Cryptographic Approaches to Complete Mempool Privacy

James Stearn
(jbstearn@gmail.com)

September 13, 2022

Abstract

MEV extraction has grown consistently since 2019, bringing with it a broad range of negative externalities and incentivising consensus instability. The majority of current MEV extraction strategies on Ethereum exploit the fact that pending transactions are visible in a public mempool. One proposed approach to minimising MEV is to encrypt transactions and decrypt after transaction order has been finalised by the block proposer. Such an encryption scheme is complicated by a number of factors that are specific to this application; in particular, the requirement that transactions cannot be decrypted *prior* to transaction order finalisation, thereby revealing MEV opportunities. In §1, we present required and desirable properties of an encryption scheme providing mempool privacy. §2 outlines a range of proposed cryptographic primitives that could be used as part of a privacy solution, and considers how these approaches compare in terms of the properties discussed in §1. Finally, we provide a summary of the approaches of §2 and conclude with a proposed direction for current development and future research.

1 Mempool Privacy

1.1 MEV & mempool privacy

Since the publication of *Flash Boys 2.0* in 2019, maximal extractable value (MEV)-related activity on the Ethereum network has grown consistently. Increased MEV-related activity leads to a range of related negative externalities, including higher transaction fees, increased blockspace usage, and network congestion. Such externalities not only result in poor user experience, but form a systemic risk to the network by incentivising miner/validator behaviour that may lead to consensus instability. The scale of MEV extraction is illustrated in Figure 1:

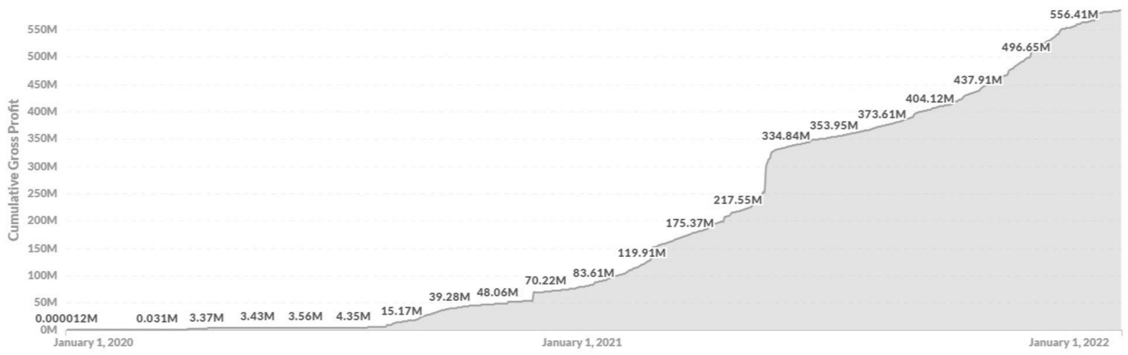

Figure 1: Cumulative Extracted MEV[1]

MEV extraction takes multiple forms, including frontrunning of arbitrage/liquidation opportunities, censoring attacks, and even potential *time-bandit attacks* [24]. MEV extractors scan the mempool in search of profit opportunities and then engage in gas bidding wars to seize the opportunity; this led Robinson & Konstantopoulos to dub the mempool a *'dark forest'* - a treacherous environment in which unsuspecting users are mercilessly exploited for maximal profit [49].

[1]MEV Explore - `https://explore.flashbots.net/`

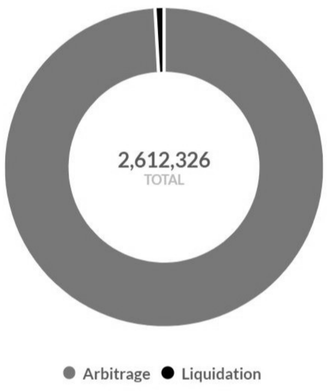

Figure 2: Extracted MEV by type

At present, the vast majority of MEV extraction activity is in the form of frontrunning bots or MEV searchers profiting from arbitrage opportunities, as shown in Figure 2. Such opportunities are found due to the public nature of the Ethereum mempool. A private mempool, in which transaction details are concealed until they have been committed to a block, would therefore mitigate against a broad range of contemporary value-extraction strategies. However, as discussed in [12], even a private mempool would still allow for adversarial value extraction in certain situations, such as auction constructs, in which just the knowledge that a concealed bid/transaction has been submitted could lead to altered strategies for other users. In effect, MEV is an unavoidable consequence of intra-user time-dependent information asymmetry.

Despite this caveat, a mempool providing complete privacy for users would constitute a substantial improvement over the status quo. In its current incarnation, Flashbots offers pre-trade privacy, as transactions pre-block inclusion are visible only to relayers and block builders. Complete privacy from all parties is a core design goal and necessary for a trustless system. A range of cryptographic approaches have been proposed to solve this problem, which we outline in §2.

1.2 Properties of a complete privacy solution

1.2.1 Privacy

A core design requirement for any solution is *complete privacy* - transactions are concealed and cannot be viewed by any intermediaries, such as block proposers or builders, until they have been included on-chain. Complete privacy mitigates against a number of current MEV extraction strategies by concealing the opportunity, including frontrunning such as sandwich attacks. Complete privacy necessarily implies pre-trade privacy and failed trade privacy, which is already provided by Flashbots.

There are two main facets of transaction privacy [23]:

- **Anonymity**: A transaction cannot be linked to its sender or receiver.
- **Confidentiality**: The contents of the transaction is concealed from all intermediaries.

1.2.2 Trustlessness

Ideally, our solution architecture will be trustless - there is no reliance upon a trusted party to set-up or co-ordinate cryptographic protocols. Enforcement of protocol may be through design (e.g. use of commitments within a protocol) or incentivisation mechanisms (e.g. bonded actors). If a trusted setup *is* required, carefully designed precautions, such as the use of multi-party computation protocols, must be taken during setup to ensure (to within some reasonable probability) that no parties retain knowledge that could be used to backdoor the system.

1.2.3 Efficiency

A privacy solution should be efficient in terms of:

- **Blockspace** - encryption should not greatly increase the blockspace required for each transaction. For example, validity proofs committed on-chain should be as small as possible and we require some mechanism for packing blockspace efficiently.
- **Scalability** - The encryption scheme must scale efficiently with both a large number of users and transactions, ideally whilst maintaining lean hardware requirements for encryption and decryption.
- **Implementation** - An ideal solution is efficient to implement (e.g. not requiring L1 protocol changes) whilst maintaining compatibility with network incentives[2] and consensus mechanisms. That is, implementation of encryption should not greatly increase the attack surface for generating consensus instability. This includes mitigating against griefing by any party in the protocol and flooding attacks.

1.2.4 Encrypted Ordering

It would be desirable for any privacy solution to be compatible with application of fair-ordering protocols [36]. We therefore require some mechanism for ordering encrypted transactions *without* revealing transaction metadata, whilst simultaneously maintaining compatibility with network incentives, such as optimising gas fees.

1.2.5 Metadata Leakage

A completely private mempool would ideally conceal *all* data about a transaction, including metadata that may potentially identify an account and therefore engender information asymmetry. A solution requires that transactions with invalid parameters can be invalidated without first being propagated to the entire network and *without* revealing the parameters themselves.

