DEDIS

Decentralized Distributed
Systems Laboratory

École Polytechnique Fédérale de Lausanne

Optimizing Commit-and-reveal for Smart Contracts

by Julie Bettens

Master Thesis

Approved by the Examining Committee:

Prof. Dr. Bryan Ford
Thesis Advisor

Yolan Romailler
External Expert

EPFL IC IINFCOM DEDIS
BC 210 (Bâtiment BC)
Station 14
CH-1015 Lausanne

21st June 2024

A wise person is mightier than a strong one;
a knowledgeable person than a powerful one.
— Proverbs 24:5

Dedicated to the wonderful people at polygl0ts, without whom I would not be who I am today.

Acknowledgments

I would like to thank Shufan Wang for prior work on the codebase, Mahsa Bastankhah for her work on the SMC code, and the Dela, Kyber, and go-ethereum maintainers. I would also like to thank my parents for supporting me.

This project would not have been possible without the work of Prof. Bryan Ford and the DEDIS team, as well as the IC faculty, administration, student advocacy associations, and my fellow students. I also thank the International Association for Cryptologic Research for organizing Eurocrypt 2024 and maintaining the Cryptology ePrint Archive.

Finally, I wish to thank Prof. Payer for the LaTeX template.

Lausanne, 21st June 2024 Julie Bettens

Abstract

Delayed execution is an alternative architecture for blockchains that allow users to encrypt the content of their transactions until they are executed. We design and evaluate Dawn: an implementation of delayed execution on Ethereum. We model the applications that benefit from Dawn, explore implementations based on threshold cryptography and sequential computation, and evaluate the feasibility of such an approach on an existing blockchain. Beyond front running prevention and censorship resistance, we show how this design change improves over pure smart contract layer solutions when it comes to sealed-bid auctions, lotteries, and voting.

Contents

Chapter 1

Introduction

In decentralized applications, most information produced by users is eventually public. However, timing of the release of information matters a lot. This is evident not only in instances where front running is a concern, but also in setting such as auctions where multiple parties reveal information. As a result, various commit-and-reveal protocols are often used: for example in the Ethereum Name Service, a popular name service application, registering a name is a two-step process where users first publish a commitment to their desired name and later open this commitment. [ENS22] Zhang et al. point out that, while the same mechanism can be extended to sealed-bid auctions, it comes with substantial drawbacks. [Zha+24]

An approach purely based on two-step commit-and-reveal inherently expects participants to send a reveal message, which raises the question of how to handle failure to reveal. There is no general solution: one cannot simply wait forever for the last user to reveal, so protocols often disincentivize failure to reveal via collateral requirements. Unfortunately, this is subject to economic attacks if the incentives can exceed the collateral, as well as denial-of-service attacks on the revealing participant. As a result, a protocol that can guarantee the reveal occurs in all cases would significantly improve the situation.

A subset of the problem has been addressed by systems such as F3B[Qu23] and Fairblock[MGZ22], in that they aim to protect a single transaction at a time from front running. Shutter Network[K22] in its current form provides temporary encryption of off-chain votes. Building on their research, we aim to explore a more generalized approach to serving commit-and-reveal applications on the blockchain.

Zhang et al. introduce the delayed execution architecture, in which transactions only need to be executed after the block attesting to their order has been finalized. The intended effect is that as long as transactions do not need to be executed, they can be temporarily encrypted. As a result, the information contained within the transactions isn't revealed too early, enabling a range of use cases such as sealed-bid auctions, lotteries, on-chain voting. This is achieved without requiring users to lock collateral, and while removing the incentive for denial-of-service attacks.

In this work, we construct a prototype of a delayed encryption blockchain based on Ethereum. We aim to show that commit-and-reveal with delayed execution is more versatile and simpler to use than hash-based commit-and-reveal, and just as efficient.

We will measure the costs and performance impact and find that Dawn is viable in a realistic setting. However, we find that the added latency adversely impacts feedback loops both inside the protocol and outside,

We will generalize front running protection to obtain a model that applies to a wider class of use cases. We will also design a data structure that allows storing past transactions in plaintext and does not require an additional fee payment mechanism.

Chapter 2

Background

2.1 Blockchain Networks

A *decentralized system* is an open computer network that allows any computer in the world to join and serve any role. In principle, no participant is a single point of failure or has special authority. Decentralized systems tolerate mutually distrusting participants, and coordinated malicious nodes up to a threshold. Decentralized systems aim for a high degree of censorship resistance and trust minimization. Bitorrentis an example of such a system.

For our purposes, a *blockchain network* is a decentralized system that is able to realize total order broadcast, i.e. arbitrary messages submitted to the network in a specific way will be included in a canonical sequence that any node in the network can learn. Given this primitive, one can create a shared ledger by specifying how people following the ledger should interpret broadcast messages as asset transfers. In practice, the ledger is often integrated with the total order broadcast functionality to incentivize it. Transfers are generally grouped into *blocks* by the total order broadcast mechanism, and those blocks contain the hash of the previous block to indicate the total order, forming a block chain. We call the node that created the block its *block producer* and the nodes actively participating in the ordering process *Consensus Group (CG).*

The earliest instance of a blockchain is Bitcoin[Nak08]. It aims to support transfers of its own native asset, bitcoins, to implement a digital cash system. To extend the functionality of blockchains, networks such as Ethereum[But14] turn the ledger into a fully fledged transactional database. Inside such a database, programs known as *smart contracts* can be placed by any participant so that other participants can interact with them. This is the basis for many *decentralized applications* which can be financial products, name systems, computational services, communication and coordination infrastructure, etc.

2.2 Auctions

In [Vic61, §2], Vickrey studies the effects of two auction mechanisms: the *progressive auction*, where bidders are allowed to announce new bids until the auction ends, and the *Dutch auction*, where the auctioneer announces decreasing prices until a single bidder announces that they accept it. Auctions force bidders to estimate the maximum price that every other bidder is willing to pay to outbid them by the smallest amount possible. In progressive auctions, bidders' maximum prices are gradually discovered as they bid, whereas in Dutch auctions, other bidders only have one chance to guess it correctly — barring repeat auctions.

Further, in [Vic61, §2], Vickrey introduces sealed bid auctions. He shows that, if the best bid executes at the price that it specifies, the procedure is equivalent to a Dutch auction. If instead, the winning bid executes at the price of the second best bid, the procedure is equivalent to a progressive auction. These are known as *first price* and *second price* sealed-bid auctions. This gives us a basis to examine existing decentralized finance applications and suggest how to take advantage of the sealed bids primitive while maintaining functionality.

Frankencoin is an application that aims to create a decentralized stable currency backed by on-chain collateral, pegged to the Swiss Franc. It allows user to create pegged coins in exchange for collateral, and uses Dutch auctions to liquidate insolvent users. Since Frankencoin does not rely on price feeds, the auctions require two phases: one where the collateral can be bought at par, thereby showing that the entity triggering the liquidation was at fault, and one where price decreases. This mechanism is sensitive to price fluctuations. We suggest that a second-price sealed-bid auction could reduce this risk. [MM22]

When a Dutch auction is conducted on a blockchain, the application should expect to sell at a discount due to the fact that discrete steps in price are offered at every block. Setting the auction parameters optimally is non-trivial. [MR24] When selling off volatile collateral, price movements during delays could adversely impact either the winning bidder or the application. In the case of a progressive auction, bidders may also repeatedly outbid each other, which results in more transaction fees. Thus, running the equivalent type of sealed-bid auction can be advantageous.

2.3 Maximum Extractable Value

On blockchain networks, users who are not able to produce a block themselves must transmit their transactions to a node that can in order to see them included. In the simpler blockchain designs, this is done by broadcasting the transaction widely so that all potential block producers will receive it. More advanced architectures involve sending the information only to a specific block producer or a subset, sometimes introducing a trust relationship between the user and the block producers.

Since at least one actor, the eventual block producer, can see the block's transactions before it is published, it is possible to simulate the effects of a particular ordering of transactions. As a result, block producers are incentivized to look for ways to reorder transactions, remove them, and/or add their own so as to profit. This phenomenon known as Maximum Extractable Value (MEV) has led to the development of a sophisticated block building ecosystem where specialized actors cooperate with block producers to find opportunities and share the profits.

We should note that while some types of MEV harm unsophisticated users, some others are beneficial. For example, if an on-chain exchange offers a price at odds with the broader market at any point in the block, the block producer is in a position to arbitrage it and give users a more representative price. This makes the market more efficient. Conversely, *front running*, which is a strategy that involves learning about a future trade and executing a trade ahead of it to benefit from the price impact, can occur and harms the target financially. [Dai+19]

To protect themselves against front running, some users use centralized services such as Flashbots Protect[1] and MEV Blocker[2], that share their transaction with trusted actors tasked with ensuring that it makes it into a block without being front run. Besides its reliance on trusted third parties, this solution is known to sometimes fail if the block is displaced by a competing block. At this point, the transaction has been revealed publicly and can be included in a new block — along with a front run. [KB21]

2.4 Commit and Reveal

Some decentralized applications employ a pattern known as commit-and-reveal to defeat front running. Namely, they require a user to commit: post a hash of their chosen value concatenated with a random value in a first transaction, and then reveal: show the preimage of the hash once the first transaction has been included. The Ethereum Name Service allows users to exclusively own human-readable names and associate them with less convenient identifiers such as public keys. When buying a domain name in ENS, two-step commit-and-reveal is used so that a malicious actor cannot take the name from the user that wants it and try to sell it back at a higher price. [ENS22] While it achieves the security goal, this pattern requires two transactions, which costs money and degrades user experience. ENS also sold names using a sealed-bid second price auction using commit-and-reveal from 2017 to 2019. [Xia+21, §3.1]

In [Zhu22] a construction for an on-chain sealed-bid auctions is proposed and implemented. It requires bidders to commit to their bid and post collateral as high as the maximum expected bid, and suffer losing their collateral if they fail to reveal.

[1]`https://protect.flashbots.net/`

[2]`https://mevblocker.io/`

With RANDAO[ran17, §III.1], randao.org explore a protocol using a blockchain to allow many mutually distrusting parties to generate an unbiased and unpredictable random number. Parties first generate their own random number. They then commit some amount of cryptocurrency as collateral and publish a commitment to their random number. Then, in the next phase, they reveal their number to reclaim their collateral. The final random number is a deterministic combination of all of the revealed random numbers. As a result, unless all participants collude, no one can bias or predict the final output. A clear drawback of this protocol is the necessity for every contributor to put up collateral and submit two transactions. Worse, the last contributor to reveal can compute the random value and decide to forfeit their collateral if they can profit from altering the final output.

Another application of commit-and-reveal schemes is shielded voting in the context of on-chain organizations. This allows voters to keep their choice confidential until the end of the voting period. Importantly, it is not by itself a secret-ballot voting system since votes are revealed and linked to their authors at the end of the process. Nonetheless, K. argues that hiding votes while voting is still open avoids influencing undecided voters, and makes some attacks more costly. [K22] We note that participants are not prevented from proving how they voted. Still, we will study shielded voting as one of our motivating applications.

