SoK: Fully-homomorphic encryption in smart contracts

Daniel Aronoff
daronoff@mit.edu
Massachusetts Institute of Technology

Adithya Bhat
aditbhat@visa.com
Visa Research

Panagiotis Chatzigiannis
pchatzig@visa.com
Visa Research

Mohsen Minaei
mominaei@visa.com
Visa Research

Srinivasan Raghuraman
srraghur@visa.com
Visa Research and Massachusetts Institute of Technology

Robert M. Townsend
rtownsen@mit.edu
Massachusetts Institute of Technology

Nicolas Xuan-Yi Zhang
nxyzhang@mit.edu
Massachusetts Institute of Technology

ABSTRACT

Blockchain technology and smart contracts have revolutionized digital transactions by enabling trustless and decentralized exchanges of value. However, the inherent transparency and immutability of blockchains pose significant privacy challenges. On-chain data, while pseudonymous, is publicly visible and permanently recorded, potentially leading to the inadvertent disclosure of sensitive information. This issue is particularly pronounced in smart contract applications, where contract details are accessible to all network participants, risking the exposure of identities and transactional details.

To address these privacy concerns, there is a pressing need for privacy-preserving mechanisms in smart contracts. To showcase this need even further, in our paper we bring forward advanced use-cases in economics which only smart contracts equipped with privacy mechanisms can realize, and show how fully-homomorphic encryption (FHE) as a privacy enhancing technology (PET) in smart contracts, operating on a public blockchain, can make possible the implementation of these use-cases. Furthermore, we perform a comprehensive systematization of FHE-based approaches in smart contracts, examining their potential to maintain the confidentiality of sensitive information while retaining the benefits of smart contracts, such as automation, decentralization, and security. After we evaluate these existing FHE solutions in the context of the use-cases we consider, we identify open problems, and suggest future research directions to enhance privacy in blockchain smart contracts.

KEYWORDS

smart contracts, fully homomorphic encryption, privacy

1 INTRODUCTION

One of the significant challenges of blockchain-based (but also generally all digital-based) transactions is the issue of privacy. The decentralized and transparent nature of blockchains especially is at odds with privacy, as all on-chain data is widely distributed and publicly visible, even when hidden behind pseudonymous addresses. Moreover, the immutable and permanent nature of blockchain records can exacerbate such privacy concerns. Once information is written into a block, it cannot be altered or deleted, which can potentially lead to permanent disclosure of sensitive information.

These privacy concerns inherently extend to blockchain smart contract applications. In traditional contract law, the terms and conditions of a contract are usually known only to the parties involved. In contrast, smart contracts are typically visible to all participants of the blockchain network. This transparency, while beneficial for verifying transactions and ensuring accountability, can inadvertently disclose sensitive information. This can include the identities of the parties involved, the value, and nature of the transactions, among other details, which can be exploited by malicious actors or even lead to competitive disadvantage in business scenarios.

Therefore, there is a need for privacy-preserving mechanisms in smart contracts, and this need has been growing even further after recent works [9, 40, 67, 68] have highlighted how smart contracts with those mechanisms can make novel protocols in economics possible. In short, privacy-preserving smart contracts would allow the benefits of this technology, such as programmability, decentralization, and security, ensure that sensitive information remains confidential, and facilitate new protocols in economics which are not feasible today.

Our contributions. In this paper, we perform a systematization on solutions that adopt fully-homomorphic encryption (FHE) as the main ingredient to enable privacy-preserving smart contracts. We first make a comprehensive study on existing FHE solutions in a smart contract setting, highlighting the nuances that might become a constraining factor in real-world deployments. We then discuss how FHE can uniquely make novel use-cases possible in economics that extend beyond the standard tokenized deposit use-case, and provide the protocols of such cases in detail. Given the above available tools and the use-cases, we examine if and how these tools can indeed realize these use-cases in practice by performing a comprehensive evaluation. Finally, based on our findings, we identify open problems and suggest future research directions.

