Leveraging Homomorphic Encryption for Maximally Extractable Value (MEV) Mitigation: Enabling Blind Arbitrage on Decentralised Exchanges

Jonathan Passerat-Palmbach[1,2] [0000−0003−3178−9502]

[1] Flashbots
[2] Imperial College London
jonathan@flashbots.net

Abstract. This paper presents a novel exploration of Fully Homomorphic Encryption (FHE) applied to the problem of Maximal Extractable Value (MEV) in blockchain transactions, explicitly focusing on arbitrage scenarios. Building upon previous work by Flashbots, we adapt their secret-sharing-based protocol to FHE to reduce data transmission overhead. We introduce a protocol that enables searchers to blindly backrun a user transaction, executing specific conditions and arithmetic operations on the transaction content using FHE. Our protocol leverages the TFHE scheme to handle the data extraction, constraint verification and arithmetic calculation on 128-bit unsigned integers representing the tokens stored on the UniswapV2 Decentralised Exchange (DEX). Despite significant runtime constraints, our work provides a solid foundation for future research, identifying key areas for improvement and expansion. This study represents a significant advancement in applying FHE to blockchain transactions, stimulating further collaboration within the FHE community to address open challenges and bring privacy-preserving protocols for MEV mitigation closer to real-world deployment.

Keywords: MEV · Arbitrage · Fully Homomorphic Encryption

1 Introduction

Blockchains function by creating blocks containing user-generated transactions, typically in a sequential order proposed by block producers. However, this order manipulation capability has spurred the emergence of Maximal Extractable Value (MEV)[14]. MEV entails extracting value beyond regular block rewards and transaction fees by controlling transaction sequencing. According to Flashbots' transparency dashboard[3], close to $750M worth of value has been extracted over the past year alone. Essentially a form of economic rent, MEV is usually a value transfer from average to elaborate users. The incentives it creates span beyond poor user experience and lead to increased gas prices, transaction delays,

[3] https://transparency.flashbots.net/

fairness and security concerns, impeding the progress of new applications on the whole network [24].

The actors ultimately extracting MEV are block producers like miners or validators, courtesy of the blockchain's consensus mechanism. Tactics to extract MEV, include front-running (prioritising personal transactions), sandwich attacks (strategically positioning personal transactions around others), and order manipulation (re-arranging transactions for personal gain). Some MEV-related activities, like arbitrage (taking advantage of a price imbalance for the same asset across multiple markets) and liquidations (selling off collateral to cover outstanding loans when their value falls below a certain threshold), are viewed more positively as providing a service to the network in return for a profit [23].

Addressing MEV involves diverse approaches, from altering consensus mechanisms to developing user protection tools. Some approaches try to prevent all forms of MEV extraction by hiding the transaction data using traditional encryption until they have been included in a committed block [27, 26]. This approach prevents advanced block building constructs and extraction strategies dependent on analysing the effects a pending (set of) transaction(s) will have on chain.

While consensus on a definitive solution remains elusive, in the rest of this work, we will focus on arbitrage as a form of MEV and try to facilitate such extraction under the condition that the user transaction creating the opportunity does not incur a financial penalty. Specifically, we examine MEV extraction scenarios called backrunning, where additional transactions are inserted after the user's original transaction, excluding front-running as a possibility.

1.1 Contributions

This paper proposes a novel approach leveraging Fully Homomorphic Encryption (FHE) to overcome these limitations. Using the tfhe-rs Rust library [29], which implements the TFHE (Torus Fully Homomorphic Encryption) [10] scheme, we present an alternative protocol targeting a specific Decentralised Exchange (DEX), UniswapV2 [1]. Unlike other recent proposals [4] powered by the so-called fhEVM [12], which aim to provide private smart contract blockchains, our goal is not to offer long-term confidential storage but to enable off-chain interactions between users and MEV searchers while ensuring pre-trade privacy.

In the following sections, we present our method and results, demonstrating the feasibility of using FHE for MEV extraction while acknowledging significant performance overhead for practical deployment (Section 3). We then evaluate the protocol, highlighting its limitations and identifying key performance bottlenecks (Section 4). In response to these challenges, we propose three algorithmic modifications to improve the efficiency of encrypted computation (Section 5), and we discuss future directions and open challenges for the FHE community to address (Section C).

[4] `https://www.fhenix.io`, `https://www.inco.org`

2 Background: Arbitrage on UniswapV2

Uniswap V2 [1] is the most commonly used decentralised exchange (DEx) on the Ethereum public blockchain. Its design is inspired by the Automated Market Maker (AMM) concept that maintains a constant k as the relationship between the quantity of each of the two crypto-tokens in the reserve of a pool. Each pool represents a trading pair and has its k value. The pool is ruled by the equation $k = x * y$ which determines the price of each token with respect to the other. Users interact with these pools by swapping a certain amount of a first token A for the corresponding amount of paired token B based on the current price in the pool. Since each Ethereum transaction executes atomically, the price of each asset in the pool is immediately updated for all the other users once the swap completes; the on chain state now reflects the new values for the token reserves.

The price quoted by the Uniswap v2 DEX is only valid for this specific pair on the DEX, and the same pair often trades at different prices on different exchanges. Traders can exploit this momentary price discrepancy by buying one token of the pair on a first exchange and selling the other back on a second exchange, pocketing the price difference as a profit. This manoeuvre is called arbitrage and is widely considered to positively affect end users since it rebalances prices across all exchanges. Arbitrage is a form of MEV that we want to enable as many traders as possible to participate in. The intuition behind this philosophy is that more arbitrageurs in the network will be more efficient at solving price differences across markets and thus provide an overall better experience to end users.