To the best of our knowledge, simple commit-and-reveal schemes are not used in production outside of ENS. RANDAO planned to go in production in 2018 but did not follow through.[3] We could not find decentralized exchanges based on this method, likely due to its clunkiness and the problems with witholding reveals. Overall, these types of applications seem to be at an impasse because of the impracticality of simple commit-and-reveal schemes.

2.5 Delayed Execution

In [Zha+24], Zhang et al. introduce the concept of *delayed execution*, which re-architectures the blockchain system in such a way that executing transactions is not necessary at any point before they are included in a canonical block. In turn, simulation of pending transactions can be prevented by temporarily encrypting them. Importantly, they specify that all transactions and not just encrypted transactions are subject to delayed execution, otherwise unprotected transactions may be used to front run. The goal is to enable sealed-bid auctions without collateral requirements. We will show that it extends naturally to our other motivating applications: on-chain voting, randomness generation and frontrunning protection.

Zhang et al. argue that, since the time-to-finality of the blockchain is interpreted by end users that receive payments as the latency of the system, introducing delayed execution has no impact on latency. We will argue that, while this is true in this case, many feedback loops are still disrupted.

[3]Ethereum itself uses a mechanism derived from randao where BLS signatures are used as random numbers. [Edg23, §2.9.2]

2.6 Cryptographic Primitives

We will now review the cryptographic primitives that we will use in Dawn.

A *commitment scheme* is akin to using sealed and opaque envelopes: it consists of the routine $\mathsf{Commit}(\mathbf{pp}, m, r)$ which outputs a commitment c given a message m and randomness r. The desired security properties are *hiding*, meaning that c does not reveal any information about m, and *binding*, meaning that it is infeasible to find distinct m_0, r_0, m_1, r_1 s.t. $\mathsf{Commit}(m_0, r_0) = \mathsf{Commit}(m_1, r_1)$. This formal treatment is consistent with the use of commitment schemes that we pointed out in decentralized applications: in ENS, users use a commitment to “lock in” their intent to register a name without revealing it by publishing c. Later, they prove that they committed to this name by revealing the randomness.

We will use *Identity-based Encryption (IBE)*: a type of asymmetric encryption scheme where decryption keys can be derived for arbitrary strings which we call *identity labels*. An IBE scheme consists of four routines: $\mathsf{Setup}(1^\lambda) \to (\mathbf{mpk}, \mathbf{msk})$ which sets up the system with a *master public key*, known to all participants, and its secret counterpart, known only by a trusted component; $\mathsf{Extract}(\mathbf{msk}, \mathrm{ID}) \to d$ which returns a decryption key matching the identity label $\mathrm{ID} \in \{0,1\}^*$; $\mathsf{Encrypt}(\mathbf{mpk}, \mathrm{ID}, m) \to c$ which encrypts a message so that it can be decrypted by the identity ID, and $\mathsf{Decrypt}(\mathbf{mpk}, d, c) \to m$ which decrypts a ciphertext using a decryption key. For our purposes, an IBE scheme is secure if the decryption key for an identity label is both necessary and sufficient to decrypt messages encrypted towards that label. We refer to [BF01] for the detailed security definition.

A *threshold network* is a group of nodes collectively possessing a secret key. A threshold parameter t is set, such that any t out of n nodes can reconstruct the secret, but no group of $t-1$ can. The network can thus tolerate up to $t-1$ malicious nodes and up to $n-t$ unavailable nodes.

In [BBH06], Boneh, Boyen, and Halevi describe *Threshold IBE (TIBE)* schemes: IBE schemes where the trusted component can be operated by a threshold network. They also describe Threshold Public Key Encryption (TPKE): asymmetric encryption schemes where the decryption key is custodied by a threshold of nodes.

A *Distributed Key Generation (DKG)* protocol allows a threshold network to create a shared secret key without any single node ever knowing the full key, and without any group of nodes smaller than the threshold being able to reconstruct it or use it. It can be used to generate a shared master key for use in TIBE or TPKE.

Per [Bon+18], a Verifiable Delay Function (VDF) consists of three routines: $\mathsf{Setup}(1^\lambda, t)$, which, given a security parameter λ and a delay parameter t outputs the public parameters $\mathbf{pp}$, defining a group $\mathbb{G}$; $\mathsf{Eval}(\mathbf{pp}, x)$, which evaluates the function on $x \in \mathbb{G}$ and outputs $y \in \mathbb{G}$ and a witness π, and $\mathsf{Verify}(\mathbf{pp}, x, y, \pi)$, which checks if y is the unique result of evaluating the function on x. It must have the following properties:

sequential Computing $(y, \pi) \leftarrow \mathsf{Eval}(\mathbf{pp}, x)$ should be possible in t sequential steps. y should be indistinguishable from random, even with parallelization.

efficiently verifiable Computing $\mathsf{Verify}(\mathbf{pp}, x, y, \pi)$ should be quick, ideally poly-logarithmic in t.

unique for all x, it is not possible to find y, π such that $\mathsf{Verify}(\mathbf{pp}, x, y, \pi) = \mathsf{yes}$ but $y \neq \mathsf{Eval}(\mathbf{pp}, x)$. (except with negligible probability and assuming a polynomially bounded adversary)

Wesolowski[Wes18] modifies Setup() to also return a trapdoor parameter. Knowing the trapdoor allows one to “cheat” and evaluate the function much more quickly. This is also at the heart of how Wesolowski constructs verifiability.

Timed-release cryptography is the problem of encrypting a message such that it can be only be decrypted at a certain date. There exist two approaches:

1. In [May93], May suggests using multiple escrow agents to custody shares of a decryption key and release them until some time passes or a specific event occurs. We will call this approach *Secret Management Committee (SMC).*
2. In [RSW96], Rivest, Shamir, and Wagner suggest using a sequential computation that can be performed to reveal the decryption key. The number of computational steps lower bounds the amount of time that will pass before the key is revealed. We will call this approach *time-lock puzzles.*

An important advantage of the SMC approach is that it supports richer decryption policies than just the passage of time. The decryption policy can be “Decrypt on the 21st of June 2024”, but in can also be “Decrypt whenever a human sets foot on Mars for the first time” for example.[May93]

May originally envisioned SMC nodes in a manner similar to anonymizing remailers, where each message would be submitted with its decryption policy attached. While this is a sound approach, it is more scalable to make use of TIBE. In this way, the decryption policy can be put in the identity label, and SMC nodes do not need to handle the ciphertexts at all. In addition, only a threshold of nodes needs to be trusted.

Zero knowledge (ZK) proof systems enable performing computation on confidential data without revealing any part of that data but with the guarantee that the computation was performed honestly. They can be used for confidentiality, but also for performance reasons, because verifying the computation can be cheaper than performing it over again. [Ben+18]

2.7 Ethereum Specifics

We will now introduce some parts of the Ethereum architecture that are important to design and evaluate our system.

The Ethereum execution environment, the *Ethereum Virtual Machine (EVM)*, uses a single, unified global state structure where program code and data are stored. Transactions are allowed to operate that state one after the other. The state also keeps balances in a native currency unit, *Ether* which is used to pay for transactions. Transactions are always initiated through an ECDSA keypair, which must have a balance of Ether.

A sophisticated metering system determines the fee for a transaction: each byte of data, as well as each instruction executed during the transaction has a cost in abstract units of *gas*. This allows enforcing upper bounds on the amount of resources that nodes reading the blockchain have to use and thus ensuring that they operate reliably. Transactions are required to specify a gas limit, which is the maximum amount of gas they expect to consume. A transaction can only be included in a block if the sender has enough Ether to cover this gas given the current gas price. Unused gas is then refunded.

Since the London update in 2021[Eth24b], gas pricing on the main Ethereum network is influenced by an automatic feedback loop, whereby a base fee value is adjusted up or down based on the gas usage in the previous block. The value is denominated in Ether per unit of gas. If transactions bid a higher base fee, the difference is returned, which saves users from engaging in an inefficient auction for gas. The base fee is burned, and an additional part of the fee is used to pay block producers for inclusion. [EIP-1559]

For our purposes, Ethereum transactions consist of the following fields. We leave aside some fields that are not relevant to the discussion.

`nonce` *number used once*, a sequential counter for transactions originating from the same sender;

`max_priority_fee_per_gas` the fee to give to the block producer upon inclusion;

`max_fee_per_gas` the maximum base fee to pay for this transaction;

`gas` the maximum units of gas to reserve for this transaction;

`to` the address of the account the transaction is directed towards;

`value` an amount of native currency to send to the receiver;

`data` arbitrary bytes to send along with the transaction: if the receiver is a smart contract, it will be able to access this data and react to it with arbitrary logic;

`v,r,s` an ECDSA signature that also allows recovering the sender's address, which is why an explicit *sender* field is not necessary.

As of now, the main Ethereum network can produce a block every 12 seconds. Blocks are allowed to consume up to 30 million gas units, but the fee mechanism targets 15 million. This corresponds to 2.5 and 1.25 million gas per second, respectively. [Eth24a]

The consensus protocol Gasper[But+20], which Ethereum uses, is able to guarantee that a block is canonical after two checkpoints are included in it or in subsequent blocks. On Ethereum, checkpoints normally occur every 384 seconds, which means under normal conditions, a block is known to be canonical after 768 seconds at most. The CG can also keep extending the chain even if finality is not being reached.

Chapter 3

Design

No one can hide as the sunrise illuminates all land at once. Dawn opens all commitments at once. Dawn is our concept and prototype for an EVM blockchain with delayed execution.

3.1 Requirements

We formalize the requirements of Dawn as follows: Dawn allows n mutually distrusting parties to compute a function of the form $f(e_1, e_2, \ldots, e_n, \mathrm{st}) \rightarrow \mathrm{st}'$ where e are shielded inputs chosen by the parties, st is the application state and st′ is the updated application state. This allows us to specify our security properties: *fairness*: no participant can alter their input based on the input of another participant, and *privacy*: no one can alter the state based on any input. It is not a design goal to keep the inputs confidential after f is computed.

Our model covers all our desired use cases: in the auction case, the inputs are the bids and f will take the highest bid an settle the auction. In the RANDAO case, the inputs are random contributions and f mixes them together. In the shielded voting case, the inputs are votes and f computes the tally. In the front running protection case, there is one input, and f executes a trade based on the input.

Indeed, our model emerged as a generalization of front running protection, which can itself be considered a generalization of standard blockchain transactions: the Ethereum Yellow Paper[Woo24] which attempts to formalize Ethereum, describes the action of a transaction as $\sigma_{t+1} = \Upsilon(\sigma_t, T)$, meaning that a single participant gets to compute the state transition function Υ to produce the next Ethereum state, with the current state and the transaction T of their choice as input. This is sufficient to implement many applications directly, but there is no guarantee that σ_t cannot be altered based on the information disclosed in T. As previously discussed, many solutions attempt to provide that guarantee. However, we provide a stronger guarantee by extending the model to multiple parties.