1.2.6 Non-interactive

A solution should be *non-interactive* - no further communication is required from the user once the transaction has been submitted to the network. Communication between other actors in the protocol, such as block proposers and builders, should also be limited, and ideally, non-interactive.

1.2.7 Optional Encryption

Depending on the encryption scheme used, it may be desirable to allow users to determine whether their transaction is encrypted or not (a feature of other privacy-focused blockchains such as Zcash). For example, users may choose not to encrypt their transaction if the gas fees incurred by encryption outweigh the possible savings generated by concealing the transaction. In certain solution architectures, optional encryption will not be possible.

1.2.8 Guaranteed Decryption

We require two related guarantees from our encryption scheme:

- Any party delegated with decryption and following the decryption protocol of the scheme is guaranteed to decrypt the transaction (possibly with some time-delay).
- Any user sending an encrypted transaction to the network is guaranteed to have it decrypted *eventually*, if it is included within a block.

1.2.9 UX

Ideally, a solution will not impact upon the UX of users interacting with the network - sending an encrypted transaction should not incur significant latency (in the form of block delays to inclusion) or greatly increased gas fees. Non-interactive protocols ensure that synchrony is not required once the transaction has been sent. The inclusion of encryption should be seamless from the end user's perspective.

[2]See §1.3 for further on incentivisation within various block proposal paradigms.

1.3 Block proposal paradigms

The block proposal and finalisation mechanism determines the roles and incentives of participants in the network. A solution to mempool privacy must be designed to maintain alignment with network incentives whilst maintaining integrity by limiting novel attack vectors and MEV extraction opportunities. For example, a mechanism in which a single known actor has unrestricted control over block transaction ordering both facilitates and incentivises censorship or re-ordering attacks, along with off-chain collusion or bribery.

1.3.1 Ethereum

In Ethereum 1.0, the block proposer's privileged position enabled both the direct exploitation of MEV opportunities, and engendered further negative externalites through arbitrage bot activity such as PGAs [24]. These outcomes were due to centralisation of the task of transaction ordering within a block. Validators maintain this privilege under PoS. As the transaction ordering process is ostensibly unchanged, MEV extraction strategies should be unaffected by the move to PoS [42]. As Obadia & Vemulapalli outline, as validator duties are assigned randomly, MEV amplifies variance in block reward distributions, producing a centralising force upon the network by incentivising the formation of large validator pools. Larger pools are further advantaged by their greater ability to leverage sophisticated MEV strategies (e.g. cross-domain [43] or multi-block MEV [42]).

1.3.2 Proposer/builder separation

In the proposer/builder separation (PBS) paradigm, the tasks of block building and proposal are separated and tasked to distinct entities.[3] Transaction ordering is delegated to block-builders, who produce blocks with an attached fee for the block proposer. The proposer's task is to select a block and sign it. [18].

PBS provides *pre-confirmation privacy* [19], as the block proposer chooses a bundle based on the header only, ensuring that the proposer cannot frontrun a builder's MEV strategy by observing their selected transactions and submitting its own bundle. In implementing PBS, we need to mitigate against censorship or DoS attacks whilst ensuring that both proposers and builders' incentives are aligned to maintain blockchain integrity.

1.3.3 MEV-Boost

Post-merge, MEV-Boost provides a permissioned form of PBS in which the relayer propagates block headers to the validators, with an additional escrow role to provide redundancy of data availability [32]. Complete privacy implies that transactions included in searchers' bundles are concealed until they have been mined. No intermediaries such as relayers or miners can see the content of a bundle prior to its inclusion on-chain. Note that transactions may still be visible in the mempool prior to inclusion in a bundle.

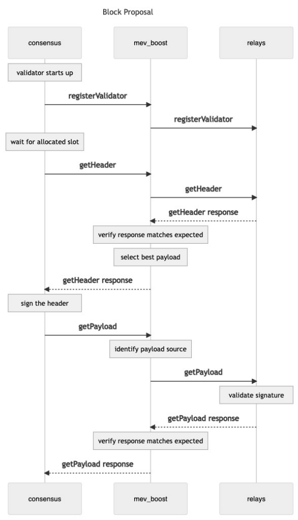

Figure 3: MEV-Boost architecture

[3]In principle, the roles of proposer and builder are distinct, but in practice, the same party may undertake both roles for a given block.

This implementation conceals MEV opportunities from the validators at the expense of an increase in block proposal latency. However, the relayers are still trusted and have full access to transactions within a block, which perpetuates a vector for collusion, bribery, or censorship. MEV opportunities remain visible in this paradigm, although pre-trade privacy is provided.

1.4 Consensus Layer vs Application Layer MEV

In the context of a given block proposal paradigm, we may consider MEV extraction strategies at both the consensus and application layers. Post-merge, at the consensus layer, a range of strategies may be employed that exploit prior knowledge of the block proposer.

Some proposed approaches to MEV minimisation at the consensus layer include:

- *single secret leader elections* (SSLE) [11], which mitigate against time-buying attacks and proposer DoS attacks [20].
- *single-slot finality* [21], to mitigate against chain reorgs for MEV extraction (*time-bandit attacks*) [35].

In §2, we we will focus on cryptographic approaches to MEV mitigation at the application layer, either through encrypted transactions or bundles. By concealing transactions, a broad range of currently employed MEV strategies can be neutralised.

2 Cryptographic Approaches to Mempool Privacy

2.1 Multi-party Computation

2.1.1 Introduction

Multi-party computation (MPC) protocols allow a set of distinct parties to compute joint computation of a function without revealing their private inputs. Each party therefore learns their respective output and nothing else [37]. A large number of protocols have been designed to implement MPC, satisfying a wide variety of security requirements. All protocols require robustness in the presence of adversaries that control a subset of corrupted parties.

Definition 1. Desired security properties of an MPC protocol include: [37]

1. **Privacy:** Each party learns only their respective output.
2. **Correctness:** Each party's respective output is guaranteed to be correct.
3. **Guaranteed Output:** Honest parties are guaranteed their output, even in the face of adversaries attempting to disrupt the protocol.
4. **Fairness:** Corrupted parties only receive outputs if honest parties receive theirs.

There are a number of models that delineate the range of adversarial behaviour and corrupted parties. For blockchain applications, we must consider *malicious adversaries* and an *adapative corruption* model, in which previously honest parties may be corrupted *during* computation. This accurately captures the possibility of an attacker hacking MPC parties and proceeding to arbitrarily deviate from the protocol.

In the case where the number of corrupted parties is unbounded, secure MPC can be achieved, but without fairness or guaranteed output [22]. These guarantees can be retained in the covert adversaries model, in which a misbehaving adversary is detected with some fixed probability ϵ. These protocols can be enhanced with *public verifiability*, such that honest parties can prove misbehaviour by adversaries [51]. Such a model could apply in a setting in which MPC parties are bonded, for example.

2.1.2 Mempool privacy using MPC

By applying an MPC protocol, HashCloak have implemented scoring bundles whilst maintaining transaction privacy [34].

