Preventing front-running attacks using timelock encryption

Venkkatesh Sekar

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Master of Science
of
University College London.

Department of Computer Science
University College London

March 2, 2022

I, Venkkatesh Sekar, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Frontrunning is an active exploit where an attacker benefits from advanced access to privileged market information about upcoming transactions. Dating back to traditional financial markets, the decentralized and open nature of the blockchain has found a new variant of frontrunning attacks with severe impact. Several solutions have been proposed to mitigate frontrunning including automatic market makers and confidential transactions.

A timelock encryption scheme allows a user to commit a secret to a particular time in the future, after which the secret will be made public. We propose a new timelock encryption scheme as a mechanism to thwart frontrunning attacks in the blockchain space. The scheme utilizes a distributed randomness beacon (drand), elliptic curve cryptography, bilinear pairings and threshold BLS signatures as building blocks. It is inspired by Boneh Franklin's Identity Based Encryption scheme and is adaptive chosen ciphertext attack secure(IND-ID-CCA) under the Computational co-Bilinear Diffie Hellman problem. As a proof of concept, we provide a web platform built in JavaScript for timelocking arbitrary messages with near-native speed. For feasibility study, a concrete integration of our construction in the Filecoin infrastructure has been undertaken. Additionally, we conduct an analysis of the performance and security of our prototype. On average, it took 62ms for encryption and 33ms for decryption for 3KB transactions on our testbench. Finally, we conclude with the possible drawbacks and other useful applications of our scheme.

Acknowledgements

I would like to thank my supervisor, Dr. Philipp Jovanovic for his invaluable support and guidance throughout the entirety of the MSc project, from mathematical understanding to efficient project planning. Besides, Dr. Nicolas Gailly from Protocol Labs was monumental in deconstructing the building blocks of the drand project and Filecoin codebase. A special thanks to Dr. Jonathan Ward from fetch.ai and Thomas Piellard from ConsenSys for their enlightening discussion on DAG based consensus and automatic market makers respectively. Finally, I am very much in debt to the members of the research staff and software development team at Protocol Labs, who helped me achieve the goals of the project successfully.

Contents

List of Figures

Chapter 1

Introduction

Blockchain technologies saw their inception in the beginning of last decade through Bitcoin. Since then, an entire $3 trillion industry has been built around it, which leverage the innovation of blockchain. Starting as naive digital currencies, modern day blockchain technologies power smart contracts, decentralized autonomous organization (DAOs), decentralized applications (dApps), decentralized exchanges (DEX), non fungible tokens (NFTs) and much more. These decentralized technologies builds the pathway to an open, trustless and permissionless network, collectively known as Web 3.0. However, the innovation also gave rise to a new range of attacks and exploits only possible due to the unique nature of the blockchain. Out of these attacks, one of the most commonly exploited attack on a large scale is frontrunning.

The goal of our research is two fold (a) To build a practical timelock encryption scheme using the drand randomness beacon. (b) Integrate the proposed timelock encryption scheme in a blockchain instance as a protection against frontrunning.

The rest of the dissertation is organised as follows : Chapter 2 defines the necessary background and cryptography primitives required for rest of the dissertation. Chapter 3 explores the motivation for our research by understanding the feasibility of various attacks against blockchain technologies. Chapter 4 looks into the literature for novel techniques enabling frontrunning mitigation. Chapter 5 describes our proposed timelock encryption scheme in detail from a theoretical perspective. Chapter 6 explores the implementation of the protocol as a (a) standalone proof of concept and (b) integration with Filecoin. Chapter 7 performs a practicality study of the scheme through performance and security analysis. Chapter 8 identifies possible drawbacks of using timelock encryption and finally, Chapter 9 discusses other applications of our scheme through future work.

Chapter 2

Background

In this chapter, we trace back to the roots of distributed consensus[1], its relevance to modern day blockchain protocols and a high level overview of certain cryptographic primitives, which are essential in reasoning our construction.

2.1 Distributed Consensus

In 2008, the article titled *Bitcoin: A peer-to-peer electronic cash system*[58] formalized the notion of distributed consensus in the context of blockchain space. However, distributed consensus have already been well documented in literature before, tracing back to the seminal *The Byzantine Generals Problem*[2][55].

> " *Imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement. The generals must have an algorithm to guarantee that.*"

To generalize, any protocol trying to achieve the *Byzantine Broadcast*, needs to satisfy three properties. The participating parties can behave arbitrarily and deviate from the agreed consensus, referenced under the pseudonym *Byzantine faults.* The three properties include :

- **Termination** - All honest parties must always decide and terminate.
- **Validity** - If the protocol leader is honest, its value is the final decision value.

[1] The term consensus is used loosely here to group any form of abstraction of reaching an agreement in a distributed setting

[2] The problem is also referred as Byzantine fault tolerance, Byzantine failure and interactive consistency

- **Agreement** - All honest parties must agree to the same value.

Lamport et al.[55] provides several solutions to achieve *Byzantine Broadcast* under different hypotheses, which were inherently expensive. Among other solutions, one of the most notable approach to achieve Byzantine Broadcast is the Dolev Strong protocol[37]. The Dolev Strong protocol, for its successful operation, depends on two key components, synchronisation and authentication.

The concept of synchronisation was solved with the introduction of logical clocks [54], which allows systems to agree on a particular ordering of events in a distributed setting. On the other hand, authentication can be achieved through a public key infrastructure and digital signatures. With these components and operational parameters, the protocol can achieve byzantine broadcast in the presence of t corrupt nodes, if the protocol is operated for $t + 1$ rounds. Collectively speaking, distributed consensus protocols can be generalized as variants of these five critical parameters[36] :

- Processors synchronous or asynchronous,
- Communication synchronous or asynchronous,
- Message order synchronous or asynchronous,
- Broadcast transmission or point-to-point transmission,
- Atomic receive/send or separate receive and send.

Modern day distributed systems, however, in contrast to single-shot protocols, require consensus repeatedly to keep the system operational over time. One such repeated consensus abstraction is known as *blockchain*.

2.2 Blockchain primitives

The world of blockchain technologies is growing at a rapid pace with state-of-the-art innovation employed everyday. However, in principal, they have key similarities between one another, which provides a high level overview of the blockchain space.

2.2.1 Blockchain

A blockchain or ledger is a data structure of linearly-ordered log of transaction blocks. A key property of the blockchain is immutability, where previously added blocks can't be modified or altered by design. Another key property is it is append only, where mechanism is provided to only add transactions to the blockchain. In practice, it is a linked list of blocks using a collision resistant hash function of the previous block as a pointer reference. Each block carry a timestamp, nonce and a payload of transactions within a given epoch. Due to hash pointers (Fig. 2.1), any modifications to a single block will propagate to all succeeding blocks in the blockchain.

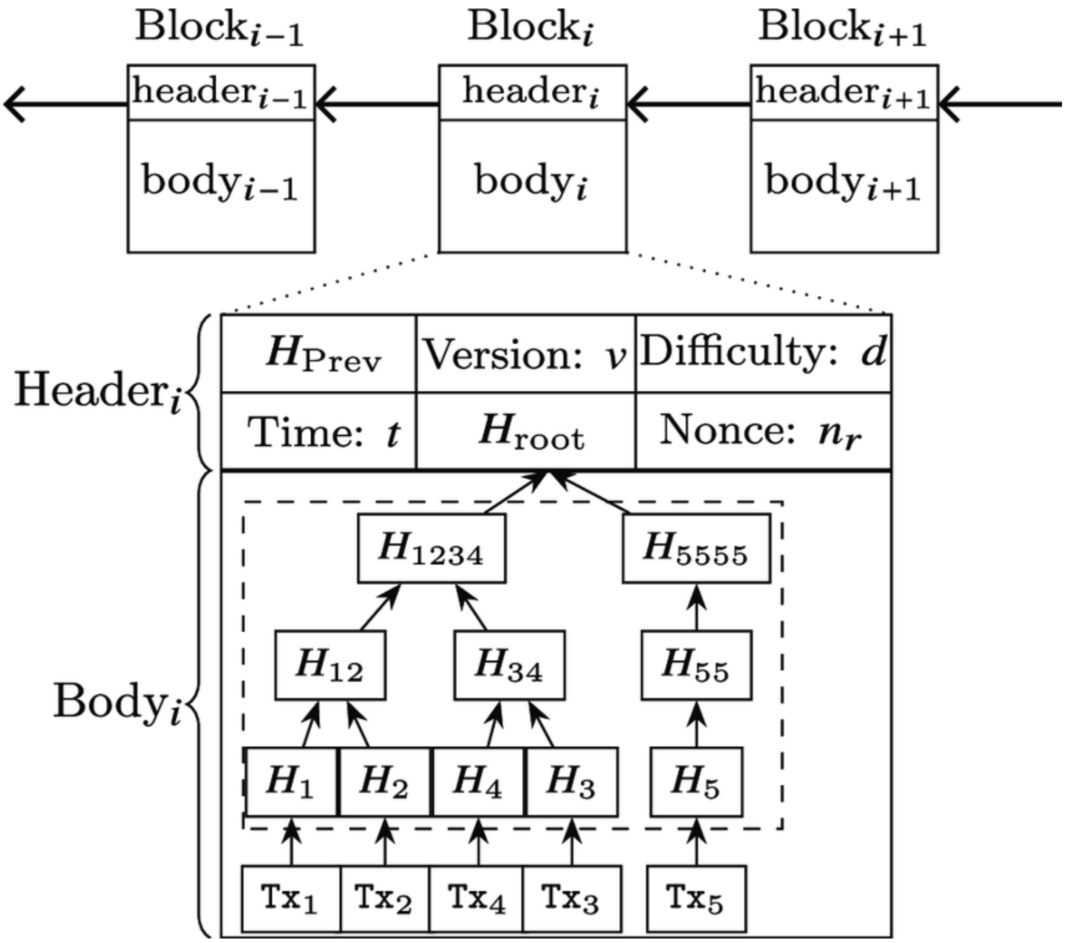


Figure 2.1: Bitcoin's blockchain structure [46]

2.2.2 Decentralized network

Decentralisation implies that the blockchain is not maintained by a central governing authority but rather by a system of distributed nodes. Without decentralisation, the properties of blockchain can't be satisfied as the central authority can publish a new copy of a modified blockchain invalidating previously existing copies. Decentralisation provides each participating node with a copy of the blockchain, and if majority are honest, a valid blockchain can be maintained through various distributed consensus protocols. In doing so, the blockchain can be made public in an open setting, allowing *transparency* of the transactions. This decentralized network is also referred to as *peer-to-peer network* or P2P, where each participating node is identified as a peer. The P2P network model allows anyone to join the network, supports fault tolerance mechanisms in case of failures and provides an efficient communication model like the *gossip* protocol between the peers.

2.2.3 Nodes

Nodes are a critical part of the blockchain network, whose key objective is to maintain the reliability of the distributed infrastructure. In an open, public blockchain network, individual participating nodes vary greatly in computational resources and availability, and hence, certain nodes offer special functionality. For example, in *Bitcoin*[3], nodes are classified into three different types namely

- **Full nodes** They form the core of the network with retaining complete history of the blockchain, validating and verifying proposing new blocks to maintain consensus. However, full nodes cannot propose new block themselves.
- **Miner nodes** Miner nodes are specialized full nodes, which can propose new blocks by solving a cryptographic challenge and validating it among other full nodes. Since the process is computation intensive, block producers are rewarded in the form of cryptocurrency or token of the network.
- **Light nodes** Light nodes (SPV) are typically payment wallets, which are small and size and carry only specific blockchain history. They depend on full nodes for verification and validation.

Nodes in the network also have a data store called **mempool**, where all the valid transactions wait to be confirmed by the decentralized network. The mempool is public and can be observed by any other nodes in the network.

2.2.4 Cryptocurrency

Cryptocurrency is a form of digital asset designed to work as a medium of exchange outside of normal currencies, where ownership to individual coins are recorded on the blockchain. Through modern cryptography, the blockchain can sufficiently secure transaction records, control the creation of additional coins, and verify the transfer of coin ownership. A transaction happens between a sender and a receiver, whom are identified through their public key addresses. Rather than validating one transaction at a time, often times the protocol would want to use batching to improve throughput — a block is exactly a batch of transactions per epoch of the protocol which are validated through their consensus mechanism.

2.2.5 Consensus

Blockchain technologies are decentralized at their core and hence, requires a intelligent distributed consensus to make decisions over the entire infrastructure. Based on the design of their protocol, many consensus algorithms have

[3] https://bitcoin.org/en/

been proposed in the last decade. Among them, the most notable ones include Proof-of-Work (PoW) and Proof-of-Stake (PoS).

- **Proof-of-Work** In PoW, at the beginning of each epoch, the existing miner nodes try to solve a cryptographic challenge for that round. The problem's difficulty is essentially defined by the parameters of the network to maintain a constant epoch (around 10 minutes in bitcoin). The first miner to solve the problem and gets it validated by majority of the full nodes is chosen as the block producer. The miner adds the next block, and new coins are minted and provided to the miner as reward.

  Repeated consensus is achieved by repeating the process for each epoch, providing a reliable distributed consensus. Recall, one of the main requirements of distributed consensus is **Validity**, where if the protocol leader is honest, it's decision is final. PoW allows the network to find a honest protocol leader each round, since, by design, it's nearly impossible to forge a solution for the cryptographic problem without investing the necessary intensive computation.

- **Proof-of-Stake** In Proof-of-Stake, the validator (aka miner in PoW) are selected in proportion to their quantity of holdings in the associated cryptocurrency. This way, instead of utilizing energy to answer PoW puzzles, a PoS miner is limited to mining a percentage of transactions that is reflective of their ownership stake. For instance, a miner who owns 3% of the coins available can theoretically mine only 3% of the blocks.[43]

2.2.6 Smart Contracts

While the inception of blockchain technologies is identified primarily through the foundation of cryptocurrencies, the past decade has seen growth far beyond the underpinning digital currency. Notably, a significant use case of blockchain technologies is *smart contracts*, pioneered by Ethereum[4]. A "smart contract" is simply a program that stored on the Ethereum blockchain and executed on the Ethereum Virtual Machine (EVM). It's a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain[1].

Each smart contract has the properties of a regular user, having a balance and ability to participate in transactions. However, they are automated, and once deployed on the blockchain, they run as programmed and enforce rules like a regular contract via code. User accounts can then interact with a smart contract by submitting transactions that execute a function defined on the smart contract. Smart contracts can not be deleted by default, and interactions with them are irreversible.

[4] `https://ethereum.org`

Deploying a contract on the blockchain is similar to a transaction and hence, costs a fee. However, the fees is significantly higher and dependent on the amount of instructions to be executed in the EVM. Since the amount of instructions executed per block is limited, a higher transaction fees provides a higher precedence in that block's execution. This is referred to as Priority Gas Auction (PGAs), which allows a special kind of exploit known as miner ex tractable value (MEV).

2.3 Group

A group G is a finite or infinite set of elements with a group operation that together satisfy the four fundamental properties of closure, associativity, identity and inverse. Let $\cdot$ represent the group operation.

- **Closure** - If $a, b \in G$, then $a \cdot b \in G$
- **Associative** - For $a, b, c \in G$, we have $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- **Identity** - $\forall a \in G$, there exists an identity element i such that $a \cdot i = i \cdot a = a$
- **Inverse** - For each $a \in G$, there exists an element $b = a^{-1} \in G$, such that $a \cdot a^{-1} = a^{-1} \cdot a = i$

A cyclic group is a group where each element can be generated by a single element using the group operation. A cyclic group is denoted by $G = \langle g \rangle$, where g is the generator of the cyclic group.