1.1 What about other PETs?

In the context of smart contracts and economic applications, several privacy-preserving technologies (PETs) such as zero knowledge proofs, homomorphic encryption, trusted execution environment (TEE) and secure multi-party computation (MPC) have been explored to address these privacy challenges. However, it turns out that fully-homomorphic encryption (FHE) in particular, can

make new protocols in economics possible in a smart-contract setting [40, 67, 68], allowing the contract to make the needed computations over encrypted data directly. Other technologies in such scenarios might introduce undesireable tradeoffs for these cases. For instance TEEs shift the assumption from the consensus safety to trusted hardware, while general purpose MPC might assume non-collusion and liveness of servers instead of the liveness and the decentralized nature of a blockchain. Looking ahead, in Sections 4.3 and 4.4 we provide more reasons on why our use-case mechanisms are a better fit for FHE. Finally, FHE is in a unique position to address blockchain-related problems such as Miner Extracrable Value (MEV) [26], which is associated with centralization, fairness and security concerns [42], and is generally considered a desireable and sought-after technology in blockchain applications [31] as well as in machine learning [52].

1.2 Smart contract operations

One of the most common standards in Ethereum smart contracts is ERC-20 [71], a standardized framework for creating and handling tokens on the Ethereum blockchain. ERC-20 specifies a set of functions and events that a contract must implement to be considered compliant, allowing different tokens to be easily integrated into decentralized applications (dApps), wallets, and exchanges. It defines several key functions, such as `transferFrom(address, address, uint256)` which transfers tokens from one address to another, using the approved amount. However, `transferFrom()` does not preserve the privacy of the sender and receiver, nor of the amount. Since all transactions and contract interactions are publicly recorded on the blockchain, the sender's and receiver's addresses, the amount transferred, and other associated metadata are visible to anyone observing the Ethereum's blockchain. In fact, the amounts any participants hold of that ERC-20 token (represented as a mapping of amounts to addresses in the ERC-20 smart contracts) are public too.

One of the first proposed solutions to address this was Zether [19], adding confidential transactions capabilities into smart contracts. It introduced a layer of privacy using zero-knowledge proofs (ZKPs) and the additively-homomorphic variant of ElGamal encryption scheme, which enabled users to hide the transaction amount while still enabling the contract (and its validators) to validate the correctness of those transactions without needing to learn their details. However, it introduced additional complexity and resource requirements, increasing the gas costs per transaction. In the variant of Zether hiding the sender and the receiver as well (i.e., anonymity) beyond just hiding the transferred amount (i.e., confidentiality), is considered impractical for deployment in the public Ethereum blockchain, despite its subsequent efficiency improvements [29].

Regardless, approaches such as Zether do not provide any additional privacy-preserving functionalities on smart contracts besides token transfers. In fact, smart contracts have much more powerful functionalities such as digital bonds [3], or more advanced standards such as ERC1155, ERC1440, ERC2020 [1, 2, 50], etc. In addition, even in standard tokenized asset contracts, there is a need for additional functionalities beyond simply minting, burning and transferring assets, such as applying interest over balance sheets, exchange assets with different rates, etc. As a result, protocols such as Zether using additive homomorphic encryption fall short of enabling more advanced smart contract functions and fulfilling more complex use cases. Therefore fully-homomorphic encryption (FHE) would be required to realize these in a privacy-preserving way.

1.3 Related Works

Works in the Computer Science field.