One technical issue with using an MPC protocol for mempool privacy is the requirement to check for transaction or bundle validity. Without this validity check, an attacker could spam the network with invalid transactions as inputs to the protocol. One potential solution is to generate a zero-knowledge proof of bundle validity against the current state, which requires a fully implemented zk-EVM. A further issue is the possibility of a malicious majority targeting the protocol, thereby not allowing for guaranteed output.

2.2 Verifiable Delay Functions

2.2.1 Introduction

A *Verifiable Delay Function* (VDF), defined by Boneh *et al.* in [7], is a function for which computing an output takes a pre-determined amount of wall-clock time. This output is efficiently verifiable, so verification is asymptotically faster than evaluation. Parallelisation of the computation of a VDF's output will not decrease the time taken to evaluate an output.

Fast verification distinguishes VDFs from other 'slow' functions such as time-lock puzzles [48], which require time-consuming naive re-evaluation of the function to verify an output. VDFs' parametrised time delay and fast verification allow for a number of useful applications in decentralised systems, such as non-manipulable randomness beacons and computational timestamping [7].

Definition 2. More formally, a *VDF **V*** is a triple of algorithms *(Setup, Eval, Verify)* [7]:

- *Setup*$(\lambda, t) \rightarrow (ek, vk)$: with λ a security parameter and t the desired puzzle difficulty producing public parameters ek: an evaluation key, and vk: a verification key.
- *Eval*$(ek, x) \rightarrow (y, \pi)$: an algorithm with input x and output y. π is an optional proof which may be used to verify y.
- *Verify*$(vk, x, y, \pi) \rightarrow \{Yes/No\}$: a deterministic algorithm which verifies that $f(x) = y$.

For useful time-delay and efficient verification, we require that *Eval* runs in time *t* and *Verify* runs in polylog*(t)* time. We further require that ***V*** satisfies the following properties:

1. **Correctness:** *Verify* correctly outputs Yes with inputs (vk, x, y, π) if $f(x) = y$.
2. **Soundness:** The probability that *Verify* outputs Yes with inputs (vk, x, y, π) if $f(x) \neq y$ is negligible.
3. **Sequentiality:** An algorithm running in time $(\sigma)t < t$ can compute y correctly with only negligible probability, even with a high degree of pre-computation and bounded parallelisation, up to bound $poly(\lambda)$ [8].

2.2.2 VDF implementations

The time-lock construction of Rivest *et al.* [48] satisfies the three conditions above, but lacks the efficient verifiability required of a VDF. Taking this scheme as a starting point, Wesolowski [56] and Pietrzak [46] propose two similar practical implementations of a VDF as defined in [7]. We outline Pietrzak's scheme below, noting that both schemes have their advantages and drawbacks relating to their respective security assumptions; for further details see [8].

We begin with the time-lock puzzle of [48]:

- Alice generates a *puzzle* (N, x, T): an RSA modulus N, a random $x \in \mathbb{Z}_N^*$ and time parameter $T \in \mathbb{N}$.
- The puzzle's *solution* is $y = x^{2^T}\ mod\ N$ [46]

Note that knowledge of the generating primes of the modulus allow efficient calculation of y. Without this knowledge, y may only be calculated by T sequential squarings of x, satisfying condition 3 of Definition 2.

Thus, without knowledge of the group order, the only way to verify a computation of $y = x^{2^T} \, mod \, N$ is to repeat the T sequential squarings. As knowing the group order is equivalent to factorising the modulus, the puzzle is conjectured to be secure.

We therefore require a more efficient way of verifying an output y. Pietrzak outlines an interactive iterative halving protocol between a verifier*V* and prover *P*, which can be made non-interactive by applying the Fiat-Shamir heuristic. The verifier needs no privileged knowledge, so the result is publicly verifiable.

In constructing the VDF, the security parameter λ determines the required bitlength of the generated RSA modulus. *Setup* generates an abelian group $\mathbb{G}$ of unknown order that allows for efficient element group membership checking.

Suppose *P* wants to prove he has calculated $y = x^{2^T}$ by computing T sequential squarings in $\mathbb{G}$. They submit y to the verifier *V*.

- *V* checks that $y, x \in \mathbb{G}$. If $T = 1$, they output *Yes* if $y = x^2$, *No* otherwise.
- If $T > 1$, *P* computes $\mu = x^{2^{T/2}}$. *V* checks $\mu \in \mathbb{G}$ and sends *P* a random $r \in \mathbb{Z}_{2^\lambda}$.
- Both *V* & *P* calculate: $x' = x^r \cdot \mu$ and $y' = \mu^r \cdot y$.

If $y = x^{2^T}$ then $y' = x'^{2^{T/2}}$, so the protocol is now iterated until this can be verified directly by *V*. For T a power of 2, the protocol thus takes $log_2 T$ rounds [8].

More recent VDF implementations by De Feo *et al* utilise bilinear pairings [26]. Burdges & De Feo introduce the concept of Delay Encryption, in which the Identity-Based Encryption (IBE) scheme of Boneh & Franklin [10] is modified to include a time-delayed private key extraction phase.

2.2.3 Mempool privacy using VDFs

To provide mempool privacy, a VDF can be employed as part of a time-delayed commitment scheme in which transactions are committed to, and may only be read by a miner once the desired amount of wall-clock time has elapsed. In this model, the security provided by our choice of parameter *t* is proportional to the transaction confirmation delay experienced by the end user. For example, if *t* is too small, miners can compute *Eval* and decrypt the transaction whilst it is still in the mempool, thereby allowing MEV opportunities to be uncovered. Conversely, a large value for *t* necessarily imposes a significant delay on transaction confirmation and correspondingly, poor UX.

One particular disadvantage of Pietrzak's construction is the reliance on a *trusted setup*. For use in decentralised trustless applications such as blockchains, this is a significant drawback as it allows for the possibility of a backdoor. Wesolowski's VDF avoids this requirement, at the expense of stronger security assumptions: namely, that finding an element of low order in the class group of an imaginary quadratic extension is hard, a problem which has not been studied as extensively as the RSA problem.

A distinct disadvantage of the VDF approach to mempool privacy is its scalability. As the number of users grows, significant computational resources will need to be expended to compute each individual function and verify outputs. In a Delay Encryption implementation, encryption is based on a unique *session identifier* [15] which is changed periodically. This makes Delay Encryption a practical primitive for application in the Ethereum epoch model. As it is a modification of Boneh-Franklin IBE, the Delay Encryption protocol of [15] requires a *trusted setup*. This issue can be mitigated by utilising multi-party computation for distributed parameter generation.

2.3 Multi-Party Timed Commitments

2.3.1 Introduction

Multi-Party Timed Commitments (MPTC) [27] utilise time-lock puzzles in a similar way to VDFs to introduce a time-delay, but add further properties that make them well-suited to use in decentralised systems. MPTC may be used in situations in which multiple users must commit to values which they want to keep hidden until all other users in the scheme have committed their respective values; such a scheme has obvious application to Flashbots Auction. Unlike other trustless commitment schemes, such as various multi-party computation protocols, MPTC has only minimal threshold requirements for honest actors.

