As far as we are aware, ours is the first full implementation of an SGX-backed blockchain. (Intel's Sawtooth Lake, which includes PoET, is implemented only as a simulation.) Our implementation supports trustworthy compilation of any desired workload. As examples, we experiment with four REM workloads, including a commonly-used protein-folding application and a machine learning application. The resulting overhead is about $5-15\%$, confirming the practicality of REM's methodology and implementation.

Paper organization

The paper is organized as follows: Section 2 provides background on proof-of-work and Intel SGX. We then proceed to describe the contributions of this work:

- *PoUW and REM*, a low-waste alternative to PoW that maintains PoW's security properties (§3).
- A *broken-chip countermeasure* consisting of a rigorous statistical testing framework that mitigates the impact of broken chips (§4).
- A *methodology for trustworthy performance instrumentation* of SGX applications using a combination of static and dynamic program analysis and SGX-backed trusted compilation (§5).
- *Design and full implementation of* REM as a resource-efficient PoUW mining system with automatic tools for compiling arbitrary code to a PoUW-compliant module. Ours is the first full implementation of an SGX-backed blockchain protocol (§5).
- A *model of consensus-algorithm resource consumption* that we use to compare the waste associated with various mining schemes. We overview the model and issues with previous schemes (§6) and defer the details to the full version [76].

We discuss related work in §7 and conclude in §8.

2 Background

2.1 Blockchains

Blockchain protocols allow a distributed set of participants, called miners, to reach a form of consensus called Nakamoto consensus. Such consensus yields an ordered list of transactions. Roughly speaking, the process is as follows. Miners collect cryptographically signed transactions from system users. They validate the transactions' signatures and generate blocks that contain these transactions plus a pointer to a parent block. The result is a chain of blocks called (imaginatively) a blockchain.

Each miner, as it generates a block, gets to choose the block's contents, specifically which transactions will be included and in what order. System participants are connected by a peer-to-peer network that propagates transactions and blocks. Occasionally, two or more miners might nearly simultaneously generate blocks that have the same parent, forming two branches in the blockchain and breaking its single-chain structure. Thus a mechanism is used to choose which branch to extend, most simply, the longest chain available [60].[1]

An attacker could naturally seek to generate blocks faster than everyone else, forming the longest chain and unilaterally choosing block contents. To prevent such an attack, a block is regarded as valid only if it contains proof that its creator has performed a certain amount of work, a proof known as a *Proof of Work* (PoW).

A PoW takes the form of a *cryptopuzzle*: In most cryptocurrencies, a miner must change an input (nonce) in the block until a cryptographic hash of the block is smaller than a predetermined threshold. The security properties of hash functions force a miner to test nonces by brute force until a satisfying block is found. Such a block constitutes a solution to the cryptopuzzle and is itself a PoW. Various hash functions are used in practice. Each type puts different load on the processor and memory of a miner's computing device [60, 58, 72].

The process of mining determines an exponentially distributed interval of time between the blocks of an individual miner, and, by extension, between blocks in the blockchain. The expected amount of work to solve a cryptopuzzle, known as its *difficulty*, is set per a deterministic algorithm that seeks to enforce a static expected rate of block production by miners (e.g., 10 minute block intervals in Bitcoin). An individual miner thus generates blocks at a rate that is proportional to its *mining power*, its hashrate as a fraction of that in the entire population of miners. Compensation to miners is granted per block generated, leading to an expected miner revenue that is proportional to the miner's hashrate.

As the mining power that is invested in a cryptocurrency grows, the cryptocurrency's cryptopuzzle difficulty rises to keep the block generation rate stable. When compensation is sufficiently high, it is worthwhile for a large number of participants to mine, leading to a high difficulty requirement. This, in turn, makes it difficult for an attacker to mine a large enough fraction of blocks to perform a significant attack.

PoW properties. The necessary properties for PoW to support consensus in a blockchain, i.e., resist adversarial control, are as follows. First, a PoW must be tied to a unique block, and be valid only for that block. Otherwise, a miner can generate conflicting blocks, allowing for a variety of attacks. A PoW should be moderately hard [10], and its difficulty should be accurately tunable so that the blockchain protocol can automatically tune the expected block intervals. Validation of PoWs, on the other hand, should be as efficient as possible, given that it is performed by the whole network. (In most cryptocurrencies today, it requires just a single hash.) It should also be possible to perform by any entity with access to the blockchain — If the proofs or data needed for validation are made selectively available by a single entity, for instance, that entity becomes a central point of control and failure.[2]

2.2 SGX

Intel Software Guard Extensions (SGX) [39, 40, 42, 43, 8, 37, 57] is a set of new instructions available on recent-model Intel CPUs that confers hardware protections on user-level code. SGX enables process execution in a Trusted Execution Environment (TEE), and specifically in SGX in a protected address space known as an *enclave*. An enclave protects the confidentiality and the integrity of the process from certain forms of hardware attack and other processes on the same host, including privileged processes like operating systems.

An enclave can read and write memory outside the enclave region as a form of inter-process communication, but no other process can access enclave memory. Thus the isolated execution in SGX may be viewed in terms of an ideal model in which a process is guaranteed to execute correctly and with perfect confidentiality, but relies on a (potentially malicious) operating system for supporting services such as I/O, etc. This model is a simplification: SGX is known to expose some internal enclave state to the OS [73]. Our basic security model assumes

[1] There are alternatives to this protocol [33, 52, 68, 72], however the differences are immaterial to our exploration here.

[2] The Bitcoin protocol is expected to soon allow for the so-called segregated witness architecture [17, 55]. Then, transaction signatures (witnesses) are kept in a data structure that is technically separate (segregated) from the blockchain data structure. Despite this separation of data structures, the data in both must be propagated to allow for distributed validation.

ideal isolated execution, but as we detail in Section 4, we have baked a defense against compromised SGX CPUs into REM.

Attestation SGX allows a remote system to verify the software running in an enclave and communicate securely with it. When an enclave is created, the CPU produces a hash of its initial state known as a *measurement*. The software in the enclave may, at a later time, request a report which includes a measurement and supplementary data provided by the process. The report is digitally signed using a hardware-protected key to produce a proof that the measured software is running in an SGX-protected enclave. This proof, known as a *quote*, is part of an *attestation* can be verified by a remote system.

SGX signs quotes in attestations using a group signature scheme called *Enhanced Privacy ID* or *EPID* [67]. This choice of primitive is significant in our design of REM, as Intel made the design choice that attestations can only be verified by accessing Intel's Attestation Service (IAS) [44], a public Web service maintained by Intel whose primary responsibility is to verify attestations upon request.

REM uses attestations as proofs for new blocks, so miners need to access IAS to verify blocks. The current way in which IAS works forces miners to access IAS on every single verification, adding an undesirable round-trip time to and from Intel's server to the block verification time. This overhead, however, is not inherent, and is due only to a particular design choice by Intel. As we suggest in Section 5.4, a simple modification, to the IAS protocol, which Intel is currently testing, can eliminate this overhead entirely.

Randomness As operating systems sit outside of the trusted computing base (TCB) of SGX, OS-served random functions such as `srand` and `rand` are not accessible to enclaves. SGX instead provides a hardware-protected random number generator (RNG) using the `rdrand` instruction. REM relies on the SGX RNG.

3 Overview of PoUW and REM

The basic idea of PoUW, and thus REM, is to replace the wasteful computation of PoW with arbitrary useful computation. A miner proves that a certain amount of useful work has been dedicated to a specific branch of the blockchain. Intuitively, due to the value of the useful work outside of the context of the blockchain supported by REM, the hardware and power are well spent, and there is no waste. A comprehensive analysis of the waste is deferred to the full version [76]. Here we describe the security model of REM and then give an overview of its system mechanics.

3.1 Security Model

A PoW solution embodies a statistical proof of an effort spent by the miner. With PoUW, however, a miner reports its own effort. The rational miner's incentive is to lie, report more work than actually performed, and monopolize the blockchain. In PoUW / REM, use of a TEE — Intel SGX in particular — prevents such attacks and enforces correct reporting of work. The resulting trust model is starkly different from that in traditional PoW.

PoET introduced, and we similarly use in REM, a *partially decentralized* blockchain model. The blockchain is *permissionless*, i.e., any entity can participate as a miner, as in a fully decentralized blockchain such as Bitcoin. It is only partially decentralized, though, in that it relies for security on two key assumptions about the hardware manufacturer's behavior.

First, we must assume that Intel correctly manages identities, specifically that it assigns a signing key (used for attestations) only to a valid CPU. It follows that Intel does not forge attestations and thus mining work. Such forgery, if detected in any context, would undermine the company's reputation and the perceived utility of SGX, costing far more than potential blockchain revenue. Second, we assume that Intel does not blacklist valid nodes in the network, rendering their attestations invalid when the IAS is queried. Such misbehavior would be publicly visible and similarly damaging to Intel if unjustified.

Even assuming trustworthy manufacturer behavior, though, a limited number of individual CPUs might be physically or otherwise compromised by a highly resourced adversary (or adversaries). Our trust model assumes the possibility of such an adversary and makes the strong assumption that she can learn the attestation (EPID signing) key for compromised machines and thus can issue arbitrary attestations for those machines. In particular, as we shall see, she can falsify random number generation and lie about work performed in REM.

Even this strong adversary, though, does have a key limitation: As signing keys are issued by the manufacturer, and given our first assumption above, it is not possible for an adversary to forge identities. We further assume that the signatures are linkable. In SGX, the EPID signature scheme for attestations has a linkable (pseudonymous) mode [44, 8, 67], which permits anyone to determine whether two signatures were generated by the same CPU. As a result, event a compromised node cannot masquerade as multiple nodes.

Outside the REM security model It is important to note that REM is a *consensus framework*, i.e., a means to generate blocks, not a *cryptocurrency*. REM can be integrated into a cryptocurrency, as we show by swapping it into the Bitcoin consensus layer. As REM has roughly the same exponentially distributed block-production interval, such integration need not change security properties above the consensus layer. For example, fork resolution, transaction validation, block propagation, etc., *remain the same in a REM-backed blockchain as in a PoW-based one*. Thus we do not expand the discussion of the security issues relevant to those elements in the REM security model.