To exploit arbitrage opportunities successfully, arbitrageurs need guarantees that the price they have identified as an opportunity on an exchange will remain the same while they purchase the other asset in the pair to trigger the first hop in the simplified arbitrage scenario described above. The only way to obtain this guarantee on a DEX is to ensure the trader's transaction initiating the arbitrage is executed right after the user's which triggered the price change. Transactions are executed in the order in which they appear in a block. We denote the action of deliberately trying to insert one's transaction after another in the same block as back-running. For simplicity in defining our encrypted backrunning protocol, we leave out the other case when a searcher can have price certainty: the top of the next block.

To summarise, this work proposes a solution for traders to blindly backrun a user transaction performing a token swap on the UniswapV2 DEX, without seeing its data in cleartext while still being able to execute certain conditions and arithmetic operations on the content of the transaction using FHE.

3 A Confidential Backrunning Protocol

Arbitrage through backrunning is a critical mechanism for exploiting price discrepancies across blockchain markets, yet competition in public mempools has driven up transaction fees and introduced risks for users. To address these issues,

we propose a confidential backrunning protocol that enables secure arbitrage on encrypted transactions. This protocol protects users who create arbitrage opportunities by ensuring pre-trade privacy and mitigating the execution costs associated with backrunning.

In our design, users submit fully encrypted transactions into an FHE-enabled backrunning protocol. Searchers securely input constraints and parameters to guide their strategy, with encrypted outputs passed to a trusted block builder for decryption and blockchain insertion. Importantly, the protocol ensures that searchers cannot directly observe transaction details.

The searcher does not obtain any observable output from the computation: the resulting backrunning transaction contains either a profitable transaction or an invalid one that will be rejected by the block builder. This guarantees user privacy while enabling the searcher to execute a backrunning strategy blindly, in line with a set of predefined preferences.

3.1 Threat Model

The scenario we consider is represented in Figure 1. There are three distinct parties: 1) a **User** who provides their transaction data but requires strong pre-trade privacy guarantees; 2) a **Searcher** who aims to exploit an arbitrage opportunity by producing a bundle comprising the user's transaction immediately followed by a backrunning transaction that capitalises on the price discrepancy introduced by the user's operation. The searcher blindly applies a predefined algorithmic strategy to the encrypted data and has control over input parameters such as a minimum profit threshold; 3) a **Block Builder** who inserts the resulting bundle, along with other externally sourced transactions and bundles, into a candidate block without modifying its contents. The builder acts as a trusted entity, decrypting the resulting bundle securely.

This work focuses on interactions between the user and the searcher, while the block builder's role remains limited to decryption and block insertion. The trust assumption for the block builder ensures that they act as a passive participant, securely decrypting and handling the transactions without interfering in the backrunning process. The parties operate under the honest-but-curious security model, where the searcher follows the protocol but may attempt to extract additional information from intermediate states. In this setting, the searcher cannot directly observe the transaction details due to encryption but could manipulate their inputs or execute modified strategies to extract sensitive information. To mitigate such risks, zero-knowledge proofs (ZKPs) of well-formed ciphertexts could be employed to validate the correctness of the user's encrypted transaction, as demonstrated in prior literature [22]. Additionally, the correctness of the encrypted computation itself could be verified using emerging verifiable FHE techniques [3]. However, these methods introduce substantial overhead with the current state of the art, making them less practical for immediate deployment.

For this study, we assume the use of a secret key shared between the user and the block builder, a reasonable assumption given that Ethereum builders are increasingly operating within Trusted Execution Environments (TEEs), as

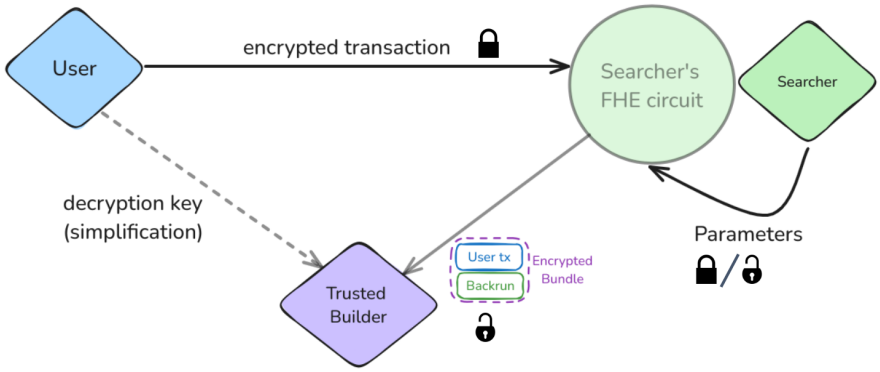

Fig. 1: High-level overview of the parties involved in the confidential backrunning protocol. The user encrypts their Ethereum transaction and propagates it to a searcher's infrastructure equipped with the FHE backrunning program. The searcher runs the FHE program on their preferences and the user's encrypted transaction to generate an encrypted backrunning transaction. Together with the user's encrypted transaction, this forms a bundle of encrypted transactions transferred to a trusted block builder for decryption and inclusion in a block. The current design assumes shared private keys, but future iterations could replace this with a public-key infrastructure managed via threshold decryption [13].

demonstrated by Flashbots' implementation of block building inside SGX [17, 16]. Notably, our protocol introduces no additional requirements for the searcher, who runs the FHE computation without the need for a TEE. Both the user and the searcher can independently verify the block builder's use of a TEE through remote attestation before securely sharing their respective ciphertexts with the builder, maintaining trust without compromising decentralisation.

The cryptographic parameters provided by TFHE-rs ensure at least 128 bits of security, as verified by the Lattice Estimator[5]. The builder generates the secret key securely and shares it with the user over a remotely attested trusted channel before the protocol begins.