2.4 Field

A field $\mathbb{F}$ is a finite or infinite set of elements defined over two operations $(+, \cdot)$ that together satisfy the following properties.

- **Associate** - $(a + b) + c = a + (b + c); (a \cdot b) \cdot c = a \cdot (b \cdot c)$
- **Commutative** - $a + b = b + a; a \cdot b = b \cdot a$
- **Additive Identity** - There exists a $0 \in \mathbb{F}$ such that $\forall a \in \mathbb{F}$, $a + 0 = a$
- **Multiplicative Identity** - There exists a $1 \neq 0 \in \mathbb{F}$ such that $\forall a \in \mathbb{F}$, $a \cdot 1 = a$
- **Additive Inverse** - $\forall a \in \mathbb{F}$, there exists an additive inverse denoted by $(-a)$ such that $a + (-a) = 0$
- **Distributivity of multiplication over addition** : $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

- **Multiplicative Inverse** - $\forall a \in \mathbb{F}$ & $a \neq 0$, there exists a multiplicative inverse denoted by a^{-1} such that $a \cdot a^{-1} = 1$

Given any prime number p, the set $\mathbb{Z}/p\mathbb{Z}$ forms a finite field under addition and multiplication. This field is denoted $\mathbb{F}_p$ and used extensively in elliptic curves.

2.5 Collision Resistant Hash Function

A hash function $\Pi = (Gen, H)$ with key generation function Gen and a keyed compression function $H^s : \{0,1\}^{l'(\lambda)} \leftarrow \{0,1\}^{l(\lambda)}$ for some key $s \leftarrow Gen(\lambda)$ is collision resistant if $l'(\lambda) > l(\lambda)$ and for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$:

$$Pr[\text{HashColl}_{\mathcal{A},H}(\lambda) = 1] \leq negl(\lambda)$$

Here, s is public and it is implicit that l is determined by λ. So, a common way of representing a hash function[5] for arbitrary lengths include $H : \{0,1\}^* \leftarrow \{0,1\}^l$

2.6 Digital Signatures

Digital Signatures are one of the primary application of public key cryptography, which serves as a backbone for functionalities provided by the blockchain. Digital signatures provide both sender verification and non repudiation and thus, making *public verifiability* and *transferability* possible.

Formally, a digital signature scheme provides the tuple of efficient algorithms $Gen, Sign, Verify$ as follows, where λ is the security parameter :

- $Gen(1^\lambda)$ - Generates a public key - private key pair pk and sk respectively.

- $Sign(sk, m)$ - The sign algorithm takes the secret key (sk) and the message (m) as input and generates a signature σ as output.

- $Verify(pk, \sigma)$ - The verify algorithm takes the public key (pk) and signature σ as input, and outputs 1 if the signature is valid, else 0

Each DS scheme satisfies the following properties :

- **Determinism** - $Verify$ algorithm must always be deterministic.

- **Correctness** - Every DS scheme must satisfy the correctness property.

$$\forall \lambda \in N, \forall (sk, pk) \leftarrow Gen(1^\lambda), \forall m \in M : Verify(pk, Sign(sk, m)) \rightarrow 1$$

[5] The HashColl game is not included here. Please refer [48]

- **Unforgeability** - A DS[6] scheme is existentially unforgeable under chosen message attacks (EUF-CMA) if for all PPT adversaries there exists a negligible function *negl* such that for all $\lambda \in N$, we have

$$Pr[Game^{euf-cma}_{\mathcal{A},DS}](\lambda) \leq negl(\lambda)$$

In our construction, we extensively use Boneh–Lynn–Shacham (BLS) signature scheme [22], which achieves unforgeability through elliptic curve cryptography and bilinear pairings.

2.7 Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is an approach to public key cryptography based on algebraic structure and operations of elliptic curves over the finite field $\mathbb{F}_p$ given by the prime number p. The main advantage of using ECC is that they allow smaller key size in compared to the other schemes like RSA and Diffie-Hellman over the plain Galois fields. For example, the equivalent security of 1024 bits of RSA keys is provided by 160 bits of ECC keys.

2.7.1 Curve Equation

The elliptic curve (Fig. 2.2) is a cloud of points, which satisfy the curve equation $y^2 = x^3 + ax + b \mod p$. In the equation, all the parameters x, y, a, b belong to $\mathbb{F}_p$. The parameters x and y are coordinates on the plane and coefficients a and b determine points on the curve[7].

Each elliptic curve must satisfy the following properties,

- It must have distinguished point at infinity $\mathcal{O}$
- The curve coefficients must satisfy the equation below to guarantee no singularities in the curve.

$$4a^3 + 27b^2 \neq 0$$

Other domain parameters which are useful in cryptographic primitives include,

- **G** - The generator or base point of the curve.
- **n** - The order of the curve generator point G.
- **h** - The co factor of the curve. It is the quotient of the number of curve-points.

[6] The DS forgery game is not included here. Please refer [48]

[7] For the sake of brevity, this section doesn't describe in detail about projective geometry

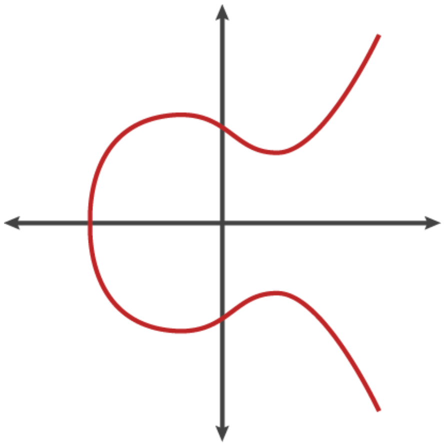


Figure 2.2: Elliptic Curve of the form $y^2 = x^3 + ax + b \mod p$

Hence, an elliptic curve is defined as tuple of six parameters **(p,a,b,G,n,h)**. The points on the curve are represented by their affine projection.[8] Points, which are represented in affine coordinates are vectors with an x and y component but, they are over $\mathbb{F}_p$. For example, the point A is represented as

$$A = \begin{pmatrix} a_x \\ a_y \end{pmatrix}$$

2.7.2 Point Addition

Point addition of two points P and Q where $P \neq Q$ is given as follows,

$$R = P + Q = \begin{pmatrix} p_x \\ p_y \end{pmatrix} + \begin{pmatrix} q_x \\ q_y \end{pmatrix} = \begin{pmatrix} r_x \\ r_y \end{pmatrix}, P \neq Q$$

$$s = \frac{p_y - q_y}{p_x - q_x}; r_x = s^2 - p_x - q_x; r_y = s(p_x - r_x) - p_y$$

Note, with point at infinity $\mathcal{O}$, for any point P

$$\mathcal{O} + P = P + \mathcal{O} = P$$

2.7.3 Point Doubling

Addition of identical points is known as point doubling and for any point P, it is given as follows,

$$R = P + P = 2P = \begin{pmatrix} r_x \\ r_y \end{pmatrix}$$

[8] In implementation, points are represented using Jacobian coordinates for better performance.

$$s = \frac{3p_x^2 + a}{2p_y}; r_x = s^2 - 2p_x; r_y = s(p_x - r_x) - p_y$$

2.7.4 Scalar Multiplication

Multiplication of a scalar over a point can be performed by repeated point addition and is defined as follows,

$$R = kP = P + P + P + ..(k \text{ times})$$

However, this is computationally inefficient and can be sped up by utilizing the exponentiation by squaring technique. The technique utilizes binary representation of the scalar value, to perform repeated point doubling or point addition based on the binary value. The technique takes only $\log(k)$ point additions in contrast to vanilla consecutive k point additions.

2.7.5 Elliptic Curve Discrete Logarithm Problem

The Elliptic Curve Discrete Logarithm Problem (ECDLP) forms the basis of mathematical hardness, upon which several cryptographic schemes are derived and built upon.

Formally, Let E be an elliptic curve over a finite field $\mathbb{F}_p$. Given two points $P, Q \in E(\mathbb{F}_p)$ such that $Q \in \langle P \rangle$, it is hard[9] to determine a scalar k such that, $Q = kP$

In our construction, we extensively use the BLS12-381 curve [10], which also satisfies a special property - *pairing friendly*

2.8 Bilinear pairing

Let $\mathbb{F}_p$ be a finite field over prime p, G_1, G_2 be two cyclic groups of prime order p and G_T be another cyclic group of order of a product of p.

A pairing is a map defined as $\hat{e} : G_1 \times G_2 \to G_T$, which satisfies the following properties:

- **Bilinearity**

$$\forall a, b \in \mathbb{F}_p^*; P, R \in G_1; Q \in G_2; \text{ we have}$$

$$\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$$

$$\hat{e}(aP, Q) = \hat{e}(P, Q)^a = \hat{e}(P, aQ)$$

$$\hat{e}(P + R, Q) = \hat{e}(P, Q) \cdot \hat{e}(R, Q)$$

[9] For computationally bound adversaries, especially probabilistic polynomial bound

[10] `https://electriccoin.co/blog/new-snark-curve/`

- **Non-degeneracy**

  $$\forall P \in G_1, \forall Q \in G_2 : \hat{e}(P, Q) \neq I_{G_T}$$

  where I_{G_T} is the identity element of G_T

- **Computability** There is an efficient method to compute $\hat{e}(P, Q)$

In the case of elliptic curves, particularly BLS12-381, we have the following groups defined

- G_1 is over the elliptic curve $E(\mathbb{F}_p)$ of the form $y^2 = x^3 + 4$

- G_2 is over the elliptic curve $E(\mathbb{F}_{p^{12}})$, where $\mathbb{F}_{p^{12}}$ is the 12th field extension of $\mathbb{F}_p$. However, $\mathbb{F}_{p^{12}}$ is extremely inefficient to compute over and hence we employ a *twist*, which is a coordinate transformation to transform higher degree field extension to a lower degree field extension[11]. In the case of BLS12-381, the curve employs a sextic twist, thus reducing the degree by a factor of six.

  Hence, G_2 is over the elliptic curve $E(\mathbb{F}_{p^2})$ of the form $y^2 = x^3 + 4(1 + i)$

- G_T is over the similar field extension $\mathbb{F}_{p^{12}}$

Bilinear pairings are primarily used in the cryptography of Identity based Encryption(IBE) and recently zkSNARKs, which are both significant in our constructions.

2.9 zkSNARKs

zkSNARKs stands for "Zero-Knowledge Succinct Non-Interactive Argument of Knowledge". Intuitively, it refers to a proof construction, where a prover can prove the possession of certain information to a verifier without leaking any information related to the secret and no interaction between the prover and the verifier. zkSNARKs comes under the research domain of Public Verifiable Computing.

In general, the zkSNARKs is described by a tuple of algorithms namely $Gen, Prove, Verify$ as follows

- *Gen* - The setup phase of the protocol which generates the two strings, namely the proving key *pk* and the verifying key *vk*. It is typically ran by a trusted third party.

- *Prove* - The prove algorithm takes proving key *pk*, statement u and witness w as input and outputs the proof π.

[11] The complex numbers are a quadratic extension of the real numbers.

- $Verify$ - The verify algorithm takes the verification key vk, statement u and the proof π as input and outputs 1/accept if the proof is valid else 0/reject.

In modern zkSNARK constructions, the program that is to be checked is compiled into a quadratic equation of polynomials : $t(x)h(x) = w(x)v(x)$, where the equality holds if and only if the program is computed correctly. The prover wants to convince the verifier that this equality holds [42]. Conservatively, construction of the above zkSNARKs comprises of 5 distinct stages of transformation[12].

- Initial computational program.
- Arithmetic circuit, where each step is composed of basic arithmetic operations.
- R1CS (Rank 1 Constraint System), where each constraint corresponds to a single logic gate.
- QAP (Quadratic Arithmetic Programs), which are obtained using Lagrange interpolation.
- zkSNARKs from QAPs through either homomorphic encryption or bilinear pairings.

The construction must satisfy the properties of zkSNARKs listed as below,

- **Succinct** - The size of message is small compared to the size of the actual computation itself. This is achieved through random sampling at the QAP stage, where the verifier randomly chooses a point and evaluates the polynomial at that point to verify the proof with very high probability (not deterministic). This reduces both the proof size and the verification time tremendously.
- **Non interactive** - In zkSNARKs, there is only the setup phase and a single transfer of proof from prover to verifier. Thus, there is little to zero interaction.
- **ARguments** - The verifier is protected against only computationally limited provers. However, this is the case for domain of public key cryptography, where schemes offer only computational soundness.
- **of Knowledge** - It is nearly impossible for the prover to construct a proof without knowing the *witness string* for that statement.

[12] Mathematical detail of these transformations are out of scope for our construction

- **zero knowledge (zk)** - The verification happens without revealing any information beyond the validity of the statement itself, particularly the *witness string* itself.

Chapter 3

Motivation

In this chapter, we motivate our work by looking at the attacks against blockchain technologies and exploring the severity of threat levels to show frontrunning attacks as one of the major threats in the space. Cloud Security Alliance [30] provides a comprehensive list of 203 vulnerabilities or attack vectors over modern blockchain platforms. However, we only discuss the most common attacks applicable to majority of the platforms and omit familiar attacks like DDoS and Phishing, which are not specific to blockchain technology.

3.1 Double Spending

Before the case of attack on blockchain technologies, the novel innovation was targeted at solving two major problems namely *The Byzantine Generals Problem*[55] and the double spending problem. The double spending problem is a scenario where the same digital currency can be spent more than once because a digital currency has no physical standing. Since, it's based of a digital record, it can falsified or duplicated.

Nakamoto's consensus [58] from 2008 tackles the problem by making the transactions public and using PoW to choose a order of transactions for a particular block. This block is then added to the ledger, and the transactions part of that block are the only confirmed transactions. Contrary to account system, Bitcoin utilizes the UTXO model (unspent transaction output), which refers to the amount of cryptocurrency someone has remaining after executing a transaction. This output, will serve as the input for the next transaction made by the same user. Once the miner solves the PoW challenge, in order for the transaction to be included in a block, the outputs that it spends from must have not been spent by any other transaction in the blockchain nor by any other transaction in the block. This mechanism is also referred to as a distributed time-stamping server.

In the case of copies of ledger with simultaneous transactions, where each report a different state, the longest one is always accepted as valid, since it was mined with the most amount of work done. Hence, to forge a transaction into an already added block, the node must redo the PoW for that block and subsequent other blocks due to the cascading effect of hash based linked list. Under a stable consensus, this is practically impossible and makes a distributed blockchain immutable.

However, there are two key takeaways from this approach,

- The transactions are protected as long as there is stable consensus.
- The miner's ordering of transactions for a particular block is final.

3.2 51% Consensus Attack

Since, the validity of the transactions depend upon the stability of the consensus, there have been several attacks theorized to destabilize consensus. One of the most notable attacks is the 51% consensus attack. In the case of PoW, the success rate of the miner solving the challenge is directly proportional to the amount of computational resources invested. Hence, if someone controls 51% of total computational resources available, it is guaranteed that they will solve the mining challenge with very high probability.

In that scenario, they could add new blocks on their own with forged transactions, thus making double spending possible. In a truly distributed network, this attack would be impossible. However, to improve the chance of mining success, mining pools combine different node's computational resources for mining and in return reap the rewards of mining in proportion to their computational resource contributed to mining. According to btc.com [1], the majority of computational resource of bitcoin is only distributed among four mining pools (Fig. 3.1). Hence, if they collude, it is possible to destabilize bitcoin's consensus.

In most mature blockchain networks, since the mining block reward is profitable, the chance of consensus destabilization is minimal. But, several networks have indeed suffered the 51% attack, most notably Bitcoin Gold [53] in 2018 and 2020.