A recent SoK paper [49] surveyed the potential of realizing privacy-preserving smart contracts with PETs such as homomorphic encryption, multi-party computation (MPC), zero-knowledge proofs (ZKPs) trusted execution environments (TEEs). However, this work is a much more high-level landscape overview without focusing on the details of actual deployments of these PETs in a smart-contract environment, and without considering advanced use-cases in economics beyond standard ones such as tokenizations or auctions. In contrast, our work has a narrower scope on applying FHE in a smart contract setting, considers existing implementations, and shows the potential of this technology to realize novel applications in economics. Another SoK work investigated the integration of PETs in blockchain applications, including smart contracts [7]. While this work extended beyond confidentiality in blockchains and considered applications which hide the computation itself ("Function Privacy"), it is limited to SmartFHE [58] as an FHE implemenation in smart contracts, without specifying how FHE can be deployed in practice in a smart contract environment or considering actual use-cases to further support the need of Function Privacy. Other works include Zkay [60] which extends Solidity smart contracts by data privacy annotations using additive homomorphic encryption and non-interactive zero-knowledge (NIZK) proofs, and ZeeStar [59], which provides a compiler to enable instantiation of privacy preserving smart contracts without substantial expertise, however it also only supports additive homomorphisms. Zexe [16] similarly implements a privacy-oriented scripting language for digital currencies similar to Zerocash [13], without however a direct support for stateful computations such as those in smart contracts.

In a more general setting, outside the field of blockchains and smart contracts, there is a plethora of works in implementing or improving FHE computations. A recent SoK paper [80] focused on the inefficiencies of FHE computations stemming from complex polynomial multiplications and maintenance operations such as bootstrapping, and how these can be improved using GPUs or Field Programmable Gate Arrays (FPGAs). In fact, several recent works have proposed methods for accelerating various FHE computations using FPGAs, such as [48, 61, 72] for the BGV FHE scheme [18], [36, 44, 62, 63] for the BFV scheme [17], [54, 73, 74] for CKKS [20] and [32, 45] for THFE [21]. These methods utilizing FGPAs can enable acceleration of FHE computations over a few orders of magnitude depending on the FPGA hardware, the encryption parameters and the FHE scheme itself. Other related works include a survey on the existing FHE compilers [70], implementing an open-source library (OpenFHE) which offers support for a wide range of FHE schemes such as leveled and bootstrappable FHE [11] and implementing libraries utilizing GPUs for efficiency [82]. Finally, [10] proposes the use of FHE to facilitate privacy in "Dark Pools", however with the absence of publicly available evaluation data it is unclear how this

would perform in a smart contract setting. Other papers followed an MPC approach for the same problem [23, 47].

Works in the Economics field.

There is a nascent literature in economics leveraging encryption for privacy-preserving computations. Previous research that has used MPC include a double auction for the Danish sugar beet market [15]; securely link Estonian education and tax databases [14]; a proposed method for protecting privacy in large-scale genome-wide association studies [37]; a simulation of a decentralized and privacy-preserving local electricity trading market [6]; an analysis of the gender wage gap in Boston using data from a large set of Boston employers [39]; use of FHE-MPC to collect financial risks [5] and cybersecurity data [27].

The combination of cryptography, distributed ledger technologies ("DLTs") and FHE can expand the frontier of feasible resource allocations. The unique elements that create this possibility are smart contracts, selective privacy and verifiable computing. The smart contract carries out instructions that are pre-agreed between participants, thereby preventing opportunistic ex-post renegotiation. Selective privacy overcomes trust deficits by controlling the disclosure of messages and information. Ensuring privacy of messages prevents third-parties from making opportunistic use of private information. FHE on a (trusted) DLT contributes to overcoming the trust deficit by enabling agents to verify that the smart contract has implemented the correct algorithm without revealing the underlying elements of the computations or the output. The combination of these attributes enable implementations of resource allocation schemes, carried out by the smart contract, that meet the incentive compatibility constraints of truth-telling revelation mechanisms [35]. We illustrate here with the concrete examples from MIT LEAD (2025) [40] of how smart contracts with FHE on DLT can improve resource allocation.