3.2 REM overview

Figure 1 presents an architectural overview of REM.

There are three types of entities in the ecosystem of REM: A *blockchain agent*, one or more *REM miners*, and one or more *useful work clients*.

The useful work clients supply useful workloads to REM miners in the form of *PoUW tasks*, each of which encompass a *PoUW enclave* and some *input*. Any SGX-compliant program can be transformed into a PoUW enclave using the toolchain we developed. Note that a PoUW enclave has to conform to certain security requirements. The most important is that it meters effort correctly, something that can be efficiently verified by a compliance checker and a novel technique we introduce called *hierarchical attestation*. We refer readers to §5.2 and §5.3 for details.

The blockchain agent collects transactions and generates a block template, a block lacking the proof of useful work (PoUW). As detailed later, a REM miner will attach the required PoUW and return it to the agent. The agent then publishes the full block to the P2P network, making it part of the blockchain and receiving the corresponding reward.

A miner takes as input a block template and a PoUW task to produce PoUWs. It launches the PoUW enclave in SGX with the prescribed input and block template. Once the PoUW task halts, its results are returned to the useful work client. The PoUW enclave meters work performed by the miner and declares whether the mining effort is successful and results in a block. Effort is metered on a per-instruction basis. The PoUW enclave randomly determines whether the work results in a block by treating each instruction as a Bernoulli trial. Thus mining times are distributed in much the same manner as in proof-of-work systems. While in, e.g., Bitcoin, effort is measured in terms of executed hashes, in REM, it is the number of executed useful-work instructions. Intuitively, REM may be viewed as *simulating* the distribution of block-mining intervals associated with PoW, but REM does so with PoUW, and thus eliminates wasted CPU effort.

When a PoUW enclave determines that a block has been successfully mined, it produces a PoUW, which consists of two parts: an SGX-generated attestation demonstrating the PoUW enclave's *compliance* with REM and another attestation that a block was successfully mined by the PoUW enclave at a given *difficulty parameter*. The blockchain agent concatenates the PoUW to the block template, forming a full block, and publishes it to the network.

When a blockchain participant verifies a fresh block received on the blockchain network, in addition to verifying higher-layer properties (e.g., in a cryptocurrency such as Bitcoin, that transactions, previous block references, etc., are valid), the participant verifies the attestations in the associated PoUW.

Intel's PoET scheme looks similar to REM in that its enclave randomly determines block intervals and attests to block production. PoET, however, lacks the production of useful work, an essential ingredient, as we explain later in the paper. We now discuss our strategy in REM for handling compromised nodes.

4 Tolerating Compromised SGX Nodes

SGX does not achieve perfect enclave isolation. While no real practical attack is known, researchers have demonstrated potentially dangerous side-channel attacks against applications [73] and even expressed concerns about whether an attestation key might be extracted [24].

Therefore, even if we assume SGX chips are manufactured in a secure fashion, some number of individual instances could be broken by well-resourced adversaries. A single compromised node could be catastrophic to an SGX-based cryptocurrency, allowing an adversary to create blocks at will and perform majority attacks on the blockchain. While she could not spend other people's money, which would require access to their private keys, she could perform denial-of-service attacks, selectively drop transactions, or charge excessive transaction fees.

In principle, a broken attestation key can be revoked through the Intel Attestation Service (IAS), but this can only happen if the break is detected to begin with. Consequently, Intel has explored ways of detecting SGX compromise in PoET [6] by statistically testing for implausibly frequent mining by a given node (using a "z-test"). Details are lacking in published materials, however, and a rigorous analytic framework seems to be needed.

For REM, we explore compromise detection within a rigorous definitional and analytic framework. The centerpiece is what we call a *block-acceptance policy*, a flexibly defined rule that determines whether a proposed block in a blockchain is legitimate. As we show, defining and analyzing policies rigorously is challenging, but we provide strong analytical and empirical evidence that a relatively simple statistical-testing policy (which we denote P_{stat}) can achieve good results. P_{stat} both limits an

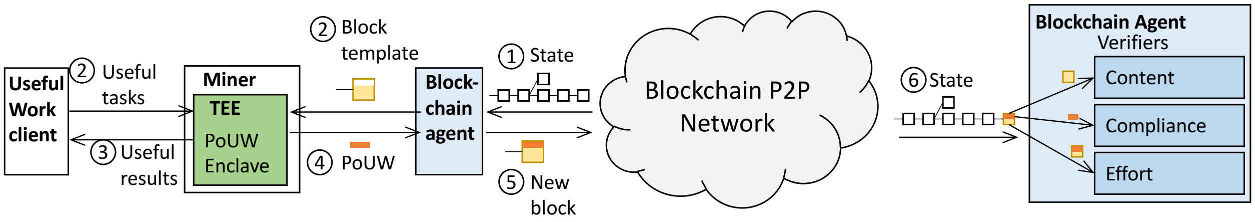


Figure 1: Architecture overview of REM

adversary's ability to harvest blocks unfairly and minimizes erroneous rejection of honestly mined blocks.

4.1 Threat Model and Definitions

4.1.1 Basic notation

To model block-acceptance policies, let $M = \{m_1, \cdots, m_n\}$ be the set of all miners, which we assume to be static. (Miners can join and leave the system; M includes all potential miners.) An adversary $\mathcal{A}$ controls a static subset $M_\mathcal{A} \in M$, where $|M_\mathcal{A}| = k$. $\text{rate}(m_i)$ specifies the *mining rate* of m_i, the number of mining operations per unit time it performs.

We define a candidate *block* to be a tuple $B = (t, m, d)$, where t is a timestamp, $m \in M$ the identity of the CPU that mines the block, and d is the block difficulty. Difficulty d is defined as the win probability per mining operation in the underlying consensus protocol (e.g. a hash in Bitcoin, a unit time of sleep in PoET, an instruction in PoUW). $\mathcal{B}$ denotes the set of possible blocks B.

A *blockchain* is an ordered sequence of blocks. At time τ, blockchain $C(\tau)$ is a sequence of accepted blocks $C(\tau) = \{B_1, B_2, \ldots, B_n\}$ for some n. We drop τ where its clear from context. We let $r(\tau)$ denote the number of rejected blocks of honest miners, i.e., miners in $M - M_\mathcal{A}$, in the history of $C(\tau)$. (Of course, $r(\tau)$ is not and indeed cannot be recorded in a real blockchain system.) Let $\mathcal{C}$ be the space of all possible blockchains C. Let C_m denote blockchain C restricted to blocks mined by miner $m \in M$.

In REM, a blockchain-acceptance policy is used to determine whether a block appears to come from a legitimate miner (CPU that hasn't been compromised).

Definition 1. *(Blockchain-Acceptance Policy) A blockchain-acceptance policy (or simply policy) $P : \mathcal{C} \times \mathcal{B} \rightarrow \{\text{reject}, \text{accept}\}$ is a function that takes as input a blockchain and a proposed block, and outputs whether the proposed block is legitimate.*

4.1.2 Security and efficiency definitions

We model the consensus algorithm for the blockchain, the adversary $\mathcal{A}$, and honest miners respectively as (ideal) programs prog_chain, $\text{prog}_\mathcal{A}$, and prog_m. Together, they define what we call a *security game* $S(P)$ for a particular policy P.

We define security games and their constituent programs formally in Appendix A.2. Where clear from context in what follows, we use the notation S, rather than $S(P)$, i.e., omit P.

A security game S may itself be viewed as a probabilistic algorithm. Thus we may treat the blockchain resulting from execution of S for interval of time τ as a random variable $C_S(\tau)$.

Normalizing the revenue from mining a block to 1, we define the *payoff* for a miner m for a given blockchain C as $\pi_m(C) = |C_m|$.

An adversary $\mathcal{A}$ seeks to maximize payoffs for its miners, as reflected in the following definition:

Definition 2. *(Advantage of $\mathcal{A}$). For a given security game S, the advantage of $\mathcal{A}$ for time τ is:*

$$\text{Adv}^S_\mathcal{A}(\tau) = \frac{\mathbb{E}[\pi_{\hat{m}}(C_S(\tau))]}{\max_{m_j \in M - M_\mathcal{A}} \mathbb{E}[\pi_{m_j}(C_S(\tau))]},$$

for any $\hat{m} \in M_\mathcal{A}$. Note that $\mathbb{E}[\pi_{\hat{m}}(C_S(\tau))]$ is equal for all such $\hat{m}$, as they all use strategy $\Sigma_\mathcal{A}$ and can emit blocks as frequently as desired (ignoring $rate(\hat{m})$).

A policy that keeps $\text{Adv}^S_\mathcal{A}(\tau)$ low is desirable, but there's a trade-off. A policy that rejects too many policies incurs high *waste*, meaning that it rejects many blocks from honest miners. We define waste as follows.

Definition 3. *(Waste of a policy). For a given blockchain $C(\tau) = \{(B_1, B_2, \ldots, B_n)\}$, the waste is defined as*

$$Waste(C(\tau)) = \frac{r(\tau)}{n + r(\tau)}.$$

For security game S, the waste at time τ is defined as

$$Waste^S(\tau) = \mathbb{E}[Waste(C_S(\tau))].$$

Our exploration of policies focuses critically on the trade-offs between low $\text{Adv}^S_\mathcal{A}(\tau)$ and low $\text{Waste}^S(\tau)$. To illustrate the issue, we give a simple example in Appendix A.3 of a policy that allows any CPU to mine only one block over its lifetime. As $\tau \rightarrow \infty$, it achieves the optimal $\text{Adv}^S_\mathcal{A}(\tau) = 1$, but at the cost of $\text{Waste}^S(\tau) = 1$, i.e., 100% waste.

```
$P_{\text{stat}}^{\alpha,\mathsf{rate}_{\text{best}}}(C,B)$:
  parse $B \rightarrow (\tau,m,d)$
  if $|C_m| > F^{-1}(1-\alpha, d\tau(\mathsf{rate}_{\text{best}}))$:
    output reject
  else
    output accept
```

Figure 2: P_{stat}^{α}. $F^{-1}(\cdot,\lambda)$ is the quantile function for Poisson distribution with rate λ.