3.3 Sybil Attack

A Sybil attack is a variant of consensus destabilization, where a large central authority impersonates as multiple nodes through fake pseudonyms. A similar protection rationale for 51% Consensus attack with respect to PoW applies to large scale Sybil attacks as well, thus making the mature networks relatively safe to Sybil attacks.

In the case of targeted Sybil attacks, the attacker can deploy a large number of fake nodes around a target miner to delay/drop the propagation of

[1] https://btc.com/stats/pool?pool_mode=year

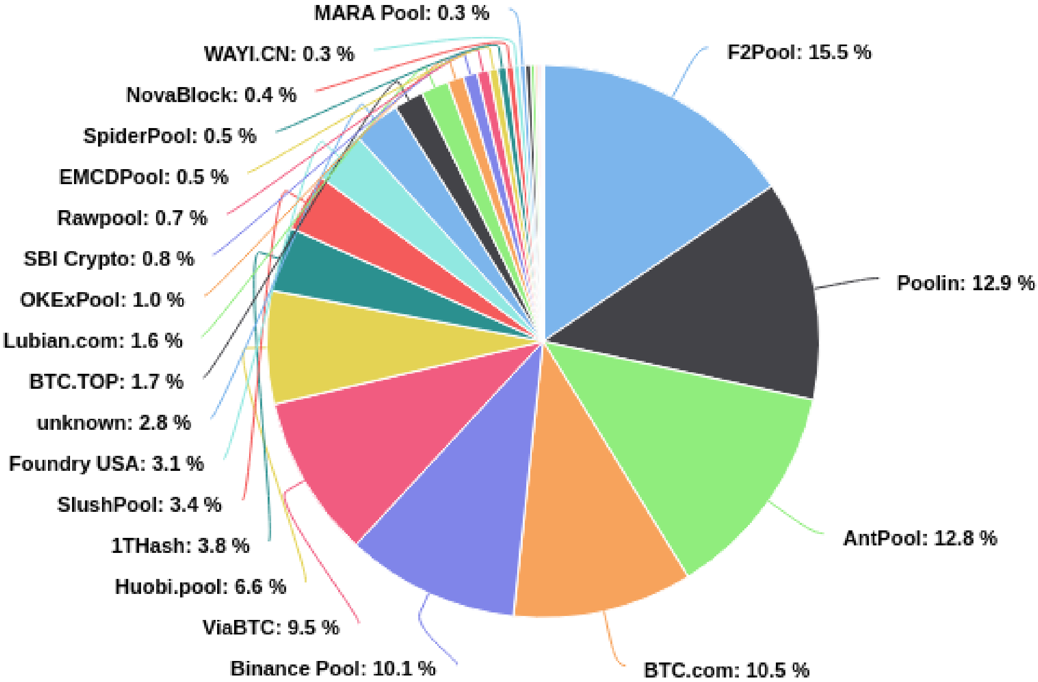


Figure 3.1: Bitcoin's mining pool distribution - 2021

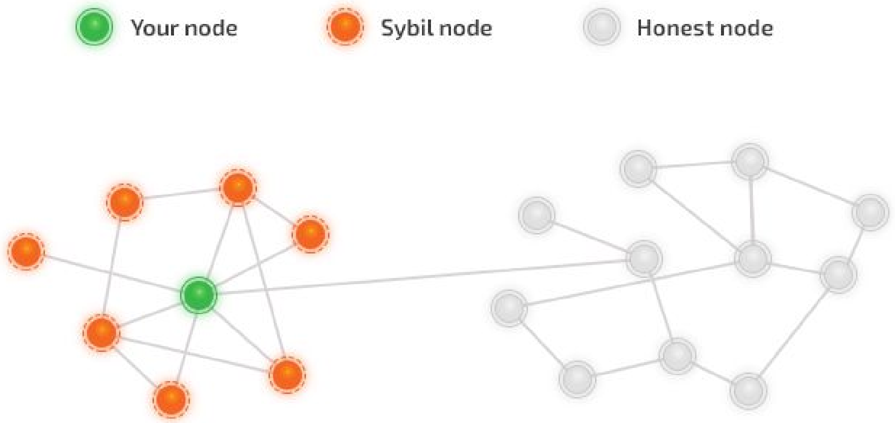


Figure 3.2: A Sybil Attack

blocks (Fig. 3.2). A variant of this attack[71] only requires 32% of the total computing power to achieve double spending.

3.4 Flawed Cryptography

Blockchain technologies are effectively powered by mathematical proven cryptographic protocols. However, in implementation, due to human error or misunderstanding, there have been several instances of flawed implementations of cryptography be actively exploited. Notably, Mt.Gox declared bankruptcy due to a sophisticated attack of transaction malleability in Bitcoin. Transaction malleability, allowed one to alter the nature of the unique identifier of the transaction by modifying the underlying user signature in the transaction data. This was fixed in 2017, by introducing SegWit (Segregated Witness), which removed the signature from the transaction identifier calculation. IOHK was proved to be using a weak hash function known as curl-p and successively it's signature scheme was broken in 2020[45]. The ECDSA signature algorithm in Bitcoin is one of it's key components. However, it suffers a weak randomness due to insufficient entropy available[24].

3.5 Vulnerable Smart Contracts

Smart Contracts are a set of rules in the form of programmable code, executed in a distributed stack based virtual machine. In the case of Ethereum, it is known as Ethereum Virtual Machine (EVM) and the programming language is Solidity. However, vulnerabilities in the source code of the smart contract or the EVM itself is a major source of attack on Ethereum. A common form of smart contract exploit occurs through a reentracy attack, where calling external contracts can take over the control flow, and make changes to your data that the calling function wasn't expecting. This exploit was successfully used in the infamous DAO hack[4] that allowed siphoning of $50 million worth of Ether. Adding to that, since blockchain is immutable, a deployed contract can't be updated when zero day exploits are identified in the source code. Hence, the irreversible damages caused by a vulnerable source code in Ethereum is of significant magnitude in comparison to traditionally deployed web services.

3.6 Frontrunning

In financial markets, frontrunning is an illegal act of utilising insider information to buy or sell stocks before the information is made public to make sizable profits. In the case of cryptocurrencies, once a transaction is made by a sender, it is sent through a series of nodes before reaching the miner node, who participates in the consensus and produces the next block with that transaction. This series of nodes is often referred to as **dark forest** and the time taken to reach the miner is called **propagation delay**. Since transaction[2] information is public through the dark forest, before it's validated, other nodes in the network (typically bots) try to find arbitrage through those transactions among other exchanges. Once identified, they utilize various mechanism to get their transaction before or after the target transaction within the same block to achieve profit. This version of frontrunning is exploited everyday among exchanges and DeFi institutions.

3.6.1 Variants

Frontrunning attacks are constantly evolving with new strategies and exploit vectors being discovered everyday. ConsenSys[31] provides a definite taxonomy on the three variants of frontrunning types common in smart contracts,

- **Displacement**- The attacker tries execute his smart contract before the target smart contract but however it's not important that the target smart contract is executed.

- **Insertion** - The attacker tries to execute his smart contract before the target smart contract and its import that the target smart contract is executed afterwards. This is the traditional front-running scenario and the most prevalent variant. Ethereum nodes intuitively, order the transactions based on their gas price and address nonce. This, however, results in a priority gas auction (PGAs) between arbitrage bots in the network to get their transaction included in the block currently being mined by paying a higher gas price.

- **Suppression** - The attacker tries to execute his smart contract before the target smart contract and suppress the execution of the target smart contract either through exhaustion of resources or high gas prices.

In all of the variants, the size of the attacker and target transactions might vary significantly to achieve maximum profits.

3.6.2 Miner Extractable Value

While arbitrage bots can take advantage of visible transactions through the dark forest, privileged nodes in the network have a significant advantage compared to full nodes. These are miner nodes[3], who essentially finalize the order of transactions and propose the block.

> *Miner extractable value (MEVs) is a special class of frontrunning attacks, where a sensible profit is made by a privileged actor like*

[2] Smart Contracts are inherently transactions and are semantically interchangeable in terms of explanation at certain places.

[3] Well-positioned relay nodes are not considered as they use propagation delay to achieve reordering of transactions, which are out of scope for our construction

> *miner through their ability to arbitrarily include, exclude, or reorder transactions from the blocks they produce.*

Other nodes in the network, as mentioned, are capable of frontrunning attacks. However, due to the unique position of miner as block producer, a miner is capable of backrunning, where he places his own transactions after a target transaction to take advantage of the market fluctuations. MEV strategies are constantly evolving and have been difficult to monitor from an analytics perspective.

3.6.3 Sandwich attack

A sandwich attack is another MEV exploit, where the miner sandwiches his own transaction between two target transactions to make a profit. The strategy is based on the idea of manipulating the price of an cryptocurrency as a result of buying/selling the same. This is a popular attack scenario over decentralized exchanges (DEX) and incurs a significant loss of profits. However, the sandwich attack is only profitable if the price before and after the trade is large enough to accommodate the network and exchange fees.

3.6.4 Time Bandit attack

An extreme case of MEV exploitation is a time-bandit attack, which can destabilize the consensus. In this scenario, the MEV exploits became so lucrative that they net a higher profit than the miner block reward. Hence, to further exploit MEVs and maximize the profit, the miner deviates from the consensus and starts restructuring previously confirmed blocks for exploitation. This is indeed achieved by using the resulting MEV to subsidize a 51% attack to rewrite required past blocks in the history, sandwiching previously confirmed transactions and thus destabilizing consensus[32].

3.6.5 Feasibility

While mounting a frontrunning attack may sound simple by definition, practically exploiting a attack requires consideration of several issues and the requirement of high frequency trading equipment and well place nodes in the network. However, MEVs have overcome this obstacle and is constantly being exploited on different networks.

Flashbots website [4] shows a cumulative representation of MEV being exploited on the Ethereum Network (Fig. 3.3). Since January, a total sum of $720 million (incl. tx fees) have been profited through MEV, which establishes the scale of the exploitation. While other attacks documented depend on obscure or zero day vulnerabilities to achieve their attack, MEV is based on open mempools. The public availability and high payoff has attracted a large number of skilled attackers to discover new ways to exploit MEV. Thus it exists

[4] https://explore.flashbots.net/

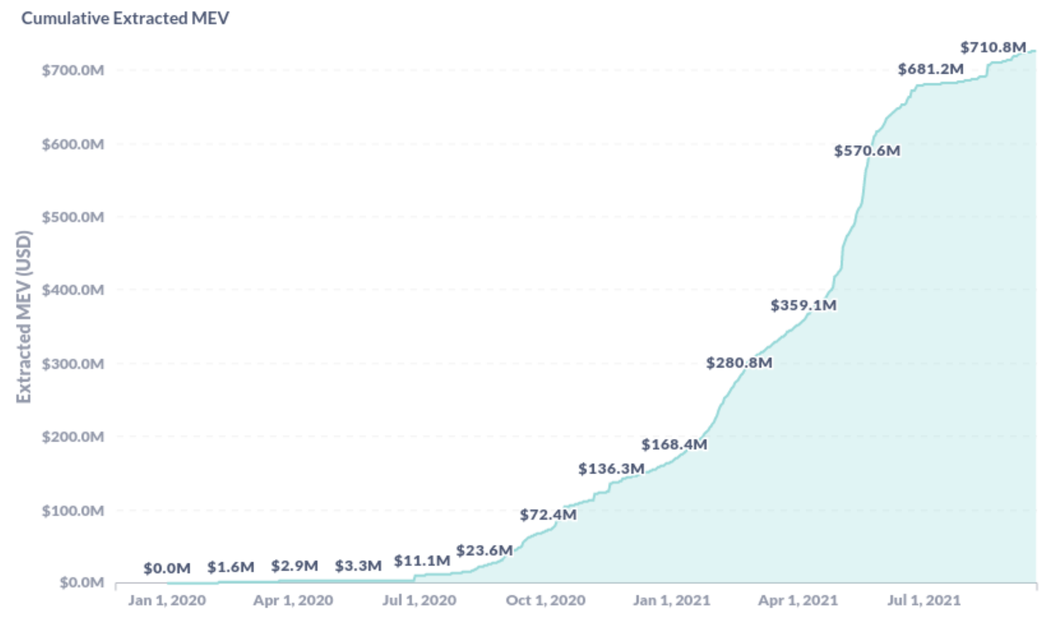


Figure 3.3: Cumulative extracted MEV since January 2020

as a recurring exploit to be mitigated, and, hence motivating our approach towards timelock encryption as a solution for mitigating frontrunning attacks.

Chapter 4

Related Work

Preventing frontrunning attacks have been well studied in the literature, since they have been present from the dawn of traditional financial markets. In this chapter, we perform a review of the literature on frontrunning mitigation techniques and identify the position our construction over the literature. Broadly speaking, the mitigation approaches are centered among three domains namely (a) *Economics*, (b) *Ordering Fairness* and (c) *Confidentiality*. Our construction is an extension of the confidentiality approach using timelock encryption.

4.1 Economics

Game theoretic approach to frontrunning protection stems from traditional financial exchanges, who employ an *order book* managed by the exchange. An order book is an electronic list of buy and sell order/bids for each stock organized by their price level. This promotes transparency among the traders and identification of market manipulations. However, a negative result of this approach allows frontrunning through network delay, by using faster connections to trading exchanges and low-latency trading algorithms[56].

In the case of decentralized exchanges for cryptocurrencies (DEX), there is no single central authority to maintain an order book. This created the need for an automated system for matching buy/sell prices based on bid/ask prices in a decentralized setting, which was solved by Automatic Market Makers (AMMs).

4.1.1 Automatic Market Maker (AMM)

AMMs are smart contracts maintaining a sufficient liquidity pool of various assets part of the DEX, where users can swap between the assets based on pricing formulas. AMMs are inherently hard to design and implement because they involve sophisticated economic investment mechanism. Due to the way AMMs work, a large liquidity pool is required to prevent less price slippage on large orders. AMMs are successfully deployed in major exchanges UniSwap[8], SushiSwap[2] and PancakeSwap, which have market cap of 20%, 13% and 6% respectively[3][1]. Most AMMs are based off Ethereum and Binance Smart Chain and use the foundation for $x * y = k$ market makers[66] or constant product market makers to protect against frontrunning.

Bartoletti et al.[13] provide a formal model of AMMs tailored for decentralized finance, formalizing the notion of frontrunning as *the arbitrage problem*. Bentov et al.[17] proposed a new DeFi exchange, which uses AMMs on trusted hardware to prevent frontrunning attacks. Zhou et al.[73] coined the protocol Automated Arbitrage Market Maker (A^2MM), proposing uniting of different AMMs under the same chain to achieve automatic MEV collection instead of competitive MEV, which could destabilize consensus. Ciampi et al. [12] promotes fair exchange through their Σ-*trade protocol*, where hybrid exchanges are carried out over both on/off chain. The frontrunning protection is achieved through cryptographic mechanisms that allow verifiability of the execution trace of the market maker.

However, AMM are still susceptible to frontrunning by miners, who can reorder transactions to limit the effect of AMMs. Several attacks [33, 38, 62, 72, 74] have been proposed to exploit currently used AMM for profitable MEVs. Angeris et al. [9] showed it's impossible to build a privacy based market makers under reasonable adversary models based on current design of the market makers.

4.2 Ordering Fairness

A second approach to tackling frontrunning attacks is ordering fairness, where transaction ordering is done under a certain set of rules to ensure fairness of profits. There are two approaches to achieving fairness in a distributed system (a) *No one receives a profit* and (b) *Everyone receives an equal profit*. The first approach is studied through various ordering mechanisms, where the total achievable profits through frontrunning is nullified or redistributed back into the system. The second approach explores the fair ecosystem for MEV extraction.