Definition 3. *MPTC* is defined by a tuple *(commit, reveal, verify)* for a set of N participants $P = \{P_1, P_2, ..., P_N\}$ and a coordinator providing: [27]

1. **Soundness:** The coordinator outputs a message m^i for participant P_i only if P_i committed to m^i during the *commit* phase.
2. **Front-running resistance:** The protocol is resistant to message ordering and censorship attacks - i.e. an adversary cannot change the order of messages or add/remove messages after the end of the *commit* phase.
3. **Time-lock:** If at least one $P_i \in P$ is an honest actor, the time taken to complete the *reveal* phase and reveal the committed message order and contents has mean τ, with τ a time parameter dependent on a chosen security parameter λ.
4. **Ensured output:** A coordinator following the protocol is guaranteed to produce a valid output after mean time τ.

2.3.2 MPTC implementation

We now describe Doweck and Eyal's [27] implementation of MPTC through their *Time Capsule* algorithm:

- The *commit* phase is a multi-stage protocol in which each participant P_i first sends a random nonce to the coordinator, who creates a Merkle tree of the nonces once all N have been received. The coordinator now sends the Merkle root as a random seed which is used by each P_i to generate a shared ElGamal public-key tuple $Pk = (p, g, b)$ using a deterministic algorithm.

  The coordinator will decrypt messages encrypted using this Pk by solving a discrete log problem via brute-force search for an a such that $b = g^a \, mod \, p$. The value of τ is therefore tunable relative to the network's computational resources by our choice of p.

  P_i now generates a commitment c_i by hashing their encrypted message m_i concatenated with salt r_i, and sends this hash to the coordinator. The coordinator places all N commitments in an ordered list L and sends this list to each P_i, who verify that their c_i is in the list. As message order is now determined, P_i sends a signed encrypted c_i, which the coordinator may verify by hashing.

- The coordinator now completes the *reveal* phase by solving the discrete log problem to recover the private key Sk and decrypt the c_i. Note that this search may begin before all values have been committed, taking mean time τ.

- Once Sk has been found, the *verify* phase proceeds with the coordinator publishing the decrypted commitments, Sk, and the signature list. Any P_i can now verify the signatures and committed messages.

For proof that this implementation satisfies the four conditions of Definition 3, see §5 of [27].

2.3.3 Mempool privacy using MPTC

One particular advantage of MPTC is that the time parameter τ is independent of the number of participants N. It therefore scales well to large sets P of participants, as required for use in a blockchain setting. With a Proof of Stake consensus mechanism, a natural choice for the coordinator is the validator, which would allow for a streamlined transaction order commitment/block verification protocol.

However, the implementation of the *commit* phase is interactive and therefore introduces possible synchrony issues and griefing attack vectors. A further difficulty with implementing the protocol is the issue of determining membership in the set P and the inefficiency of encrypting transactions which may not be included within a particular validator's block due to limited blockspace. This latter issue may be resolved by separation of the roles of coordinator in the MPTC protocol and block validator.

In the protocol, the coordinator is privileged as they determine the order of the encrypted transactions. Although the contents of each transaction is concealed, this opens up a possible vector for novel and unforeseen MEV strategies, as discussed in [12].

As the security mechanism is time-lock based, a coordinator may hold the position until all their corresponding m_i have been decrypted and subsequently verified/rejected by validators. Further, any party can decrypt the messages by independently completing the reveal phase. This flexibility allows for more complicated protocols in which the bulk of the computation can be outsourced to specialised entities.

In the context of Flashbots Auction, for complete privacy, we require that no intermediaries have privileged access to transaction data pre-block validation. As such, we cannot directly use MPTC with relayers as coordinators, as this would privilege miners and allow transaction reordering.

2.4 Threshold Encryption

2.4.1 Introduction

One approach that may resolve the issue of deadweight loss caused by encrypted transactions that are not included in a particular epoch is threshold encryption. A threshold encryption scheme consists of a sender and a set of n receivers, of which at least t must cooperate to decrypt. In an ideal scheme for blockchain implementation, both the encryption and decryption phases would be non-interactive and the set of receivers may be chosen ad-hoc. Daza *et al.* call such a scheme *threshold broadcast encryption* (TBE) [25].

The basic idea is that a transaction is encrypted, with the set of n receivers chosen upon encryption - the validator committee is a natural choice for the set of receivers. Once an ordered list of transactions[4] has been committed to (e.g. through an on-chain commitment), a subset of t receivers compute verifiable partial decryptions (*'decryption shares'*). These partial decryptions are combined to obtain a complete decryption of the original transaction, which can then be executed. By employing a threshold scheme, the user's issue of non-inclusion by one particular proposer can be mitigated, as any validator within the epoch may decrypt a transaction in cooperation with $(t-1)$ others.

Definition 4. A threshold public key encryption system consists of five algorithms: [9]

- *Setup*$(n, k, \Lambda) \to (PK, VK, SK)$: with n players, k the threshold and Λ a security parameter. The setup algorithm outputs a public key PK, verification key VK and a vector of n private key shares $SK = (SK_1, \ldots SK_n)$.
- *Encrypt*$(PK, M) \to C$: Encrypts a message M with using public key PK.
- *ShareDecrypt*$(PK, i, SK_i, C) \to \mu$: Outputs a decryption share $\mu = (i, \hat{\mu})$ for player i.
- *ShareVerify*$(PK, VK, C, \mu) \to \{Valid, Invalid\}$: Verifies that a decryption share μ is valid.
- *Combine*$(PK, VK, C, \{\mu_1, \ldots \mu_k\}) \to M$: Takes k decryption shares and outputs a cleartext M.

2.4.2 Threshold Encryption implementation

An implementation of threshold encryption for the purposes of mempool privacy consists of an initial distributed key generation (DKG) protocol. Such a protocol allows a bonded committee to trustlessly generate the keypair of a public-key cryptosystem without reliance on a privileged dealer to generate and distribute the secret key, as in variants of Shamir's secret sharing [53]. Schindler *et al* provide a practical implementation of a DKG scheme using Ethereum smart contracts in [50].

Users encrypt a message using the master public key that may only be decrypted by a set of t members of the committee collaborating to recover the master secret key. One issue with this implementation is the possibility of collusion between committee members; we therefore some require some notion of co-ordinated information release (e.g. decryption shares). Stengele *et al* adapt the protocol of [50] to accommodate scheduled secret key reconstruction.

[4]The block order proposer here will depend on the block proposal paradigm employed: either a dedicated proposer or a miner/validator.

The essential idea is that the encryption committee generates an ElGamal public key and an *(n,n)*-sharing of the corresponding secret key. That is, *all* parties must collaborate to recover the key. Through a partial information exchange protocol, the members of the committee now generate a *(t,n)*-sharing of the secret key using Verifiable Secret Sharing [45]. Disputes at the verification stage are completed in a single transaction using a non-interactive zero-knowledge proof (NIZK), as in [50]. Members breaking protocol are publicly verifiable and can be slashed and disqualified from the remaining protocol.