4.2 The REM policy: P_{stat}

REM makes use of a statistical-testing-based policy that we denote by P_{stat}. P_{stat} is compatible not just with PoUW, but also with PoET and potentially other SGX-based mining variants.

There are two parts to P_{stat}. First, P_{stat} estimates the rate of the fastest honest miner(s) (fastest CPU type), denoted by $\mathsf{rate}_{\text{best}} = \max_{m \in M - M_{\mathcal{A}}} rate(m)$. There are various ways to accomplish this; a simple one would be to have an authority (e.g., Intel) publish specs on its fastest CPUs' performance. (In PoET, mining times are uniform, so $\mathsf{rate}_{\text{best}}$ is just a system parameter.) We describe an empirical approach to estimating $\mathsf{rate}_{\text{best}}$ in REM in Appendix A.1.

Given an estimate of $\mathsf{rate}_{\text{best}}$, P_{stat} tests submitted blocks statistically to determine *whether a miner is mining blocks too quickly and may thus be compromised*. The basic principle is simple: On receiving a block B from miner m, P_{stat} tests the null hypothesis

$$H_0 = \{rate(m) \leq \mathsf{rate}_{\text{best}}\}.$$

We use $|C_m(\tau)|$, the number of blocks mined by m at time τ, as the test statistic. Under H_0, $|C_m|$ should obey a Poisson distribution with rate $d\tau(\mathsf{rate}_{\text{best}})$, denoted as $\text{Pois}[d\tau(\mathsf{rate}_{\text{best}})]$. P_{stat} rejects H_0 if $|C_m|$ is greater than the $(1-\alpha)$-quantile of the Poisson distribution. The false rejection rate for a single test is therefore at most α. We specify P_{stat} (for a given $\mathsf{rate}_{\text{best}}$) in Figure 2.

An important property that differentiates P_{stat} from canonical statistical tests is that P_{stat} repeatedly applies a given statistical test to an accumulating history of samples. *The statistical dependency between samples makes the analysis non-trivial, as we shall show.*

4.3 Analysis of P_{stat}

We now analyze the average-case and worst-case waste and adversarial advantage of P_{stat}. We assume for simplicity that $\mathsf{rate}_{\text{best}}$ is accurately estimated. We remove this assumption in the worst-case analysis below. We also assume that the difficulty $d(t)$ is stationary over the period of observation.

Waste Under P_{stat}, a miner generates blocks according to a Poisson process; whether a block is accepted or rejected depends on whether the miner has generated more blocks than a time-dependent threshold. This process is obviously not memoryless and thus not directly representable as a Markov process. We can, however, achieve a close approximation using a discrete-time Markov chain. Indeed, as we show, we can represent waste in P_{stat} using a discrete-time Markov chain that is *periodically identical* to the process it models, meaning that its expected waste is identical at any time $n\tau$, for $n \in \mathbb{Z}^+$ and τ a model parameter specified below. This Markov chain has a stationary distribution that yields an expression upper-bounding waste in P_{stat}. (We believe, and the periodic identical property suggests, that this bound is very tight.)

To construct the Markov Chain, we partition time into intervals of length τ; we regard each such interval as a discrete timestep. Assuming that all honest miners mine at rate $rate$, let $\lambda = d\tau(rate)$. Thus an honest miner generates an expected $\text{Pois}[\lambda]$ blocks in a given timestep i, which we may represent as a random variable Y_i. Without loss of generality, we may set $\tau = 1/(d \times rate)$ and thus λ = 1 and $E[\text{Pois}[\lambda]] = 1$.

We represent the state of an honest miner at timestep n by a random variable $X_n = \sum_{i=1}^{n}(Y_i - E[Y_i]) = (\sum_{i=1}^{n} Y_i) - n$. Thus $X_n \in \mathbb{Z}$ is simply difference between the miner's actually mined blocks and the expected number.

Our Markov chain consists of a set of states $C = \mathbb{Z}$ representing possible values of X_n (we use the notation C here, as states represent $|C_m|$ for an honest miner m). Figure 3 gives a simple example of such a chain (truncated to only four states).

Our statistical testing regime may be viewed as rejecting blocks when a transition is made to a state whose value is above a certain threshold $thresh$. We denote the set of such states $C_{\text{rej}} = \{j \mid j \geq thresh\} \in C$ and depict corresponding nodes visually in our example in Figure 3 as red. P_{stat} sets $thresh$ according to the statistical-testing regime we describe above and a desired false-rejection (Type-I) parameter α. Specifically,

$$C_{\text{rej}}[\alpha] = \{j \in \mathbb{Z} \mid j \geq F^{-1}(1-\alpha, \tau \times rate)\}. \qquad (1)$$

The transition probabilities in our Markov chain are:

$$P[i \rightarrow j \mid i \in C \setminus C_{\text{rej}}[\alpha]] = \begin{cases} P(j-i+1) & \text{if } j \geq i-1 \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

$$P[i \rightarrow j \mid i \in C_{\text{rej}}[\alpha]] = \begin{cases} P(j+1) & \text{if } j \leq -1 \\ 0 & \text{otherwise.} \end{cases} \qquad (3)$$

An example of transitions is given in Figure 3. For instance, from state -1, the next state can be -2 if the

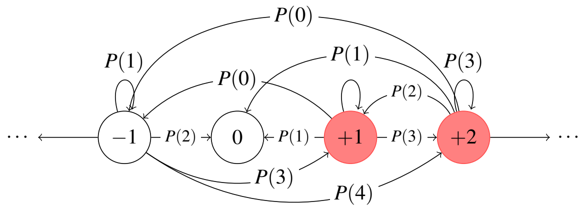
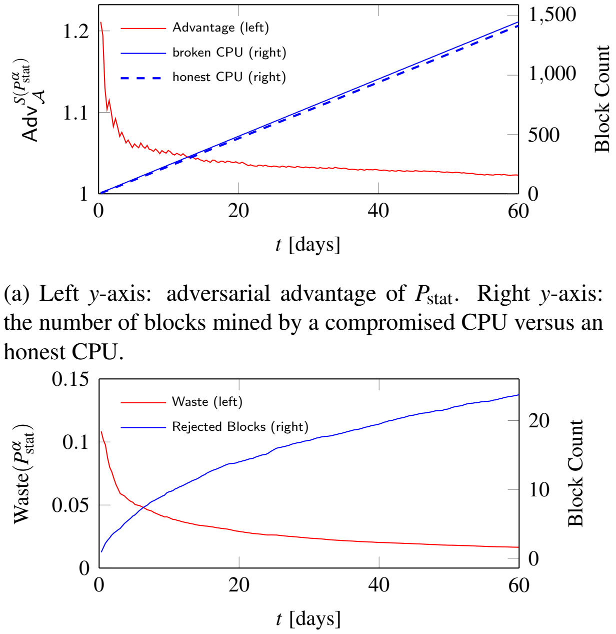


Figure 3: Markov chain with states C representing P_{stat}. Red nodes show the rejection set $C_{\text{rej}} = \mathbb{Z}^+$, i.e., *thresh* $= 1$. Outgoing edges from 0 are omitted for clarity.



(a) Left y-axis: adversarial advantage of P_{stat}. Right y-axis: the number of blocks mined by a compromised CPU versus an honest CPU.

(b) Left y-axis: the waste of P_{stat}. Right y-axis: the number of rejected blocks.

Figure 4: 60-day simulation of P_{stat}. The fastest honest CPU mines one block per hour. The Markov chain analysis yields a long-term advantage upper bound of 1.006 and waste of 0.006.

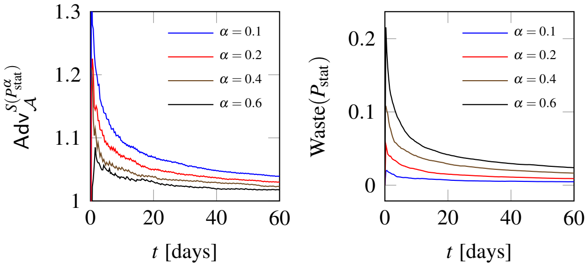
miner doesn't produce any block in this step with probability $P(0)$, or state $-2+i$ if the miner produces $i+1$ blocks in this step, thus with probability $P(i+1)$.

Finally, an upper bound on the false rejection rate can be derived from the stationary probabilities of the Markov chain. Letting $q(s)$ denote the stationary probability of state s,

$$\text{Waste}(P_{\text{stat}}^{\alpha}) = \sum_{s \in C_{\text{rej}}[\alpha]} s q(s). \tag{4}$$

We compare our analytic bounds with simulation results in below.

Adversarial Advantage We denote by Σ_{stat} the strategy of an adversary that publishes blocks as soon as they will be accepted by P_{stat}. In Appendix A.4, we show the following:



(a) The adversarial advantage of P_{stat} under different α

(b) The waste of P_{stat} under different α

Figure 5: 60-day simulation of P_{stat}, under various α. The fastest honest CPU mines an expected one block per hour.

Theorem 1. *In a (non-degenerate) security game S where $\mathcal{A}$ uses strategy Σ_{stat},*

$$\mathsf{Adv}_{\mathcal{A}}^{S(P_{stat}^{\alpha})} = \frac{1}{1 - Waste(P_{stat}^{\alpha})}.$$

Simulation We simulate P_{stat} to explore its efficacy in both the average case and the worst case. Figure 4 shows the result of 1000 runs of a 60-day mining period simulation under P_{stat}. We set $\alpha = 0.4$. We present statistics with respect to the fastest (honest) CPU in the system, which for simplicity we assume mines one block per hour in expectation and refer to simply as "the honest miner." The adversary uses attack strategy Σ_{stat}.

In Figure 4a, the solid blue line shows the average aggregate number of blocks mined by the adversary, and the dashed one those of the honest miner. The attacker's advantage is, of course, the ratio of the two values. Initially, the adversary achieves a relatively high advantage ($\approx 127\%$), but this drops below 110% within 55 blocks, and continues to drop. Our asymptotic analytic bound on waste (given below) implies an advantage of 100.6%.