In the absence of delayed execution, the security properties are typically implemented as a protocol with three phases: during the commit phase, a smart contract collects commitments from all parties. During the reveal phase, parties open their commitments. In the execution phase, the contract can act upon the revealed values. Additionally, the case where a participant does not open their commitment during the allotted time requires a design decision. It is also important to ensure the state is locked during the reveal phase. Note that by instantiating this protocol for random contributions and in such a way that parties who do not reveal their randomness forfeit their collateral, we obtain the original RANDAO protocol.

By contrast, under delayed execution, the functionality can be achieved in two phases: during the contribution phase, each participant sends a protected transaction containing their input. During the execution phase, the contract can compute f and update the state. This is safe assuming that the contribution phase duration is shorter than the delay, and that all transactions are subject to delayed execution so that the state is protected. As a result, we sidestep the problem of unopened commitments, and require half the amount of transactions in the typical case.

3.2 Timed Release

Timed release cryptography, introduced in Section 2.6, allows us to limit the encryption of transactions in time.

The first approach that we will use is based on TIBE: the sender provides the block number of an upcoming block, and the IBE label is set to that number. When the target block number is about to be reached, the SMC will release the decryption key. We will select a *verifiable* TIBE scheme such that the decryption key can be verified to be the correct decryption key for the label efficiently, i.e. in a constant number of steps.

The second approach is a TPKE approach: we take the first approach, and replace the label with the concatenation of the sender's address and the transaction nonce. In turn, the SMC must release decryption keys for transactions when they appear in a finalized shadow block. We inherit the verifiability from the first approach.

The third approach is based instead on time-lock puzzles. We require the sender to generate a puzzle of a pre-set difficulty such that it can be solved in roughly the time it takes to finalize a block, and we require the next block producer to compute it. We use a VDF construction to provide verifiability.

To abstract and formalize those approaches, we say that the timed release component implements a protocol with four routines: $\mathsf{Setup}(1^\lambda) \to (\mathbf{pp}, \mathbf{sp})$ which generates the public parameters and secret parameters — really a DKG procedure in the first two approaches and no operation in the VDF case; $\mathsf{Share}(\mathbf{pp}, a, n, n_b) \to (k, e)$ used by the transactor to generate an encryption key k along with its encapsulation e needed for timed release to decrypt it, given their account address a, the nonce n, and the target block number n_b; $\mathsf{Reveal}(\mathbf{sp}, a, n, n_b, e) \to r$, which allows the SMC (or anybody in the VDF case) to decapsulate an encryption key, and $\mathsf{Recover}(\mathbf{pp}, a, n, n_b, e, r) \to k$, which allows anybody to recover the same encryption key, or find out that the r is incorrect.

The important properties we should have in this formalism are *key agreement*: given a secret from Share, running Reveal and Recover on the encapsulation should yield the same secret; *robustness*: Recover should fail for invalid r arguments, and *context binding*: Recover should either fail or output an unrelated key if a or n are changed.

3.3 Symmetric Encryption

We should now explain how transactions are temporarily protected. Importantly, the usual definition of encryption does not provide the *binding* property of commitments. As a result, taking an existing ciphertext and using a different key will often decrypt fine, albeit likely to a garbage plaintext. More importantly, it is often possible to "maul" a ciphertext and obtain a valid ciphertext which corresponds to a different plaintext. In [Zha+24, §2.1], Zhang et al. describe such an attack where, in the case of an auction, a user looks at the encrypted bid of another user and constructs their own bid by blindly adding 1 unit to the original bid. This is something that we want to prevent.

Still, our encryption needs to be binding even in the presence of a malicious sender. Key-commitment [Gue20] really captures this property, specifying that it should be unfeasible to find a ciphertext that can be validly decrypted under two different keys. Len, Grubbs, and Ristenpart provide concrete examples of such attacks against AES-GCM and ChaCha20-Poly1305 in [LGR20]. Thus, we should use a symmetric primitive that satisfies Committing Authenticated Encryption (CAE).

A higher-level attack could also occur where a participant simply copies a commitment from another and submits it unmodified as their own transaction. In the randomness generation case, assuming XOR is used for mixing, this could allow canceling out another participant's contribution. More generally, this violates the fairness property because a participant has submitted a contribution identical to another's without it having been disclosed. To solve this, we should enforce *context binding* in the higher-level protocol, making it possible to check that a ciphertext-commitment was intended in a given transaction.

Finally, we require *verifiability* of the decryption process: when given the decrypted form of the transaction, a node should be able to verify that there exists an encrypted transaction that decrypts to it. This is to ensure that when a block producer introduces decrypted transactions in the block, we can quickly verify that they are correct, just like we can verify that all transactions are signed correctly.

Overall, we combine a CAE primitive with a timed release primitive to obtain the end-to-end scheme. We will benchmark and evaluate multiple CAE primitives.

Chapter 4

Implementation

Our prototype of Dawn is based on go-ethereum, an implementation of Ethereum, and on prior work by Wang, who implemented front running protection for Ethereum. [Wan23] It is architectured as a potential update to Ethereum, named the Lausanne Hard Fork, such that an already existing chain can switch to it at a given block. We are thus able to add new transaction types with new functionality, and change the execution environment or any other aspect of the system. We should note that our prototype uses Clique as a consensus protocol, which is not used in production by Ethereum. [EIP-225]

Before the Lausanne Hard Fork, the block producer of block n was free to fill block n with any sequence of valid transactions, so long as the gas consumed did not exceed the block limit. The Hard Fork removes this ability from the block producer. Instead, block n must include exactly the transactions from a new structure called the *shadow block*, which was included in block $n - d$ by another block producer. d is the execution delay parameter, which should be set so that a block which has d-length chain leading to it assured to be in the canonical chain, modulo security assumptions.

Unlike the normal block body, the shadow block can contain encrypted transactions. As soon as it does, it is no longer possible to accurately predict the future state of the blockchain because those transactions cannot be simulated. As a result, we specify that the cumulative gas reserved by transactions in the shadow block must not exceed the block gas limit. This value upper bounds the eventual gas consumption of the block, which cannot be computed in advance, but the difference can be arbitrarily big.

To ensure that the transactions in the shadow block can actually be included in the block, any sequence of transactions from the same address in shadow blocks should have consecutive nonce, and the first such transaction should match the nonce in the blockchain state at the same block. One way to implement this constraint is to maintain a “shadow nonce” for each active address which can be ahead of the regular nonce. The real nonce will be incremented as transactions are executed, and transactions are stored in regular blocks, so it is not necessary to retain shadow blocks indefinitely. Ether balance requirements must also be checked.

The priority fee for the transaction goes to the proposer of block $n - d$. This incentivizes them to fill the shadow block. However, they should receive a fee proportional to the amount of gas reserved instead of consumed. Otherwise, this allows anyone with Ether to submit an encrypted transaction with a very high gas limit that does nothing when decrypted, thereby giving much less than advertised to the proposer, unexpectedly.

The base fee is more interesting: since it is either burned or refunded, the only incentive is with the sender of the transaction. We believe that the mechanism doesn't need to be changed: we can refund both the unused gas and the difference between the bid and the base fee. However, the base fee value can change over d blocks, which means that some committed transactions may no longer be executable. We chose to disable the base fee for the sake of the prototype, but a redesign of the feature should be the subject of future research.

4.1 Data Structures

To represent encryptable transactions, we introduce three new types of transactions to the Ethereum execution environment. We have described existing transactions in Section 2.7. Compared to those, *encrypted transactions* lack `to` and `data` and contain the following new fields.

`ciphertext` the encryption of `to` and `data`;

`tag` an authentication tag for the ciphertext;

`enc_key` the encapsulated encryption key;

`target_block` a block number, only used in the TIBE case.

In addition, the message signed in `v`, `r`, `s` is adapted to include the new fields.

Encrypted transactions cannot be executed and thus can only be included in the shadow block. Decryption outputs *decrypted transactions* that do not contain `ciphertext` and `tag`, but instead `to` and `data` in plaintext like normal transactions. They retain `enc_key` and `target_block`, and add the `reveal` field which contains the secret used to decapsulate the decryption key. Because the sender signs the transaction in encrypted form, checking the signature on a decrypted transaction requires re-encrypting the transaction to recover the ciphertext and tag and build the signature payload. Deriving the decryption key requires the sender's address in the general case, so we also add the `from` field to provide the address of the sender.

Finally, to handle the case where the ciphertext is invalid, we can put on chain an *undecrypted transaction* with the fields of the encrypted transaction and `reveal`. This avoids DoS attacks with undecryptable transactions by penalizing the sender with gas costs.

4.2 Secret Management Committee

Building on [Bet23], our SMC is a group of $n \geq 3$ DEDIS Ledger Architecture (Dela) instances managing a BN254 BLS key pair with a threshold of $t = \lfloor n \rfloor + 1$. This allows us to implement both the TIBE and TPKE schemes.

To process a decrypted transaction when executing a block, it is necessary to compute the shared secret $k = e(\sigma, U)$ where σ is the decryption key supplied by the SMC, e is the bilinear pairing operation, and U is the encapsulated key. It is also necessary to check the following equation to ensure σ is valid, where **mpk** is the SMC key and L is the identity label.

$$e(\sigma, g_2) = e(\mathrm{H}_1(L), \mathbf{mpk}) \tag{4.1}$$

The algorithm we use to compute pairings consists of two main steps: the Miller loop and the final exponentiation. Scott[Sco05, §4.3] shows that when computing products of pairings, the Miller loop must be performed for each pairing, but the final exponentiation can be performed only once. This is also encouraged in [EIP-197] and [EIP-2537] by making the gas cost of products of pairings an affine function. Thus, re-arranging our formulas to compute products of pairings may allow us to amortize the final exponentiation.

We expect that the pairing operation will dominate the cost, so we optimize by changing (4.1) to take advantage of pairing checks as follows. This should be less expensive than doing two complete pairings.

$$e(\sigma, g_2) \cdot e(\mathrm{H}_1(L), -1 \cdot \mathbf{mpk}) = g_T \tag{4.2}$$

4.3 Time-Lock Puzzles

To build a time-lock puzzle, we use repeated squaring in a 4096-bit RSA group. The sender generates the group, so they know its multiplicative order and are able to shortcut the computation. Then, the block producer has to spend sequential time computing the squarings. To ensure verifiability, We can succinctly prove that the result of a squaring is correct using a public-coin argument described in [Wes18, §4].

Our parameter is 2^{32768} iterations, such that it takes $0.4s$ to compute the function on our experiments machine and $0.75s$ to compute the function with a proof. Clearly, this would not be sufficient in a real world scenario, but due to practical constraints we had to settle for this.