Users now encrypt their transactions using the published ElGamal public key. To achieve a scheduled reconstruction of the secret key, committee members are incentivised to provide their decryption shares at a pre-defined time, parametrised by block height. Once at least *t* committee members submit their decryption shares, the master secret key is recoverable and transactions can be decrypted by any party, including other users. Shutter employ a similar scheme to provide opt-in transaction encryption via a DAO tasked with selecting the encryption committee [54].

2.4.3 Mempool privacy using threshold encryption

As discussed, a threshold encryption scheme integrates well with a validator committee paradigm, as the decryption phase can be consolidated with the consensus mechanism. The cryptographic security of the scheme relies on robust, well-studied cryptographic primitives and scales well, e.g. for use in layer 2 solutions. However, a threshold scheme requires well-designed incentivisation to ensure that a quorum of participants are well-behaved. Incentives mitigate against adversarial behaviour such as off-chain collusion or decryptors griefing by not providing decryption shares. Such mechanisms are feasible in the validator committee paradigm. However, although unlikely with larger committees, off-chain collusion between a quorum of decryptors and a block proposer could allow for pre-finalisation decryption and consequent MEV extraction that would be difficult to both detect and punish, as identifying the contravening entities is non-trivial

Requiring an on-chain commitment prior to transaction decryption introduces a 1-block execution delay. This negatively impacts UX and introduces price-discovery issues if threshold encryption is not employed on all transactions in the network. However, in principle, applying threshold encryption is opt-in and can be applied only to certain classes of transaction if desired. Such flexibility is advantageous to users and developers. Smart contract implementations of threshold encryption also impose increased gas fees for the user, making optional encryption desirable.

2.5 Zero-Knowledge Proofs

2.5.1 Introduction

Zero-knowledge (ZK) proofs allow the correctness of a computation to be efficiently verified via a succinct proof *without* revealing any information about the inputs of the computation itself. Currently, the majority of ZK development is focused on leveraging succinctness properties to scale blockchains, but an increasing number of privacy focused projects are also deploying ZK proofs. If we think of a transaction as a function f with input x that induces some output state y, a ZK proof allows us to prove that we know some valid x such that $f(x) = y$ for some y, without having to reveal x itself [16]. Although ZK proofs have been utilised by a number of projects to e.g. preserve privacy in the UTXO model, implementing fully private smart contracts is a challenging problem. Approaches such as Zether [14] provide privacy at the expense of restricted functionalities.

Using a zero-knowledge proof, a *prover* proves some statement to a *verifier*. For blockchain applications, we also require that the proof be publically verifiable by any interested party. Two main varieties of ZK proofs - SNARKS & STARKS - are currently utilised for blockchains, each with their respective advantages and disadvantages. We discuss each variety below, and consider how they may be applied to the problem of mempool privacy.

2.5.2 zk-SNARKS

A zero-knowledge **S**uccinct **N**on-Interactive **Ar**gument of **K**nowledge (zk-SNARK) has the following properties: [47]

1. **Zero-Knowledge:** The prover does not reveal anything about their statement, beyond its validity, to the verifier.

2. **Succinctness:** The proof is exponentially smaller than the size of the computation itself.
3. **Non-interactive:** The prover does not need to interact with the verifier to prove a statement.
4. **Argument:** As distinct from a proof of knowledge, the protocol is protected only against computationally limited attackers.
5. **Knowledge:** The prover can construct a valid proof of a particular statement only if they have knowledge of a corresponding *witness*.

2.5.3 zk-SNARK implementation

SNARKs are a family of proof systems that attest to computational integrity (CI). [41] provides a summary of the recent developments in SNARK technology. We outline the approach used in Pinocchio, Groth16 [33] and other QSP-based SNARKs, which have small proof sizes and efficient verifier runtime.

To prove the correctness of a given computation using a SNARK, we first perform arithmetisation, which translates the computation into a special polynomial form. The very broad intuition is that arithmetisation allows us to verify a computation by checking for equality between polynomials, rather than having to check each step of the computation itself.

The prover wants to demonstrate knowledge of a particular *target* polynomial which satisfies the equality. To do this succintly, we employ a protocol in which the prover commits to their polynomial, and the verifier randomly selects a point to be evaluated. This exploits the Schwartz-Zippel lemma [52], which implies that the probability of an incorrect polyomial evaluating to the target polynomial's value at the chosen point is negligible. The critical point is that the prover does not know which point the verifier will choose to test before making their commitment. By only needing to test a single point, verification is extremely efficient (Groth16 proofs are under 300 bytes, independently of the size of the circuit.) [44]

The first step is to encode the computation that is to be checked into a special polynomial form. By Theorem 10 of Genarro *et al.*'s paper [30], we are able to construct this form for any arithmetic circuit. For a detailed account of this encoding process, see [30] or Buterin's more accessible article [16]. For our purposes, it is sufficient to know that the prover wants to show that they have knowledge of some *satisfying* assignment.

Definition 5. *An assignment* $(c_1, \dots, c_m)$ *satisfies* Q if T divides P, for $P = L \cdot R - O$, where $L = \sum_{i=1}^{m} c_i \cdot L_i, R = \sum_{i=1}^{m} c_i \cdot R_i, O = \sum_{i=1}^{m} c_i \cdot O_i$ [28]

In order for the verifier to check the prover's assignment, without revealing it directly, we employ an encoding function E with homomorphic properties. This can be thought of as a special type of one-way function which supports linear combinations and multiplication, i.e. given $E(x)$ and $E(y)$ we can compute a linear combination $E(ax + by)$ and $E(xy)$, without knowing x and y. This allows the prover to demonstrate knowledge of a satisfying assignment by committing to polynomials L, R, O, U where $P = U \cdot T$. For efficient verification, these polynomials are combined into a single polynomial F by addition.

The verifier can now test that the polynomials satisfy the polynomial equality at a secret random point $s \in \mathbb{F}_p$. To achieve the non-interactive property of a SNARK, the verifier first performs a setup phase and shares a *Structured Reference String* (SRS):

$$E(s^0), E(s^1), \dots, E(s^d)$$

The element s must then be destroyed, as knowledge of s allows construction of false proofs. The prover can now evaluate their (encoded) polynomial at s, calculating $E(F(s))$ as it is a linear combinations of the elements of the SRS, and the encoding E supports linear combinations [47]. To verify this calculation without knowledge of s, the verifier chooses another random element $\alpha \in \mathbb{F}_p$ and also publishes:

$$E(\alpha s^0), E(\alpha s^1), \dots, E(\alpha s^d)$$

The prover similarly calculates $E(\alpha F(s))$. The verifier checks for equality of the prover's published values using a pairing:

Definition 6. For cyclic groups $\mathbb{G}, \mathbb{G}_1$ with generator $g \in \mathbb{G}$, a pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ is a map with the properties: [9]

1. **Bilinearity**: $\forall u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$
2. **Non-degeneracy**: $e(g, g) \neq 1$

Using this pairing function, the verifier checks that the following holds:

$$e[E(F(s)), g^\alpha] = e[E(\alpha F(s)), g]$$

Using this pairing function, the verifier can check that the prover did in fact evaluate their polynomial at s. We require some further values to be published as part of the SRS to ensure that the correct polynomials are evaluated by the prover, but this is the general idea. The critical point is that the prover can generate all required encrypted values using only elements published in the SRS. The verifier finally verifies that the prover's published values satisfy the QAP conditions using the pairing function.