Figure 4b shows the average waste of P_{stat} and absolute number of rejected blocks. The waste quickly drops below 10%. As blocks accumulate, the statistical power of P_{stat} grows, and the waste drops further. Analytically, we obtain $\text{Waste}(P_{\text{stat}}^{\alpha}) = 0.006$, or 0.6% from Eqn. 4.

Setting α Setting the parameter α imposes a trade-off on system implementers. As noted, α corresponds to the Type-I error for a single test in P_{stat}. As P_{stat} performs continuous testing, however, a more meaningful security measure is $\text{Waste}(P_{\text{stat}}^{\alpha})$, the rate of falsely rejected blocks. Similarly there is no notion of Type-II error—particularly, as our setting is adversarial. $\mathsf{Adv}_{\mathcal{A}}^{S(P_{\text{stat}}^{\alpha})}$ captures the corresponding notion in REM. As shown in Figure 5, raising α results in a lower $\mathsf{Adv}_{\mathcal{A}}^{S(P_{\text{stat}}^{\alpha})}$, but higher $\text{Waste}(P_{\text{stat}}^{\alpha})$, and vice versa.

Algorithm 1: Miner Loop. The green highlighted line is executed in a TEE (e.g., an SGX enclave).

```
while True do
    template ← read from blockchain agent
    hash, difficulty ← process(template)
    task ← get from useful work client
    outcome, PoUW ← TEE(task, hash, difficulty)
    send outcome to useful work client
    if $PoUW \neq \perp$ then
        block ← formBlock(template, PoUW)
        send block to blockchain agent
```

5 Implementation Details

We have implemented a full REM prototype using SGX (§5.1), and as an example application swapped REM into the consensus layer of Bitcoin-core [18]. We explain how we implemented secure instruction counting (§5.2), and our hierarchical attestation framework (§5.3) that allows for arbitrary tasks to be used for work. We explain how to reduce the overhead of attestation due to SGX-specific requirements (§5.4). Finally (§5.5) we present two examples of PoUW and evaluate the overhead of REM.

5.1 Architecture

Figure 1 shows the architecture of REM. As discussed in §3.2, the core of REM is a miner program that does useful work and produces PoUWs. Each CPU instruction executed in the PoUW is analogous to one hash function computation in PoW schemes. That is, each instruction has some probability of successfully mining a block, and if the enclave determines this is the case, it produces a proof — the PoUW.

Pseudocode of the miner's iterative algorithm is given in Algorithm 1. In a given iteration, it first takes a block template from the agent and calculates the previous block's hash and difficulty. Then it reads the task to perform as useful work. Note that the enclave code has no network stack, therefore it receives its inputs from the miner untrusted code and returns its outputs to the miner untrusted code. The miner calls the TEE (SGX enclave) with the useful task and parameters for mining, and stores the result of the useful task. It also checks whether the enclave returned a successful PoUW; if so, it combines the agent-furnished template and PoUW into a legal block and sends it to the agent for publication. In REM, the miner untrusted layer is implemented as a Python script using RPC to access the agent.

To securely decide whether an instruction was a "winning" one, the PoUW enclave does the equivalent of generating a random number and checking whether it is smaller than value `target` that represents the desired system-wide block rate, i.e., difficulty. For this purpose, it uses SGX's random number generator (SRNG). However, calling the SRNG and checking for a win after every single instruction would impose prohibitive overhead. Instead, we batch instructions by dividing useful work into subtasks of short duration compared to the inter-block interval (e.g. 10 second tasks for 10 minute average block intervals). We let each such subtask run to completion, and count its instructions. The PoUW enclave then calls the SRNG to determine whether at least one of the instructions has won, i.e., it checks for a result less than `target`, weighted by the total number of executed instructions. If so, the enclave produces an attestation that includes the input block hash and difficulty.

Why Count Instructions While instructions are reasonable estimates of the CPU effort, CPU cycles would have been a more accurate metric. However, although cycles are counted, and the counts can be accessed through the CPU's performance counters, they are vulnerable to manipulation. The operating system may set their values arbitrarily, allowing a rational operator, who controls her own OS, to improve her chances of finding a block by faking a high cycle count. Moreover, counters are incremented even if an enclave is swapped out, allowing an OS scheduler to run multiple SGX instances and having them double-count cycles. Therefore, while instruction counting is not perfect, we find it is the best method for securely evaluating effort with the existing tools available in SGX.

5.2 Secure Instruction Counting

As we want to allow arbitrary useful work programs, it is critical to ensure that instructions are counted correctly even in the presence of malicious useful work programs. To this end, we adopt a hybrid method combining static and dynamic program analysis. We employ a customized toolchain that can instrument any SGX-compliant code with dynamic runtime checks implementing secure instruction counting.

Figure 6 shows the workflow of the PoUW toolchain. First, the useful work code (`usefulwork.cpp`), C / C++ source code, is assembled while reserving a register as the instruction counter. Next, the assembly code is rewritten by the toolchain such that the counter is incremented at the beginning of each basic block (a linear code sequence with no branches) by the number of instructions in that basic block. In particular, we use the `LEA` instruction to perform incrementing for two reasons. First, it completes in a single cycle, and second, it doesn't change flags and therefore does not affect con-

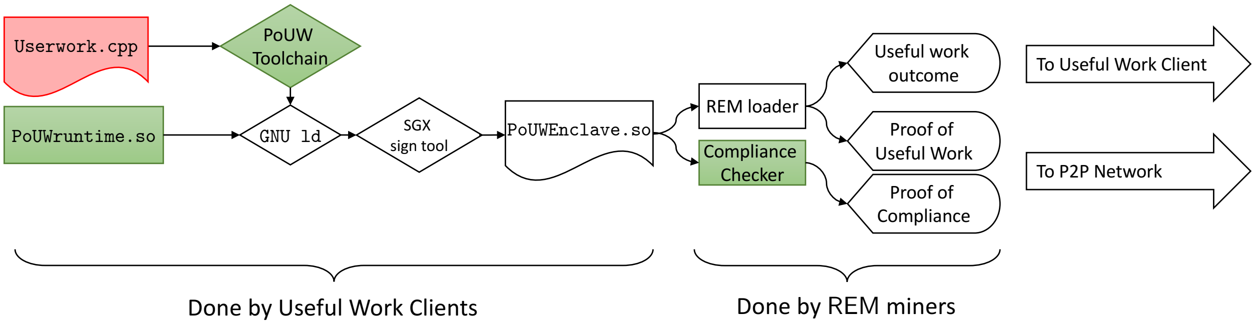


Figure 6: REM toolchain to transfer a useful work to an PoUW-ready program. Everything in the diagram has been implemented besides existing tools such as ld and SGX signing tool.

Algorithm 2: PoUW Runtime

```
Function TEE(task, hash, diff)
    $outcome, n := task.run()$
    $win := 0$
    $PoUW := \perp$
    /* simulating $n$ Bernoulli tests */
    $l \leftarrow \mathcal{U}[0,1]$                /* query SGX RNG */
    if $l \geq 1 - (1 - diff)^n$ then
        $PoUW = \Sigma_{\text{intel}}[hash \mid diff \mid 1]$
    return outcome, PoUW
```

```
...
.LEHB0:
    leaq    1(%r15), %r15   # added by PoUW
    call    _ZN11stlpmtx_std12basic_stringIcNS...
.LEHE0:
    .loc 7 70 0 is_stmt 0 discriminator 2
    leaq    3(%r15), %r15   # added by PoUW
    leaq    -80(%rbp), %rax #, tmp94
    movq    %rax, %rsi  # tmp94,
    movq    %rbx, %rdi  # _4,
.LEHB1:
    leaq    1(%r15), %r15   # added by PoUW
    call    _ZN11stlpmtx_std12out_of_rangeC1ER...
.LEHE1:
...
```

Figure 7: A snippet of assembly code instrumented with the REM toolchain. Register r15 is the reserved instruction counter; it is incremented at the beginning of each basic block in the lines commented added by PoUW.

ditional jumps. The count is performed at the beginning of a block rather than its end to prevent a cheater from jumping to the middle of a block and gaining an excessive count.

Another challenge is to ensure the result of instruction counting is used properly—we cannot rely on the useful work programs themselves. The solution is to wrap the useful work with a predefined, trusted PoUW runtime, and make sure to the enclave can only be entered through the PoUW runtime. The logic of PoUW runtime is summarized in Algorithm 2, and it is denoted as PoUWruntime.so in Figure 6. The PoUW runtime serves as an "in-enclave" loader that launches the useful work program with proper input and collects the result of instruction counting. It takes the block hash and difficulty and starts mining by running the mining program. Once the mining program returns, the PoUW runtime extracts the instruction counter from the reserved register. Then it draws a random value from SRNG and determines whether a new block should be generated, based on the instruction counter and the current difficulty. If a block should be generated, the PoUW runtime produces an attestation recording the template hash that it is called with and the difficulty.

The last step of the toolchain is to compile the resultant assembly and link it (using linker GNU ld) with the PoUW runtime (PoUWruntime.so), to produce the PoUW enclave. Figure 7 shows a snippet of instrumented assembly code. This PoUW enclave is finally signed by an Intel SGX signing tool, creating an application PoUWEnclave.so that is validated for loading into an enclave.

The security of instruction counting relies on the assumption that once instrumented, the code cannot alter its behavior. To realize this assumption in SGX, we need to require two invariants. First, code pages must be non-writable; second, the useful work program must be single threaded.

Enforcing Non-Writable Code Pages Writable code pages allow a program to rewrite itself at runtime. Although necessary in some cases (e.g. JIT), writable code opens up potential security vulnerabilities. In particular, writable code pages are not acceptable in REM because they would allow a malicious useful work program to easily bypass the instrumentation. A general memory protection policy would be to require code pages to have W⊕X permission, namely to be either writable or executable, but not both. However, W⊕X permissions are

```
.section data
ENCLAVE_MTX:
    .long 0

.section text
...
enclave_entry:
    xor %rax, %rax
    xchgl ENCLAVE_MTX(%rip), %rax
    cmp %rax, 0
    jnz enclave_entry
```

Figure 8: Code snippet: a spinlock to allow only the first thread to enter `enclave_entry`

not enforced by the hardware. Intel has in fact acknowledged this issue [5] and recommended that enclave code contain no relocation to enable the W⊕X feature.