While this assumption simplifies implementation, it has limitations in real-world deployments. Specifically, it assumes a trusted channel for key sharing and does not account for scenarios involving multiple, potentially untrusted builders. Future iterations of the protocol aim to address these limitations by adopting a public-key cryptographic model. In particular, we propose transitioning to a threshold decryption setup, as outlined in [18], where a distributed network of parties collaboratively manages the private key. This approach enhances resilience against builder compromise and aligns with a permissionless deployment model.

[5] `https://github.com/malb/lattice-estimator`

3.2 FHE Parsing of User Transactions

The FHE program begins by parsing the encrypted user transaction, serialized in Recursive-Length Prefix (RLP[6]) format. This process is critical because it ensures that the data matches specific acceptance criteria established by the searcher in their strategy. The searcher typically focuses on transactions involving particular token pools, with this focus being encoded in the search strategy's parameters.

During this initial phase, the protocol evaluates aspects such as gas price, gas limit, and token pool addresses, ensuring that transactions do not incur prohibitive execution costs for the searcher. The FHE-enabled logic systematically reads bytes from the input transaction, comparing them with the predefined values from the searcher program. An encrypted boolean, 'match', tracks whether all conditions are met. If any condition fails, 'match' is set to 'false', signaling that the transaction does not meet the searcher's criteria.

In the event of a failed condition, the protocol outputs an empty transaction (filled with zeroes). This approach accommodates FHE's limitations, notably the inability to handle branching logic directly. Thus, the protocol can terminate early when conditions aren't satisfied, avoiding unnecessary computational overhead.

3.3 Arithmetic Calculations for Profit Optimization

Once the transaction is validated, the next step involves calculating the optimal amount of tokens to buy for a profitable backrun. This calculation relies on the Uniswap V2 pricing formula, adapted to the FHE setting.

Uniswap V2 Pricing Function In Uniswap V2, token prices are determined using a constant product formula. The price of a token is the ratio of reserves in the liquidity pool, adjusted for a 0.3% fee. For arbitrage, searchers aim to exploit price differences between decentralized exchanges like Uniswap and centralized exchanges such as Binance. To backrun a trade, the searcher needs to calculate the number of tokens to buy at the updated pool price, which is determined by the pending user transaction.

In our protocol, the searcher inputs three constants into the strategy: 'FEE' (the trading fee in Uniswap V2), 'PRICE' (the maximum price they are willing to pay for the asset), and 'PREC' (precision for on-chain calculations). The pool reserves, 'X' and 'Y', depend on the encrypted user transaction and are required to compute the updated price after the pending swap.

The optimal amount equation 1 used in this work and obtained from [25] is as follows:

[6] RLP is the format used to serialise Ethereum transactions - `https://ethereum.org/en/developers/docs/data-structures-and-encoding/rlp/`

$$\text{amount} = \frac{\sqrt{\text{PREC}\cdot X\cdot(\text{FEE}^2\cdot\text{PREC}\cdot X+4\cdot\text{PRICE}\cdot Y\cdot(1-\text{FEE}))}+\text{PREC}\cdot X\cdot(\text{FEE}-2)}{2\cdot\text{PREC}\cdot(1-\text{FEE})} \tag{1}$$

This equation calculates the optimal token amount for the searcher to purchase based on the new state of the pool's reserves 'X' and 'Y' after the user's transaction.

FHE Challenges: Square Roots and Divisions The main technical challenge in applying this equation under FHE lies in two costly operations: division by a scalar and calculating the square root of an encrypted number. These operations are computational bottlenecks, particularly the square root calculation, which we approximate using the Newton-Raphson method, limiting precision to five iterations. Each iteration involves additional homomorphic divisions, significantly increasing computation time. In recent benchmarks using the tfhe-rs library[7], this square root approximation required approximately 20 seconds on an AWS *hpc7a.96xlarge* server equipped with an AMD EPYC 9R14 CPU running at 2.60GHz, highlighting the performance costs of FHE in such scenarios.

3.4 Data Types and Precision Considerations

Uniswap V2 uses 112-bit unsigned integers to represent token reserves, while our FHE protocol handles these values as 128-bit unsigned integers to ensure compatibility with homomorphic encryption libraries like tfhe-rs. Moreover, the calculations use 256-bit integers internally to avoid overflow during encrypted multiplications and other operations. These larger data types help manage the inherent complexity of FHE while preserving numerical accuracy during the backrun.

To handle fractions (which are unsupported by the Ethereum Virtual Machine), Uniswap relies on a fixed-precision representation over 224-bit values equally split between integer and fractional parts. Our FHE implementation similarly adopts this strategy, using 256-bit unsigned integers to ensure safe computations across the backrun.

4 Evaluation of the confidential backrunning solution

4.1 RLP Data Extraction

In this first set of experiments, we evaluate the performances of different configurations of a single backrun by focusing on the RLP data extraction part. We report performances of the encrypted workload where the strategy's constants are represented using 128-bit cleartext unsigned integers in the setting where the

[7] `https://docs.zama.ai/tfhe-rs/0.6-3/get-started/benchmarks`

Table 1: Runtime comparison for different configurations of the confidential backrun protocol averaged over 5 runs. The first two rows depict the runtime and memory utilization during the RLP extraction phase, contrasting settings with searcher constants in cleartext (*Clear*) or encrypted. The third row showcases the runtime and memory consumption when enabling AVX512 insructions. The last two rows present the same breakdown for the amount calculation phase. Note: Key generation and encryption impacts are considered negligible compared to the reported metrics, and as such are included as part of the overall runtime.

Phase	**Runtime (seconds)**	**Memory (kB)**
RLP (Clear)	36.706 (±0.193)	331,992
RLP (Encrypted)	38.782 (±0.336)	334,000
RLP (Clear + AVX512)	31.787 (±0.921)	337,100
Amount	1024.928 (±0.925)	432,116
Amount (AVX512)	846.409 (±3.134)	462,146

workload runs on the searcher's machine, as well as their homomorphic equivalent, to understand the possibility of running the protocol on a third party's machine.