It is worth mentioning here, that in a distributed setting, achieving perfect fairness is impossible. This is a consequence of Condorcet paradox from social choice theory, in which collective preferences can be cyclic, even if the preferences of individual voters are not cyclic [49].

4.2.1 Transaction ordering

Transaction reordering stems from the fact that one of the remediation is to remove the benefit of frontrunning from the protocol. First In First Out (FIFO) model of transaction order is a fair ordering, but, however, not possible

[1] dYdX with market cap 20% is not considered, since it uses *limit order books* instead of AMMs

in decentralized setting due to the propagation delay to different nodes in the network. Although, certain protocols have achieved FIFO by sacrificing true decentralization and employing a trusted third party to assign sequence number to transactions[68].

Another form of ordering can be achieved from certain uninfluenced pseudorandomness part of the transaction. Canonical Transaction Ordering Rule (CTOR)[6] in Bitcoin ABC is an example, where transactions are ordered based on the lexicographical order of their hashes. However, this can still be frontrun by adding void data to the transaction to manipulate hashes and bypass CTOR with relatively less complexity. Other proposals include transactions forcing the order themselves through requiring a particular state for execution.

Several protocols use a directed acyclic ordering (DAG)[7, 34] ordering of transactions, specifically from the context of improving consensus latency and throughput of the protocol. However, compared to a linear ordering in a block, a DAG ordering can have relatively higher complexity to frontrun.

Finally, certain protocols define their consensus protocol over fair ordering of transactions. HashGraph[11] implements the *gossip the gossip* protocol, where nodes gossip the transactions history itself in the form of a hashgraph instead of traditional individual transactions. Aequitas[50] protocol provides a fair-ordering consensus for both leader and leaderless settings with asynchronous and synchronous communication mechanism. The protocol also models their upper-bound on fairness using the Condorcet paradox.

4.2.2 Flashbots

The second component of ordering fairness has its origin from distributing the profits equally. Flashbots[2][3] is a research organization particularly focused on this activity to achieve permissionless, transparent, and fair ecosystem for MEV extraction. It stems from their three main goals (a) *Democratizing Access to MEV Revenue*, (b) *Bringing Transparency to MEV Activity* and (c) *Redistributing MEV Revenue.*

Two notables proposals from Flashbots research include *MEV-SGX*[18] and *Proposer/block builder separation*[67]. In MEV-SGX, the nodes part of the auction are required to run their software in a secure enclave like Intel SGX to guarantee that the software utilized for the auction is not tampered or modified. This is done by remote attestation handshakes prior to the engagement on the auction software. In *Proposer/block builder separation*, the goal is to separate the role of transaction ordering from the block proposer. Instead of selfish extraction of MEVs, the proposer depends on entities called

[2] https://docs.flashbots.net/

[3] Flashbots mainly concentrate on Ethereum, however the research is applicable to several blockchain technologies

block-builder who propose variants of the block and associate a fee to it for the proposer. Here, the choice of the proposer is narrowed down to choosing the variant with highest fee, which can be enforced by cryptographic techniques. On the other hand, it promotes an open market for MEV extraction through block-builders and thus achieving the primary goals of flashbots.

4.3 Confidential Transactions

The final section of frontrunning mitigation explore over cryptographic techniques to achieve confidentiality of transactions, so attackers would not be able to explore the contents of the transactions and execute their attack. While zkproofs and timelock encryption are two major areas used to research mitigation for front running, secure multiparty computation protocols have recently been shown capable. Notably P2DEX[14] uses *Insured MPC* to build a universally composable privacy preserving decentralized exchange, while being financially incentivised to behave honestly.

4.3.1 Zero knowledge proofs

The notion of Confidential transaction was first coined by Gregory Maxwell for Bitcoin[69], which allows the amount transferred visible only to the participants of the transaction using Pedersen's commitment. To facilitate public validation of the blockchain, zero knowledge proofs of the validity of the transaction must be submitted.

Zero knowledge proofs suitable for transactions include zkSNARKs (requires a trusted setup) [16], zkSTARKs (large proof size) [15] and Bulletproofs (short and doesn't require trusted setup)[26]. Variants of these zero knowledge proofs have been used in privacy preserving smart contracts like Hawk[52] and Ekiden[29] and decentralized applications (dApps) like AZTEC[70] and a proposed merge between Zcash and Ethereum[41].

4.3.2 Commit/Reveal

In a *commit/reveal* scheme, a user can commit to a digital value while keeping it secret (*hiding*) and reveal only that value (*binding*) at a later time of his choosing. In the case of the blockchain transactions, the transactions are hidden from miners and other frontrunning nodes using a commitment, which is only revealed after a ordering of transactions for that block has been decided. Ideally, this should allow the adversaries to learn no information about the transactions and hence, influence the ordering for frontrunning[4].

A practical commit/reveal scheme was first seen in NameCoin[47], which allowed users to commit to a domain name. After the first transaction is

[4] While the transaction itself is hidden, the metadata surrounding it needs to be public for consensus, which promotes certain kind of attacks. This is one of the major drawbacks of the scheme.

confirmed, the details of the transactions are revealed. This protects the domain name being frontrunned by privileged actors. An enhanced version of commit/reveal scheme is Submarine Commitments[25]. In submarine commits, a user initiates a commit transaction and locks up the required gas to a Submarine address, which is indistinguishable from a freshly minted Ethereum address. Once the transaction order is confirmed, the user issues a reveal transaction, which reveals the details of the contract and the Submarine address issues a unlock transaction (based off a smart contract), which relocates the funds for the target smart contract's execution, essentially thwarting influence of frontrunners (Fig. 4.1)[5]

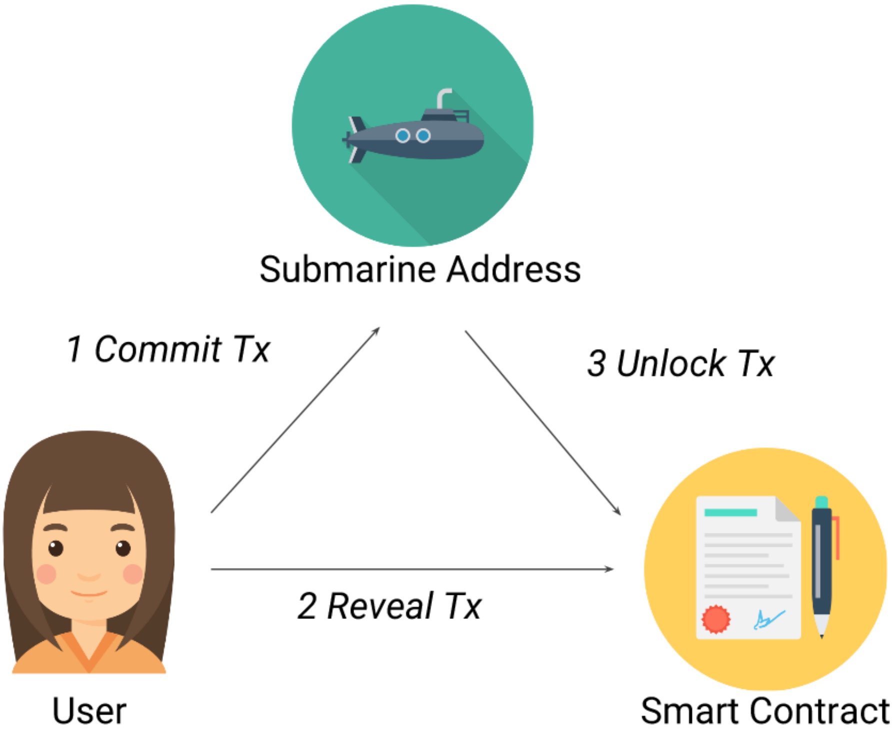


Figure 4.1: Submarine commitment

4.3.3 Timelock Encryption

Timelock Encryption is a variant of commit/reveal scheme, where the reveal is not bound to the user, but to time. In other words, the secret will be automatically revealed after a certain amount of time in contrast to atomic reveals. After the agreed time elapses, the secret would be deemed public. The timelocking mechanism is a perfect fit for frontrunning protection, as the transaction needs to protected only until the sequencing is finalized, which is a fixed period of time.

[5] `https://libsubmarine.org/`

Timelock encryption can be realized by the notion of a *randomness beacon* coined by Rabin[63] or a distributed key generation (DKG) protocol. In the protocol, multiple parties co-operate to provide a verifiable, public source of randomness with a very high entropy at regular intervals of time. A timelock encryption scheme essentially uses this source of randomness at regular intervals and other novel cryptographic techniques to lock a transaction to a particular round of randomness. The key requirements for the scheme is honest majority, where if majority parties collude, the scheme collapses. While timelock encryption has been theorized, most protocols apply different applications to their randomness beacons or there have been no concrete proof of concepts.

Helix protocol uses randomness beacon to elect the leaders for fair consensus mechanism [10]. drand is a publicly verifiable randomness beacon [65], which powers Filecoin's proof of replication [40]. Ouroboros[51] and Ouroboros Praos[35] use a randomness beacon to power their provable Proof of Stake protocol.

Novelty in randomness beacons construction stems from several years of research on Verifiable Secret Sharing and threshold cryptography. SCRAPE[28] protocol proposes an optimized construction for the beacon using linear exponentiations independent of the threshold. Groth[44] gave a new distributed key generation construction in a non interactive setup, a new paradigm is threshold cryptography. While randomness beacon constructions depend on honest majority, Verifiable Delay function can work on stronger notions of security[19].

A Verifiable Delay Function (VDF)[19] requires a specified number of sequential steps to evaluate, yet produces a unique output that can be efficiently and publicly verified. A timelock encryption can be built around VDFs by committing a message m to the value of the unique output, which is produced after a certain delay. The set of sequential steps determine the delay or in the other words the round time of the randomness beacon. VDF construction with reduced computation was shown by Pietrzak[61] using a trusted setup.

4.4 Our Construction

In our construction, we will be modeling a timelock encryption based on the drand randomness beacon and Identity Based Encryption (IBE). Within the time of our research, there have been new protocols which similarly utilize a distributed key generation algorithm to achieve timelocking. Shutter[59] and Anoma[5] are parallel implementations, with a similar goal of removing frontrunning from the consensus using timelock encryption.

Chapter 5

Timelock Encryption

In this chapter, we delve into detail of our practical construction for timelock encryption using various building blocks of innovation. The novelty of our scheme stems from its construction, practicality and proposed road maps for integration into various blockchain technologies.

5.1 Distributed randomness - drand

drand[1] is a public distributed randomness beacon, which provides verifiable, unpredictable and unbiased randomness with high entropy. In this section, we look under the hood of drand's composition to deconstruct its building blocks.

5.1.1 BLS Signature

A core part of drand's protocol and our construction is built from the BLS digital signature scheme[22]. The BLS Signature Scheme provides a tuple of three algorithms $(Gen, Sign, Verify)$ as follows. Since, elliptic curve cryptography is used to build the construction, we will use its primitives to define the scheme.

- *Setup* - The protocol generates the following public parameters which are agreed among all participants[2].

    1. Cyclic Groups G_1, G_2 and G_T of prime order p, where $G_1 = \langle G \rangle$
    2. A pairing $\hat{e} : G_1 \times G_2 \to G_T$
    3. A hash function $H : \{0,1\}^* \to G_2$

- $Gen(1^\lambda)$ - Generates a public key pair of $sk \xleftarrow{\$} Z_q$ and $pk = skG$

- $Sign(m, sk)$ - Generates $\sigma \leftarrow skH(m)$

[1] https://drand.love/

[2] While the original scheme uses G_1 for hashing to a curve, we will use G_2 to stay synchronous with the BLS12-381 curve

- $Verify(\sigma, m, pk)$ - The algorithm outputs 1/accept if

  $$\hat{e}(pk, H(m)) = \hat{e}(G, \sigma)$$

  else 0/reject.

The correctness of the scheme is straightforward from the bilinear pairings. The scheme is secure under the Computational co-Diffie Hellman problem in a Gap Diffie-Hellman Group (co-GDH) [23].

5.1.2 Threshold BLS Signature

While BLS signature is suited for a two party setting, it can be extended into a n party scheme. Now, we define the threshold BLS signature scheme, who's security model is dependent on a t honest parties (threshold) of n parties $(t-of-n)$. This model of signatures is also referred by aggregates signatures.

The threshold BLS signature defines a tuple of five algorithms $(Gen, PartialSign, PartialVerify, SigCombine, SigVerify)$ as follows,

- $Setup$ - Same as BLS signature
- $Gen(1^\lambda)$ - A deterministic algorithm generates a $sk \xleftarrow{\$} Z_q$ and a public key skG. The secret is then made into n shares of $s_i \in Z_q$, such that $pk_i = sk_i G$. The secret key is destroyed after the distribution of secret shares.
- $PartialSign(sk_i, m)$ - Generates $\sigma_i \leftarrow sk_i H(m)$
- $PartialVerify(\sigma_i, m, pk_i)$ - The algorithm outputs 1/accept if

  $$\hat{e}(pk_i, H(m)) = \hat{e}(G, \sigma_i)$$

  else 0/reject.
- $SigCombine$ - A deterministic algorithm that generates σ from t distinct partial signature σ_i for a message m.
- $SigVerify(\sigma, m, pk)$ - The algorithm outputs 1/accept if

  $$\hat{e}(pk, H(m)) = \hat{e}(G, \sigma)$$

  else 0/reject.

5.1.3 Building Blocks

The core component of the drand randomness is the threshold BLS signatures. Notice that threshold BLS signature requires a deterministic algorithm to share the secret and combine the signature without corruption of the shares. This is essentially done in the *setup phase* of the randomness beacon.

In the setup phase, the n parties use Feldman's Verifiable Secret Sharing scheme[39], which allows a secret sk to be shared among n parties using Shamir's secret Sharing[64] and a verification mechanism to check if the shares are consistent with other nodes, so it could be reconstructed later. The reconstruction of shares is enabled by Lagrange's interpolation.

However, there is a second requirement that the dealer who initiates the secret sharing must delete the secret sk after the protocol. This is a strong requirement as any dealer could behave maliciously and store the secret. To overcome this, drand uses Pedersen's Secret Sharing[60], which executes n instances of Feldman's VSS and additional verification to guarantee that no individual gets any information about the secret.

To produce unique randomness for every specific interval with this setup, we enter the *beacon phase*. From now, we identify the intervals as round denoted by r and it increases iteratively from genesis. In the beacon phase, each node generate their own BLS partial signatures for some predetermined message m using $PartialSign$, which changes every round. Then they use a broadcast protocol to communicate the partial signatures to other nodes. Each individual node can create the signature σ using $SigCombine$ after receiving t valid partial signatures.

After recreating the signature, each node can provide the randomness of that round r is given by $\mathbb{H}(\sigma)$, where $\mathbb{H}$ is a collision resistant hash function. It is worth noting here that the epoch between successive rounds must be sufficiently long enough to accommodate the computation delay and the propagation delay for the partial signatures.

At any time after genesis, a beacon provides the following information on request (a) *current round* r, (b) *current round randomness* $\mathbb{H}(\sigma_r)$, (c) *current round signature* σ_r and (d) *previous round signature* σ_{r-1}

5.1.4 Time dependent Signature

As mentioned earlier, for every round, there must be a message m known to all nodes upon which the partial signatures are built. In the current deployed version of drand,

$$m = \mathbb{H}(r||\sigma_{r-1}), \sigma_r = skH_1(\mathbb{H}(r||\sigma_{r-1}))$$

Referring timelock encryption description, one must be able to commit to a time in the future after which the secret will be public. Hence, if we make the *message only dependent on time*, one could build a timelock using that message. With the current message, one could only build a timelock of one epoch where $m = \mathbb{H}((r+1)||\sigma_r)$, since you need the knowledge of σ_r.