REM thus explicitly requires code pages in the enclave code (`usefulwork.so`) to have W⊕X permission. This is straightforward to verify, as with the current implementation of the SGX loader, code page permissions are taken directly from the ELF program headers [4].

Enforcing Single Threading Another limitation of SGX is that the memory layout is largely predefined and known to the untrusted application. For example, the State Save Area (SSA) frames are a portion of stack memory that stores the execution context when handling interrupts in SGX. This also implies that the SSA pages have to be writable. The address of SSA frames for an enclave is determined at the time of initialization, as the Thread Control Structure (TCS) is loaded by the untrusted application through an EADD instruction. In other words, the address of SSA is always known to the untrusted application. This could lead to attacks on the instruction counting if a malicious program has multiple threads that interact via manipulation of the execution context in SSA. For example, as we will detail later, REM stores the counter in one of the registers. When one thread is swapped out, the register value stored in an SSA is subject to manipulation by another thread.

While more complicated techniques such as Address Space Layout Randomization (ASLR) for SGX could provide a general answer to this problem, for our purposes it suffices to enforce the condition that an enclave can be launched by at most one thread. As an SGX enclave has only one entry point, we can instrument the code with a spinlock to allow only the first thread to pass, as shown in Figure 8.

Known entry points REM expects the PoUW toolchain and compliance checker to provide and verify a subset of Software Fault Isolation (SFI), specifically indirect control transfers alignment [26, 53, 74, 38]. This ensures that the program can only execute the instruction stream parsed by the compliance checker, and not jump to the middle of an instruction to create its own alternate execution that falsifies the instruction count. Our implementation does not include SFI, as off the shelf solutions such as Google's Native Client could be integrated with the PoUW toolchain and runtime with well quantified overheads [74].

5.3 Hierarchical Attestation

A blockchain participant that verifies a block has to check whether the useful work program that produced the block's PoUW followed the protocol and correctly counted its instructions. SGX attestations require such a verifier to obtain a fingerprint of the attesting enclave. As we allow arbitrary work, a naïve implementation would store all programs on the blockchain. Then a verifier that considers a certain block would read the program from the blockchain, verify it correctly counts instructions, calculate its fingerprint, and check the attestation. Beyond the computational effort, just placing all programs on the blockchain for verification would incur prohibitive overhead and enable DoS attacks via spamming the chain with overly large programs. The alternative of having an entity that verifies program compliance is also unacceptable, as it puts absolute blockchain control in the hands of this entity: it can authorize programs that deterministically win every execution.

To resolve this predicament, we form PoUW attestations with what we call *two-layer hierarchical attestations*. We hard-code only a single program's fingerprint into the blockchain, a static-analysis tool called *compliance checker*. The compliance checker runs in a trusted environment and takes a user-supplied program as input. It validates that it conforms with the requirements defined above. First, it confirms the `text` section is non-writable. Then it validates the work program's compliance by disassembling it and confirming that the dedicated register is reserved for instruction counting and that counts are correct and appear where they should. Next, it verifies that the PoUW runtime is correctly linked and identical to the expected PoUW runtime code. Finally, it verifies the only entry point is the PoUW runtime and that this is protected by a spinlock as shown in Figure 8. Finally, it calculates the program's fingerprint and outputs an attestation including this fingerprint.

Every PoUW then includes two parts: The useful work program attestation on the mining success, and an attestation from the compliance checker of the program's compliance (Figure 9). Note that the compliance attestation and the program's attestation must be signed by the same CPU. Otherwise an attacker that compromises a single CPU could create fake compliance attestations for invalid tasks. Such an attacker could then create blocks at

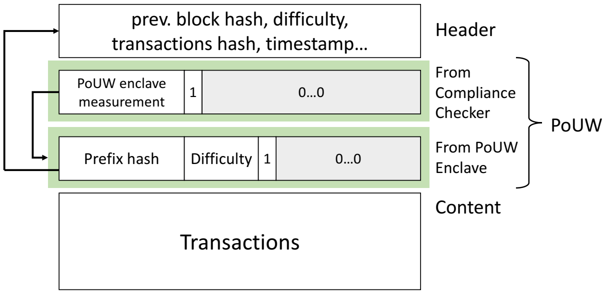


Figure 9: Block structure with a proof comprising the quotes from the compliance enclave and a work enclave.

will from different uncompromised CPUs, circumventing the detection policy of Section 4.

In summary, the compliance enclave is verified through the hard-coded measurement in the blockchain agent. Its output is a measurement that should be identical to the measurement of the PoUW enclave `PoUWEnclave.so`. PoUW Enclave's output should match the block template (namely the hash of the block prefix, up to the proof) and the prescribed difficulty.

Generalized Hierarchical Attestation The hierarchical attestation approach can be useful for other scenarios where participants need to obtain attestations to code they do not know in advance. As a general approach, one hard-codes the fingerprint of a *root compliance checker* that verifies its children's compliance. Each of them, in turn, checks the compliance of its children, and so on, forming a tree. The leaves of the tree are the programs that produce the actual output to be verified. A hierarchical attestation therefore comprises a leaf attestation and a path to the root compliance checker. Each node attests the compliance of its child.

5.4 IAS access overhead

Verifying blocks doesn't require trusted hardware. However, due to a design choice by Intel, miners must contact the IAS to verify attestations. Currently there is no way to verify attestations locally. This requirement, however, does not change the basic security assumptions. Moreover, a simple modification to the IAS protocol, which is being tested by Intel [3], could get rid of the reliance on IAS completely on verifiers' side.

Recall that the IAS is a public web service that receives SGX attestations and responds with verification results. Requests are submitted to the IAS over HTTPS; a response is a signed "report" indicating the validation status of the queried platform [44]. In the current version of IAS, a report is not cryptographically linked with its corresponding request, which makes the report only trustworthy for the client initiating the HTTPS session. Therefore an IAS access is required for every block verification by every blockchain participant.

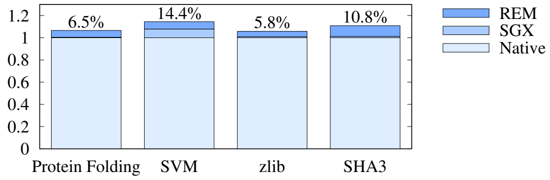


Figure 10: REM Overhead

However, the following modification can eliminate this overhead: simply echoing the request in the body of the report. Since the report is signed by Intel using a published public key [44, 45], only one access to IAS would be needed globally for every new block. Other miners could use the resulting signed report. Such a change is under testing by Intel for future versions of the IAS [3].

5.5 Experiments

We evaluate the overhead of REM with four examples of useful work benchmarks in REM as mining programs: a protein folding algorithm [1], a Support Vector Machine (SVM) classifier [22], the zlib compression algorithm (iterated) [2], and the SHA3-256 hash algorithm (iterated) [7]. We evaluate each benchmark in three modes:

Native We compile with the standard toolchain.

SGX We port to SGX by removing system calls and replacing system libraries with SGX-compliant ones. Then we compile in SGX-prerelease mode and run with the SGX driver v1.7 [43].

REM After porting to SGX, we instrument the code using our REM toolchain. We then proceed as in the SGX mode.

We use the same optimization level (`-O2`) in all modes. The experiments are done on a Dell Precision Workstation with an Intel 6700K CPU and 32GB of memory. For more details on the experiment setup, we refer readers to the full version [76].

We compared the running time in three modes and the results are shown in Figure 10. The running time of the native mode is normalized to one as a baseline. For all four useful workloads, we observe a total overhead of 5.8% $\sim$ 14.4% in REM relative to the native mode. Because the code is instrumented at control flow transfers, workloads with more jumps will incur more counting overhead. For example, SHA3-256 is highly iterative compared with the other workloads, so it incurs the most counting overhead.

We note that overhead for running in SGX is not uniform. For computation-bound workloads such as protein folding, zlib, and SHA3, SGX introduces little overhead ($< 1\%$) because the cost of switching to SGX and obtaining attestations is amortized by the longer in-enclave execution time of the workload. In the shorter SVM benchmark, the cost of entering SGX is more significant.

In summary, we observe an overhead of roughly $5-15\%$ for converting useful-work benchmarks into REM PoUW enclave.

6 Waste Analysis

To compare PoUW against PoET and alternative schemes, we explore a common game-theoretic model (with details deferred to the appendix). We consider a set of operators / agents that can either work locally on their own useful workloads or utilize their resource for mining. Based on the revenue from useful work and mining, and the capital and operational costs, we compute the equilibrium point of the system. We calculate the waste in this context as the ratio of the total resource cost (in U.S. dollars) spent per unit of useful work on a mining node compared with the cost when mining is not possible and all operators do useful work. We plug in concrete numbers for the parameters based on statistics we collected from public data sources.

Initial study of PoET identified a subtle pitfall involving miner's ability to mine simultaneously on multiple blockchains, a problem solved by He et al. [59] in a scheme we call *Lazy-PoET*. Our analysis, however, reveals that even Lazy-PoET suffers from what we call the *stale-chip problem*. Miners are better off maintaining farms of cheap, outdated CPUs just for mining than using new CPUs for otherwise useful goals.

We consider instead an approach in which operators utilize their CPUs while mining, making newer CPUs more attractive due to the added revenue from the useful work done. We call this scheme *Busy PoET*. We find that it improves on Lazy Poet, but remains highly wasteful.

This observation leads to another approach, *Proof of Potential Work* (*PoPW*). PoPW is similar to Busy-PoET, but reduces mining time according to the speed of the CPU (its potential to do work), and thus rewards use of newer CPUs. Although PoPW would greatly reduce waste, SGX does not allow an enclave to securely retrieve its CPU model, making PoPW theoretical only.