Please note that we do not report the key generation and encryption steps separately in the results table as they are negligible (a few seconds) in light of the core workload's runtime. They could also take place as part of a preprocessing phase where users and searchers would encrypt once, and this already minimal time would be amortised over many executions.

All results reported in Table 1 were obtained using a 128-core Intel Xeon Platinum 8375C CPU running at 2.90GHz with AVX2 instructions enabled.

Not surprisingly, the amount calculation phase dominates the RLP extraction by two orders of magnitude due to the series of homomorphic divisions it has to perform as part of Equation 1 to approximate the square root using Newton's method. The present setting is particularly challenging since it is not trivial to initialise the square root approximation algorithm with a reasonable initial estimate without computing even more encrypted operations. Indeed, the square root entirely depends on the user transaction, which can significantly impact the new reserves' amounts in the equation. This performance discrepancy highlights the gap in computational efficiency that will need to be bridged to support any input data for protocols running in the EVM.

4.2 SIMD optimisation: AVX2 vs AVX512

Another significant aspect of our analysis focused on the impact of SIMD (Single Instruction, Multiple Data) instruction sets on the execution speed of the FHE algorithm. The TFHE scheme relies on a fast bootstrapping operation implemented via external products that can be accelerated using Fast Fourier Transforms (FFTs).

Our investigation included the AVX2 and AVX512 instruction sets. Notably, the AVX512 instruction set demonstrated substantial performance improvements over the AVX2, providing a speed-up of approximately 20%. However, AVX512 instructions are less widely available in customer-grade CPUs and would require running the workload on high-end servers. This strong hardware requirement reinforces our flexible design goal of being able to run the protocol on the searcher's machine or a third party's.

4.3 Key and Ciphertexts Size

Fully Homomorphic Encryption (FHE), notably TFHE, faces ciphertext expansion, causing data transfers significantly larger than plaintext versions. However, the data payload of Ethereum transactions is relatively small, curbing transfer demands to the encrypted data payload only, compared to MPC requirements for the protocol itself, even under weaker security models. The server key currently consumes around 105MB, while the ciphertexts for the transaction and strategy contain 47MB and 20MB of data, respectively.

5 Methodological Enhancements for FHE Performance

5.1 Reducing Security to 80 bits

In the realm of fully homomorphic encryption (FHE), a fundamental trade-off exists between the level of security and the computational performance. Traditionally, the security parameter for FHE schemes has been set to 128 bits, providing robust protection against brute-force attacks and ensuring long-term security. However, some applications make for a compelling case to lower the security parameter to 80 bits or less when long-term confidentiality is not required.

This is typically the case in our current work, for which we can leverage the inherently public nature of Ethereum transactions. Once included in a block, transactions become part of the public ledger, and their contents are openly accessible to all parties. Consequently, the need for long-term confidentiality of the encrypted data diminishes, as the sensitive information will inevitably be revealed upon block inclusion.

As such, we experiment with the performance benefits of lower security parameters when encrypting user transactions for our confidential backrun protocol. Reducing the security parameter to 80 bits still provides substantial protection against brute-force attacks and remains more than enough in the context of short-term data confidentiality.

It is important to note that the reduction in security parameters does not compromise the integrity or correctness of the FHE scheme itself. While 80-bit encryption has been studied and can reuse existing publicly available parameter sets carefully chosen for tools such as Zama's Concrete [7], there is no guarantee we can venture lower down the security level and still guarantee correctness.

Table 2: Runtime comparison for different configurations of the confidential backrun protocol using lower 80-bit security. The speed-up column is calculated as a ratio of the Cleartext 128-bit alternative.

Phase	**Runtime (seconds)**	**Speed-up over 128-bit**
RLP (Cleartext)	22.607 (±0.143)	1.62
RLP (Cleartext + AVX512)	19.277 (±0.098)	1.90
Amount	619.325 (±0.429)	1.65
Amount (AVX512)	504.351 (±10.25)	2.03

Indeed, generated parameters need to balance security properties with the probability of correctly decrypting the result of a computation. This type of decryption error is completely silent and, as a result, would not be detected. In the context of the backrunning application, the searcher would miss potential opportunities without knowing whether this is due to his choice of parameters or an unfortunate decryption error that tampered with the result and changed the outcome of the computation. As such, we experiment with lower security settings while remaining overly conservative about the level of security we still guarantee.

Here, we select 80-bit security parameters from the publicly available set made available along with Concrete's open source code. Our algorithm uses the standard encoding of 2-bit precision to store the message and 2-bit precision for the carry. This leads us to select a set of parameters in the 4-bit precision table and a log2 of 2-norm = 3. Table 2 presents the new runtimes for the 80-bit security parameters. All other parameters remain identical to the original 128-bit parameter set.

We observe a 62-65% speedup over the original settings for both phases of the algorithm. When coupled with the AVX512 instructions, the workload's runtime is about twice as fast as the original 128-bit setting. Considering the very short time our application needs to guarantee the confidentiality of its input data, we could lower the security level even further. However, this requires careful investigation to generate a correct set of parameters that will preserve the correctness of the encrypted calculation.

5.2 Circuit-Level Optimisation through Levelled Operation Grouping

Another promising optimisation technique for enhancing the performance of FHE schemes is circuit-level optimisations, specifically the grouping of levelled operations. This approach aims to minimise unnecessary Programmable Bootstrapping (PBS) operations, which can significantly improve the overall computational efficiency of the FHE scheme.

The TFHE scheme [10] exposes two types of operations: levelled operations and operations implemented via Programmable Bootstrapping (PBS). Levelled

operations, like additions of ciphertexts with scalars, are typically less computationally intensive and can be performed without bootstrapping. On the other hand, operations like multiplications or divisions require PBS to maintain the noise level within acceptable bounds and implement their functionality. This means that contrary to levelled operations, evaluating PBS-based operations will require performing a bootstrapping, which, while fast in TFHE relative to other schemes, remains a performance bottleneck at the application scope.