Hence, we propose a new message, which is only time dependent to essentially build a timelock encryption scheme around it.

$$m_2 = \mathbb{H}(r), \sigma_r = skH_1(\mathbb{H}(r))$$

where r is the current round of the beacon. This allows us to timelock a input to a future round $r^{'}$, who's secret can be revealed when $r = r^{'}$. The signatures made using m_2 will be referred as v2 (version 2) signatures, which enables us to perform timelock encryption.

5.2 Timelock Encryption with drand

With v2 signatures, we can now commit to a future round of the randomness beacon. To build an practical encryption scheme around it, we explore the notion of identity based encryption in the seminal paper *Identity-Based Encryption from the Weil Pairing* by Boneh and Franklin[20].

5.2.1 Identity Based Encryption

In this paper[20], Boneh et al. show the mechanism of using an arbitrary string ID as a public key to build a secure public key encryption scheme. The idea of adaption arises from the fact that our message $m_2 = \mathbb{H}(r)$ can be incorporated as the arbitrary string.

It is worth noting here that Boneh et al.[20] did include possible extensions of IBE in the paper with a special note on a digital signature scheme, which later came to be the BLS signature scheme. Since it builds over the security models of IBE, they are indeed compatible schemes, which can indeed be used together without any lapse in theoretical security.

Some of the notable properties of the proposed IBE scheme include,

- The scheme is secure under the adaptive chosen ciphertext attacker in the random oracle model (IND-ID-CCA).
- It is based on bilinear pairings.
- IBE derives its security from Bilinear Diffie-Hellman assumption (BDH) on elliptic curves.[20]

5.2.2 Definition

Until now, we have explored all the working parts of a practical timelock encryption scheme. To put all together, we define the notion of timelock encryption as a tuple of three algorithms (Gen, Enc, Dec) defined as follows,

- *Gen* - The Gen algorithm mimics drand. The algorithm generates a secret key s and distributes it among the n nodes and a public key pk.

- *Enc(m, r, pk)* - The algorithm takes the round to timelock r, message to encrypt m and the public key pk as input and outputs a ciphertext c.
- *Dec(c, r, σ)* - The algorithm takes the ciphertext c, round to which the ciphertext was timelocked r and the signature produced by the beacon σ as input and outputs m if r is less than current round else discards it.

5.2.3 Setup

To build the timelock encryption, we require the following cryptographic primitives.

- An elliptic curve with domain parameters $(p, a, b, G_1, G_2, G_T, n, h)$ with a pairing defined as $\hat{e} : G_1 \times G_2 \rightarrow G_T$. $G_1 = \langle \mathcal{G}_1 \rangle$, $G_2 = \langle \mathcal{G}_2 \rangle$
- A randomness beacon with distributed public key Pub that produces a threshold BLS signature σ at round r using $\mathbb{H}(r)$ as message.
- Hash functions

$$\mathbb{H} : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$$
$$H_1 : \{0,1\}^* \rightarrow G_2$$
$$H_2 : G_T \rightarrow \{0,1\}^l$$
$$H_3 : \{0,1\}^l \times \{0,1\}^l \rightarrow \mathbb{Z}_p^*$$
$$H_4 : \{0,1\}^l \rightarrow \{0,1\}^l$$

  where l is the length of the message.
- A semantically secure encryption scheme E

5.2.4 Encryption

The encryption $Enc(m, r, Pub)$ is given as follows :

$$Q_r = H_1(\mathbb{H}(r)) \in G_2^*$$
$$\phi \xleftarrow{\$} \{0,1\}^l \ ; \ k \leftarrow H_3(\phi, m)$$
$$\mathcal{G}_r = \hat{e}(Pub, Q_r) \in G_T$$
$$C = \langle k\mathcal{G}_1, \phi \oplus H_2({\mathcal{G}^k}_r), E_{H_4(\phi)}(m) \rangle$$

5.2.5 Decryption

The decryption $Dec(c, r, \sigma)$ is given as follows :

1. If r is greater than current round, REJECT
2. Parse C as $\langle U, V, W \rangle$

3. Compute $\phi = V \oplus H_2(\hat{e}(U, \sigma))$

4. Compute $m = W \oplus E_{H_4(\phi)}(m)$

5. Set $k = H_3(\phi, m)$. Test if $U = k\mathcal{G}_1$. If not, REJECT.

6. Output m as the message.

5.2.6 Correctness

To validate the correctness of the scheme, we check for the equality of second term in the ciphertext, since other two are straightforward to verify correctness if the middle term is valid.

As noted earlier, for some round r, the signature $\sigma = sH_1(\mathbb{H}(r))$ and $Pub = s\mathcal{G}_1$ by construction, where s is the secret which was shared among n participants in setup.

To validate,

$$\phi = V \oplus H_2(\hat{e}(U, \sigma))$$

$$= \phi \oplus H_2({\mathcal{G}^k}_r) \oplus H_2(\hat{e}(U, \sigma))$$

$$= \phi \oplus H_2(\hat{e}(Pub, Q_r)^k) \oplus H_2(\hat{e}(k\mathcal{G}_1, sH_1(\mathbb{H}(r))))$$

$$= \phi \oplus H_2(\hat{e}(s\mathcal{G}_1, H_1(\mathbb{H}(r)))^k) \oplus H_2(\hat{e}(k\mathcal{G}_1, sH_1(\mathbb{H}(r))))$$

By definition of bilinear pairings, we get

$$= \phi \oplus H_2(\hat{e}(\mathcal{G}_1, H_1(\mathbb{H}(r)))^{ks}) \oplus H_2(\hat{e}(k\mathcal{G}_1, sH_1(\mathbb{H}(r))))$$

$$= \phi \oplus H_2(\hat{e}(k\mathcal{G}_1, sH_1(\mathbb{H}(r)))) \oplus H_2(\hat{e}(k\mathcal{G}_1, sH_1(\mathbb{H}(r))))$$

$$= \phi$$

Hence, we have a valid and practical timelock encryption scheme secure under the IND-ID-CCA enabled by pairing friendly elliptic curves.

Chapter 6

Implementation

In this chapter, we explore the practical nature of our timelock encryption through concrete implementations. The approach is two fold (a) A proof of concept of timelock encryption for any arbitrary message. (b) Integration with Filecoin for timelocking transactions.

6.1 Proof of Concept

In our proof of concept (PoC), we build a website for timelock encryption, where one can timelock any arbitrary message m to a round r of drand's randomness beacon. The encrypted message can only be decrypted if the beacon's round has elapsed r. The code for the implementation is at the following Github repository (`https://anonymous.4open.science/r/timelock-encryption`). The platform was built using JavaScript and NodeJS.

The project is standalone and can be built on any platform with NodeJS support. The build instructions are provided in the README of the repository.

6.1.1 BLS12-381 Curve

The core of our protocol depends on elliptic curve cryptography and bilinear pairings. For our implementation, we use the BLS12-381 curve, which is used by drand's randomness beacon as well (compatibility) and is pairing friendly.

Deconstructing the BLS12-381 curve, the BLS stems from the authors name Barreto, Lynn, and Scott. 12 is the embedding degree the curve, which is the smallest integer k required to transform the ECDLP problem of an elliptic curve over $\mathbb{F}_p$ into a discrete logarithm problem (DLP) of finite field $\mathbb{F}_{p^k}$, courtesy of the MOV attack[57]. A higher embedding degree offers better security but increase computational complexity. 381 is roughly the number the bits required to represent the point coordinate on the curve. We have already defined the deconstruction of the groups from the field extension in pairing background (Sec. 2.8). We will be using G_1 for keying operations and G_2 for signing operations.

We will be using the *noble-bls12-381*[1] library for performing elliptic curve operation on the BLS12-381 curve including pairing. The library was chosen because it provided :

1. Fast curve operations in JavaScript, which was required to build the application in NodeJS

2. Compatibility with the drand implementation. To implement BLS signatures on the BLS12-381 curve, a set of four parameters[2] must match among implementations for cross compatibility. In our case, the *noble-bls12-381* library and the primitives used in drand's implementation must be compatible. The parameters are

   - **PK_IN** - Public key in G_1
   - **HASH_OR_ENCODE** - true. Both libraries use HashToCurve instead of EncodeToCurve. Recall our definition of H_1,

     $$H_1 : \{0,1\}^* \rightarrow G_2$$

     The hash function maps any arbitrary string to a point on the curve defined from G_2. Both implementations use *hash-to-curve-11*.[3]
   - **DST (Domain Separation Tag)** - BLS_SIG_BLS12381G2_XMD:SHA-256_SSWU_RO_NUL_
   - **RAND_BITS** - 64

With that compatibility check, we can safely use the *noble-bls12-381* to finish our implementation.

6.1.2 Building blocks

From our setup definition in Sec. 5.2.3, we still require concrete implementations of certain crypto primitives to achieve the guaranteed theoretical security. They are implemented as follows,

- $\mathbb{H}$ and H_3 were built from SHA256, a strong hash function

- H_2 and H_4 were built using SHAKE256, which is a standard extensible output function (XOF). XOF is a cryptographic hash function which can provide an output of any arbitrary large number of random bits.

- H_1 is built from *hash-to-curve-11*.

[1] https://github.com/paulmillr/noble-bls12-381

[2] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bls-signature

[3] https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-11

- For semantically secure Encryption E, we use AES in GCM mode with 256 bits of security. Since it's authenticated encryption, we use other parts of the ciphertext from the timelock encryption as additional authenticated data (AAD). This provides an authentication tag to validate that the other part of the ciphertext weren't tampered.
- *Encoding* - Ciphertexts are wrapped in base64 encoding.

With the crypto primitives implemented, we construct our encrypt and decrypt functions. They were exposed as APIs through NodeJS and can be interacted from the website using AJAX requests. The encryption and decryption are done on the server side and hence, the website is very lightweight and with good cloud servers, the website can provide timelock as a service (TLaaS).

6.1.3 Integrating drand

drand provides a JS client library *drand-client*[4], which allows integration of the drand API interfaces easier into the PoC. Since, drand can be self hosted as well, we provide a configuration file under `config/default.json`, which allows any drand hosting to be integrated with our PoC. However, it must have v2 signatures enabled for a working timelock encryption. We have forked the primary repository to add this functionality and some other tweaks to host a functional drand locally for testing our PoC (`https://anonymous.4open.science/r/drand`).

To integrate a drand beacon, we require two parameters,

1. *chainHash* - An unique identifier for a particular distributed randomness beacon. The most common publicly hosted drand version is known as League of Entropy [5].
2. *urls* - Set of URLs for interacting with the beacon's API.

Both these parameters must be added in `config/default.json` for the drand-client library to identify and interact with the correct distributed randomness beacon.

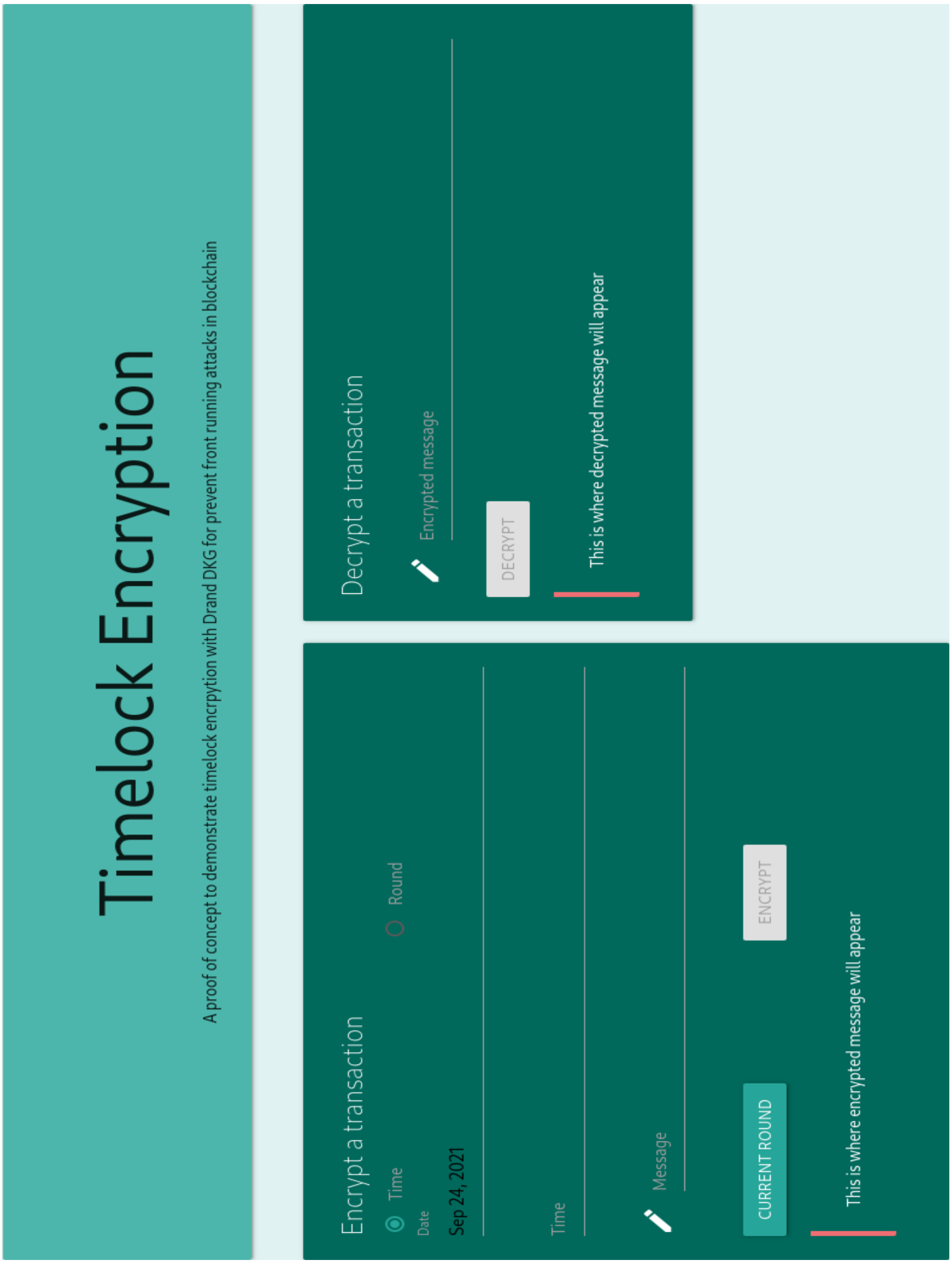
6.1.4 User interface

Fig.6.1 is the website for our PoC of timelock encryption. It provides an easy to use interface for encrypting and decrypting messages using timelock encryption. A message can be encrypted using either a particular round (Fig. 6.2) or a particular time (Fig. 6.3) in the future. There's a button present to fetch the current round of the drand beacon, so the user can use it to timelock

[4] `https://drand.love/developer/clients/#js`

[5] `https://www.cloudflare.com/leagueofentropy/`

to a round after that. If time is provided, the backend automatically converts the time in future to the correct future round of the drand beacon, since each round occurs at regular epochs of time. Attempting a message to decrypt a message before the round has elapsed (Fig. 6.4) will result in a warning to wait for that round to be elapsed. If the round has elapsed, the message will be decrypted (Fig. 6.5).