We conclude that PoUW incurs the smallest amount of waste among the options under study. For full details on our model, parameter choices, and analyses of the various mining schemes, we refer the reader to the full version [76].

7 Related Work

Cryptocurrencies and Consensus. Modern decentralized cryptocurrencies have stimulated strong interest in Proof-of-Work (PoW) systems [12, 29, 46] as well as techniques to reduce their associated waste.[3]

An approach similar to PoET [41], possibly originating with Dryja [27], is to limit power waste by so-called Proof-of-Idle. Miners buy mining equipment and get paid for proving that their equipment remains idle. Beyond the technical challenges, as in PoET, an operator with a set budget could redirect savings from power to purchase more idle machines, producing capital waste.

Alternative approaches, like PoUW, aim at PoW producing work useful for a secondary goal. Permacoin [58] repurposes mining resources as a distributed storage network, but recycles only a small fraction of mining resources. Primecoin [49] is an active cryptocurrency whose "useful outputs" are Cunningham and Bi-twin chains of prime numbers, which have no known utility. Gridcoin [36, 35], an active cryptocurrency whose miners work for the BOINC [9] grid-computing network, relies on a central entity. FoldingCoin [65] rewards participants for work on a protein folding problem, but as a layer atop, not integrated with, Bitcoin.

Proof-of-Stake [71, 14, 48, 16] is a distinct approach in which miners gain the right to generate blocks by committing cryptocurrency funds. It is used in experimental systems such as Peercoin [50] and Nxt [23]. Unlike PoW, however, in PoS, an attacker that gains majority control of mining resources for a bounded time can control the system forever. PoS protocols also require that funds, used as stake, remain frozen (and unusable) for some time. To remove this assumption, Bentov et al. [15] and Duong et al. [28] propose hybrid PoW / PoS systems. These works, and the line of hybrid blockchain systems starting with Bitcoin-NG [32, 51, 63], can all utilize PoUW as a low-waste alternative to PoW.

Another line of work on PoW for cryptocurrencies aims at PoWs that resist mining on dedicated hardware and prevent concentration of mining power, e.g., via memory-intensive hashing as in Scrypt [54] and Ethereum [19]. Although democratization of mining power is not our focus here, PoUW in fact achieves this goal by restricting mining to general-use CPUs.

SGX. Due to the complexity of the x86-64 architecture, several works [24, 70, 73] have exposed security problems in SGX, such as side-channel attacks [73]. Tramer et al. [70] consider the utility of SGX if its confidentiality guarantees are broken. Similar practical concerns motivate REM's tolerance mechanism of compromised SGX chips.

Ryoan [38] is a framework that allows a server to run code on private client data and return the output to

[3] "Permissioned" systems, as supported in, e.g., Hyperledger [20] and Stellar [56], avoid waste by using traditional consensus protocols at the cost of avoiding decentralization.

the client. The (trusted) Ryoan service instruments the server operator's code to prevent leakage of client data. In contrast, in REM, the useful-workload code is instrumented in an *untrusted* environment, and an attestation of its validity is produced within a trusted environment.

Haven [13] runs non-SGX applications by incorporating a library OS into the enclave. REM, in contrast, takes code amenable to SGX compilation and enforces correct instrumentation. In principle, Haven could allow for non-SGX code to be adapted for PoUW.

Zhang et al. [75] and Juels et al. [47] are the first works we are aware of to pair SGX with cryptocurrencies. Their aim is to augment the functionality of smart contracts, however, and is unrelated to the underlying blockchain layer in which REM operates.

8 Conclusion

We presented REM, which supports permissionless blockchain consensus based on a novel mechanism called Proof of Useful Work (PoUW). PoUW leverages Intel SGX to significantly reduce the waste associated with Proof of Work (PoW), and builds on and remedies shortcomings in Intel's innovative PoET scheme. PoUW and REM are thus a promising basis for partially-decentralized blockchains, reducing waste given certain trust assumptions in a hardware vendor such as Intel.

Using a rigorous analytic framework, we have shown how REM can achieve resilience against compromised nodes with minimal waste (rejected honest blocks). This framework extends to PoET and potentially other SGX-based mining approaches.

Our implementation of REM introduces powerful new techniques for SGX applications, namely instruction-counting instrumentation and hierarchical attestation, of potential interest beyond REM itself. They allow REM to accommodate essentially any desired workloads, permitting flexible adaptation in a variety of settings.

Our framework for economic analysis offers a general means for assessing the true utility of mining schemes, including PoW and partially-decentralized alternatives. Beyond illustrating the benefits of PoUW and REM, it allowed us to expose risks of approaches such as PoET in the use of stale chips, and propose improved variants, including Proof of Potential Work (PoPW). We found that small changes to the TEE framework would be significant for reduced-waste blockcain mining. In particular, allowing for secure instruction (or cycle) counting would reduce PoUW overhead, and a secure chip-model reading instruction would allow for PoPW implementation.

We reported on a complete implementation of REM, swapped in for the consensus layer in Bitcoin core in a prototype system. Our experiments showed minimal performance impact (5-15%) on example benchmarks. In summary, our results show that REM is practically deployable and promising path to fair and environmentally friendly blockchains in partially-decentralized blockchains.

Acknowledgements

This work is funded in part by NSF grants CNS-1330599, CNS-1514163, CNS-1564102, CNS-1601879, CNS-1544613, and No. 1561209, ARO grant W911NF-16-1-0145, ONR grant N00014-16-1-2726, and IC3 sponsorship from Chain, IBM, and Intel.

References

[1] A Genetic Algorithm for Predicting Protein Folding in the 2D HP Model. https://github.com/alican/GeneticAlgorithm. Accessed: 2016-11-11.

[2] A Lossless, High Performance Implementation of the Zlib (RFC 1950) and Deflate (RFC 1951) Algorithm. https://code.google.com/archive/p/miniz/. Accessed: 2017-2-16.

[3] Attestation Service for Intel Software Guard Extensions (Intel SGX): API Documentation. Revision 2.0. Section 4.2.2. https://software.intel.com/sites/default/files/managed/7e/3b/ias-api-spec.pdf. Accessed: 2017-2-21.

[4] Intel(R) Software Guard Extensions for Linux OS. https://github.com/01org/linux-sgx. Accessed: 2017-2-16.

[5] Intel Software Guard Extensions Enclave Writer's Guide. https://software.intel.com/sites/default/files/managed/ae/48/Software-Guard-Extensions-Enclave-Writers-Guide.pdf. Accessed: 2017-2-16.

[6] Sawtooth-core source code (validator). https://github.com/hyperledger/sawtooth-core/tree/0-7/validator/sawtooth_validator/consensus/poet1. Accessed: 2017-2-21.

[7] Single-file C implementation of the SHA-3 implementation with Init/Update/Finalize hashing (NIST FIPS 202). https://github.com/brainhub/SHA3IUF. Accessed: 2017-2-16.

[8] Anati, I., Gueron, S., Johnson, S., and Scarlata, V. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2^{nd} International Workshop on Hardware and Architectural Support for Security and Privacy* (2013), vol. 13.

[9] Anderson, D. P. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on* (2004), IEEE, pp. 4–10.

[10] Aspnes, J., Jackson, C., and Krishnamurthy, A. Exposing computationally-challenged Byzantine impostors. *Department of Computer Science, Yale University, New Haven, CT, Tech. Rep* (2005).

[11] Azure, M. Blockchain as a service. https://web.archive.org/web/20161027013817/https://azure.microsoft.com/en-us/solutions/blockchain/, 2016.

[12] Back, A. Hashcash – a denial of service counter-measure. http://www.cypherspace.org/hashcash/hashcash.pdf, 2002.

[13] Baumann, A., Peinado, M., and Hunt, G. Shielding applications from an untrusted cloud with Haven. *ACM Trans. Comput. Syst. 33*, 3 (Aug. 2015), 8:1–8:26.

[14] Bentov, I., Gabizon, A., and Mizrahi, A. Cryptocurrencies without proof of work. *CoRR abs/1406.5694* (2014).

[15] Bentov, I., Lee, C., Mizrahi, A., and Rosenfeld, M. Proof of activity: Extending Bitcoin's proof of work via proof of stake. Cryptology ePrint Archive, Report 2014/452, 2014. http://eprint.iacr.org/2014/452.

[16] Bentov, I., Pass, R., and Shi, E. Snow White: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. http://eprint.iacr.org/2016/919.

[17] Bitcoin community. Bitcoin source. https://github.com/bitcoin/bitcoin, retrieved Nov. 2016.

[18] Bitcoin community. Bitcoin source. https://github.com/bitcoin/bitcoin, retrieved Mar. 2015.

[19] Buterin, V. A next generation smart contract & decentralized application platform. https://www.ethereum.org/pdfs/EthereumWhitePaper.pdf/, retrieved Feb. 2015, 2013.

[20] Cachin, C. Architecture of the Hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers* (2016).

[21] Carlsten, M., Kalodner, H., Weinberg, S. M., and Narayanan, A. On the instability of Bitcoin without the block reward. In *ACM CCS* (2016).

[22] Chang, C.-C., and Lin, C.-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology 2* (2011), 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[23] Community, T. N. Nxt whitepaper, revision 4. https://web.archive.org/web/20160207083400/https://www.dropbox.com/s/cbuwrorf672c0yy/NxtWhitepaper_v122_rev4.pdf, 2014.

[24] Costan, V., and Devadas, S. Intel SGX Explained. *Cryptology ePrint Archive* (2016).

[25] Deetman, S. Bitcoin could consume as much electricity as Denmark by 2020. http://tinyurl.com/yc4r9k3k, Mar. 2016.

[26] Donovan, A., Muth, R., Chen, B., and Sehr, D. PNacls: Portable native client executables.

[27] Dryja, T. Optimal mining strategies. SF Bitcoin-Devs presentation. https://www.youtube.com/watch?v=QN2TPeQ9mnA, 2014.

[28] Duong, T., Fan, L., Veale, T., and Zhou, H.-S. Securing Bitcoin-like backbone protocols against a malicious majority of computing power. Cryptology ePrint Archive, Report 2016/716, 2016. http://eprint.iacr.org/2016/716.