- We start with the implementation of a simple model from Lee, Martin, Townsend (2024) [41] which illustrates the differences between the "legacy settlement system" and a DLT-based one. In that simple model, a broker needs to decide whether he agrees to carry inventory (hence, financial risks) to facilitate trade between two parties which do not know of each other's coincidental needs, and who could renege or renegotiate the terms for this trade. In today's systems the broker has to earmark portions of its balance sheet to insure against these risks. In a DLT world the broker can enter into atomic and composed smart contracts with both parties, such that the asset never touches its balance sheet (as the asset can flow atomically along reverse payment flows from the buyer to the seller).
- We then show how a smart contract can implement a self-reporting insurance mechanism, with a version of the model first described in Townsend (1988) [66], then described with FHE in [67] and finally inscribed in the context of DLT and FHE in [68].
- We finally extend this example by adding a repayment leg, which corresponds to a repo market. The FHE on DLT solution presented here is the tokenized implementation of MIT LEAD (2025) [40] inspired by the economic model in Aronoff and Townsend (2022) [9].

All three examples show how FHE on DLT can potentially reduce frictions in financial intermediation by facilitating multilateral coordination with limited commitment. A privacy-preserving smart contract with verifiability of computation on a public blockchain solves trust issues thereby incentivizing agents to reveal their preferences to the smart contract. This, in turn, enables the design of smart contracts that simultaneously increase trade volume (and in the case of self-reporting insurance, make feasible an entirely new market) and reduce the level of inventories an intermediary must carry.

2 CRYPTOGRAPHIC PRIMITIVES - FULLY HOMOMORPHIC ENCRYPTION

We now provide a basic background on FHE as a cryptographic primitive and the schemes used by the smart contract FHE solutions.

A public key encryption scheme for a message space $\mathcal{M}$ is a triple of (probabilistic polynomial time) algorithms $(KeyGen, Enc, Dec)$ given by

- The key generation algorithm $KeyGen$ which, on input a security parameter 1^λ, outputs a pair of secret and public keys (sk, pk).
- The encryption algorithm Enc which, on input the public key pk and message $m \in \mathcal{M}$, outputs a ciphertext c. When clear from context, we suppress the input pk to Enc.
- The decryption algorithm Dec which, on input the secret key sk and a ciphertext c, outputs either a message $m \in \mathcal{M}$ or $\perp$.

We say that the encryption scheme is correct if

$$Dec(sk, Enc(pk, m)) = m$$

for all messages $m \in \mathcal{M}$ and key pairs $(sk, pk) \leftarrow KeyGen(1^\lambda)$.

We say that the encryption scheme is IND-CPA secure if for any probabilistic polynomial time adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins the following game is at most $\frac{1}{2} + negl(\lambda)$:

- Sample $b \leftarrow \{0, 1\}$, $(sk, pk) \leftarrow KeyGen(1^\lambda)$.
- Send pk to $\mathcal{A}$ and receive $m_0, m_1 \in \mathcal{M}$ from $\mathcal{A}$.
- Send $Enc(m_b)$ to $\mathcal{A}$ and receive $b' \in \{0, 1\}$ from $\mathcal{A}$.
- $\mathcal{A}$ wins if $b = b'$.

Let $\mathcal{F} \subseteq \cup_{w>0}\{f : \mathcal{M}^w \to \mathcal{M}\}$ be a set of functions over message tuples. A public key encryption scheme is $\mathcal{F}$-homomorphic if it has an evaluation algorithm $Eval$ which, on input the public key pk, a function $f \in \mathcal{F}$ with $f : \mathcal{M}^w \to \mathcal{M}$ for some $w > 0$, and a ciphertext tuple $\vec{c} = (c_1, \ldots, c_w)$, outputs a ciphertext c. We say that the homomorphic encryption scheme is correct if

$$Dec(sk, Eval(pk, f, \vec{c})) = f(\vec{m})$$

for all functions $f \in \mathcal{F}$ with $f : \mathcal{M}^w \to \mathcal{M}$ for some $w > 0$, message tuples $\vec{m} = (m_1, \ldots, m_w) \in \mathcal{M}^w$, key pairs $(sk, pk) \leftarrow KeyGen(1^\lambda)$, and $\vec{c} = (c_1, \ldots, c_w)$ where $c_i \leftarrow Enc(pk, m_i)$ for all $i \in \{1, \ldots, w\}$.