4.4 Application

To understand how our system responds in a realistic scenario, we perform a sealed-bid first price auction for a single item using an Ethereum smart contract. First we evaluate a version using hash-based commit-and-reveal and overcollateralization as a baseline. Then, we modify our contract to take advantage of delayed execution.

In the pre-Lausanne version, the auctioneer starts the auction by locking the item in the contract. Then, there is a commit phase where bidders can commit to their bids with a hash while locking collateral, which is an amount of cryptocurrency equal to the `max_bid` parameter. The commitment is a hash of the bid amount, the sender's address, and a random blinding factor. Then, a reveal phase runs where the bidders reveal the preimage for the hash. As the reveals are processed, the contract refunds either the full collateral if the bid has already been beaten, or the difference to the collateral amount. In that case it also refunds the previous bidder. Finally, after the deadline for the reveal phase, the auction can settle and transfer the item to the winner.

In the delayed execution version, we replace the commit phase and the reveal phase with a single phase, which must not be longer than the execution delay. This ensures that the bidding is already closed as the first transactions start executing. In that phase, bidders submit a single transaction that offers their bid amount. The contract reacts similarly to the reveal phase, either accepting the bid and refunding the previous bidder, or rejecting it.

One can use Ether directly as a currency on Ethereum using the `value` field of transactions. However, it is not encrypted in our design, defeating the purpose of sealed-bid auctions. Instead, we bid using an ERC-20 token, which is a widely used representation for fungible token on the EVM. [EIP-20]

We test and deploy the application using the Foundry toolchain[Kon22] and the Solidity language[1] which are widely used by smart contract developers. They do not need to be modified to work on a Dawn chain.

[1] `https://soliditylang.org/`

Chapter 5

Evaluation

As reported by git, our modifications to go-ethereum itself amount to 1'325 inserted lines and 218 line deletions. The codebase comprises over 550'000 lines of code according to cloc[1] meaning that our changes are modest. This does not include the cryptographic primitives we have implemented, or the 4'000 lines of code for the SMC nodes.

Our contract for Dawn-powered auctions is 27% shorter by bytes than our contract using the traditional method. Less code means less surface for bugs to appear, so we argue that Dawn makes smart contract development safer for our motivating applications.

5.1 Gas Pricing

As a first step to evaluate and refine our design, we want to learn how expensive the decryption and verification operations that every node following the chain must perform are. For that, we try to follow the existing gas system in Ethereum closely.

In [EIP-1108], Cardozo and Williamson use the metric that one microsecond of computation should cost at least 25.86 gas, based on their benchmark of an already priced operation. In [EIP-2537], Vlasov et al. use 30M gas per second, so 30 gas per microsecond. No rationale is given. Both could be equally valid choices given different hardware and software assumptions. This can also vary over time.

We performed our benchmarks on a Raspberry Pi 4B with 8 gigabytes of RAM, running NixOS Linux 23.11. This is intended to reflect the intent of the Ethereum community to support running nodes on low-end machines such as cheap single-board computers, in order to lower the barrier of en-

[1] `https://github.com/AlDanial/cloc` version 2.00

Machine	time μs/op	gas cost (gas)	throughput ($m$gas/s)
Raspberry Pi 4B	293.862	3000	10.20
Reference	116	3000	25.86

Table 5.1: Calibration benchmark using `BenchmarkPrecompiledEcrecover`

try to run a node. To compare our setup to the EIP-1108 benchmark, we first reproduce the reference benchmark which is available in the go-ethereum codebase as `BenchmarkPrecompiledEcrecover`.

The result is presented in Table 5.1. We can therefore stick with the methodology of EIP-1108 by enforcing that a microsecond of computation on our machine should cost at least 10.2 gas units to try to maintain consistency.

The schemes we consider are

ChaCha20-HMAC-SHA256 the ChaCha20 stream cipher paired with HMAC for committing authentication.

AES-CTR-HMAC-SHA256 the AES256 cipher in counter mode, similarly paired with HMAC.

RK-ChaCha20-Poly1305 the authenticated encryption scheme ChaCha20-Poly1305, modified following Gueron[Gue20] to ensure commitment.

RK-AES-GCM the authenticated encryption scheme AES-GCM, similarly modified.

First, we pit the selected algorithms against each other. In figure 5.1, we first note that the RK variants are twice as fast on large messages as the HMAC variants. We also note that RK-ChaCha20-Poly1305 has the lowest start-up overhead and per-byte computation cost on the Raspberry Pi platform.

Then, we investigate round reduction. In [Aum19], Aumasson argues that when standardizing symmetric primitives such as hash functions and ciphers, the number of rounds is chosen very conservatively, and that one could confidently reduce this number later on. Concretely, he suggests that ChaCha with 8 rounds is secure and 2.5 times faster than ChaCha20. To investigate this, we run ChaCha unauthenticated with 3 different round settings. We present the results in Figure 5.2. We see that, for our largest payloads, round reduction provides an appreciable speedup, going from 16 milliseconds to 12 (ChaCha12) and 10 (ChaCha8) for a 2 megabyte message. The efficiency gain is most likely lesser when authentication is added back. In the best case scenario, there is a reduction of 25% and 37.5% respectively in the computation time.

For informational purposes, we quantify the cost of key-commitment by benchmarking authenticated but non-committing versions of our chosen algorithms in Figure 5.3. We find no discernible

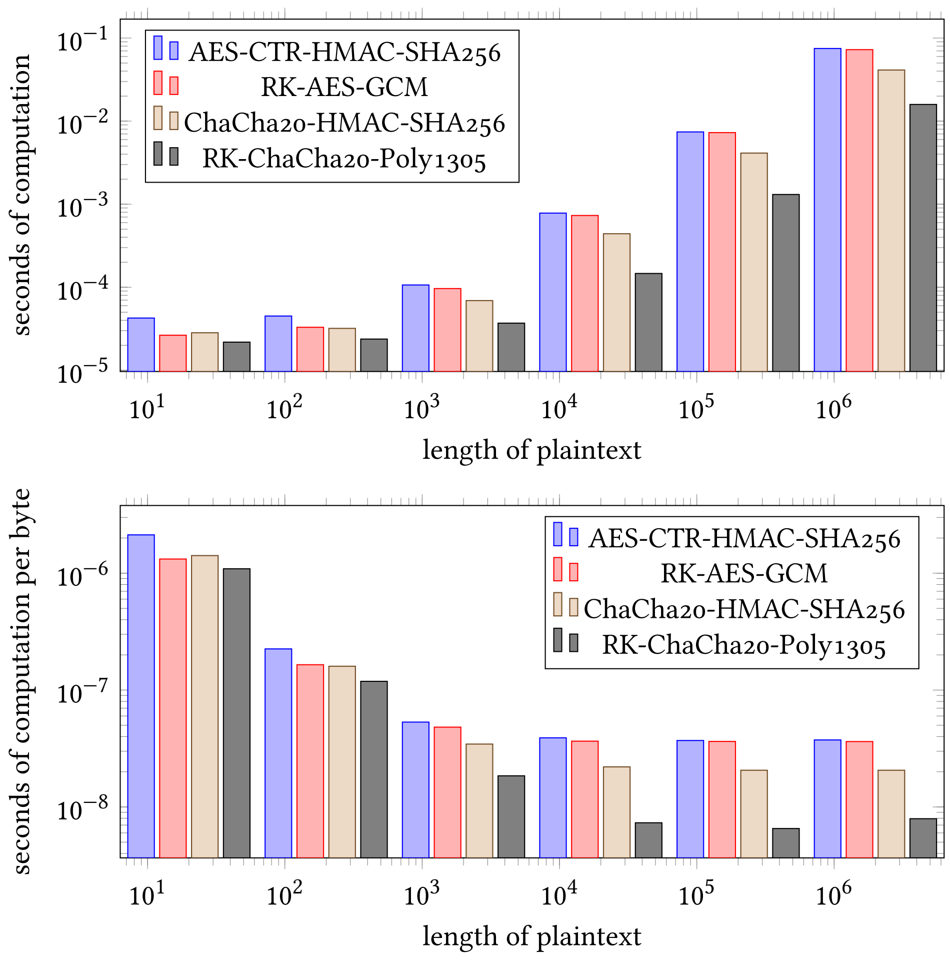
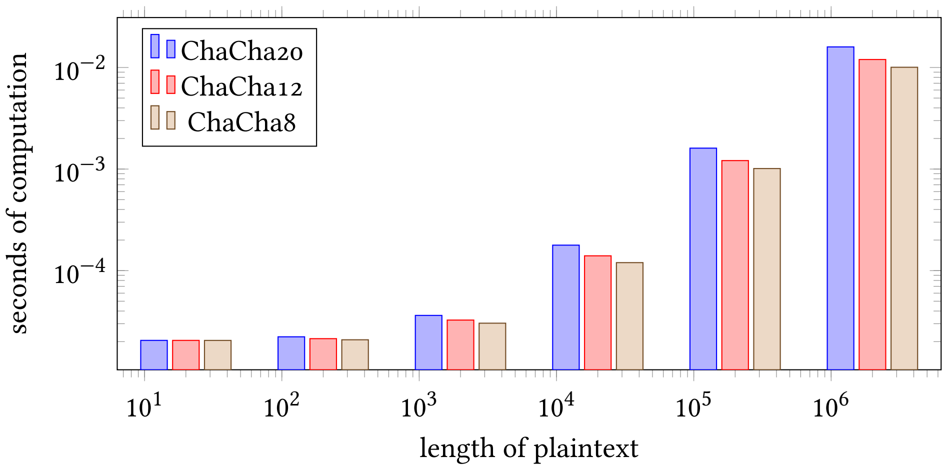


Figure 5.1: gas benchmarks for candidate algorithms

difference. Since non-malicious transactions can be stored in decrypted form, there is also no long-term storage cost, meaning that the mitigation is effectively free.

After evaluation, we choose to retain RK-ChaCha20-Poly1305 as our CAE scheme since it performs the best out of the candidates we selected. Round reduction could deliver a small performance improvement, at the cost of a less standard scheme, some engineering time, and some controversy. We leave it as an exercise for future researchers. We note that this decision is based on regular CPUs. Should Ethereum decide to optimize for ZK proof systems in the future as suggested in [EIP-7667], other algorithms should be considered.