Timelock Encryption

A proof of concept to demonstrate timelock encryption with Drand DKG for prevent front running attacks in blockchain

Decrypt a transaction

Encrypted message

This is where decrypted message will appear

Encrypt a transaction

● Time ○ Round

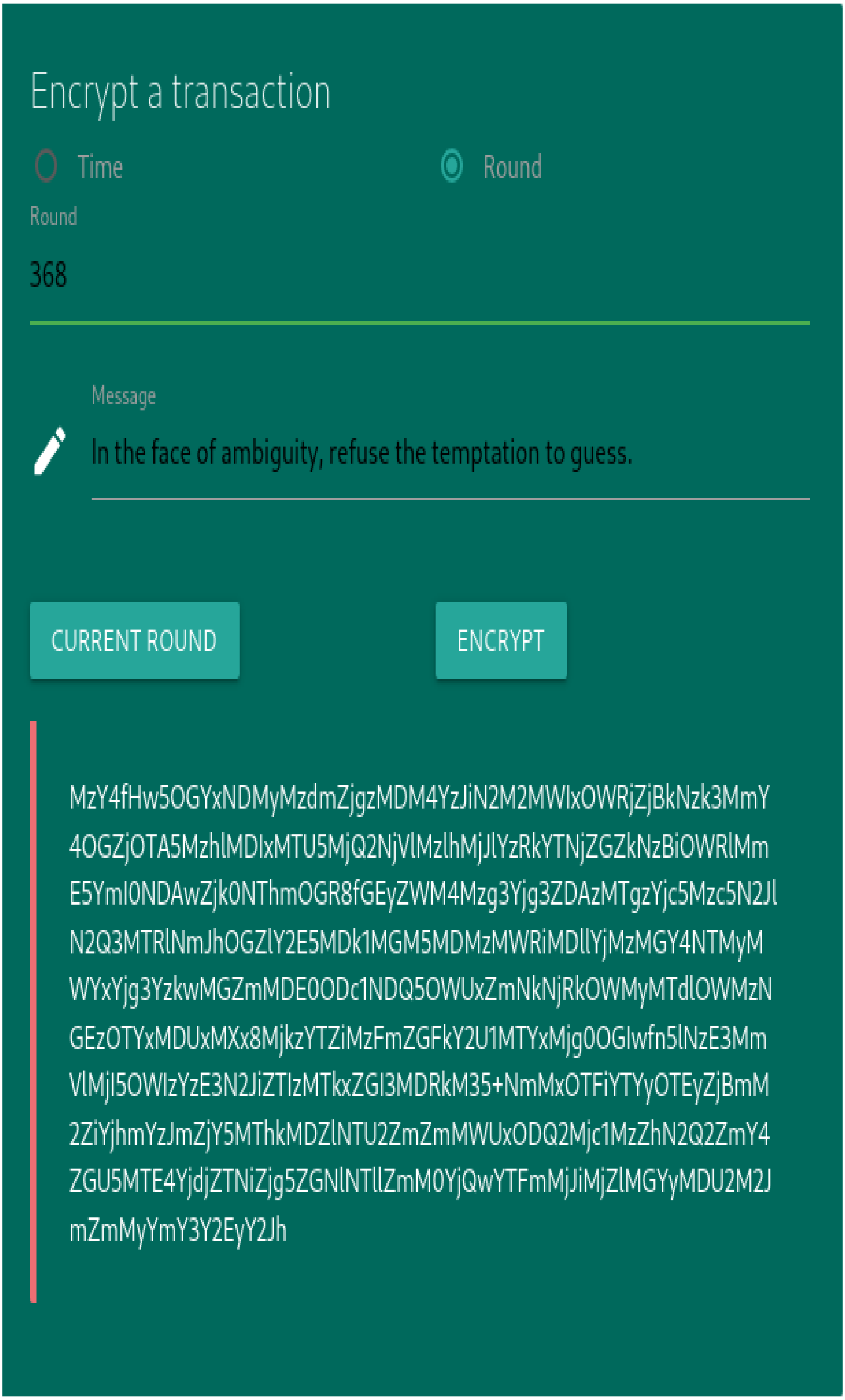
Date: Sep 24, 2021

Time

Message

This is where encrypted message will appear

Figure 6.1: User Interface



Encrypt a transaction

○ Time ● Round

Round

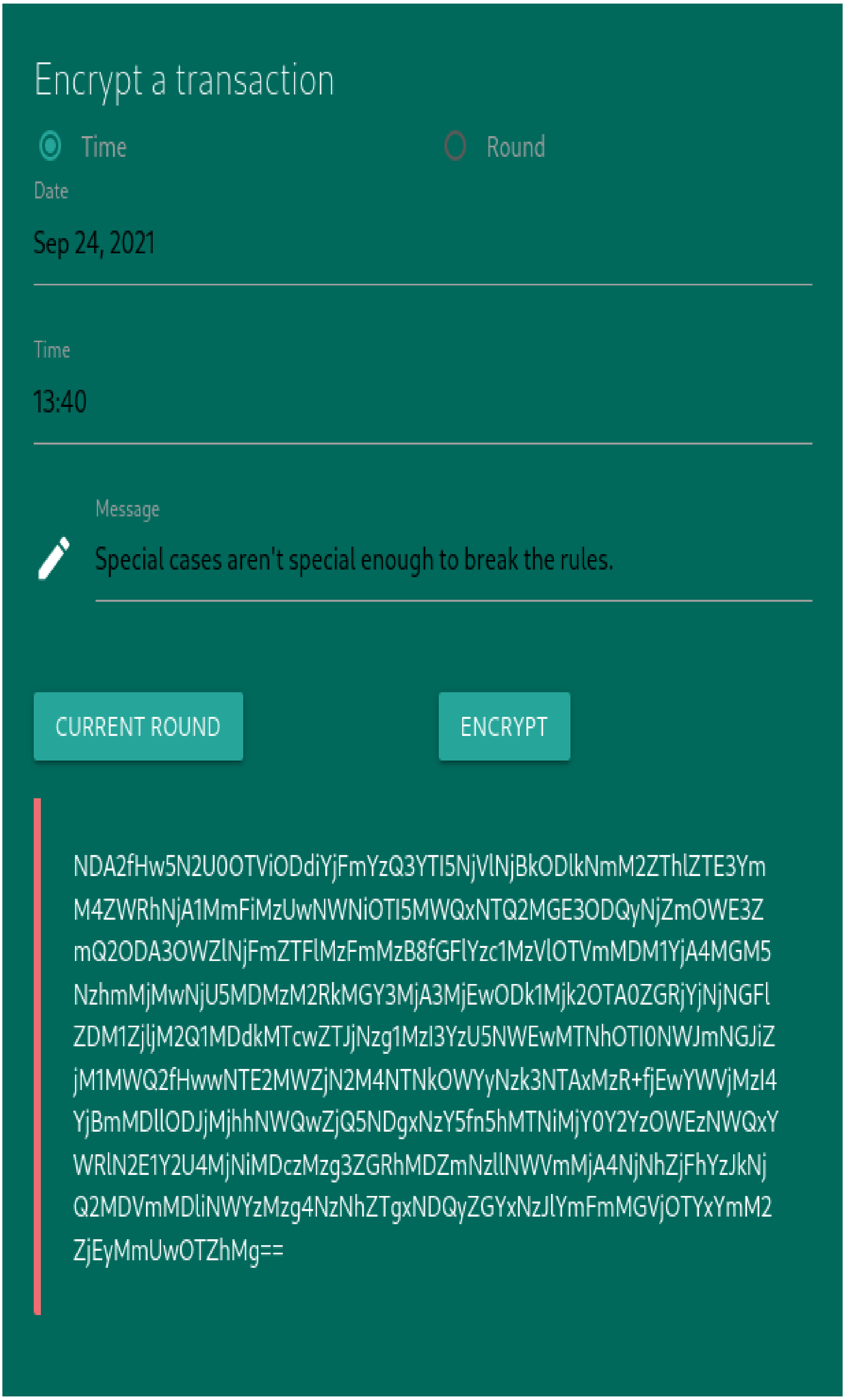
368

Message

In the face of ambiguity, refuse the temptation to guess.

```
MzY4fHw5OGYxNDMyMzdmZjgzMDM4YzJiN2M2MWIxOWRjZjBkNzk3MmY
4OGZjOTA5MzhlMDIxMTU5MjQ2NjVlMzlhMjJlYzRkYTNjZGZkNzBiOWRlMm
E5YmI0NDAwZjk0NThmOGR8fGEyZWM4Mzg3Yjg3ZDAzMTgzYjc5Mzc5N2Jl
N2Q3MTRlNmJhOGZlY2E5MDk1MGM5MDMzMWRiMDllYjMzMGY4NTMyM
WYxYjg3YzkwMGZmMDE0ODc1NDQ5OWUxZmNkNjRkOWMyMTdlOWMzN
GEzOTYxMDUxMXx8MjkzYTZiMzFmZGFkY2U1MTYxMjg0OGIwfn5lNzE3Mm
VlMjI5OWIzYzE3N2JiZTIzMTkxZGI3MDRkM35+NmMxOTFiYTYyOTEyZjBmM
2ZiYjhmYzJmZjY5MThkMDZlNTU2ZmZmMWUxODQ2Mjc1MzZhN2Q2ZmY4
ZGU5MTE4YjdjZTNiZjg5ZGNlNTllZmM0YjQwYTFmMjJiMjZlMGYyMDU2M2J
mZmMyYmY3Y2EyY2Jh
```

Figure 6.2: Encrypt a message using round number



Encrypt a transaction

● Time ○ Round

Date

Sep 24, 2021

Time

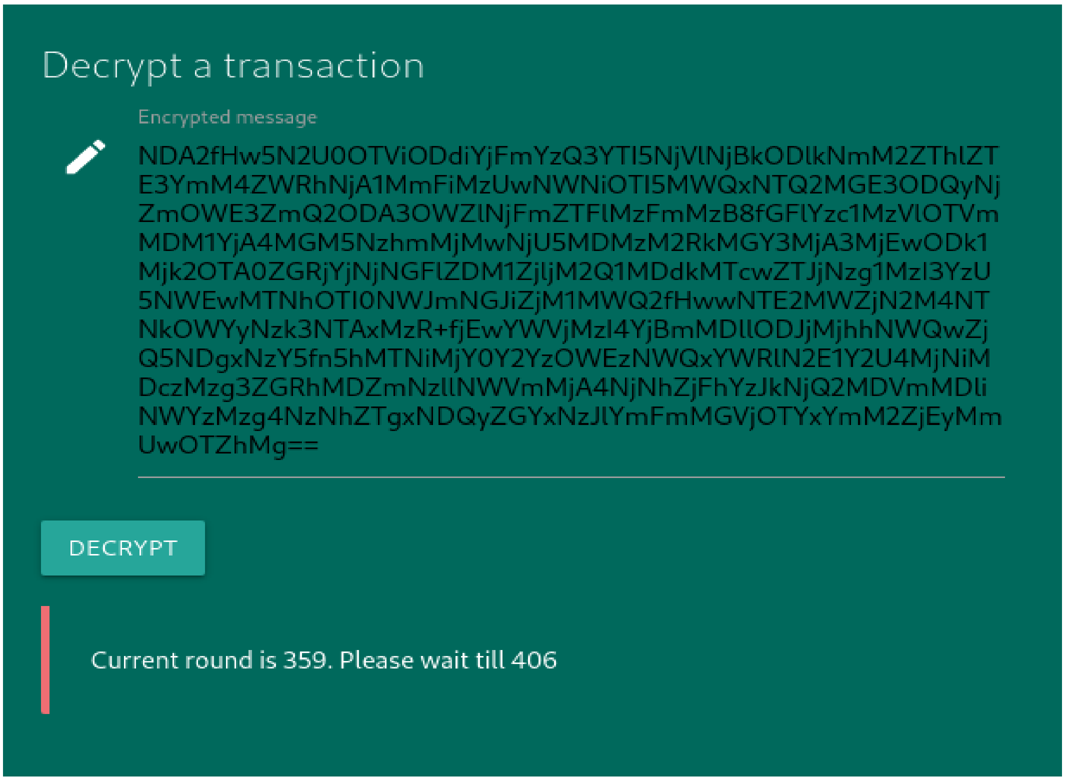
13:40

Message

Special cases aren't special enough to break the rules.

NDA2fHw5N2U0OTViODdiYjFmYzQ3YTI5NjVlNjBkODlkNmM2ZThlZTE3Ym
M4ZWRhNjA1MmFiMzUwNWNiOTI5MWQxNTQ2MGE3ODQyNjZmOWE3Z
mQ2ODA3OWZlNjFmZTFlMzFmMzB8fGFlYzc1MzVlOTVmMDM1YjA4MGM5
NzhmMjMwNjU5MDMzM2RkMGY3MjA3MjEwODk1Mjk2OTA0ZGRjYjNjNGFl
ZDM1ZjljM2Q1MDdkMTcwZTJjNzg1MzI3YzU5NWEwMTNhOTI0NWJmNGJiZ
jM1MWQ2fHwwNTE2MWZjN2M4NTNkOWYyNzk3NTAxMzR+fjEwYWVjMzI4
YjBmMDllODJjMjhhNWQwZjQ5NDgxNzY5fn5hMTNiMjY0Y2YzOWEzNWQxY
WRlN2E1Y2U4MjNiMDczMzg3ZGRhMDZmNzllNWVmMjA4NjNhZjFhYzJkNj
Q2MDVmMDliNWYzMzg4NzNhZTgxNDQyZGYxNzJlYmFmMGVjOTYxYmM2
ZjEyMmUwOTZhMg==

Figure 6.3: Encrypt a message using time



Decrypt a transaction

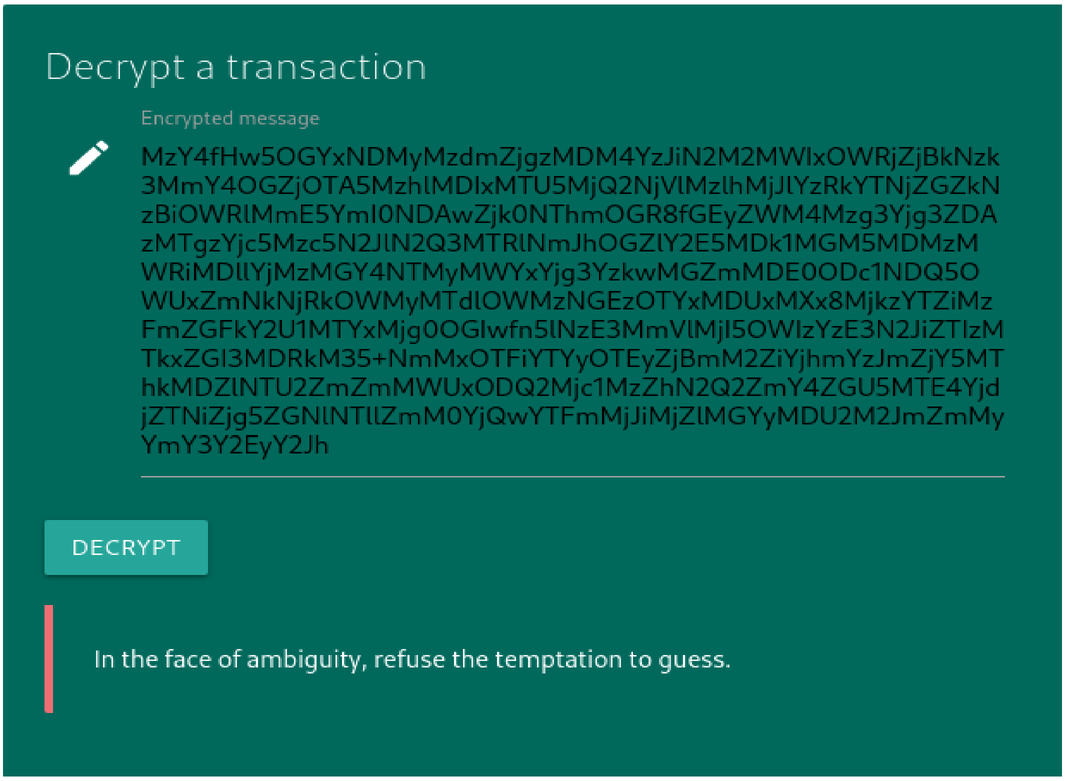
Encrypted message

NDA2fHw5N2U0OTViODdiYjFmYzQ3YTI5NjVlNjBkODlkNmM2ZThlZT
E3YmM4ZWRhNjA1MmFiMzUwNWNiOTI5MWQxNTQ2MGE3ODQyNj
ZmOWE3ZmQ2ODA3OWZlNjFmZTFlMzFmMzB8fGFlYzc1MzVlOTVm
MDM1YjA4MGM5NzhmMjMwNjU5MDMzM2RkMGY3MjA3MjEwODk1
Mjk2OTA0ZGRjYjNjNGFlZDM1ZjljM2Q1MDdkMTcwZTJjNzg1MzI3YzU
5NWEwMTNhOTI0NWJmNGJiZjM1MWQ2fHwwNTE2MWZjN2M4NT
NkOWYyNzk3NTAxMzR+fjEwYWVjMzI4YjBmMDllODJjMjhhNWQwZj
Q5NDgxNzY5fn5hMTNiMjY0Y2YzOWEzNWQxYWRlN2E1Y2U4MjNiM
DczMzg3ZGRhMDZmNzllNWVmMjA4NjNhZjFhYzJkNjQ2MDVmMDli
NWYzMzg4NzNhZTgxNDQyZGYxNzJlYmFmMGVjOTYxYmM2ZjEyMm
UwOTZhMg==