A homomorphic encryption scheme is fully homomorphic if it is $\mathcal{F}$-homomorphic, where $\mathcal{F}$ is the set of all (efficiently computable) functions.

A homomorphic encryption scheme is leveled homomorphic if all algorithms take in an auxiliary parameter ℓ, run in time polynomial in ℓ (and λ), and are (say) $\mathcal{F}_\ell$-homomorphic, where $\mathcal{F}_\ell$ is the set

of all "depth-ℓ computations". Since the details of this will not be pertinent to our discussion, we do not elaborate further and instead refer the reader to [34] for further details. The standard technique of bootstrapping (homomorphically decrypting and re-encrypting) can be used to turn leveled homomorphic encryption schemes into fully homomorphic ones. However, the bootstrapping is extremely time consuming.

There are several homomorphic encryption schemes known today. Of relevance to our discussion, the BFV encryption scheme [17] is a homomorphic encryption scheme based on the Ring-Learning with Errors (RLWE) assumption [43], while the TFHE encryption scheme [21] is a homomorphic encryption scheme based on the Learning with Errors (LWE) assumption [51]. Different homomorphic encryption schemes have distinct advantages. BFV is efficient when it comes to integer arithmetic, while TFHE is efficient when it comes to Boolean operations. Both support Single Instruction Multiple Data (SIMD) batching, which allows a vector of messages to be encrypted as a single ciphertext. TFHE can implement not only (linear) additions and multiplications, but also non-linear operations, e.g., ReLU (rectified linear unit) activations. Also, TFHE performs very fast bootstrapping after every homomorphic operation (a TFHE bootstrapping requires only about 10ms on a CPU). Thus, depending on the application at hand, one picks the appropriate encryption scheme with which to work.

3 SMART CONTRACT FHE SOLUTIONS

3.1 Sunscreen

Sunscreen implements an FHE compiler to enable web3 (and web2) engineers to write programs using FHE without requiring extensive knowledge of the underlying FHE mechanics (such as arithmetic circuits, polynomial parameters, etc.) while remaining efficient. The compiler is based on Microsoft's SEAL library [53] and uses the BFV-FHE scheme [18]. Its core cryptographic library [65] (written in Rust) implements the compiler (Rust decorator) to compile circuits into a program, a runtime to evaluate the generated program on encrypted values, and a codec to encrypt and decrypt 256-bit numbers. Sunscreen's technology also includes an Ethereum Virtual Machine (Rust EVM) [64] with adds two additional opcodes: `FHE_ADD` and `FHE_MULTIPLY`.

The SmartFHE framework [57, 58] is similar to Sunscreen because it utilizes the same underlying cryptography (e.g., BFV). The difference is that SmartFHE adds zero-knowledge proof systems to prove the properties of ciphertexts. Therefore, for our systematization purposes, we treat SmartFHE in a fashion similar to that of Sunscreen.

3.2 Zama

Zama offers a comprehensive software suite to make FHE accessible and practical. Zama's technology includes the following components:

- The core cryptographic library TFHE-rs (written in Rust) [78] which uses Fully Homomorphic Encryption over the Torus (TFHE) encryption scheme [21], and includes:
  - A codec to encrypt and decrypt 32-bit numbers
  - API to run FHE operations in Rust
- An SDK [24, 77] to develop and compile FHE enabled smart contracts in Solidity
- An SDK to develop and compile FHE programs written in Python [75]
- An Ethereum client (Go-lang) [76] that can understand execute the smart contracts.

Zama's cryptographic library offers a wide range of programmability features, such as high-precision encrypted integers (up to 256 bits), a full range of operators (such as '+', '-', '*', '/', '<', '>', '==' etc.), encrypted "if-else" conditionals to check conditions on encrypted states, on-chain PRNG to generate secure randomness without using oracles, unbounded compute depth for consecutive FHE operations and a look-up table optimization [22].