A second observation we need to make is that TFHE does not have native support for integers beyond 8 bits of precision. When it comes to handling non-natively supported big integers, one approach that TFHE offers is to decompose a large integer using a radix representation in blocks of the same size, each containing an LWE ciphertext encoding a plaintext of low precision (usually 4 bits) [7]. This plaintext space is further split into bits for the actual message and bits available to store carries resulting from arithmetic operations. These carries need to be propagated from one block to the next to ensure the correctness of arithmetic operations under the radix representation for large integers. TFHE propagates carries through PBS calls, one to extract the message and a second for the carry itself, resulting in fresh LWE ciphertexts where needed in the chain of radix blocks. Depending on the arithmetic operations performed, carries may or may not need to be propagated. Thus, optimally handling when carry should be propagated can be a source of performance gain, avoiding superfluous calls to the costly PBS operation.

On top of the complex management of carries, an algorithm's logic might naturally present intertwined levelled operations and operations that can only be done via PBS, such as encrypted comparisons. PBS-based operations expect carries to be cleaned before running; hence, a ciphertext might still be able to support further levelled arithmetic computation but will see unnecessary PBS invocations to support a comparison.

The security parameters discussed in the previous section also control the number of levelled operations that can be performed without the need for bootstrapping or propagating carries through radix blocks. By strategically grouping levelled operations together, we can reduce the number of PBS operations required, thereby improving the overall performance of the FHE scheme. By lowering the number of PBS invocations that would otherwise be induced by unnecessary carry cleansing, we are effectively trading code readability for performance.

Figure 2 shows the example of a function extracted from the protocol's RLP decoding phase. The function `right_shift_equal_block` will be called multiple times in a row with different input parameters. By doing so, the subtraction on the last line will be followed by a PBS call corresponding to the first line of the function's next call in the actual FHE circuit. This process will break the series of levelled operations and force two extra PBS calls to propagate the carries through the radix blocks.

However, carefully observing the code snippet in Figure 2 reveals a pattern that allows rewriting the same function differently. The subtraction is indepen-

```
fn right_shift_equal_block(in_tx_cipher, ...) {
    let shifted: FheUint =
        in_tx_cipher >> right_shift_const;
    let eq = shifted.eq(eq_const);
    let enc_is_match = in_match_ctxt & eq;
    let rse: FheUint = in_tx_cipher - sub_const;
    (res, enc_is_match)
}
```

Fig. 2: Example of intertwined levelled and PBS-based operations.

dent of the other operations and, as such, does not need to be computed right after the PBS-based operations. Our solution consists of collecting the input parameters for all the different operations in each call to the function before processing them out of order after the final instance of this repeated pattern. Thanks to this approach, we can compute all the subtractions back to back, thus minimising the number of PBS calls across the sequence of calls to the `right_shift_equal_block` function without changing its result.

The process of grouping levelled operations can be perceived as a circuit- or graph-level optimisation, where the algorithm is represented as a computational graph, and levelled operations are strategically merged together to minimise the number of PBS operations required. This optimisation technique requires a deep understanding of the underlying FHE scheme and its internals, such as the radix representation and carry propagation mechanisms.

By minimising the number of PBS operations, the overall execution time of the FHE computation can be significantly reduced. In our backrun context, this approach is particularly efficient in the RLP decoding phase of the algorithm, where subtractions often follow comparisons as part of the program's logic. However, these operations do not need to be computed in this order and can be re-arranged to benefit from this optimisation technique. Our experiments confirmed that we process 8 fewer PBS calls in the deferred execution version over 4 consecutive calls to the `right_shift_equal_block` function, saving 2*4 superfluous carry propagation steps.

It is important to note that this optimisation technique is generic and can be applied to a wide range of FHE-enabled applications. Furthermore, while it requires a deep understanding of the underlying FHE scheme, better tooling and compiler-level support could automate this optimisation process. With proper tooling, graph-level optimisation through levelled operation grouping could be systematically applied without requiring developer intervention at the code level.

5.3 Efficient Amount Calculations Using Uniswap V3 Arithmetic

In this section, we present a method for calculating the output amount of a Uniswap V2 trade using the arithmetic principles of Uniswap V3. Our approach

centers on the key observation that Uniswap V3's internal state is built around the square root of the product of reserves and price. This representation leads to a more computationally efficient arithmetic model, which eliminates the need for costly square root approximations. This feature is even more prevalent when speeding up calculations in cryptographic environments, since we have previously identified the square root approximation function as the bottleneck of the V2 pool simulation method in the FHE context.

Uniswap v3 was initially designed to optimize liquidity deployment through concentrated liquidity, where liquidity providers (LPs) selectively contribute to price ranges, maximizing capital efficiency. Each of these virtual liquidity pools within Uniswap v3 behaves similarly to a standard Uniswap v2 pool, meaning we can apply the same arithmetics to a v2 pool if we construct an initial state consistent with v3 expectations.

Unlike Uniswap v2, Uniswap v3 does not directly store the reserves (x and y) of the two trading assets. Instead, it stores the square root of the product of reserves ($\sqrt{x \cdot y}$), known as the liquidity, and the square root of the current price. The key innovation here is that once a v3 pool is initialized with the square root of the current price, subsequent price updates can be performed using only integer multiplications and divisions. This approach avoids recalculating the square root—previously a performance bottleneck in FHE contexts—thus improving both the speed and accuracy of our trade simulations.