One further technical point - to make the SNARK zero-knowledge, the prover does not want to reveal any information about their polynomials. To achieve this, the prover adds a random δ-shift to their published values. These shifts will not affect the equalities checked for by the verifier using the pairing function due to the homomorphic properties of the encryption function, but they conceal the prover's witness. In practice we also wish to generate a proof non-interactively. To do this, we use the Fiat-Shamir heuristic and hashes of our data for randomness.

Groth16 benefits from constant proof size (3 group elements) and efficient verification at the expense of a trusted setup. More recent SNARK constructions which do not require trusted setups, such as Halo and Aurora, achieve trustlessness at the expense of an increase in proof size, as outlined in table 1. [39]

SNARK	Proof Size	Trusted Setup?
Groth16	0.2kb	✓
PLONK	1kb	✓
Sonic	2kb	✓
Halo	20kb	✗
Fractal	250kb	✗

Table 1: SNARK proof sizes

2.5.4 Mempool privacy using zk-SNARKs

The properties of zk-SNARKs make them potentially useful in achieving the goal of mempool privacy. However, despite their many advantages, there are a number of technical issues with implementing SNARKs on Ethereum. Ethereum's modularity means we require a new SRS (and corresponding trusted setup) for each program. General-purpose updateable SNARKs such as Sonic and PLONK allow for arbitrary computation, at the expense of verification efficiency and proof size when benchmarked to Groth16. The verifier's task of computing pairing functions is computationally intensive and expensive given Ethereum's gas limitations.

Despite these technical issues, zk-SNARKs may still be useful as part of a modular architecture to achieve complete mempool privacy. Through optimisations such as precompiled contracts that implement pre-defined pairings on certain elliptic curves, it may be feasible to implement zk-SNARKs for dedicated tasks such as solving meta-data leakage issues by proving account balance validity for a transaction without first requiring it be gossiped to the network.

An important philosophical issue in the current implementations of zk-SNARKs is the implicit trust required from the user due to the trusted setup generating the SRS. Essentially, the user must trust that developers are honest and do not retain knowledge of the secret values (so-called *toxic waste*) used to generate the SRS. See ZCash's elaborate parameter generation ceremony [57] for further on this issue. In practice, this is resolved by generating parameters using a permissionless, updatable MPC.

More recent SNARK developments that do not require trusted setups rely on newer cryptographic primitives which have not had the security benefits of years of research. Transparency (lack of trusted setup) also comes at the expense of greatly increased proof sizes, meaning higher blockspace requirements for proofs. This problem leads us to zk-STARKs, which retain many of the innovative features of SNARKs, but dispense with the requirement of a trusted setup and utilise well-studied, robust cryptographic assumptions.

2.5.5 zk-STARKs

2.5.6 Introduction

Similarly to a zk-SNARK, a zero-knowledge **S**calable **T**ransparent **Ar**gument of **K**nowledge (zk-STARK) has the following properties: [5]

1. **Zero-Knowledge:** The prover does not reveal anything about their statement, beyond its validity, to the verifier.
2. **Scalable:** The prover's overhead for generating a proof is relatively low (i.e. generating a proof of the integrity of a computation does not take much longer than performing the computation itself). Verification is (poly)logarithmic in the computation time (i.e. verification is exponentially faster than performing the computation itself).
3. **Transparent:** All verifier messages are public - there is no trusted setup required or toxic waste generated.
4. **Argument:** As distinct from a proof of knowledge, the protocol is protected only against computationally limited attackers.
5. **Knowledge:** The prover can construct a valid proof of a particular statement only if they have knowledge of a corresponding *witness*.

2.5.7 zk-STARK implementation

The goal in implementing a STARK is ostensibly the same as for a SNARK: a prover can efficiently generate a proof that attests to the correctness of some computation C to a verifier, with verification poly-logarithmic in the runtime of C. We now outline the principal elements of a STARK as described in Ben-Sasson *et al.*'s paper [5].

As with a SNARK, the initial step is to reduce our computational statement to a statement about polynomials via arithmetisation. Using the parameters of the computation, we can define polynomial constraints on the *execution trace*, which is effectively an array representing the state of the computation at each step C_i. These polynomial constraints are satisfied if and only if the execution trace is valid. In generating these constraints, we require that they can be verified *locally* (that is, each row can be verified by checking nearby rows only).

When making this reduction, the constraints are defined for polynomials over some finite field $\mathbb{F}$ with a particular algebraic structure. By expressing the elements of the execution trace as powers g^i of a generator g of some subgroup $G < F$, we can exploit this algebraic structure to construct a polynomial with evaluations *'close'* to some low-degree polynomial on all but the i values represented by the g^i. See §B.2 of [5] for details on this construction.

The prover's task is therefore reduced to one of proving knowledge of some low-degree polynomial. The verifier's task is then to verify that this polynomial *is* of low-degree at randomly chosen points. STARKs use an innovative protocol, FRI[5], based on error correction techniques to verify the low-degree of the polynomial. Through only a limited number of queries, the verifier can be convinced that the prover's polynomial is in fact of low-degree. As proving this conclusively would require checking the polynomial across its entire domain, we in fact construct a type of *probabilistically checkable proof* (PCP) with negligible error probability [2]. If a prover committed to an incorrect polynomial, they are overwhelmingly likely to fail the verifier's low-degree checks. FRI achieves linear prover complexity and *logarithmic* verifier complexity [6] - thereby attaining the scalable property of STARKs.

To achieve logarithmic verification, the prover now expresses their polynomial f as a linear combination of two polynomials P_0, P_1 with degree $< \frac{d}{2}$: [6]

[5]Fast Reed-Solomon Interactive Oracle Proof of Proximity.

$$f(x) = P_0(x^2) + x \cdot P_1(x^2)$$

The prover commits, via a Merkle tree, to evaluation of the polynomial at a random value provided by the verifier. By iterating this protocol, the polynomial's degree is repeatedly halved, until we reach a constant polynomial which the verifier can check directly. The verifier must therefore check that:

1. The prover's committed polynomial satisfies the low-degree test.
2. The prover did in fact use the verifier's values to evaluate the polynomial.

As with a zk-SNARK, this interactive protocol can be made non-interactive by applying the Fiat-Shamir heuristic and using a hash of the Merkle root for pre-sampling.

2.5.8 Mempool privacy using zk-STARKs

A distinguishing feature of STARKs is their minimal cryptographic assumptions, utilising well-studied, robust cryptographic hash functions as opposed to pairings. As stated, STARKs do not require a trusted setup but their scalability and transparency comes at the cost of larger proof sizes compared to SNARKs ($\sim$45kb vs $<$300B) [40]. It therefore makes sense to use STARKs for proving extremely large computations to amortise the larger blockspace cost of the proof. A particular technical advantage of STARKs is that by relying on Merkle trees for commitments, arithmetisation can be over any finite domain, allowing for faster proving and verification. The relatively larger proof size of STARKs would lead to increased blockspace usage and higher gas fees if used as part of a privacy solution.