[29] Dwork, C., and Naor, M. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference* (1992), Springer, pp. 139–147.

[30] Dwyer, J. P., and Hines, P. Beyond the byzz: Exploring distributed ledger technology use cases in capital markets and corporate banking. Tech. rep., Celent and MISYS, 2016.

[31] Eyal, I. The miner's dilemma. In *IEEE Symposium on Security and Privacy* (2015), pp. 89–103.

[32] Eyal, I., Gencer, A. E., Sirer, E. G., and Van Renesse, R. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (2016), pp. 45–59.

[33] Eyal, I., and Sirer, E. G. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security* (2014).

[34] Garay, J. A., Kiayias, A., and Leonardos, N. The Bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2015), pp. 281–310.

[35] Gridcoin. Gridcoin. https://web.archive.org/web/20161013081149/http://www.gridcoin.us/, 2016.

[36] Gridcoin. Gridcoin (grc) – first coin utilizing boinc – official thread. https://web.archive.org/web/20160909032618/https://bitcointalk.org/index.php?topic=324118.0, 2016.

[37] Hoekstra, M., Lal, R., Pappachan, P., Phegade, V., and Del Cuvillo, J. Using innovative instructions to create trustworthy software solutions. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy* (New York, NY, USA, 2013), HASP '13, ACM, pp. 11:1–11:1.

[38] Hunt, T., Zhu, Z., Xu, Y., Peter, S., and Witchel, E. Ryoan: A distributed sandbox for untrusted computation on secret data. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (GA, Nov. 2016), USENIX Association, pp. 533–549.

[39] Intel. *Intel Software Guard Extensions Programming Reference*, 2014.

[40] Intel. *Intel 64 and IA-32 Architectures Software Developer's Manual: Volume 3 (3A, 3B, 3C & 3D): System Programming Guide*, 325384-059us ed., June 2016.

[41] Intel. Sawtooth lake – introduction. https://web.archive.org/web/20161025232205/https://intelledger.github.io/introduction.html, 2016.

[42] Intel Corporation. Intel® Software Guard Extensions SDK. https://software.intel.com/en-us/sgx-sdk, 2015.

[43] Intel Corporation. Intel SGX for Linux. https://01.org/intel-softwareguard-extensions, 2016.

[44] Intel Corporation. Intel Software Guard Extensions: Intel Attestation Service API. https://software.intel.com/sites/default/files/managed/3d/c8/IAS_1_0_API_spec_1_1_Final.pdf, 2016.

[45] Intel Corporation. Public Key for Intel Attestation Service. https://software.intel.com/en-us/sgx/resource-library, 2016.

[46] Jakobsson, M., and Juels, A. Proofs of work and bread pudding protocols. In *Secure Information Networks*. Springer, 1999, pp. 258–272.

[47] Juels, A., Kosba, A., and Shi, E. The Ring of Gyges: Investigating the future of criminal smart contracts. In *ACM CCS* (2016), pp. 283–295.

[48] Kiayias, A., Konstantinou, I., Russell, A., David, B., and Oliynykov, R. A provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Report 2016/889, 2016. http://eprint.iacr.org/2016/889.

[49] King, S. Primecoin: Cryptocurrency with prime number proof-of-work. https://web.archive.org/web/20160307052339/http://primecoin.org/static/primecoin-paper.pdf, 2013.

[50] King, S., and Nadal, S. PPcoin: Peer-to-peer crypto-currency with proof-of-stake. https://web.archive.org/web/20161025145347/https://peercoin.net/assets/paper/peercoin-paper.pdf, 2012.

[51] Kogias, E. K., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., and Ford, B. Enhancing Bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)* (2016), pp. 279–296.

[52] Lewenberg, Y., Sompolinsky, Y., and Zohar, A. Inclusive block chain protocols. In *Financial Cryptography* (Puerto Rico, 2015).

[53] Li, Y., McCune, J., Newsome, J., Perrig, A., Baker, B., and Drewry, W. Minibox: A two-way sandbox for x86 native code. In *2014 USENIX annual technical conference (USENIX ATC 14)* (2014), pp. 409–420.

[54] Litecoin Project. Litecoin, open source P2P digital currency. https://litecoin.org, retrieved Nov. 2014.

[55] Lombrozo, E., Lau, J., and Wuille, P. BIP141: Segregated witness (consensus layer). https://web.archive.org/web/20160521104121/https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki, 2015.

[56] Mazieres, D. The Stellar consensus protocol: A federated model for Internet-level consensus. https://web.archive.org/web/20161025142145/https://www.stellar.org/papers/stellar-consensus-protocol.pdf, 2015.

[57] McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C. V., Shafi, H., Shanbhogue, V., and Savagaonkar, U. R. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy* (2013), p. 10.

[58] Miller, A., Shi, E., Juels, A., Parno, B., and Katz, J. Permacoin: Repurposing Bitcoin work for data preservation. In *Proceedings of the IEEE Symposium on Security and Privacy* (San Jose, CA, USA, 2014), IEEE.

[59] Milutinovic, M., He, W., Wu, H., and Kanwal, M. Proof of luck: An efficient blockchain consensus protocol. In *Proceedings of the 1st Workshop on System Software for Trusted Execution* (New York, NY, USA, 2016), SysTEX '16, ACM, pp. 2:1–2:6.

[60] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf, 2008.

[61] Nayak, K., Kumar, S., Miller, A., and Shi, E. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. *IACR Cryptology ePrint Archive 2015* (2015), 796.

[62] Pass, R., Seeman, L., and Shelat, A. Analysis of the blockchain protocol in asynchronous networks. Tech. rep., Cryptology ePrint Archive, Report 2016/454, 2016.

[63] Pass, R., and Shi, E. Hybrid consensus: Efficient consensus in the permissionless model. Cryptology ePrint Archive, Report 2016/917, 2016. http://eprint.iacr.org/2016/917.

[64] Popper, N. Central banks consider Bitcoin's technology, if not Bitcoin. New York Times, Oct. 2016.

[65] Ross, R., and Sewell, J. Foldingcoin white paper. https://web.archive.org/web/20161022232226/http://foldingcoin.net/the-coin/white-paper/, 2015.

[66] Sapirshtein, A., Sompolinsky, Y., and Zohar, A. Optimal selfish mining strategies in Bitcoin. *CoRR abs/1507.06183* (2015).

[67] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. Intel Software Guard Extensions: EPID Provisioning and Attestation Services, 2015.

[68] Sompolinsky, Y., and Zohar, A. Accelerating Bitcoin's transaction processing. fast money grows on trees, not chains. In *Financial Cryptography* (Puerto Rico, 2015).

[69] SWIFT, and Accenture. Swift on distributed ledger technologies. Tech. rep., SWIFT and Accenture, 2016.

[70] Tramer, F., Zhang, F., Lin, H., Hubaux, J.-P., Juels, A., and Shi, E. Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. Cryptology ePrint Archive, Report 2016/635, 2016. http://eprint.iacr.org/2016/635.

[71] User "QuantumMechanic". Proof of stake instead of proof of work. https://web.archive.org/web/20160320104715/https://bitcointalk.org/index.php?topic=27787.0.

[72] Wood, G. Ethereum: A secure decentralised generalised transaction ledger (EIP-150 revision). https://web.archive.org/web/20161019105532/http://gavwood.com/Paper.pdf, 2016.

[73] Xu, Y., Cui, W., and Peinado, M. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Proc. IEEE Symp. Security and Privacy* (May 2015), pp. 640–656.

[74] Yee, B., Sehr, D., Dardyk, G., Chen, J. B., Muth, R., Ormandy, T., Okasaka, S., Narula, N., and Fullagar, N. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *Proceedings of the 30th IEEE Symposium on Security and Privacy* (May 2009), pp. 79–93.

[75] Zhang, F., Cecchetti, E., Croman, K., Juels, A., and Shi, E. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2016), CCS '16, ACM, pp. 270–282.

[76] Zhang, F., Eyal, I., Escriva, R., Juels, A., and van Renesse, R. REM: Resource-Efficient Mining for Blockchains. Cryptology ePrint Archive, Report 2017/179, 2017. http://eprint.iacr.org/2017/179.

A Tolerating Compromised SGX Nodes: Details

A.1 Mining Rate Estimation

We start by discussing how to statistically infer the power of a CPU from its blocks in the blockchain. Reading the difficulty of each block in the main chain and the rate of blocks from a specific CPU, we can estimate a lower bound of that CPU's power – it follows directly from the rate of its blocks. It is a lower bound since the CPU might not be working continuously, and the estimate's accuracy increases with the number of available blocks.

Recall C_{m_i} is the blocks mined by miner m_i so far. C_{m_i} may contain multiple blocks, perhaps with varying difficulties. Without loss of generality, we write the difficulty as a function of time, $d(t)$. The difficulty is the probability for a single instruction to yield a win. Denote the power of the miner, i.e., its mining rate, by rate_i. Therefore in a given time interval of length T, the number of blocks mined by a specific CPU obeys Poisson distribution (since CPU rates are high and the win probability is small, it's appropriate to approximate a Binomial distribution by a Poisson distribution,) and with rate $\mathsf{rate}_i T d(t)$. Further, under independence assumption, the mining process of a specific CPU is specified by a Poisson process with rate $\lambda_i(t) = \mathsf{rate}_i d(t)$, the product of the probability and the miner's rate rate_i.

There are many methods to estimate the mean of a Poisson distribution. We refer readers to the full version [76] for more details. Knowing rates for all miners, the rate of the strongest CPU ($\mathsf{rate}_{\text{best}}$) can be estimated. The challenge here is to limit the influence of adversarial nodes. To this end, instead of finding the strongest CPU directly, we approximate $\mathsf{rate}_{\text{best}}$ based on rate_ρ (e.g. $f_{90\%}$), namely the $\rho-$percentile fastest miner.