Additionally, the liquidity parameter remains constant for any given price-range in a V3 pool, hence for a whole V2 pool. We can pre-calculate this value from public blockchain data and inject it as a constant in price calculations. This results in lighter computational requirements: for price updates following a swap, we require one encrypted multiplication, one scalar multiplication, and one encrypted division; for the searcher to obtain the input amount to bid to obtain receive the maximum number of tokens at a target price, one scalar multiplication, one encrypted multiplication, and one encrypted division; and for the corresponding output amount, just one scalar multiplication with an optional division.

The searcher in our system can specify the next target price and provide the pool's liquidity, reducing the FHE circuit's input size and complexity. Notably, unlike Uniswap v2 simulations that require computing the user's exact trade output, in this protocol only the updated price matters, so the searcher does not need to compute the exact amount of assets received by the user. This optimization fits naturally within the RLP decoding phase of our framework, allowing the searcher to only specify liquidity values for the targeted pools, thus minimizing the input data required by the FHE algorithm.

The impact of the new calculation method's performance is presented in Table 3, which demonstrates a drastic reduction in the number of operations required to simulate the user's transaction and calculate the optimal price and amount for the searcher. To provide a clearer picture of where these improvements originate, we chose to present the breakdown of operations individually along with their respective runtime costs for 128-bit and 256-bit unsigned inte-

Table 3: Detailed comparison of the arithmetic operations performed in the two versions of our FHE backrunning program: one using UniswapV2's traditional reserve-based pricing function and the other employing UniswapV3-style arithmetic, which updates the square root of the price in a virtual pool. The operations are classified into four types: ciphertext-scalar multiplication, ciphertext-scalar division, ciphertext-ciphertext multiplication, and ciphertext-ciphertext division. Each operation is evaluated for both 128-bit and 256-bit unsigned integer variants. The total runtime for each type is calculated using Zama's per-operation benchmarks from the tfhe-rs v0.6.3 library. The results illustrate significant differences in computational overhead, demonstrating that UniswapV3's arithmetic reduces the cost of encrypted computations by nearly 12X, primarily due to the elimination of the square root approximation.

		128-bit unsigned integers				**256-bit unsigned integers**			
		Scalar Multiplication	**Ciphertext Multiplication**	**Scalar Division**	**Ciphertext Division**	**Ciphertext Multiplication**	**Ciphertext Division**	**Total**	**Total**
UniV2 style	**Number of Operations**	6	1	2	2	2	9	22	22
	Contribution To Runtime	2.226	0.961	1.498	41.2	6.4	484.2	536.485	536.485
UniV3 style	**Number of Operations**	3	2	1	2	0	0	8	8
	Contribution To Runtime	1.113	1.922	0.749	41.2	0	0	44.984	44.984

gers. This breakdown allows us to isolate and emphasize the specific reductions in computational overhead that would have been harder to observe at the whole application level. Profiling a complex FHE library like tfhe-rs is challenging, as its execution engine is highly optimized and involves intricate processing pipelines, making it difficult to extract fine-grained operation-level details from typical profiling tools.

By categorizing the operations and presenting them separately, we can more effectively highlight the improvements achieved through the use of UniswapV3-style arithmetics—most notably, the elimination of the square root approximation, which was a major bottleneck in the UniswapV2-based approach, and results in a nearly 12X overhead decrease in the new version. The total runtime table, summarizing these results, is available in Appendix A for further reference.

5.4 Performance and Comparison with the MPC version

The backrunning protocol based on Multi-Party Computation (MPC), as proposed in [25], demonstrates significant computational and communication overheads that limit its practicality. Under the most stringent MPC security model (MASCOT protocol [21]), the protocol requires approximately 22.5 hours of computation and 33.7 PB of data transfer spread over 9.6 million communication rounds for a single transaction. In the semi-honest security model [2], the runtime is reduced to 4 minutes, but still requires an overwhelming 6 GB of data transfer over 3.5 million communication rounds.

While the MPC version demonstrates the feasibility of confidential MEV extraction, its extreme communication demands make it impractical, especially in high-frequency scenarios. These limitations are even more problematic in the context of high-frequency MEV extraction scenarios, where reduction of communication costs will be sought via co-location at the expense of geographic decentralisation. It is also important to view the MPC protocol as a proof of concept and not a definitive representation of MPC performance in all MEV contexts.

In comparison, our FHE-based protocol eliminates the need for communication rounds during the encrypted computation phase, significantly reducing network dependence. This architectural advantage not only addresses bandwidth constraints but also avoids incentives for co-location, preserving the level of decentralisation of the unencrypted protocol. However, the computational overhead introduced by homomorphic operations remains a significant challenge that will be a focus for future work.

Our FHE protocol operates under a semi-honest model, and its most optimised configuration (leveraging UniswapV3-style arithmetic, 80-bit security, and a 128-core server) produces a backrunning transaction in 5.59 minutes as is detailed in Table 4. The associated data transfers include 47MB for the encrypted user transaction, 20MB for the searcher's strategy, and another 47MB for the resulting encrypted backrun transaction, totalling approximately 114MB.

This comparison highlights the trade-offs: while the FHE protocol achieves a substantial reduction of 54X in communication requirements and aligns better with real-world deployment constraints as this exchange happens over a single round of communications.

6 Conclusion

In this work, we have explored the application of Fully Homomorphic Encryption (FHE) to the problem of Maximal Extractable Value (MEV) in blockchain transactions. We have focused on the specific case of arbitrage, where a transaction swapping two tokens on a given market can create a price discrepancy in another market. Our approach builds upon previous work by Flashbots, adapting their secret-sharing-based protocol with FHE to reduce the overhead in data transmission.

We have presented a protocol that allows traders to blindly backrun a user transaction, executing certain conditions and arithmetic operations on the content of the transaction using FHE to mimic what is known as searcher's strategy. This protocol uses the tfhe-rs Rust library and targets the UniswapV2 Decentralised Exchange (DEX).