2.6 Witness Encryption

2.6.1 Introduction

Witness Encryption schemes are powerful cryptographic tools that allow an encrypted message to be decrypted by anyone who knows a *witness* to some NP relation. This is in contrast to public-key schemes, in which a message is encrypted for a pre-determined decryptor (or set of decryptors). This flexibility makes witness encryption potentially a very useful primitive for blockchain applications, allowing for non-interactive decryption of encrypted messages. Such a property allows for guaranteed decryption given the satisfying of some defined condition, allowing the construction of modular encryption schemes. For example, witness encryption can theoretically be used to construct a system combining a threshold scheme with delay encryption parameters.

Definition 7. A *witness encryption* scheme for an NP language L consists of two algorithms: [29]

- *Encrypt*$(1^\lambda, x, m) \rightarrow (c)$: with inputs a security parameter λ, an instance x, and message m, the algorithm outputs a ciphertext c.
- *Decrypt*$(c, w) \rightarrow m$: With w a corresponding witness to the instance x, the algorithm outputs the message m.

2.6.2 Witness encryption implementation

Garg *et al.* in [29] introduced both the formal definition of witness encryption above and a range of applications for the primitive, including building public-key encryption schemes with lightweight key generation algorithms. They also proposed a candidate construction utilising graded encoding systems, which are impractical to implement for blockchain purposes. A number of attacks on these systems have been found, summarised in [1].

Bartusek *et al.* introduce a candidate scheme based on *affine determinant programs* (ADPs) [4], which are conceptually simpler to work with but still produce impractically large ciphertexts when used to construct a public-key encryption scheme. Witness encryption is therefore an extremely promising primitive which requires further research and development to be efficiently implementable for blockchain uses.

3 Summary of Cryptographic Approaches to Mempool Privacy

We summarise the cryptographic approaches discussed in §2, with respect to the design properties outlined in §1.2. Note that the efficiency of each approach does not take into account the practical complexity of implementation.

	MEV Security	**Trustlessness**	**Efficiency**	**Optional encryption?**	**UX**
MPC	Malicious majority can grief	✓	Increased communication overhead	✓	Small latency increase
VDF/MPTC	Trade-off with UX	✓	Efficient blockspace usage Less efficient scalability	✗	Higher confirmation latency
Threshold	MEV discovery possible for colluding quorum	✓	Efficient scalability Small increase in blockspace usage	✓	Small increase in gas fees Negligible latency increase
Zero-knowledge proofs	Strong MEV security	✓(STARKs) ✗ Trusted setup (SNARKs)	Increased blockspace usage Implementation challenges to arbitrary computation	?	Higher gas fees Small latency increase

Table 2: Summary of Cryptographic Approaches

3.1 Conclusion

Complete privacy is a critical design goal towards constructing a trustless, transparent and fair MEV extraction ecosystem. Cryptographic approaches can minimise trust assumptions across the range of participants in the MEV ecosystem. As outlined, there are a number of theoretical and practical challenges to implementation of complete privacy for searchers, builders, and end users. Each of the methods outlined in §2 exist along multiple trade-off axes.

In terms of practical implementation, threshold schemes arguably represent the most efficient solution at present. There exist efficient DKG and threshold encryption schemes that allow for practical encryption with only small latency penalties. These benefits are achievable with an honest majority trust assumption, with the downside that collusion amongst decryptors is hard to police. Implementation of such a scheme requires careful consideration of incentives of network participants, although much of this has been completed in prior works.

Although currently impractical, witness encryption represents perhaps the most streamlined solution to achieving mempool privacy. At present, the proposed schemes are not efficient enough for this application and their security relies on novel hardness assumptions that have not been as thoroughly researched as for more established primitives. We propose further security analysis of existing witness encryption schemes and development of more efficient schemes as a direction for further research.

Acknowledgements

Thanks to Alejo Salles, Yael Doweck, Justin Drake, Alex Obadia, Michal Zajac & Jonathan Passerat-Palmbach for discussion, clarifications and feedback.

References

[1] Albrecht, M. R. (2018). Are Graded Encoding Schemes broken yet? malb.io. https://malb.io/are-graded-encoding-schemes-broken-yet.html

[2] Arora, S., Lund, C., Motwani, R., Sudan, M., & Szegedy, M. (1998). Proof verification and the hardness of approximation problems. Journal of the ACM (JACM), 45(3), 501-555.

[3] Aumann, Y., & Lindell, Y. (2010). Security against covert adversaries: Efficient protocols for realistic adversaries. Journal of Cryptology, 23(2), 281-343.

[4] Bartusek, J., Ishai, Y., Jain, A., Ma, F., Sahai, A., & Zhandry, M. (2020). Affine determinant programs: a framework for obfuscation and witness encryption. In 11th Innovations in Theoretical Computer Science Conference.

[5] Ben-Sasson, E., Bentov, I., Horesh, Y., & Riabzev, M. (2018). Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive.

[6] Ben-Sasson, E., Bentov, I., Horesh, Y., & Riabzev, M. (2018). Fast reed-solomon interactive oracle proofs of proximity. In 45th international colloquium on automata, languages, and programming (icalp 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[7] Boneh, D., Bonneau, J., Bünz, B., & Fisch, B. (2018, August). Verifiable delay functions. In Annual international cryptology conference (pp. 757-788). Springer, Cham.

[8] Boneh, D., Bünz, B., & Fisch, B. (2018). A Survey of Two Verifiable Delay Functions. IACR Cryptol. ePrint Arch., 2018, 712.

[9] Boneh, D., Boyen, X., & Halevi, S. (2006, February). Chosen ciphertext secure public key threshold encryption without random oracles. In Cryptographers' Track at the RSA Conference (pp. 226-243). Springer, Berlin, Heidelberg.

[10] Boneh, D., & Franklin, M. (2001, August). Identity-based encryption from the Weil pairing. In Annual international cryptology conference (pp. 213-229). Springer, Berlin, Heidelberg.

[11] Boneh, D., Eskandarian, S., Hanzlik, L., & Greco, N. (2020, October). Single secret leader election. In Proceedings of the 2nd ACM Conference on Advances in Financial Technologies (pp. 12-24).

[12] Breidenbach, L., Daian, P., Tramèr, F., & Juels, A. (2018). d and Exploit-Resistant Smart Contracts. In 27th USENIX Security Symposium (USENIX Security 18) (pp. 1335-1352).

[13] Bünz, B., Fisch, B., & Szepieniec, A. (2020, May). Transparent SNARKs from DARK compilers. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 677-706). Springer, Cham.

[14] Bünz, B., Agrawal, S., Zamani, M., & Boneh, D. (2020, February). Zether: Towards privacy in a smart contract world. In International Conference on Financial Cryptography and Data Security (pp. 423-443). Springer, Cham.

[15] Burdges, J., & Feo, L. D. (2021, October). Delay encryption. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 302-326). Springer, Cham.