In the TPKE and TIBE modes, verification requires some elliptic curve pairing operations. In Section 4.2 we hypothesized that the run times of those routines would be dominated by the Miller loop (M) and the final exponentiation (F), and we described our optimization to avoid one F. This would mean that RecoverSecret is $M+F$, the slow VerifyIdentity is $2M+2F$, and the fast VerifyIdentity



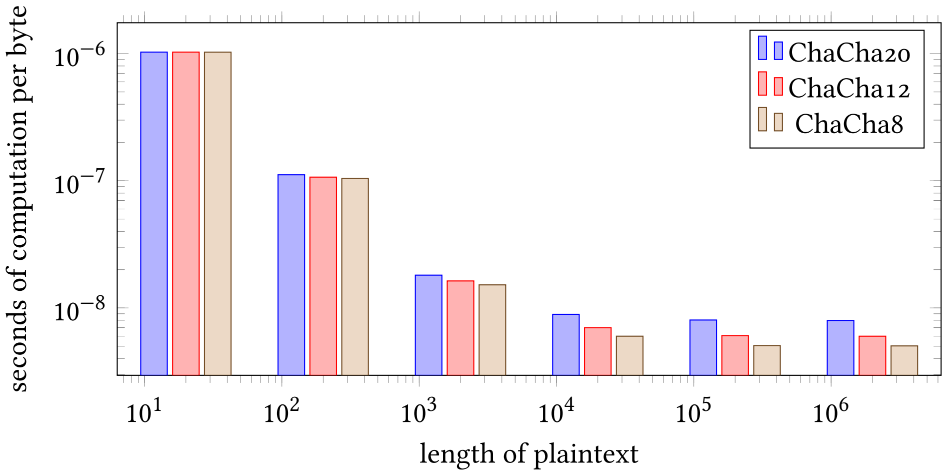
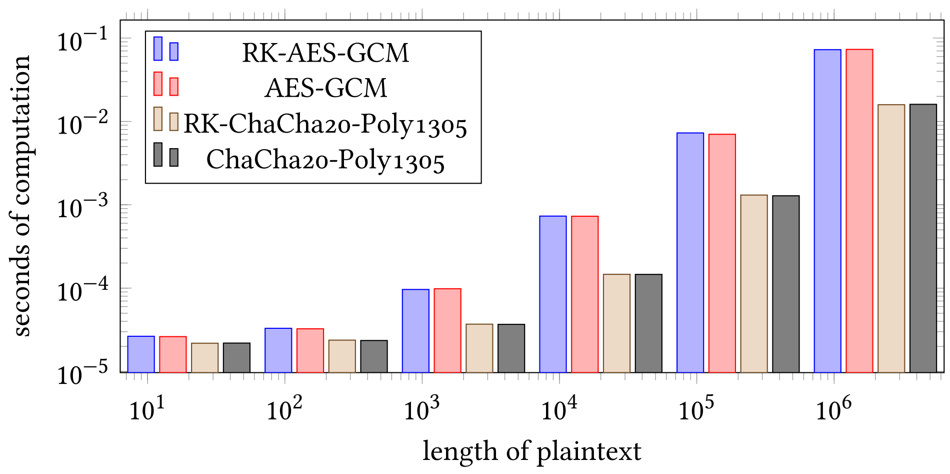


Figure 5.2: gas benchmarks for round reduction



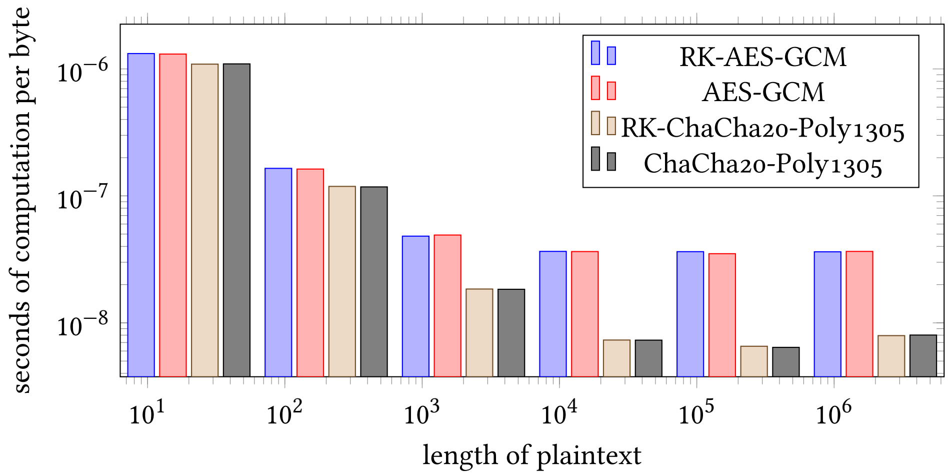


Figure 5.3: gas benchmarks for key commitment

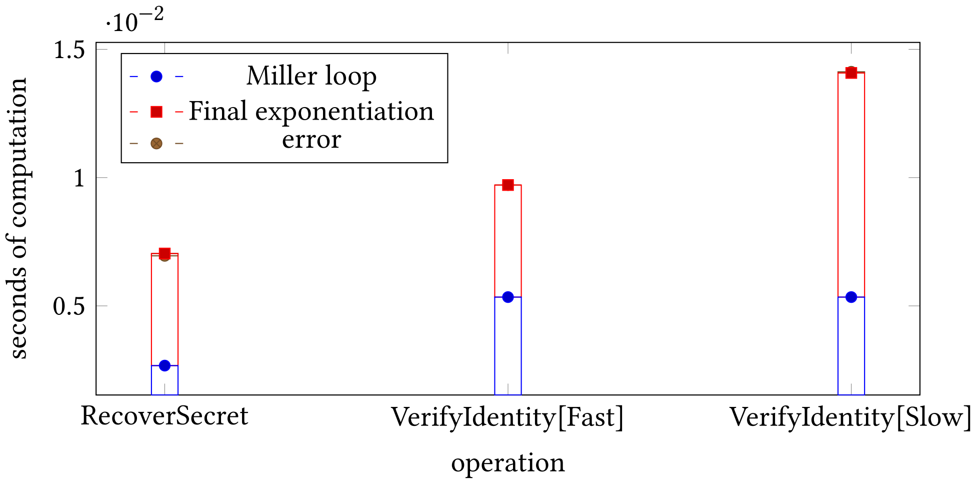


Figure 5.4: benchmarks for gas-relevant IBE algorithms

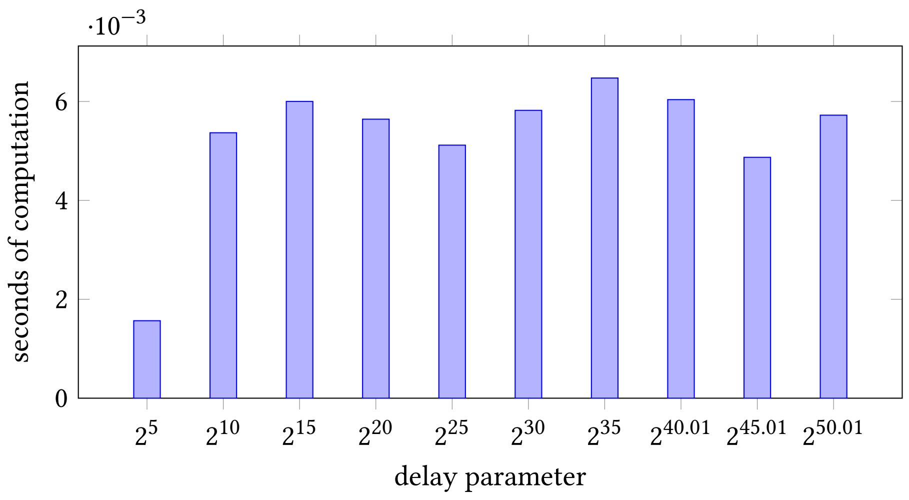


Figure 5.5: benchmarks for VDF proof verification

is $2M+F$. In Figure 5.4, we benchmark both routines. This gives us $F \approx 4368.787$ and $M \approx 2670.824\mu s$ with a maximum error of 1.26%, which is consistent with our hypothesis.

We can convert these values to gas: $F \equiv 44562$ and $M \equiv 27242$. The costs in EIP-1108 are $F \equiv 45000$ and $M \equiv 34000$. The former is spot on, but the latter is cheaper in our implementation by 20%. Again in EIP-1108, Parity is referred to as "the less performant client", which implies that go-ethereum's implementation is more performant. We use Cloudflare's BN254 library, and so does go-ethereum. This explain our observations very well.

We decided to align with the existing practice when pricing these elliptic curve operations. Thus, we charge $3M + 2F \equiv 192000$gas per transaction. Alternatively, we could have picked $M \equiv 28000$ to reflect performance in Go and charge $3M + 2F \equiv 174000$gas.

Finally, we benchmark VDF verification and output recovery in figure 5.5. We find that the runtimes oscillate for an unknown reason, but are broadly consistent with the idea that the operation is constant time in its worst case. The highest value we record is $6474\mu s$, which suggest a gas cost of 66'000 for the operation.

Overall, an encrypted transaction should clearly cost more gas. Increasing the *intrinsic gas*, which always charged to every transaction to cover validation and long-term storage, by 192'000, or 66'000 in the time-lock puzzle case, would conservatively account for the key decapsulation and verification. The cost of RK-ChaCha20-Poly1305 of 220 gas units plus 0.144 gas units per byte could be added to that, but we opt not to since it is negligible relative to the rest of the cost, suggesting to us that it is not a likely DoS vector.

The impact on the fee paid by transactors can be calculated like so: assume a gas price of 10

Scheme	`target_block`	`enc_key`	`reveal`	`from`	total	cost
TPKE	0	128	64	20	212	\$0.119
TIBE	8†	128	64	20	220	\$0.123
VDF	0	256	544	20	820	\$0.459

†: assuming 64-bit representation, actual encoding may be more efficient.

Table 5.2: Sizes of additional fields in bytes

gwei (1 gwei = 10^{-9} Ether) and an Ether price of 3'500 USD. Multiply with 192'000 or 66'000 gas to obtain \$6.72 or \$2.31. For a big trader at risk of losing thousands to front running, this would likely be a no-brainer. However, for auctions and other use cases, it must be multiplied by the number of participants. Especially in the case of small value auctions and non-financial use cases, this may be prohibitive.

In Table 5.3, we use data about a typical swap transaction in an existing decentralized exchange application, and find that if Dawn was deployed on Ethereum, the additional gas cost of front running protection would be 35% with SMC and 15% with VDF.

We should note that, due to the high costs of computation on Ethereum, so-called layer-2 solutions exist to move computation off of the main chain. In particular many *rollups* are being used in production. A rollup removes the need to execute the transaction logic on the main chain but still posts data back to the main chain[TSH22]. For instance, one way to build a rollup is to use a ZK proof that the computation was performed correctly. As a result, computation costs are reduced, but data costs can remain high. This means that the additional witness data produced by our scheme also needs to be evaluated.

We present the lengths of the new fields in Table 5.2. By this metric, the most efficient protocol is TPKE, closely followed by TIBE. However, since the `reveal` field of TIBE is the same for transactions with the same target block, optimisations could be applied although the table still presents the worst case. Due to the size of RSA group elements, the time-lock scheme is nearly four times as expensive. To estimate the costs, we use the rule that, on Ethereum, one nonzero byte of data costs 16 gas[2], again with 10 gwei and \$3'500 Ether. Overall, these costs are much lower than the computation costs, but may irritate some layer-two users if the computation costs are small in comparison.