A vital feature also is "Configurable Decryption", which users can instantiate with threshold [25], centralized, or on a "Key Management System" (KMS) based decryption [79]. The purpose is to alleviate the problem of having a single "global" public key used for FHE operations, where a naive approach would require a single centralized private key contradicting the decentralized setting of a blockchain. Zama's KMS combines a threshold MPC protocol [25] to facilitate key generation and decryption, a Proof of Authority consensus [8] to ensure the integrity of decryptions, and TEEs to store private key shares. However, at the time of writing, KMS is still in the early stages and is not deployed by Zama.

Fhenix [81] is a similar framework built on top of Zama's TFHE-rs; however, it serves as a layer-2 blockchain solution rather than a layer-1. Therefore, we treat Fhenix similarly to Zama's layer-1 SDK for our systematization purposes.

4 APPLICATIONS

We now consider use-cases where implementations with FHE and smart contracts on distributed ledgers enable improvements in resource allocations that are not otherwise obtainable. After reviewing the basics of FHE applied to exchanges of tokenized assets in Section 4.1, we compare the performance of Sunscreen and Zama on trading protocols in two domains with significant economic impact. Section 4.2 presents a first and very general model of financial intermediation. Section 4.3 extends the previous model to a more automated situation without intermediaries, and links that to insurance. Section 4.4 extends the previous models with also future repayment legs, and links that to repo and collateral markets.

4.1 Some first use-cases with privacy

A straightforward application for FHE would be to implement privacy-preserving tokenized assets in smart contracts, such as ERC20, ERC1155 extensions [50, 71]. In such contracts, the asset values are hidden with FHE, therefore providing confidentiality for the sender and receiver. Essentially this approach would enhance existing confidential smart contracts [30, 56], i.e. it would not be limited to only adding and subtracting amounts (e.g. when minting, burning, sending or receiving assets) but also performing multiplication operations (e.g., applying interest, or performing computation as in Automated Market Makers (AMMs) [12]). The costs of these operations is typically associated with the cost of a single `add()` or `mul()`. Note that one could still implement such contracts using semi-homomorphic encryption instead of FHE, if

the expected `mul()` operations will be sparse - in this approach semi-homomorphic encryption would be used for everyday `add()` operations while ZK proofs would be used for showing correct transitions between states for `mul()` operations.

Other potential use-cases for privacy-preserving smart contracts include a collateral pledge between funds with ERC1155 token contracts [46], dark pools in securities trading [10, 23, 47], or information flows to deal with crisis contagion [55].

4.2 A model of intermediation without inventories

Lee, Martin, Townsend (2024) [41] describe a model where a smart contract on a DLT eliminates the requirement of intermediaries to pre-fund trades between clients. The model demonstrates how these technologies enable an expansion of trade and reduction of inventories at the same time. The model is as follows:

- Participant A has an asset that participant C desires.
- A and C do not interact directly with each other.
- A broker B intermediates between A and C, interacting with either one first with equal probability (50%).

If B interacts with A first, B must decide whether to take on risk and purchase the asset to sell it to C later (potentially profiting if C buys it or incurring losses if C does not). Conversely, if B interacts with C first, they can agree that B will acquire the asset from A for C. However, this leads to a hold-up problem. After B has purchased the asset from A, C could opportunistically reneg and lower its bid, knowing that B will suffer a loss if it is unable to resell the asset. In response, C could mitigate its risk of loss by lowering the price at which it is willing to purchase from A. This will reduce the volume of trade.[1]