We introduced three improvements to accelerate the performance of our FHE algorithm. Combining the reduced security parameter approach discussed in the previous section with circuit-level optimisation through levelled operation grouping can achieve significant performance gains in FHE schemes. These optimisa-

tions are generic to any FHE program and can be adopted in addition to the new mode of calculation inspired from Uniswap V3.

Our evaluation has shown that while the protocol is feasible, it faces significant runtime constraints. We have identified several areas for future research, including handling multiple input transactions, overcoming ECDSA signature challenges, and extending the protocol to other forms of MEV and other DEXs.

In conclusion, this work represents a significant step forward in applying FHE to blockchain transactions. While many challenges remain, our work provides a solid foundation for future research at the interface of MEV and encrypted computing. Our findings will stimulate further collaboration within the FHE community to address the open challenges and bring privacy-preserving protocols for MEV mitigation closer to real-world deployment.

References

1. Adams, H., Zinsmeister, N., Robinson, D.: Uniswap v2 core. Tech. rep. (mar-2020)
2. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 805–817. CCS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978331, https://doi.org/10.1145/2976749.2978331
3. Aranha, D.F., Costache, A., Guimarães, A., Soria-Vazquez, E.: Heliopolis: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. In: Chung, K.M., Sasaki, Y. (eds.) Advances in Cryptology – ASIACRYPT 2024. Lecture Notes in Computer Science, vol. 15488, p. 302–334. Springer Nature Singapore, Singapore (2025). https://doi.org/10.1007/978-981-96-0935-2_10, https://link.springer.com/10.1007/978-981-96-0935-2_10
4. Bebel, J., Ojha, D.: Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive
5. Benarroch, D., Gillespie, B., Lai, Y.T., Miller, A.: Sok: Programmable privacy in distributed systems. Cryptology ePrint Archive (2024)
6. Bentov, I., Ji, Y., Zhang, F., Breidenbach, L., Daian, P., Juels, A.: Tesseract: Real-time cryptocurrency exchange using trusted hardware. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1521–1538 (2019)
7. Bergerat, L., Boudi, A., Bourgerie, Q., Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Parameter optimization and larger precision for (t) fhe. Journal of Cryptology **36**(3), 28 (2023)
8. Bormet, J., Faust, S., Othman, H., Qu, Z.: Beat-mev: Epochless approach to batched threshold encryption for mev prevention. Cryptology ePrint Archive (2024)
9. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: CHIMERA: Combining ring-LWE-based fully homomorphic encryption schemes. Journal of Mathematical Cryptology **14**(1), 316–338 (2020). https://doi.org/doi:10.1515/jmc-2019-0026, https://doi.org/10.1515/jmc-2019-0026
10. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)

11. Choudhuri, A.R., Garg, S., Piet, J., Policharla, G.V.: Mempool privacy via batched threshold encryption: Attacks and defenses. Cryptology ePrint Archive (2024)
12. Dahl, M., Danjou, C., Demmler, D., Frederiksen, T., Ivanov, P., Joye, M., Rotaru, D., Smart, N., Thibault, L.T.: Confidential evm smart contracts using fully homomorphic encryption. Tech. rep., Zama (2023)
13. Dahl, M., Demmler, D., El Kazdadi, S., Meyre, A., Orfila, J.B., Rotaru, D., Smart, N.P., Tap, S., Walter, M.: Noah's ark: Efficient threshold-fhe using noise flooding. In: Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. p. 35–46. ACM, Copenhagen Denmark (Nov 2023). https://doi.org/10.1145/3605759.3625259, https://dl.acm.org/doi/10.1145/3605759.3625259
14. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 910–927. IEEE (2020)
15. Dujmovic, J., Garg, R., Malavolta, G.: Time-lock puzzles with efficient batch solving. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 311–341. Springer (2024)
16. Flashbots: Introducing buildernet (2024), https://buildernet.org/blog/introducing-buildernet
17. Hager, C., Paape, F.: Block building inside sgx (2023), https://writings.flashbots.net/block-building-inside-sgx
18. Joye, M.: TFHE public-key encryption revisited. Cryptology ePrint Archive, Paper 2023/603 (2023), https://eprint.iacr.org/2023/603
19. Kavousi, A., Le, D.V., Jovanovic, P., Danezis, G.: Blindperm: Efficient MEV mitigation with an encrypted mempool and permutation.
20. Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1575–1590. CCS '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372297.3417872, https://doi.org/10.1145/3372297.3417872
21. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 830–842. CCS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976749.2978357, https://doi.org/10.1145/2976749.2978357
22. Libert, B.: Vector commitments with proofs of smallness: Short range proofs and more (2023/800) (2023), https://eprint.iacr.org/2023/800
23. McLaughlin, R., Kruegel, C., Vigna, G.: A large scale study of the ethereum arbitrage ecosystem. In: Proceedings of the 32nd USENIX Conference on Security Symposium. SEC '23, USENIX Association (2023)
24. Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: How dark is the forest? In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 198–214 (2022). https://doi.org/10.1109/SP46214.2022.9833734
25. Robert Annessi: Backrunning Private Transactions Using Multi-Party Computation (2023), https://writings.flashbots.net/backrunning-private-txs-MPC
26. Rondelet, A., Kilbourn, Q.: Mempool privacy: An economic perspective (2023)
27. Shutter: Introducing shutter network - Combating front running and malicious MEV using threshold cryptography (2021), https://blog.shutter.network

28. Van Schaik, S., Seto, A., Yurek, T., Batori, A., AlBassam, B., Genkin, D., Miller, A., Ronen, E., Yarom, Y., Garman, C.: Sok: Sgx. fail: How stuff gets exposed. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 4143–4162. IEEE (2024)
29. Zama: TFHE-rs: A pure rust implementation of the TFHE scheme for boolean and integer arithmetics over encrypted data (2022)
30. Zhang, H., Merino, L.H., Qu, Z., Bastankhah, M., Estrada-Galiñanes, V., Ford, B.: F3b: A low-overhead blockchain architecture with per-transaction front-running protection. arXiv preprint arXiv:2205.08529 (2022)

A Total runtime for Uniswap V3 style computations

Table 4: Total runtime when using Uniswap V3 arithmetics to compute the backrunning amount in FHE across various configurations.