[16] Buterin, V. (2021, January 26). An approximate introduction to how zk-SNARKs are possible. vitalik.ca. https://vitalik.ca/general/2021/01/26/snarks.html

[17] Buterin, V. (2018, June 21). Quadratic Arithmetic Programs: from Zero to Hero - Vitalik Buterin. Medium. https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649

[18] Buterin, V. (2018, June 17). Separating proposing and confirmation of collations. Ethereum Research. https://ethresear.ch/t/separating-proposing-and-confirmation-of-collations/1000

[19] Buterin, V. (2022). Increasing censorship resistance of transactions under proposer/builder separation (PBS). Ethereum. https://notes.ethereum.org/@vbuterin/pbs_censorship_resistance

[20] Buterin, V. (2022, January). Secret non-single leader election. Ethereum Research. https://ethresear.ch/t/secret-non-single-leader-election/11789

[21] Buterin, V. (2022). Paths toward single-slot finality. HackMD. https://notes.ethereum.org/@vbuterin/single_slot_finality

[22] Cleve, R. (1986, November). Limits on the security of coin flips when half the processors are faulty. In Proceedings of the eighteenth annual ACM symposium on Theory of computing (pp. 364-369).

[23] Dai, W. (2022). Navigating Privacy on Public Blockchains. Wei Dai. https://wdai.us/posts/navigating-privacy/

[24] Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., ... & Juels, A. (2019). Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. arXiv preprint arXiv:1904.05234.

[25] Daza, V., Herranz, J., Morillo, P., & Rafols, C. (2007, November). CCA2-secure threshold broadcast encryption with shorter ciphertexts. In International Conference on Provable Security (pp. 35-50). Springer, Berlin, Heidelberg.

[26] De Feo, L., Masson, S., Petit, C., & Sanso, A. (2019, December). Verifiable delay functions from supersingular isogenies and pairings. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 248-277). Springer, Cham.

[27] Doweck, Y., & Eyal, I. (2020). Multi-party timed commitments. arXiv preprint arXiv:2005.04883.

[28] Gabizon, A. (2018, October 1). Explaining SNARKs Part V: From Computations to Polynomials. Electric Coin Company. https://electriccoin.co/blog/snark-explain5/

[29] Garg, S., Gentry, C., Sahai, A., & Waters, B. (2013, June). Witness encryption and its applications. In Proceedings of the forty-fifth annual ACM symposium on Theory of computing (pp. 467-476).

[30] Gennaro, R., Gentry, C., Parno, B., & Raykova, M. (2013, May). Quadratic span programs and succinct NIZKs without PCPs. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 626-645). Springer, Berlin, Heidelberg.

[31] Goldreich, O., Micali, S., & Wigderson, A. (2019). How to play any mental game, or a completeness theorem for protocols with honest majority. In Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali (pp. 307-328).

[32] Gosselin, S. (2022, January 19). MEV-Boost: Merge ready Flashbots Architecture. Ethereum Research. https://ethresear.ch/t/mev-boost-merge-ready-flashbots-architecture/11177

[33] Groth, J. (2016, May). On the size of pairing-based non-interactive arguments. In Annual international conference on the theory and applications of cryptographic techniques (pp. 305-326). Springer, Berlin, Heidelberg.

[34] HashCloak (2021). Using MPC to provide full privacy for flashbots. https://hackmd.io/@hashcloak/rkXxDglMK

[35] Konstantopoulos, G., & Buterin, V. (2021, July 20). Ethereum Reorgs After The Merge. Paradigm. https://www.paradigm.xyz/2021/07/ethereum-reorgs-after-the-merge

[36] Kelkar, M., Zhang, F., Goldfeder, S., & Juels, A. (2020, August). Order-fairness for byzantine consensus. In Annual International Cryptology Conference (pp. 451-480). Springer, Cham.

[37] Lindell, Y. (2020). Secure multiparty computation (MPC). Cryptology ePrint Archive.

[38] Maller, M., Bowe, S., Kohlweiss, M., & Meiklejohn, S. (2019, November). Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (pp. 2111-2128).

[39] Mannak, R. (2021, December 12). Comparing General Purpose zk-SNARKs - Coinmonks. Medium. https://medium.com/coinmonks/comparing-general-purpose-zk-snarks-51ce124c60bd?

[40] Matter Labs (2019, August). GitHub-matter-labs/awesome-zero-knowledge-proofs: A curated list of awesome things related to learning Zero-Knowledge Proofs (ZKP). al

[41] Nitulescu, A. (2020). zk-SNARKs: A Gentle Introduction. Technical report.

[42] Obadia, A., & Vemulapalli, T. (2021, August). MEV in eth2 - an early exploration. HackMD. https://hackmd.io/@flashbots/mev-in-eth2

[43] Obadia, A., Salles, A., Sankar, L., Chitra, T., Chellani, V., & Daian, P. (2021). Unity is Strength: A Formalization of Cross-Domain Maximal Extractable Value. arXiv preprint arXiv:2112.01472.

[44] Parno, B., Howell, J., Gentry, C., & Raykova, M. (2013, May). Pinocchio: Nearly practical verifiable computation. In 2013 IEEE Symposium on Security and Privacy (pp. 238-252). IEEE.

[45] Pedersen, T. P. (1991, August). Non-interactive and information-theoretic secure verifiable secret sharing. In Annual international cryptology conference (pp. 129-140). Springer, Berlin, Heidelberg.

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

[47] Reitwiessner, C. (2016). zkSNARKs in a nutshell. Ethereum blog, 6, 1-15.

[48] Rivest, R. L., Shamir, A., & Wagner, D. A. (1996). Time-lock puzzles and timed-release crypto.

[49] Robinson, D., & Konstantopoulos, G. (2020, August 28). Ethereum is a Dark Forest. Paradigm. https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest

[50] Schindler, P., Judmayer, A., Stifter, N., & Weippl, E. (2019). Ethdkg: Distributed key generation with ethereum smart contracts. Cryptology ePrint Archive.

[51] Scholl, P., Simkin, M., & Siniscalchi, L. (2021). Multiparty computation with covert security and public verifiability. Cryptology ePrint Archive.

[52] Schwartz, J. T. (1980). Fast probabilistic algorithms for verification of polynomial identities. Journal of the ACM (JACM), 27(4), 701-717.

[53] Shamir, A. (1979). How to share a secret. Communications of the ACM, 22(11), 612-613.

[54] Shutter Network. (2021, August 18). In-Depth Explanation of How Shutter Prevents Frontrunning. https://blog.shutter.network/shutter-in-depth-explanation-of-how-we-prevent-frontrunning

[55] Stengele, O., Raiber, M., Müller-Quade, J., & Hartenstein, H. (2021, November). ETHTID: Deployable threshold information disclosure on Ethereum. In 2021 Third International Conference on Blockchain Computing and Applications (BCCA) (pp. 127-134). IEEE.

[56] Wesolowski, B. (2019, May). Efficient verifiable delay functions. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 379-407). Springer, Cham.

[57] Zcash. (2019, August 14). Parameter Generation. https://z.cash/technology/paramgen/