DECRYPT

Current round is 359. Please wait till 406

Figure 6.4: Decrypt a message before time



Decrypt a transaction

Encrypted message

MzY4fHw5OGYxNDMyMzdmZjgzMDM4YzJiN2M2MWIxOWRjZjBkNzk
3MmY4OGZjOTA5MzhlMDIxMTU5MjQ2NjVlMzlhMjJlYzRkYTNjZGZkN
zBiOWRlMmE5YmI0NDAwZjk0NThmOGR8fGEyZWM4Mzg3Yjg3ZDA
zMTgzYjc5Mzc5N2JlN2Q3MTRlNmJhOGZlY2E5MDk1MGM5MDMzM
WRiMDllYjMzMGY4NTMyMWYxYjg3YzkwMGZmMDE0ODc1NDQ5O
WUxZmNkNjRkOWMyMTdlOWMzNGEzOTYxMDUxMXx8MjkzYTZiMz
FmZGFkY2U1MTYxMjg0OGIwfn5lNzE3MmVlMjI5OWIzYzE3N2JiZTIzM
TkxZGI3MDRkM35+NmMxOTFiYTYyOTEyZjBmM2ZiYjhmYzJmZjY5MT
hkMDZlNTU2ZmZmMWUxODQ2Mjc1MzZhN2Q2ZmY4ZGU5MTE4Yjd
jZTNiZjg5ZGNlNTllZmM0YjQwYTFmMjJiMjZlMGYyMDU2M2JmZmMy
YmY3Y2EyY2Jh

DECRYPT

In the face of ambiguity, refuse the temptation to guess.

Figure 6.5: Decrypt a message after time

6.2 Integration with Filecoin

The second part of our implementation is to integrate timelock encryption into a blockchain technology, which can utilise the scheme for frontrunning protection. We chose Filecoin[6] as our platform since drand was already implemented inside the Filecoin ecosystem for their consensus, the availability of compatible elliptic curve primitives and the possibility of extending timelock encryption to support other functionalities of the platform, which are described in future work.

In this implementation, due to the complexity of the existing code base and the dependency to learn the Go programming language, the integration is still ongoing. Since the project is open-sourced, the goal is continue working on the project and finish the implementation. However, in this section, we provide a concrete road map of the work plan and the possible challenges that have been identified to affirm the interest to continue the implementation.

6.2.1 Architecture

Filecoin is a distributed storage network based on blockchain technology. Filecoin miners can elect to provide storage for the network and receive the Filecoin cryptocurrency (FIL) as a reward if they provide cryptographic proofs that prove that the agreed storage by the miner is indeed provided. Outside of this, FIL cryptocurrency can also be sent among users through transactions which are recorded on the blockchain. The consensus mechanism is known as Expected Consensus (EC), which elects a leader at random from a weighted set of miners based on storage provided. Hence, the consensus is powered by the proof of storage itself ie a miner's stake in the consensus protocol is proportional to the storage provided by him.

Filecoin's functionality is powered through the Filecoin virtual machine (Filecoin VM). The decentralized network generates a series of blocks and uses consensus to agree which chain is the correct one. Each block contains a series of state transitions called messages[7], and a checkpoint of the global state after the application of those messages. The global state contains a set of actors and is deterministically generated from the initial sate and the set of messages generated by the system.

6.2.2 Actors

A actor is essentially the equivalent of an Ethereum smart contract. Each actor has their own state, an associated address, a Filecoin balance, a set of methods designed to interact with the actor and a nonce which tracks the number of messages sent by the actor.

[6] `https://filecoin.io/`

[7] Transactions are messages within filecoin ecosystem as it initiates a state transition.

For an external user to interact with the actor, he must send a signed message to the filecoin network with the actor's address and a method reference. In turn, he must pay a fee to miner to include the message in the block. The address used to sign the message must also have sufficient filecoin to execute the message in the VM, which is proportional to the amount of computation required to process the message. There are 11 builtin system actor (not user deployable) present in the filecoin system.

Our goal is to build a new system actor part of the filecoin system called the *timelock actor*, which can receive an timelocked message from an user, decrypt it after the secret is revealed and invoke the actual actor part of the original message. The address of the user signing the message must have enough filecoin associated with him for the cost of decryption.

6.2.3 Roadmap

To achieve our goal, the integration is spread across three repositories, which all needs to be completed to have a working timelock encryption.

The first part is on the client side, where a prototype for users must be built to timelock their messages and send it to the filecoin network. Since we already have a PoC built in JavaScript, the `filecoin.js` JavaScript library[8] can be used to integrate the encryption functionality in the client library. Additionally, it must include the proper address and method reference for the *timelock actor* to have the timelocked message processed by the Filecoin network.

The second part of the work stems from designing the actor itself. Filecoin maintains all 11 system actors under the spec-actors repository (`https://anonymous.4open.science/r/specs-actors`), which will be the focus for this part. The new *timelock actor* will comprise of the following state and methods

- `State` - The state will be a hashmap of timelocked messages.
- `Method 1` - The first method will be the external interface of the actor. Users will send their messages to this method reference. The method will check for fee validity for decryption with the associated user address and add the transaction to the actor's state.
- `Method 2` - The second method is the crux of our logic and will be executed when the block is produced. The method interacts with the drand randomness beacon to get the signature and decrypts the message. After decryption, the actual actors are invoked for their respective messages.

The final part is to integrate the *timelock actor* in the filecoin infrastructure. This will be done in the lotus repository (`https://anonymous.4open.`

[8] `https://filecoin-shipyard.github.io/filecoin.js/`

`science/r/lotus`), which is the code for filecoin network, implemented in Go. On the lotus codebase, three distinct functionalities will be focused.

- The new actor must be registered as a system actor at the genesis of the filecoin instantiation, so that it will be part of the global state. This is required to dispatch the timelocked message to `Method 1`, when a user issues a message to the timelock actor's address. The implementation requires several abstractions to be added, such that timelock actor can be invoked from other actors to offer a different functionality.

- `Method 2` must be called at the end of an epoch before a block is produced to achieve frontrunning protection.

- Add a runtime interface to fetch the v2 signatures from the drand randomness beacon. This method will be used for decrypting the timelocked message.

6.2.4 Challenges

While working on the integration, three challenges have been identified that have an impact on the implementation.

- *Validity of messages* - While the timelock actor can validate if the associated address has enough filecoin for executing the decryption, the original message can be validated only at the end of the epoch. Hence, there must be a mechanism to validate the original message, such that it wont be discarded at the end. zkSNARKs can be used for proving that the original message costs a particular amount of filecoin and that the user has the required amount. The proof validation must be done by the timelock actor when it receives the message.

- *Round identification* - The client must have a mechanism to identify the right round of drand for appropriately timelocking the message. If the round is still in the future at the time of block production, the message must be discarded. However, this technique is counter-productive and a better mechanism is required for handling such messages.

- *Epoch Synchronization* - Filecoin and drand have different epoch as they are independent entities. Hence, the actual deployment must have a δ (difference in epochs) smaller than network propagation, so that a miner can't decrypt and frontrun the message when it enters the filecoin network.

Chapter 7

Analysis

In this chapter, we perform various analysis of our construction to capture the practicality and reliability of the protocol. The analysis is approached from two fronts (a) Performance and (b) Security.

7.1 Performance analysis

A cryptographic scheme could only be made practical if it can be built from previously existing cryptographic primitives and provides high performance in implementation. In this section, we explore the performance of the scheme both from theoretical and practical outlook.

7.1.1 Theoretical performance

Theoretical performance of the scheme stems from the collective number of costly computation that compose the scheme. Since our construction is based on elliptic curve cryptography, curve operations are predominantly computation intensive in comparison to their hash function and symmetric encryption counterparts.

Hence, we define the performance of the scheme using the number of curve operations required to compute the encryption and decryption. Also, HashToCurve is meant to run using a constant number of steps to prevent side channels attacks. Hence, in our analysis, we will consider HashToCurve as a single operation, instead of its multiple separate curve operations, which vary between implementations.

To encrypt a message, our scheme requires 2 scalar multiplications, 1 HashToCurve and 1 pairing operation. Similarly, for decryption, the scheme requires only 1 scalar multiplication and 1 pairing operation.

7.1.2 Practical performance

To obtain a sense of practical performance, we stress test our PoC to achieve appropriate benchmarks. However, we study the encryption and decryption operation as a whole and not by their individual curve operations. The *noble-*

bls12-381 library has already performed the benchmarks on standalone curve operations[1].

The benchmarks were run on system with AMD Ryzen 9 3900XT 12-Core Processor CPU and 32GB System Memory. The system also hosted a local version of drand for v2 signatures to minimize the additional execution time caused due to network delay.

On our first test, we ran the encryption and decryption functions for 1000 turns on randomly generated 3KB messages with a fixed round. The 3KB message size was chosen, as on average, that is the size of the smart contract on Ethereum. On average, the encryption took 62.4ms with a standard deviation of 10ms and the decryption took 32.9ms with a standard deviation of 4ms.

In our second test, we wanted to study the growth of the length of ciphertext as a function of the input message. Since computation onchain might be expensive for decryption, it is possible to add just the encrypted transaction to the block and provide zero knowledge proofs for validity of the transaction. The transaction can be decrypted at a later stage offchain, once the effects of front runners has been averted. But, the transaction indeed preserves its position in the blockchain.

In this approach, however, block space is of importance. The amount of block space available per round is limited in size and hence the encrypted transaction shouldn't occupy too much space. For our test, we run the encryption function for 1500 rounds with increasing byte size of the message. The result is plotted below as a graph (Fig. 7.1). On average, a 3KB message or transaction produces a 16KB encoded ciphertext, which is well within range.

7.2 Security Analysis

Another component of practicality stems from reliability, which could be studied by analysing of security of the construction. A similar approach to previous section is taken, exploring both the theoretical and practical aspects of security of our construction.

7.2.1 Theoretical Security

Since our construction is based on a composition of different protocols, the hardness problems which power the security of the scheme are mentioned at their respective sections.

The original IBE paper[20] is adaptive chosen ciphertext attack secure(IND-ID-CCA) under the Bilinear Diffie-Hellman assumption. However, the bilinear pairings defined in the paper are of the form $\hat{e} : G_1 \times G_1 \to G_2$. However, our construction uses the pairing of the form $\hat{e} : G_1 \times G_2 \to G_T$, which updates the security problem. However, the proof from IBE still holds

[1] `https://github.com/paulmillr/noble-bls12-381#speed`

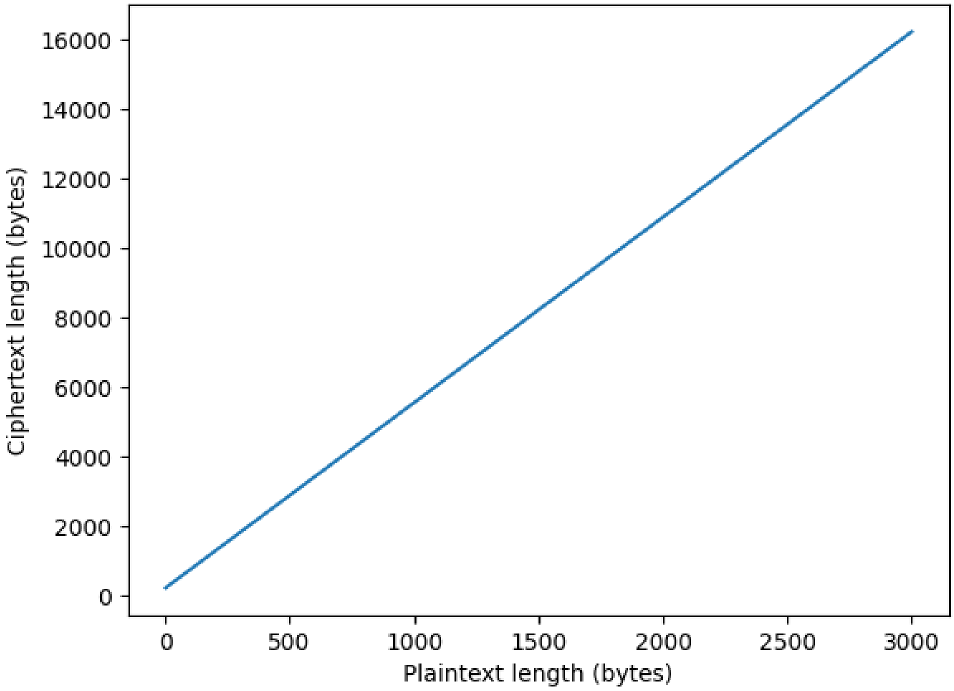


Figure 7.1: Growth of ciphertext length w.r.t plaintext size

for our construction to show its secure under adaptive chosen ciphertext attack (IND-ID-CCA).

Our construction is secure under the updated Computational Co-Bilinear Diffie-Hellman Problem [21] which is defined as follows,

> Let G_1, G_2 and G_T be three groups of prime order p. Let $\hat{e} : G_1 \times G_2 \rightarrow G_T$ be an admissible bilinear map and let P be a generator of G_1 and Q be a generator of G_2. The Computational Co-Bilinear Diffie-Hellman in $\langle G_1, G_2, G_T, \hat{e} \rangle$ is as follows : Given $\langle P, aP, bP \rangle \in G_1$ and $\langle Q, aQ, cQ \rangle \in G_2$ for some random elements $a, b, c \in Z_q^*$, compute $W = \hat{e}(P, Q)^{abc}$. An algorithm $\mathcal{A}$ has an advantage in solving the problem in $\langle G_1, G_2, G_T, \hat{e} \rangle$ if
>
> $$Pr[\mathcal{A}(P, aP, bP, Q, aQ, cQ) = \hat{e}(P, Q)^{abc}] \geq \epsilon$$
>
> where the probability is over the random choice of a, b, c in Z_q^*, the random choice of $P \in G_1$, $Q \in G_2$ and the random bits of $\mathcal{A}$.

An alternative proof approach for the security of the scheme would be to use *Universally composable security*[27], which guarantees secure composition of arbitrary protocols.