5.2 Throughput

For a given level of throughput to be sustainable on a Dawn chain, two conditions must be satisfied: the block producers must be able to receive all the decryption keys in time, and the gas consumption

[2]There exists an alternative way to publish data on Ethereum, known as blobs, which may be cheaper .[EIP-4844]

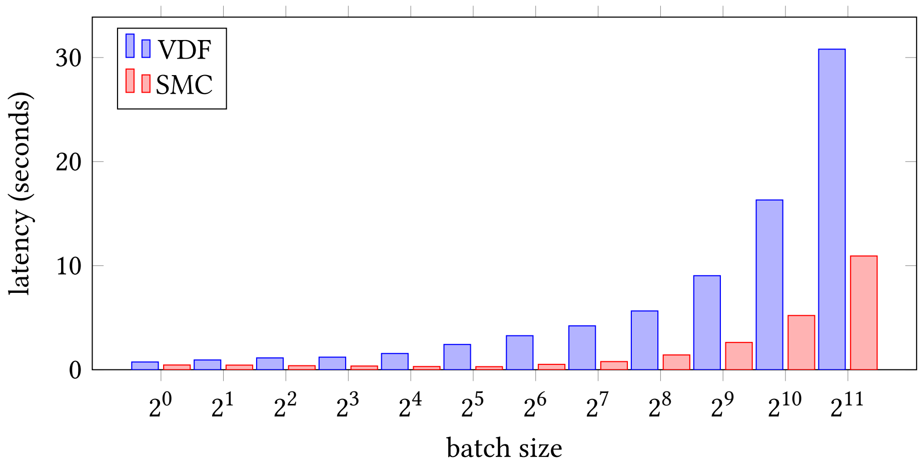


Figure 5.6: Batch Decryption Latency

must stay within the predefined limit.

To address the first condition, we measure in Table 5.6 how quickly our schemes can decrypt every transaction in a batch. We use a 40-core KVM virtual machine with 32GB of RAM running Debian GNU/Linux provided by the DEDIS laboratory. Since blocks on Ethereum come every 12 seconds, we chose 10 seconds as a conservative threshold. Accordingly, the VDF scheme can process up to 512 transactions per block and the SMC schemes up to 2'048. This corresponds to 42 and 170 transactions per second, respectively.

To determine how many bids the gas limit allows, we will look at the gas units consumed by transactions in our prototype. In Table 5.3, we collect the highest amount of gas that can be consumed during a given action, and analyse which components are responsible for it. The "base" item is the original intrinsic gas. "smc" and "vdf" cover verifiable decryption as per the previous section. Since different code paths can incur different costs, we distinguish the case where a bid is the new highest bid and so funds have to be transferred to and from the auction, and the case where it is not and the operation is less expensive.

When using overcollateralized auctions, a commit and a reveal transaction together amount to either 134'443 or 150'814 gas, which is thus amount of gas per bidder. In the worst case scenario, bids are processed in ascending order and each consume the higher amount. In the best case scenario, they are processed in descending order and consume the lower amount.

When using Dawn, the bidder makes a single transaction. When the CG fill the shadow block, they do not know if the bid will execute as the current highest or not, hence they reserve the higher amount. Still, in the VDF case, the higher gas expense is closer to the lower end of the range for a pair of overcollateralized auction transactions. However, in the SMC case, the higher expense is 75% to 96% higher than baseline. This means that in the current state, the TIBE and TPKE variants will

Routine	Gas cost	
Commit bid	base + 46200	67200
Reveal bid	base + 46243	67243
Reveal highest bid	base + 62614	83614
Bid (SMC)	base + smc + 11487	224487
Bid highest (SMC)	base + smc + 50537	263537
Bid (VDF)	base + vdf + 11487	98487
Bid highest (VDF)	base + vdf + 50537	137537
Swap†	base + 335190	356190
Swap (SMC)‡	base + smc + 335190	548190
Swap (VDF)‡	base + vdf + 335190	422190

base = 21000 smc = 192000 vdf = 66000

†: data according to [Eth24c] ‡: extrapolated

Table 5.3: Gas cost breakdown for typical transactions

exhibit at best half the throughput as the baseline, but the VDF variant could be roughly equal. It is important to note that those results are mainly determined by the gas pricing that we established in the previous section, meaning that any change of policy there will greatly influence them.

Putting this together, at 2'500'000 gas per second, we are limited to 18 and 9 bids per second respectively. This is the bottleneck since decryption can easily keep up.

5.3 Observations

When running under delayed execution, Operations such as deploying the contracts or preparing funds for the auction take more time. Unsurprisingly, the program has to wait for confirmation that the transaction executed as expected, which means waiting for inclusion and then waiting for the block delay. This does not affect actions that can be performed in parallel or optimistically in sequence, except for the final latency.

As previously stated, the TIBE scheme introduces a rule that a transaction cannot be included if its target block is too old. This is important so that nodes do not need to store old decryption keys indefinitely, but has the side effect of creating time-limited transactions. When using the auction application, we set the target block for bids to be the block where the auction opens. That way, a bid is either included on time, or it expires. This is advantageous because if it were included, the transaction would revert anyways. Moreover, it seems to us that for other use cases, this is not such a bad outcome: late votes and randomness contributions will most likely be discarded. Trades are more nuanced: the trade will not be able to be front run, but the user's intent will still be revealed. For that reason, this use case will likely prefer TPKE-style guarantees.

We found that, under normal circumstances, a go-ethereum node, even without participating in consensus, will try to construct a potential next block locally. This is useful for gas estimation, and more generally for speculative execution. Delayed execution disrupts this process however. Additionally, we had to manually set gas limits when sending the bid transactions because the simulation would respond with an error because the auction had not opened yet in the simulated state, even though it would be open by the time the transaction is executed.

Chapter 6

Related Work

In [Qu23], Qu designs and implement a threshold decryption scheme for use with The threshold decryption scheme is based on [SG02], which does not rely on pairings and can thus use the Ed25519 curve. Qu also discusses a variation that enables the sender to construct their own SMC, but we will refer to his paper.

Qu also proposes to implement *verifiability* despite the lack of pairings in a clever way: the sender is required to include a hash of the symmetric key. In the honest scenario, the key can be provided when executing the transaction. In case the sender is dishonest, they can be slashed by posting the whole decryption proof. We believe that there is a weakness in this construction in that, with the complicity of a block producer, a sender can craft a transaction where the hashed key doesn't match the encapsulated key, an obtain the capacity to a posteriori cancel their transaction: they can reveal the preimage and the transaction will decrypt and execute correctly, or they can let the decryption proof be posted, making the transaction invalid. In private correspondence, Qu noted that the misbehavior would still be detectable by using the decryption shares, and suggested that further study would be needed to reach a conclusion. We concur.

Looking more closely at the decryption step in the threshold scheme, we see that it computes $\mathbf{mpk}^r$ given (among other things) $u = g^r$. A succinct proof of validity would be a Schnorr discrete logarithm equality proof $\log_g(u) = \log_{\mathbf{mpk}}(\mathbf{mpk}^r)$. It might be possible to produce such a proof with a protocol similar to FROST[KG20], albeit after multiple rounds of communication within the SMC. This is an avenue for future research. We note that this protocol would also be a threshold version of ECVRF. [RFC9381]

Another difference between our work and Qu's is context binding: Qu did not consider auctions and protocols with multiple participants and thus did not involve the sender address and nonce in the encryption and decryption routines. Thankfully, we believe this can be addressed with a minor modification.

Finally, we differ in the way we penalize a malicious sender: we chose to turn a malformed transaction into a no-operation transaction at the expense of the sender using the existing gas mechanic. This integrates more directly into the existing architecture and does not require an additional deposit. However, if this mechanic turns out to be impractical, Qu's can still be used. Overall, the scheme provides similar guarantees to our TPKE scheme.

Shutter Network is working on a real-world deployment of front-running protection. Shutter does not introduce new transaction types like we do, instead choosing to encrypt normal transactions entirely and publishing them to a contract using another transaction. This has the advantage of not requiring a hard fork, in fact, since a subset of the CG nodes can opt-in to relay decrypted transactions, it is really a soft fork. [KS24]

On rollups, Shutter can instead submit transactions to a third-party termed the *collator*, who will commit to their order and obtain the decryption key from the SMC.[Shu22] This shows that Shutter is more similar to our TIBE scheme, with batches as the identity labels.

Based on code inspection, it seems like in Shutter, SMC members communicate using a specialized blockchain using the Tendermint[BKM19] consensus algorithm. Contracts on the main blockchain will trust that the majority of SMC nodes is honest when receiving signed messages from that chain.

Still based on code inspection, the encryption scheme of shutter is interesting because it implements key commitment and authentication in a different way than our solution.

```
Enc(msg, mpk, label):
  sigma := get_random_bytes(32)
  r := hash(sigma || msg)
  C1 := r * G2
  C2 := sigma xor hash_point(r * e(hash_to_g1(label), mpk))
  C3 := symmetric_encrypt(sigma, msg)
  return (C1, C2, C3)
```

This means that, upon release of the decryption for the label, parties are able to recover `sigma`, which is the symmetric key, from `C1` and `C2`, then recompute `r` to check consistency of the key and plaintext. We note that, since the label is the epoch number, this scheme does not perform context binding, but since the ciphertext is a signed transaction, this does not lead to a vulnerability.

Finally, Shutter provides a production-ready SMC setup, something that is very valuable and was not studied in this work. The important difference is that we choose to enforce that correct decryption is verified on-chain and not just by the consensus group, and generalize the approach for multiple participants.

In [Cho+24], Choudhuri et al. comment on Shutter's scheme among others, and suggest a *batched* TPKE scheme that can decrypt multiple transactions with the same communication costs as one. It combines the flexibility of our TPKE scheme with the efficiency of our TIBE scheme, albeit at the cost of setup assumptions, and thus appears to be a significant breakthrough.

In [Sta+21], Stathakopoulou et al. use *Trusted Execution Environments (TEE)* to implement front running protection. TEE are hardware-based solutions that allow a program to run in an untrusted environment and produce a signed attestation of the process. TEE require trust in the hardware manufacturer. A TEE-based setup can replace a decentralized SMC and potentially perform processing more complex than just decryption. We note that the paper features a more formal definition of a front running attack, which is equivalent to our privacy guarantee, and also guarantees *sender obfuscation*, meaning the sender of a transaction is private before the transaction is ordered, which we do not provide.

In [EIP-7547] Neuder et al. present a mechanism for block producers to force subsequent producers to include certain transactions. The idea is that, since Ethereum producers often delegate block building to a few specialized entities for monetary reasons, those specialized entities have too much control over the network, potentially calling into question its decentralization. In particular, block builders have the power to censor transactions arbitrarily. To mitigate this risk, an upgrade could allow the block proposer of block n to select a few transactions and enforce that they are included in block $n + 1$ at the latest.