Bootstrapping. During the launch of a cryptocurrency, it could be challenging to estimate the mining power of the population accurately, potentially leading to poisoning attacks by an adversary. At this early stage, it makes sense to hardwire a system estimate of the maximum mining power of honest miners into the system and set conditions (e.g., a particular mining rate or target date) to estimate $\mathsf{rate}_{\text{best}}$ as we propose above. If the cryptocurrency launches with a large number of miners, an even simpler approach is possible before switching to $\mathsf{rate}_{\text{best}}$ estimation: We can cap the total number of blocks that any one node can mine, a policy we illustrate below. (See P_{simple}.)

A.2 Security game definition

We model REM as an interaction among three entities: a blockchain consensus algorithm, an adversary, and a set of honest miners. Their behavior together defines a *security game*, which we define formally below. We characterize the three entities respectively as (ideal) programs $\mathsf{prog}_{\text{chain}}$, $\mathsf{prog}_{\mathcal{A}}$, and prog_m, which we now define.

Blockchain consensus algorithm ($\mathsf{prog}_{\text{chain}}$). A consensus algorithm determines which valid blocks are added to a blockchain C. We assume that underlying consensus and fork resolution are instantaneous; loosening this assumption does not materially affect our analyses. We also assume that block timestamping is accurate. Timestamps can technically be forged at block generation, but in practice miners reject blocks with large skews [18], limiting the impact of timestamp forgery.

Informally, $\mathsf{prog}_{\text{chain}}$ maintains and broadcasts and authoritative blockchain C. In addition to verifying that block contents are correct, $\mathsf{prog}_{\text{chain}}$ appends to C only blocks that are valid under a policy P. We model the blockchain consensus algorithm as the (ideal) stateful program specified in Figure 11.

Adversary $\mathcal{A}$ ($\mathsf{prog}_{\mathcal{A}}$). In our model, an adversary $\mathcal{A}$ executes a *strategy* $\Sigma_{\mathcal{A}}$ that coordinates the k miners $M_{\mathcal{A}}$ under her control to generate blocks. Specifically:

```
                    $\mathsf{prog}_{\text{chain}}[P]$

State:
  $C$: the chain
On receive "init":
  $C := \varnothing$
  $d := d_0$
  Send $(C, P, d)$ to all miners
On receive "submit" $B$ from $m$:
  if $P(C, B) = \mathsf{accept}$:
    $C \leftarrow C \cup \{B\}$
    $d \leftarrow \mathsf{adjust}(C, d)$
    Send $(C, P, d)$ to all miners
```

Figure 11: The program for a blockchain. We omit details here on how difficulty d is set, i.e., how d_0 and adjust are chosen.

```
                    $\mathsf{prog}_{\mathcal{A}}[\Sigma_{\mathcal{A}}]$

On receive $(C, P, d)$ from $\mathsf{prog}_{\text{chain}}$
  $\hat{t}, \hat{m} \leftarrow \Sigma_{\mathcal{A}}(M_{\mathcal{A}}, C, P, d)$
  if $\hat{t}$ is not $\perp$:
    wait until $\hat{t}$
    send "submit" $(\hat{t}, \hat{m}, d)$ to $\mathsf{prog}_{\text{chain}}$
```

Figure 12: The program for an adversary $\mathcal{A}$ that controls k nodes $M_{\mathcal{A}} = \{m_{\mathcal{A}1}, \cdots, m_{\mathcal{A}k}\}$.

Definition 4. *(Adversarial Strategy). An* adversarial strategy *is a probabilistic algorithm $\Sigma_{\mathcal{A}}$ that takes in a set of identities, the current blockchain and the policy, and outputs a time-stamp and identity for block submission. Specifically, $(M_{\mathcal{A}}, C, t, P) \rightarrow (\hat{t}, \hat{m}) \in \mathbb{R}^+ \times M_{\mathcal{A}}$.*

In principle, $\Sigma_{\mathcal{A}}$ can have dependencies among individual node behaviors. In our setting, this would not benefit $\mathcal{A}$, however. As we don't know $M_{\mathcal{A}}$ *a priori*, though, the only policies we consider operate on individual miner block-generation history.

As a wrapper expressing implementation by $\mathcal{A}$ of $\Sigma_{\mathcal{A}}$, we model $\mathcal{A}$ as a program $\mathsf{prog}_{\mathcal{A}}$, specified in Figure 12.

Honest miners (prog_m). Every honest miner $m \in M - M_{\mathcal{A}}$ follows an identical strategy, a probabilistic algorithm denoted Σ_h. In REM, Σ_h may be modeled as a simple algorithm that samples from a probability distribution on block mining times determined by $rate(m)$ (specifically in our setting, an exponential distribution with rate $rate(m)$). We express implementation by honest miner m of Σ_h as a program $\mathsf{prog}_m[\Sigma_h]$ (Figure 13).

To understand the security of REM, we consider a *security game* that defines how an adversary $\mathcal{A}$ interacts with honest miners, a blockchain consensus protocol,

```
$\mathsf{prog}_m[\Sigma_h]$

On receive $(C,P,d)$ from $\mathsf{prog}_{\mathsf{chain}}$
    $\hat{t} \leftarrow \Sigma_h(C,d)$
    Send "submit" $(\hat{t},m,d)$ to $\mathsf{prog}_{\mathsf{chain}}$
```

Figure 13: The program for an honest miner. Σ_h is the protocol defined by $\mathsf{prog}_{\mathsf{chain}}$(e.g. PoET or PoUW).

```
$P_{\mathrm{simple}}(C,B)$:
    parse $B \to (\tau,m,d)$
    if $|C_m| > 0$:
        output reject
    else
        output accept
```

Figure 14: A simple policy that allows one block per CPU over its lifetime.

and a policy given the above three ideal programs. Formally:

Definition 5. *(Security Game) For a given triple of ideal programs* $(\mathsf{prog}_{chain}[P], \mathsf{prog}_{\mathcal{A}}[\Sigma_{\mathcal{A}}], \mathsf{prog}_m[\Sigma_h])$*, and policy P, a security game* $S(P)$ *is a tuple* $S(P) = ((M, M_{\mathcal{A}}, rate(\cdot)); (\Sigma_{\mathcal{A}}, \Sigma_h))$.

We define the *execution* of $S(P)$ as an interactive execution of programs $(\mathsf{prog}_{\mathrm{chain}}[P], \mathsf{prog}_{\mathcal{A}}[\Sigma_{\mathcal{A}}], \mathsf{prog}_m[\Sigma_h])$ using the parameters of $S(P)$. As P, $\Sigma_{\mathcal{A}}$ and Σ_h are randomized algorithms, such execution is itself probabilistic. Thus we may view the blockchain resulting from execution of S for interval of time τ as a random variable $C_S(\tau)$.

A *non-degenerate* security game S is one in which there exists at least one honest miner m with $rate(m) > 0$.

A.3 Warmup policy

As a warmup, we give a simple example of a potential block-acceptance policy. This policy just allows one block throughout the life of a CPU, as shown in Figure 14.

Clearly, an adversary cannot do better than mining one block. Denote this simple strategy Σ_{simple}. For any non-degenerate security game S, therefore, the advantage $\mathsf{Adv}_{\mathcal{A}}^{S(P_{simple})}(\tau) = 1$ as $\tau \to \infty$. This policy is optimal in that an adversary cannot do better than an honest miner unconditionally. However the asymptotic waste of this policy is 100%.

Another disadvantage of this policy is that it discourages miners from participating. Arguably, a miner would stay if the revenue from mining is high enough to cover the cost of replacing a CPU. But though a CPU is still valuable in other contexts even if it is is blacklisted forever in *this* particular system, repurposing it incurs operational cost. Therefore chances are this policy would cause a loss of mining power, especially when the initial miner population is small, rendering the system more vulnerable to attacks.

A.4 Adversarial advantage

A block-acceptance policy depends only on the number of blocks by the adversary since its first one. Therefore an adversary's best strategy is simply to publish its blocks as soon as they won't be rejected. Denote this strategy as Σ_{stat}.

Clearly, an adversary will submit $F^{-1}(1-\alpha, td \cdot \mathsf{rate}_{\mathrm{best}})$ blocks within $[0,t]$. On the other hand, the strongest honest CPU with rate $\mathsf{rate}_{\mathrm{best}}$ mines $td \cdot \mathsf{rate}_{\mathrm{best}}$ blocks in expectation. Recall that according to our Markov chain analysis, P_{stat} incurs false rejections for honest miners with probability $w_h(\alpha)$, which further reduces the payoff for honest miners. For a (non-degenerate) security game S, in which $\mathcal{A}$ uses strategy Σ_{stat}, the advantage is therefore:

$$\mathsf{Adv}_{\mathcal{A}}^{S(P_{\mathrm{stat}}^{\alpha})} = \lim_{t\to\infty} \frac{F^{-1}(1-\alpha, td \cdot \mathsf{rate}_{\mathrm{best}})}{(1-w_h(\alpha))\, td \cdot \mathsf{rate}_{\mathrm{best}}} \qquad (5)$$

Theorem 1. *In a (non-degenerate) security game S where* $\mathcal{A}$ *uses strategy* Σ_{stat}*,*

$$\mathsf{Adv}_{\mathcal{A}}^{S(P_{stat}^{\alpha})} = \frac{1}{1 - Waste(P_{stat}^{\alpha})}.$$

Proof. Let $\lambda = td \cdot \mathsf{rate}_{\mathrm{best}}$. It is known that as λ for a Poisson distribution goes to infinity, it converges in the limit to a normal distribution with mean and variance λ. Therefore,

$$\lim_{\lambda\to\infty} \frac{F^{-1}(1-\alpha,\lambda)}{(1-w_h(\alpha))\lambda} = \lim_{\lambda\to\infty} \frac{\lambda + \sqrt{\lambda} z_p}{(1-w_h(\alpha))\,\lambda} = \frac{1}{1-w_h(\alpha)}.$$

□

Early in a blockchain's evolution, the potential advantage of an adversary is relatively high. The confidence interval is wide at this point, allowing the adversary to perform frequent generation without triggering detection. As the adversary publishes more blocks, the confidence interval tightens, forcing the adversary to reduce her mining rate. This is illustrated by our numerical simulation in Section 4.3.