7.2.2 Practical Security

The practical security of our scheme come from the building blocks of our implementation. SHA256, AES256GCM and SHAKE256 are all well studied and tested protocols in the industry for their practical security. With our hardness guarantee from last section, we could argue that a proper implementation of the construction from the building blocks will be practical secure. For example, in our construction $\phi \xleftarrow{\$} \{0,1\}^l$ requires a random string to be generated for the size of the message. To achieve this, we use nodeJS's `randomBytes` API[2] from the crypto library which is secure pseudo random generator using true source of randomness like `/dev/urandom` for seeding.

However, implementations can often go wrong (ex. insecure randomness) and hence, a thorough security audit must be performed before practical adoption. Drand has performed a thorough analysis for their randomness beacon and provide a comprehensive list of attack vectors that threaten the security of their protocol[3].

[2] https://nodejs.org/api/crypto.html#crypto_crypto_randombytes_size_callback

[3] https://drand.love/docs/security-model/#randomness-generation

Chapter 8

Drawbacks

Timelock encryption offers a practical solution to fight the frontrunning crisis, while preserving the open nature of blockchain technology. However, the decentralized nature of the ecosystem, brings certain drawbacks to the scheme, outside the influence of timelock encryption

1. *Transaction censorship* - Since transactions are broadcasted over the p2p network, an attacker seeing an encrypted transaction in the network can censor its propagation by withholding it. Since outside the timelock encryption, the transaction still requires certain metadata to travel through the network, the attacker can enforce censorship rules through them (for ex, the IP address). Transaction censorship is an already existing problem in blockchain protocols and most propose a penalty approach as the viable solution.

2. *Blind Reordering* - A miner can choose to blindly reorder the encrypted transactions and propose the block. The blind reordering could inadvertently affect the users, which is outside of their control. A proposed solution is to enforce an sequencing order based on the properties of the ciphertext (for ex. lexicographically).

3. *Collusion* - There is no mechanism to prevent collusion between miners and devious users. This can be done by sharing ciphertexts and requesting a reordering to bench a profit. However, this is an extremely rare scenario in a decentralized setting with a low impact factor.

4. *Destabilization of Consensus* - It's worth noting here that timelock encryption, if applied to all transactions, removes MEV completely. In that scenario, miners could collude to reorder transactions and rewrite the blockchain history after the transactions are decrypted and the block is produced, to net a significant MEV. Essentially, this can cause destabilization of consensus and break the protocol.

Chapter 9

Future Work

In this chapter, we explore the possible avenues of future work leveraging the timelock encryption construction. A major focus of the future work, however, will be integrating the timelock encryption into the Filecoin ecosystem.

9.1 Integration with Ethereum

Ethereum is by far, the biggest decentralized platform for smart contracts with a market cap of $340 billion. Hence, it would be a great platform to implement the timelock encryption and study the scalability of the construction. However, there is no precompile currently available on the Ethereum Virtual Machine (EVM) to execute elliptic curve operations on BLS12-381 curve. The precompile has been proposed in EIP-2537[1] but is still under review.

However, there are EVM compatible platforms like Celo[2], which carry the functionality of Ethereum on a new blockchain protocol. Celo has the precompile EIP-2537 already accepeted under cip-0031[3] and can be used as an avenue to build the integration. Once EIP-2537 is made available, the implementation can then directly be tested on the mainnet.

The amount of computation carried out on EVM per block is limited and a wide scale adaption of the timelock encryption could slow down the mainnet and drive up the fees for decryption onchain. Hence, an alternative approach to decrypting the transaction would be using a layer 2 technology. Layer 2 technology[4] is proposed to scale the Ethereum performance, by offloading computation from the mainnet. An approach to layer 2 technology for timelock encryption would look as follows :

- Decryption of the transaction is carried outside layer 1

[1] `https://eips.ethereum.org/EIPS/eip-2537`

[2] `https://celo.org/`

[3] `https://github.com/celo-org/celo-proposals/blob/master/CIPs/cip-0031.md`

[4] `https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/`

- Proof of validity of transaction is on layer 1 (the smart contract inside costs a particular amount of gas).
- A rollup smart contract in layer 1 that can enforce correct decryption on layer 2 by using the transaction data on layer 1

The most common form of validity rollups are zk-Rollups, which uses zero knowledge proofs like zkSNARKs to validate the encrypted transaction. Optimistic rollups is another approach, which assumes that the transaction is valid and executes it off the mainnet and proposes the new state after execution to layer 1. In case someone dispute that the transaction is fraudulent, they can invoke a fraud challenge and the transaction will be executed outside layer 1 to answer the challenge. If the transaction is indeed found invalid, the challenger proposes the new state and gets the collateral used before as reward. The layer 2 is rolled back to the last non fraudulent state.

9.2 Proof of replication

Proof of replication[40] is a new paradigm in data storage, which enforce a storage provider to store a unique replica of the data through cryptographic proofs, even if the data is available from other sources. Our timelock encryption can essentially be used for proof of replication with an idea similar to the one proposed here[19]. In this construction, we will be using the v1 signatures, as a timelock of 1 epoch of drand is only required and is significant to the construction (Sec. 5.1.4). The core of the proof goes as follow : Consider a file f and a unique replica identifier id. The file f is broken down down into l-bit blocks $B_1, B_2..B_n$. The proof for a block B_i is $y_i = Enc(r+1, B_i \oplus H(id||i), pk)$, where r is the current round, H is a hash function and pk is the drand public key.

Once the drand produces the σ_r for round r, the verifier chooses a file block B_x at random and requests for y_x from the storage provider. The provider uses our timelock encryption scheme to encrypt the necessary file block and sends it to the verifier. Once the drand passes to round $r + 1$, the proof can be decrypted and if it is B_x, the proof is valid. The strength of the proof arises from two factors (a) Since we are using version 1 signature, the provider could not encrypt the block before σ_r is made available. (b) The timelock is only for 1 epoch. If the file block is large enough, he couldn't retrieve it from other sources and compute the encryption within that epoch. The validity of the proof holds only if the encryption is published before the v1 signature is published for the next round. The process can be continued at each epoch to provide a contiguous proof of replication.

Chapter 10

Conclusion

Timelock encryption offers a secure mechanism to mask transactions over the dark forest from threat actors interested in exploiting the information in the transaction for frontrunning attacks. To mitigate them, we proposed a practical timelock encryption scheme, building over threshold BLS signatures and identity based encryption. The feasibility of the scheme was realized through a proof of concept and an integration roadmap in the filecoin infrastructure. The analysis established the scalability of the protocol and offered a checkpoint for further fine tuning the implementation. Finally, we concluded the work with possible drawbacks of the scheme with suitable fixes and new applications of interest for timelock encryption, marking the future work of the research.

Appendix A

Anonymized Repositories

As required, the Github repositories of codebases relevant to the thesis have been anonymized using `https://anonymous.4open.science/`. The links provided will stay valid until 25/03/2022. After that, the links will redirect back to the original repository. A selection of four separate codebases have been anonymized :

- *timelock-encryption* - `https://anonymous.4open.science/r/timelock-encryption`
- *drand* - `https://anonymous.4open.science/r/drand`
- *spec-actors* - `https://anonymous.4open.science/r/specs-actors`
- *lotus* - `https://anonymous.4open.science/r/lotus`

Bibliography

[1] Introduction to smart contracts. https://ethereum.org/en/developers/docs/smart-contracts/. Accessed: 20-09-2021.

[2] Sushiswap. https://docs.sushi.com/products/amm-exchange. Accessed: 20-09-2021.

[3] Top cryptocurrency decentralized exchanges ranked. https://coinmarketcap.com/rankings/exchanges/dex/. Accessed: 20-09-2021.

[4] The dao: An analysis of the fallout - coindesk. https://www.coindesk.com/markets/2016/06/18/the-dao-an-analysis-of-the-fallout/, Jun 2016. Accessed: 20-09-2021.

[5] Alex Flory Samartino May 03. Anoma blog: Ferveo: A distributed key generation scheme for front running protection. https://anoma.network/blog/ferveo-a-distributed-key-generation-scheme-for-front-running-protection/. Accessed: 20-09-2021.

[6] Bitcoin ABC. Benefits of canonical transaction order. https://bitcoin-abc-org.medium.com/benefits-of-canonical-transaction-order-ec30ae62d955, Aug 2018. Accessed: 20-09-2021.

[7] Marcin Abram, David Galindo, Daniel Honerkamp, Jonathan Ward, and Jin-Mann Wong. Democratising blockchain: A minimal agency consensus model, 06 2020.

[8] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core. Technical report, Tech. rep., Uniswap, 2021.

[9] Guillermo Angeris, Alex Evans, and Tarun Chitra. A note on privacy in constant function market makers. *CoRR*, abs/2103.01193, 2021.

[10] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. A fair consensus protocol for transaction ordering. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 55–65, 2018.

[11] Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep*, 2016.

[12] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. Maximizing extractable value from automated market makers. *arXiv preprint arXiv:2106.01870*, 2021.

[13] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. A theory of automated market makers in defi. *CoRR*, abs/2102.11350, 2021.

[14] Carsten Baum, Bernardo David, and Tore Kasper Frederiksen. P2dex: privacy-preserving decentralized cryptocurrency exchange. In *International Conference on Applied Cryptography and Network Security*, pages 163–194. Springer, 2021.

[15] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.

[16] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Annual cryptology conference*, pages 90–108. Springer, 2013.

[17] Iddo Bentov, Ari Juels, Fan Zhang, Philip Daian, and Lorenz Breidenbach. Real-time cryptocurrency exchange using trusted hardware, May 23 2019. US Patent App. 16/198,223.

[18] Bertmiller, Vbuterin, Kladkogex, Wminshew, Yaeldo, Shymaa-Arafat, Pmcgoohan, Mightypenguin, Yoavw, lsankar4033, and et al. Mev-sgx: A sealed bid mev auction design. `https://ethresear.ch/t/mev-sgx-a-sealed-bid-mev-auction-design/9677`, May 2021. Accessed: 20-09-2021.

[19] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[20] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001.

[21] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *International conference on the theory and applications of cryptographic techniques*, pages 416–432. Springer, 2003.

[22] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, pages 514–532, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[23] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17(4):297–319, July 2004.

[24] Joppe W. Bos, J. Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. Elliptic curve cryptography in practice. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*, pages 157–175, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[25] Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1335–1352, 2018.

[26] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[27] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[28] Ignacio Cascudo and Bernardo David. Scrape: Scalable randomness attested by public entities. In *International Conference on Applied Cryptography and Network Security*, pages 537–556. Springer, 2017.

[29] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution. *CoRR*, abs/1804.05141, 2018.

[30] Cloud Security Alliance. `https://cloudsecurityalliance.org/blog/2020/10/26/blockchain-attacks-vulnerabilities-and-weaknesses/`. Accessed: 20-09-2021.

[31] ConsenSys. Smart contracts known attacks. `https://consensys.github.io/smart-contract-best-practices/known_attacks/`. Accessed: 20-09-2021.

[32] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges, 2019.

[33] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927, 2020.

[34] George Danezis, Eleftherios Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: A dag-based mempool and efficient BFT consensus. *CoRR*, abs/2105.11827, 2021.

[35] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[36] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, January 1987.

[37] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[38] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. *CoRR*, abs/1902.05164, 2019.

[39] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

[40] Ben Fisch. Poreps: Proofs of space on useful data.

[41] Ethereum Foundation. An update on integrating zcash on ethereum (zoe). `https://blog.ethereum.org/2017/01/19/update-integrating-zcash-ethereum/`. Accessed: 20-09-2021.

[42] Ethereum Foundation. zksnarks in a nutshell. `https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/`. Accessed: 20-09-2021.

[43] Jake Frankenfield. Proof of stake (PoS). `https://www.investopedia.com/terms/p/proof-stake-pos.asp`, May 2021. Accessed: 20-09-2021.

[44] Jens Groth. Non-interactive distributed key generation and key resharing. *IACR Cryptol. ePrint Arch.*, 2021:339, 2021.

[45] Ethan Heilman, Neha Narula, Garrett Tanzer, James Lovejoy, Michael Colavita, Madars Virza, and Tadge Dryja. Cryptanalysis of curl-p and other attacks on the iota cryptocurrency. *IACR Transactions on Symmetric Cryptology*, pages 367–391, 2020.

[46] Bin Hu, Zongyang Zhang, Jianwei Liu, Yizhong Liu, Jiayuan Yin, Rongxing Lu, and Xiaodong Lin. A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems. *Patterns*, 2:100179, 02 2021.

[47] Harry A Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau, and Arvind Narayanan. An empirical study of namecoin and lessons for decentralized namespace design. In *WEIS*. Citeseer, 2015.

[48] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[49] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. Cryptology ePrint Archive, Report 2020/269, 2020. `https://ia.cr/2020/269`.

[50] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 451–480, Cham, 2020. Springer International Publishing.

[51] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[52] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[53] MIT Media lab. Bitcoin Gold (BTG) was 51% attacked. `https://gist.github.com/metalicjames/71321570a105940529e709651d0a9765`. Accessed: 20-09-2021.

[54] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*, pages 179–196. 2019.

[55] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[56] Michael Lewis. *Flash boys: a Wall Street revolt*. WW Norton & Company, 2014.

[57] Alfred J Menezes, Tatsuaki Okamoto, and Scott A Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *iEEE Transactions on information Theory*, 39(5):1639–1646, 1993.

[58] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[59] Shutter Network. Introducing shutter network - combating front running and malicious mev using threshold cryptography. `https://shutter.ghost.io/introducing-shutter-network-combating-frontrunning-and-malicious-mev-using-threshold-cryptography/`, Sep 2021. Accessed: 20-09-2021.

[60] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[61] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[62] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? *CoRR*, abs/2101.05511, 2021.

[63] Michael O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267, 1983.

[64] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[65] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 444–460. Ieee, 2017.

[66] Vbuterin, Kladkogex, MicahZoltu, Haydenadams, a4nkit, k06a, Imkharn, Mulitwfn, Napcalx, Shymaa-Arafat, and et al. Improving front running resistance of x*y=k market makers. `https://ethresear.ch/t/improving-front-running-resistance-of-x-y-k-market-makers/1281`, Mar 2018. Accessed: 20-09-2021.

[67] Vbuterin, Nazariyv, Shymaa-Arafat, Jannikluhn, Yoavw, JustinDrake, and Lekssays. Proposer/block builder separation-friendly fee market designs. `https://ethresear.ch/t/proposer-block-builder-separation-friendly-fee-market-designs/9725`, Jun 2021. Accessed: 20-09-2021.

[68] Will Warren. Front-running, griefing and the perils of virtual settlement (part 1). `https://blog.0xproject.com/front-running-griefing-and-the-perils-of-virtual-settlement-part-1-8554ab283e97`, Jan 2019. Accessed: 20-09-2021.

[69] WeUseCoins. Confidential transactions - gregory maxwell. `https://www.weusecoins.com/confidential-transactions/`. Accessed: 20-09-2021.

[70] Zachary J Williamson. The aztec protocol. *URL: https://github.com/AztecProtocol/AZTEC*, 2018.

[71] Shijie Zhang and Jong-Hyouk Lee. Double-spending with a sybil attack in the bitcoin decentralized network. *IEEE Transactions on Industrial Informatics*, 15(10):5715–5722, 2019.

[72] Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. On the just-in-time discovery of profit-generating transactions in defi protocols. *CoRR*, abs/2103.02228, 2021.

[73] Liyi Zhou, Kaihua Qin, and Arthur Gervais. A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. *arXiv preprint arXiv:2106.07371*, 2021.

[74] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V. Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. *CoRR*, abs/2009.14021, 2020.