Inclusion lists are interestingly similar to our shadow blocks. They are lists of transaction that constrain future block producers. However, unlike inclusion list, shadow blocks need to wait for finality. In principle, transactions on inclusion lists could be encrypted, but this is not a design goal. If that were the case, they would need to enforce some ordering guarantees to prevent front running. Inclusion lists do not replace the existing block building mechanism, but instead complement it. As a result, they don't prevent MEV extraction like we do, which is not their goal.

Neuder et al. also work more closely with Ethereum as it exists in the field, addressing concrete data structures, interoperability with block builder software, data retention, interaction with EIP-1559, and enforcement of consensus rules. These are useful solutions to problems that were not addressed in this work.

Cicada[Gla+23] uses *homomorphic* time-lock puzzles to implement sealed-bid auctions and voting. It differs from our solution in that, instead of decrypting each participant's contribution on its own and executing it on chain, it can allow an off-chain actor to perform the application logic on the encrypted inputs without solving the puzzle, then solve a puzzle that directly produces the output. The process can then be verified on-chain.

Compared to Dawn with VDF, Cicada reduces the amount of puzzles that must be solved to one instead of n, thereby saving a substantial amount of electricity. It can also be implemented on top of any Ethereum-compatible chain instead of requiring an upgrade. We note that it currently requires sigma protocols to be designed by hand for each program, and that the lowest gas cost per voter is reportedly 418'358, which is substantially more than the cost of a bid in our experiment even in the worst case. However, further development could both deliver a generalized execution environment and bring costs down. We suggest it may make sense to integrate some of the proof schemes into the underlying execution environment (likely a rollup) to avoid the overhead inherent in an EVM implementation.

In [DGM23], Dujmovic, Garg, and Malavolta provide a construction for time-lock puzzles that support batch-solving, meaning multiple puzzles using the same parameters can be solved as efficiently as a single one. If applied to our VDF scheme, given a set of puzzle parameters for each block, senders could pick a target block using the same logic as in the TIBE variant and the CG could solve one puzzle per block. The drawback is that, to our knowledge, setting up this scheme requires a trusted setup phase — specifically a RSA group where the private parameters are used and erased[MT19], which may be hard to do in a decentralized setting.

6.1 Final Considerations

Thanks to our design choice with regards to decrypted transactions, we can store and transmit regular blocks in plaintext form. Currently, the size of plaintext Ethereum blocks can be reduced three- to fivefold with compression. [EIP-706] Losing this property by storing transactions in encrypted form would likely have a practical impact. As long as shadow blocks are discarded after some time, just like inclusion lists would be, Dawn should not adversely impact performance in this area.

In our design, we decide that all transactions would be subject to delayed execution. This is a simple way to prevent front running even if a committed transaction is decrypted a bit early, but it forces even applications that do not benefit from Dawn to undergo the execution delay before knowing what will be in the EVM state when they are executed. One could counter that that knowledge is not final, but many applications may make stronger assumptions about the likelihood of a block being reorged than us. As a result, one could allow a fast path where unencrypted transactions can be executed directly in the block after the shadow block has been exhausted. This also avoids wasting the gas reserved but not used by the shadow block and partially restores the ability to extract MEV. However, it makes it more complex for commit-and-reveal applications to enforce the delay, especially in case participants try to cancel their own bids by changing some external conditions, e.g. moving their tokens.

A different approach to avoid wasting reserved gas would be to require the block producer to include as many consecutive shadow block transactions as will fit. This may also help restoring EIP-1559 functionality by giving better feedback, albeit with a delay. For this reason and to decrease end-user latency, faster finality on Ethereum would benefit Dawn.

We propose to allow transactions to access their decryption key from the EVM. This would allow contracts to enforce that only delayed transactions may touch a given piece of state. It would also serve, in the TPKE case as a per-transaction source of randomness that even the sender cannot predict, since the verifiable decryption key is unique. This is also true in the TIBE case but it is shared with other transactions with the same target block.

An issue that we did not address in this work is how the SMC node operators should be incentivized. Clearly they must be compensated in some way since they dedicate resources to the activity for the benefit of the blockchain system. Whether issuing tokens or awarding them a cut of gas fees is more desirable is up to the implementers.

We do not achieve sender obfuscation in Dawn: the address of the sender is known from the encrypted transaction. A possible way to remediate this, along with hiding fields such as the gas limit, is to encrypt these fields and submit a zero-knowledge proof that the transaction is valid. This requires showing that, in a recent blockchain state, the balance requirement is met and the nonce matches. Allowing consecutive transactions from the same account is subtle however, since it requires checking that the most recent pending balance is sufficient and that the nonce is consecutive.

Since its inception, Ethereum has sought to remove the special status that ECDSA secp256k1 key pairs possess in the protocol and instead allow accounts to be defined by EVM code. We should explore if such a mechanism is compatible with Dawn. In our opinion, a proposal such as [EIP-7701] could fit: the proposal makes it so that transaction validation is expressed as a call into the EVM. The payload of this call could remain in plaintext, whereas the main call, which would be subject to delayed execution, can be encrypted. To enable sender obfuscation, the validation contract itself could require and verify a ZK proof.

Another obfuscation angle is balance obfuscation: with ERC-20 tokens, every user's balance is public. This could enable bidders to make guesses as to other bidders' future moves. Dawn may be combined with a confidential payments mechanism such as Zether[Bün+19]. Assume the user can withdraw from Zether and use the funds to bid in a single transaction: then Zether protects the balance prior to the auction, and Dawn prevents the bid from being read in-flight.

In the VDF scheme, the sender of the transaction is in a privileged position: they already know the decryption key without needing to perform the slow computation and they can even generate a proof. One may want to incentivize them to cooperate with the CG to reduce computational costs. Wesolowski shows that proofs can be attributed to an identifier ("watermarked") by including it in the Fiat-Shamir input. [Wes18, §7.2] Thus, it can be shown whether the sender cooperated or not. The exact incentive mechanism remains undetermined for now.

Although we pretended to upgrade Ethereum itself for the project, an upgrade such as the Lausanne Hard Fork is unlikely to occur in the near future for a major reason: it adds an honest majority assumption on the SMC, which in our estimation, the Ethereum community will not want. This may be different in the case of rollups and other chains. Observe that the chain itself becomes dependent on the SMC's availability: if transactions can no longer be decrypted, new blocks will not be produced because we require them to include finalized encrypted transactions. Worse still: if the SMC is malicious, it can choose which block producers are allowed to decrypt and cause the others to be blamed for unavailability. We might imagine an automated process whereby validators can bear witness that the SMC is unavailable and disable the functionality as an emergency measure, but further research is needed to determine if this can be abused. This is less pronounced in the VDF case, but it assumes all validators are able to perform the VDF computation efficiently enough.

We also note that reusing the CG to be the SMC may resolve both of the previously highlighted issues. However, we reserve our judgement until more research is available.

Chapter 7

Conclusion

In this work, we described Dawn: a blockchain architecture that enables multiple users to provide their input to a smart contract in a temporarily private way, and its prototype based on Ethereum. We simulated realistic sealed bid auctions using this prototype, and discovered the trade-offs between the TPKE, TIBE, and VDF methods: TPKE is most indicated for front running protection, while TIBE is more suited for auctions and votes. VDF is much harder to put in production at this stage due to the energy consumption and uncertainty about computational limits, but is computationally cheaper to verify.

A Dawn chain has backwards compatibility with existing tooling, Under realistic conditions, the price of the delayed execution and front running protection feature is reasonable for an average blockchain user. We speculate that future developments or deployment on layer-twos may reduce the cost even more.

Drawbacks of Dawn are the increased latency when observing transaction execution without waiting for finality, the reduced throughput due to verification costs and block building inefficiencies, and the disruption of helpful MEV. Compared with existing solutions, we eliminate long-term storage of ciphertexts, remove the need for a separate fee payment mechanism, and explicitly target use cases involving multiple users.

In this work, we provided a rigorous treatment of verifiable decryption, committing authenticated encryption, and context binding. We also abstracted and formalized the concept of a commit-and-reveal application. We also highlighted works related to batched threshold decryption, homomorphic time-lock puzzles, sender obfuscation, and time-lock puzzles with batch solving. We briefly discuss interactions between delayed execution and ongoing developments in Ethereum such as account abstractions and inclusion lists. Finally, we highlighted future areas of inquiry such as more MEV-friendly designs, more efficient block building, randomness provision, SMC and VDF incentivization, sender and balance obfuscation, and account abstraction.

In 2022, John Wang wrote that "Since the execution/settlement layer is a highly competitive, race-to-the-bottom of tight margins, the rollups with the best MEV mitigation mechanisms will have an edge". [Wan22] Stimulated by the competition taking place in the market, this field of study will certainly remain active and foster fascinating research and exciting developments.

Bibliography

[Aum19] Jean-Philippe Aumasson. *Too Much Crypto.* Cryptology ePrint Archive, Paper 2019/1492. 2019. URL: https://eprint.iacr.org/2019/1492.

[BBH06] Dan Boneh, Xavier Boyen, and Shai Halevi. *Chosen Ciphertext Secure Public Key Threshold Encryption Without Random Oracles.* 2006. URL: https://crypto.stanford.edu/~dabo/papers/ibethresh.pdf.

[Ben+18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Scalable, transparent, and post-quantum secure computational integrity.* Cryptology ePrint Archive, Paper 2018/046. 2018. URL: https://eprint.iacr.org/2018/046.

[Bet23] Julie Bettens. *Optimizing Front-running Protection.* 2023. URL: https://www.epfl.ch/labs/dedis/wp-content/uploads/2024/06/Bettens2023_FrontRunningProtection.pdf.

[BF01] Dan Boneh and Matthew Franklin. *Identity Based Encryption From the Weil Pairing.* 2001. URL: https://crypto.stanford.edu/~dabo/papers/bfibe.pdf.

[BKM19] Ethan Buchman, Jae Kwon, and Zarko Milosevic. *The latest gossip on BFT consensus.* 2019. arXiv: 1807.04938 [cs.DC].

[Bon+18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. *Verifiable Delay Functions.* Cryptology ePrint Archive, Paper 2018/601. 2018. URL: https://eprint.iacr.org/2018/601.

[Bün+19] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. *Zether: Towards Privacy in a Smart Contract World.* Cryptology ePrint Archive, Paper 2019/191. 2019. URL: https://eprint.iacr.org/2019/191.

[But+20] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. *Combining GHOST and Casper.* 2020. arXiv: 2003.03052 [cs.CR].

[But14] Vitalik Buterin. *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.* 2014. URL: https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf.

[Cho+24] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. *Mempool Privacy via Batched Threshold Encryption: Attacks and Defenses.* Cryptology ePrint Archive, Paper 2024/669. 2024. URL: https://eprint.iacr.org/2024/669.