Settings	Runtime (mins)	Speedup
Laptop 4 cores	28.4	1
128 cores	11.36	2.50
128 cores & AVX512	9.38	3.03
(80-bit) 128 cores	6.87	4.14
(80-bit) 128 cores & AVX512	5.59	5.08

B Related work

Efforts to mitigate Miner Extractable Value (MEV) can be broadly categorised into methods that entirely block MEV extraction and those that preserve pre-trade privacy while allowing controlled MEV extraction.

Threshold encryption approaches [4, 19, 30] hide transaction details by distributing shares of a decryption key across a committee. Transactions are decrypted only after block finalisation, ensuring blind transaction ordering. While these methods provide strong confidentiality guarantees, their scalability is limited by the high communication overhead of generating and sharing partial decryption keys for each transaction [26]. Protocols like Shutter [27] reduce this overhead by introducing epoch-based encryption but fail to protect the confidentiality of pending transactions in these encrypted transaction pools.

More recently, [11, 8] proposed to address these limitations by introducing Batched Threshold Encryption (BTE). BTE maintains transaction confidentiality until their inclusion in a block and scales better since committee members need to decrypt fewer transactions at once and even eliminate setup overheads for [8].

Time-lock encryption, another method, conceals transaction data until a predefined time elapses, avoiding the need for trusted committees. While early designs incurred significant computational costs, advancements like linearly homomorphic time-lock puzzles [15] optimise batch decryption.

Methods allowing MEV extraction are now commonly referred to as Programmable Privacy [5]. They encompass software cryptography and Trusted Execution Environments (TEEs), such as Intel SGX. TEEs have long made their way in MEV mitigation solutions [6] because these methods are efficient and practical, with industry solutions like Flashbots [17, 16] deploying TEE-centric products, thus demonstrating the appeal for these tools despite their different

threat model. Indeed, TEEs on hardware trust assumptions, making them vulnerable to various software and hardware-level attacks [28], and harder to reason about from a security standpoint. Both threshold and time-lock encryption systems could also be paired with programmable privacy, controlling the timing of MEV extraction.

The present work proposes a programable-privacy approach based on Fully Homomorphic Encryption (FHE). Unlike threshold encryption, we enable confidential arbitrage-MEV extraction and consider a different threat model to TEE-based systems at the cost of scalability.

Building on prior work by Flashbots [25], we aim to improve upon a protocol that extracts arbitrage-MEV from confidential transactions without financial harm to the user. The original study demonstrated the feasibility of deriving optimal arbitrage opportunities from confidential user transactions using a semi-honest, honest-majority, 3-party protocol based on the MP-SPDZ [20] framework. While effective, the protocol suffers from high bandwidth requirements, as is typical for secret-sharing-based methods.

C Discussion

C.1 Real-world Applicability

The readiness of our FHE protocol for real-world deployment also faces significant constraints, necessitating a thorough discussion in this section to elucidate existing gaps and catalyse collaboration within the FHE community. Addressing open challenges crucial for privacy-preserving protocols in MEV mitigation is paramount.

Practical deployment in blockchain environments mandates considerations for several aspects not addressed in this version of the protocol. First, searchers often combine more than one transaction to generate novel MEV opportunities. Handling multiple transactions would necessitate further runtime reduction and the identification of batching possibilities as a scaling solution.

Second, Ethereum's inclusion requirement for signed transactions poses complexities. Searchers must inject their private keys as part of encrypted inputs, demanding ECDSA signature operations within the encrypted realm. This feature poses notable challenges due to the modular inverses intrinsic to ECDSA's signing algorithm.

C.2 Further Research Avenues

Our results are deliberately delineated per protocol stage to underscore the formidable challenge posed by the hybrid workload while highlighting prospects for leveraging sequential phases to bolster and rely on hybrid FHE schemes beyond TFHE alone. The optimal derivation of amounts could benefit from a protocol finely tuned for arithmetic computations. Our findings advocate further exploration into hybrid schemes akin to Chimera[9], capitalising on this aspect

of our workload while still enabling fast encrypted comparisons for the RLP extraction phase.

The protocol's current confinement to a single use case—limited to UniswapV2—underscores the necessity for expansion. Extending its scope to encompass diverse MEV forms, such as liquidations or other DEXs, presents an avenue to unearth novel challenges pertinent to FHE adoption within MEV-related workloads. This expansion is pivotal for comprehensive FHE applicability assessment in diverse MEV scenarios. It is however likely that to obtain the best performance out of an FHE protocol, we will need to implement a custom circuit for each new DEX or use case. Let us consider UniswapV3, which we have evoked earlier by exploiting its arithmetics to optimise our own protocol. It is harder to search for arbitrage opportunities on UniswapV3 compared to UniswapV2. As we have seen when exploiting its price and amount calculation equations, UniswapV3 has multiple liquidity pools for the same token pair with different fee structures, requiring arbitrageurs to analyze numerous pools to identify price discrepancies. Additionally, the concentrated liquidity in UniswapV3, where liquidity is limited to specific price ranges, makes it harder to predict when and where arbitrage opportunities will arise, unlike UniswapV2, which has a more uniform distribution of liquidity across all prices. This complexity makes it more challenging for arbitrageurs to find opportunities on UniswapV3, and would in turn introduce great challenges to implement a blind arbitrage protocol targetting V3 pools.