[Dai+19] 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. arXiv: 1904.05234 [cs.CR].

[DGM23] Jesko Dujmovic, Rachit Garg, and Giulio Malavolta. *Time-Lock Puzzles with Efficient Batch Solving.* Cryptology ePrint Archive, Paper 2023/1582. 2023. URL: https://eprint.iacr.org/2023/1582.

[Edg23] Ben Edgington. *Upgrading Ethereum.* Commit ebfcf50. 2023. URL: https://eth2book.info/capella.

[EIP-1108] Antonio Salazar Cardozo and Zachary Williamson. *EIP-1108: Reduce alt_bn128 precompile gas costs.* https://eips.ethereum.org/EIPS/eip-1108. Ethereum Improvement Proposal. 2018.

[EIP-1559] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. *EIP-1559: Fee market change for ETH 1.0 chain.* https://eips.ethereum.org/EIPS/eip-1559. Ethereum Improvement Proposal. 2019.

[EIP-197] Vitalik Buterin and Christian Reitwiessner. *EIP-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128.* https://eips.ethereum.org/EIPS/eip-197. Ethereum Improvement Proposal. 2017.

[EIP-20] Fabian Vogelsteller and Vitalik Buterin. *ERC-20: Token Standard.* https://eips.ethereum.org/EIPS/eip-20. Ethereum Improvement Proposal. 2015.

[EIP-225] Péter Szilágyi. *EIP-225: Clique proof-of-authority consensus protocol.* https://eips.ethereum.org/EIPS/eip-225. Ethereum Improvement Proposal. 2017.

[EIP-2537] Alex Vlasov, Kelly Olson, Alex Stokes, and Antonio Sanso. *EIP-2537: Precompile for BLS12-381 curve operations.* https://eips.ethereum.org/EIPS/eip-2537. Ethereum Improvement Proposal. 2020.

[EIP-4844] Vitalik Buterin, Dankrad Feist, Diederik Loerakker, George Kadianakis, Matt Garnett, Mofi Taiwo, and Ansgar Dietrichs. *EIP-4844: Shard Blob Transactions.* https://eips.ethereum.org/EIPS/eip-4844. Ethereum Improvement Proposal. 2022.

[EIP-706] Péter Szilágyi. *EIP-706: DEVp2p snappy compression.* https://eips.ethereum.org/EIPS/eip-706. Ethereum Improvement Proposal. 2017.

[EIP-7547] Michael Neuder, Vitalik Buterin, Francesco Damato, Terence, Potuz, and Manav Darji. *EIP-7547: Inclusion lists.* https://eips.ethereum.org/EIPS/eip-7547. Ethereum Improvement Proposal. 2023.

[EIP-7667] Vitalik Buterin. *EIP-7667: Raise gas costs of hash functions.* https://eips.ethereum.org/EIPS/eip-7667. Ethereum Improvement Proposal. 2024.

[EIP-7701] Vitalik Buterin, Yoav Weiss, Alex Forshtat, Dror Tirosh, and Shahaf Nacson. *EIP-7701: Native Account Abstraction with EOF.* https://eips.ethereum.org/EIPS/eip-7701. Ethereum Improvement Proposal. 2024.

[ENS22] ENS. *Controller — ENS Documentation.* https://docs.ens.domains/contract-api-reference/.eth-permanent-registrar/controller. 2022.

[Eth24a] Ethereum.org Contributors. *Blocks.* Last updated February 27, 2024. 2024. URL: https://ethereum.org/en/developers/docs/blocks/.

[Eth24b] Ethereum.org Contributors. *The history of Ethereum.* Last updated March 14, 2024. 2024. URL: https://ethereum.org/en/history.

[Eth24c] Etherscan.io. *Ethereum Gas Tracker.* 2024. URL: https://etherscan.io/gastracker (visited on 06/20/2024).

[Gla+23] Noemi Glaeser, István András Seres, Michael Zhu, and Joseph Bonneau. *Cicada: A framework for private non-interactive on-chain auctions and voting.* Cryptology ePrint Archive, Paper 2023/1473. 2023. URL: https://eprint.iacr.org/2023/1473.

[Gue20] Shay Gueron. *Key Committing AEADs.* Cryptology ePrint Archive, Paper 2020/1153. 2020. URL: https://eprint.iacr.org/2020/1153.

[K22] Tatu K. *Shutter brings shielded voting to Snapshot.* 2022. URL: https://blog.shutter.network/shutter-brings-shielded-voting-to-snapshot.

[KB21] Georgios Konstantopoulos and Vitalik Buterin. *Ethereum Reorgs After The Merge.* 2021. URL: https://www.paradigm.xyz/2021/07/ethereum-reorgs-after-the-merge.

[KG20] Chelsea Komlo and Ian Goldberg. *FROST: Flexible Round-Optimized Schnorr Threshold Signatures.* Cryptology ePrint Archive, Paper 2020/852. 2020. URL: https://eprint.iacr.org/2020/852.

[Kon22] Georgios Konstantopoulos. *Announcing Foundry v0.2.0.* 2022. URL: https://www.paradigm.xyz/2022/03/foundry-02.

[KS24] Tatu K. and Shutter Network. *The First Shutterized Testnet Is Now Live on Chiado!* 2024. URL: https://blog.shutter.network/the-first-shutterized-testnet-is-now-live-on-chiado/.

[LGR20] Julia Len, Paul Grubbs, and Thomas Ristenpart. *Partitioning Oracle Attacks.* Cryptology ePrint Archive, Paper 2020/1491. 2020. URL: https://eprint.iacr.org/2020/1491.

[May93] Timothy C. May. *Timed-Release Crypto.* 1993. URL: https://mailing-list-archive.cryptoanarchy.wiki/archive/1993/02/a421c6fc805dfb4ae4197521e8a9e91dd456e3deab855f12af31a4b1ccccf6cb.

[MGZ22] Peyman Momeni, Sergey Gorbunov, and Bohan Zhang. *FairBlock: Preventing Blockchain Front-running with Minimal Overheads.* Cryptology ePrint Archive, Paper 2022/1066. 2022. URL: https://eprint.iacr.org/2022/1066.

[MM22] Luzius Meisser and Basile Maire. "Frankencoin." In: (2022). URL: https://www.snb.ch/dam/jcr:2cc1b322-39d7-45ea-a91c-b57dd1e43e43/sem_2022_06_03_maire.n.pdf.

[MR24] Ciamac C. Moallemi and Dan Robinson. *Loss-Versus-Fair: Efficiency of Dutch Auctions on Blockchains.* 2024. arXiv: 2406.00113 [q-fin.TR].

[MT19] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. *Homomorphic Time-Lock Puzzles and Applications.* Cryptology ePrint Archive, Paper 2019/635. 2019. URL: https://eprint.iacr.org/2019/635.

[Nak08] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* 2008. URL: https://bitcoin.org/bitcoin.pdf.

[Qu23] Ziyan Qu. "Rethinking Execution Layer Front-Running Protection with Threshold Encryption." Master's Thesis. KTH Royal Institute of Technology, 2023. URL: https://kth.diva-portal.org/smash/get/diva2:1801732/FULLTEXT01.pdf.

[ran17] randao.org. *Randao: Verifiable Random Number Generation.* 2017. URL: https://randao.org/whitepaper/Randao_v0.85_en.pdf.

[RFC9381] Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Včelák. *Verifiable Random Functions (VRFs).* RFC 9381. Aug. 2023. DOI: 10.17487/RFC9381. URL: https://www.rfc-editor.org/info/rfc9381.

[RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner. *Time-lock puzzles and timed-release Crypto.* 1996. URL: https://people.csail.mit.edu/rivest/pubs/RSW96.pdf.

[Sco05] Michael Scott. "Computing the Tate Pairing." In: *Topics in Cryptology – CT-RSA 2005.* Ed. by Alfred Menezes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 293–304. ISBN: 978-3-540-30574-3.

[SG02] Victor Shoup and Rosario Gennaro. "Securing Threshold Cryptosystems against Chosen Ciphertext Attack." In: *J. Cryptology* 15 (2002), pp. 75–96. DOI: 10.1007/s00145-001-0020-9.

[Shu22] Shutter Network. *Rolling Shutter: MEV Protection Built Into Layer 2.* 2022. URL: https://blog.shutter.network/announcing-rolling-shutter/.

[Sta+21] Chrysoula Stathakopoulou, Signe Rüsch, Marcus Brandenburger, and Marko Vukolić. "Adding Fairness to Order: Preventing Front-Running Attacks in BFT Protocols using TEEs." In: *2021 40th International Symposium on Reliable Distributed Systems (SRDS).* 2021, pp. 34–45. DOI: 10.1109/SRDS53918.2021.00013.

[TSH22] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. "Blockchain Scaling Using Rollups: A Comprehensive Survey." In: *IEEE Access* 10 (2022), pp. 93039–93054. DOI: 10.1109/ACCESS.2022.3200051.

[Vic61] William Vickrey. "Counterspeculation, Auctions, and Competitive Sealed Tenders." In: *The Journal of Finance* 16.1 (1961), pp. 8–37. DOI: https://doi.org/10.1111/j.1540-6261.1961.tb02789.x. URL: https://sci-hub.st/https://doi.org/10.1111/j.1540-6261.1961.tb02789.x.

[Wan22] John Wang. *A Succinct Deconstruction of L2 MEV*. 2022. URL: https://x.com/j0hnwang/status/1489268468771872775 (visited on 06/20/2024).

[Wan23] Shufan Wang. "Execution Layer Based Front-running Protection on Ethereum." Master's Thesis. École Polytechnique Fédérale de Lausanne, 2023. URL: https://www.epfl.ch/labs/dedis/wp-content/uploads/2023/01/report-2022-3-ShufanWang-FrontRunningProtection.pdf.

[Wes18] Benjamin Wesolowski. *Efficient verifiable delay functions*. Cryptology ePrint Archive, Paper 2018/623. 2018. URL: https://eprint.iacr.org/2018/623.

[Woo24] Gavin Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. Paris version 705168a. 2024. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

[Xia+21] Pengcheng Xia, Haoyu Wang, Zhou Yu, Xinyu Liu, Xiapu Luo, and Guoai Xu. *Ethereum Name Service: the Good, the Bad, and the Ugly*. 2021. arXiv: 2104.05185 [cs.CR].

[Zha+24] Haoqian Zhang, Michelle Yeo, Vero Estrada-Galinanes, and Bryan Ford. *ZeroAuction: Zero-Deposit Sealed-bid Auction via Delayed Execution*. Cryptology ePrint Archive, Paper 2024/189. 2024. URL: https://eprint.iacr.org/2024/189.

[Zhu22] Michael Zhu. *On Paper to On-Chain: How Auction Theory Informs Implementations*. 2022. URL: https://a16zcrypto.com/posts/article/how-auction-theory-informs-implementations/.