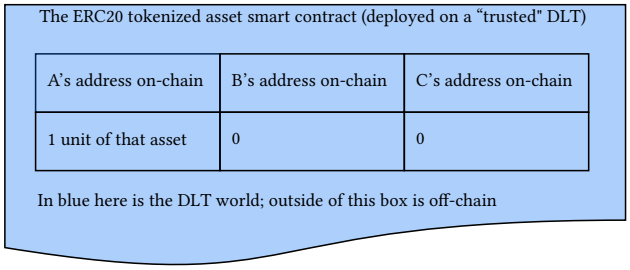
Building on the set up of Lee, Martin, Townsend (2024), [40] show that the hold-up problem can be eliminated and the trade volume increased if the following two conditions are obtained; (i) C does not know whether B has acquired the asset when it agrees to the price and (ii) B and C place the asset and money in escrow. There are two obstacles to implementing this solution. One is that B must trust that the escrow will not leak information to C. The other is that the transaction must be executed atomically, so that C's money is not locked in escrow without consummating a transaction. Under current technology trust is reputational. One way to overcome these obstacles is with an FHE smart contract that moves tokenized financial objects on a blockchain. In that case trust resides in the guaranteed execution of the smart contract and the guaranteed atomicity. [40] propose the following simplest smart contract implementation, which effectively leveraged blockchain-based programmability and privacy-preserving smart contracts:

Assume the tokenized asset resides on the blockchain as an ERC20 token. The ERC20 contract employs Fully Homomorphic Encryption (FHE), keeping allocations private. Initially, A owns the token, as shown in Figure 1:

A and B meet. They can agree on conditions for B to potentially sell to C later, but B may not want to commit to buying the asset outright. They draft a smart contract specifying the conditions under which B would buy the asset (e.g., a minimum price). A also

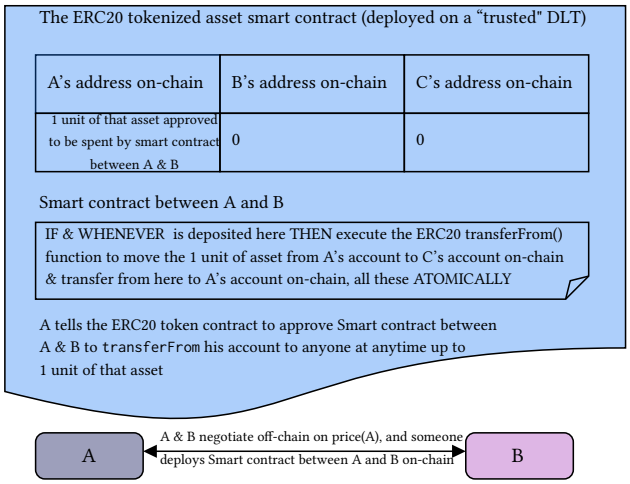
[1]There is an analogous hold-up problem in A's transaction with B. If A knows that B has a buyer (C) lined up, then A can strategically renegotiate its price with B.

Figure 1: Initial allocation before any trade.



authorizes the ERC20 smart contract to transfer 1 unit of the asset from A to B when the conditions are met, as illustrated in Figure 2:

Figure 2: Smart contract negotiated between A and B

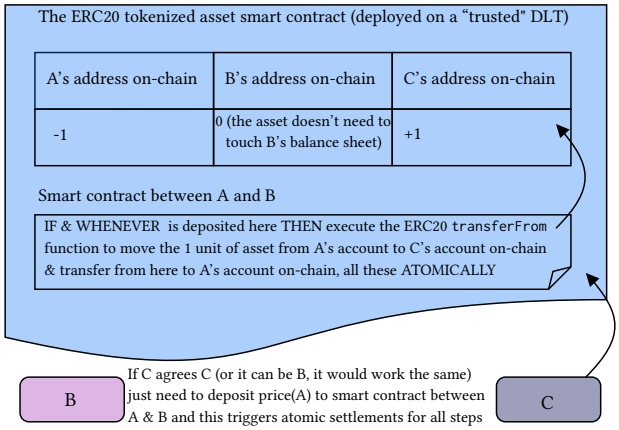


B and C meet later. If they agree on a deal, C deposits the price into the smart contract B drafted with A. This smart contract will then transfer 1 unit of the asset from A to B. The transaction can be made atomic, ensuring that either both transfers (price from C to B and asset from A to B) occur simultaneously, or neither happens. This atomicity can also be extended to the transfer of the price from C to B and the asset from B to C. This mitigates the risk of reneging and incentivizes all participants to announce their true prices, as illustrated in Figure 3:

4.3 Smart contracts to replace B of the previous example, and links to the insurance industry

One can ask the question of how A and C could negotiate directly without having B in the middle. This is similar to a problem asked in Townsend (1988) [66] and Townsend and Zhang (2020) [67] aiming at creating incentive compatible "self-reporting" contracts. Indeed, we want A and C to share with each other their true preferences, so that the trade they end up with is jointly optimal. This problem is quite general; in fact, this forms the basis of negotiating the terms of any risk sharing contract between adverse parties. Here

Figure 3: B and C using the smart contract negotiated between A and B



the specificity of Lee, Martin Townsend's model is that to encourage C to share its real preference, A must not be able to know how much C wants the asset, so that A doesn't take advantage of that knowledge. So this "self-reporting" contract must also be non-verifiable, e.g. A shouldn't be able to verify whether C is in bad need for that asset. This can be related to the insurance industry, in which historically contracts have required that claims must be verifiable e.g. building damage, hospitalization, or drop in asset price. It is desirable however to provide an insurance mechanism for unverifiable claims, or for claims that participants would want to keep private even if these could be verified by third parties (for instance when a bank that has a liquidity shortfall). Examples of such claims could be "the data I got hacked is worth $\$x$ to me" or "I held $\$y$ of Bitcoin and lost my secret key". Insuring such claims is difficult since self-reporting of loss creates incentives to either overstate claims (for instance to get more insurance payouts), or to understate claims (for instance if reputation could be affected negatively). An especially impactful area of potential application of self-reporting claims is the provision of loans and contingent money transfers to low wealth people in amounts that are too small to justify incurring the cost of verification [38]. A key feature that has, up to the present time, prevented the development of a self-reporting insurance market is the difficulty of designing a market mechanism wherein a policyholder is assured that its claim will not be revealed to any party, including the insurer.

We delegate the details of the exact incentives to [66] [2] and trust here (from the results of [66]) that C, who is now equivalent to a policyholder, reveals their true need for the asset A has, in one of either two prices for simplicity - a high need h which is higher than the price A wants to sell the asset, and a low need l which would be lower than the priccec A wants to sell the asset.

The FHE smart contract scheme description. As in Lee Martin Townsend (2024), there are two time periods, time 1 before A and C meets, and time 2 for delivery of the asset if the trade was agreed upon; two agents, C (now akin to a policyholder) and A (now akin to an insurer, who deploys a smart contract to negotiate with C). As stated above, C has two policyholder states (ie of how much C needs the asset, akin to whether C incurred high losses requiring high disbursement from the insurer, or low losses requiring less disbursement from the insurer) h and l, where $h > l$. At time 1 the agents agree to contract terms that will be encoded (for instance by equation 6 from Figure 4 with all the encrypted parameters into the smart contract's *transferFrom()* function), which determine the contingent payouts at t_2, (h or l). At time 2 the policyholder C messages the insurer A loss claim h or l (which, by assumption, is its true state)[3] and the payout is a random function of the policyholder claim. The high need, h, which is higher than the price A wants to sell the asset, is imperfectly correlated with the policyholder reporting a large claim (i.e. there is a small probability the policyholder will receive the asset even for a low price or in the insurance analogy in the case of claim of small loss). This is a salient point of [66], as this small probability, encoded as a random variable in the smart contract, that the policyholder receives the asset regardless of its true need, which makes it impossible to infer with certainty the policyholder's state. A accepts this small probability to provide the incentives for C to announce its true need without fearing that A can infer it (and say broadcast to competitors that C needs that asset badly). The smart contract algorithm is the following:
The smart contract payout formula is $Enc(h).Enc(b) + [Enc(1) - Enc(b)].(Enc(l) + [Enc(h-l)].Enc(r_3))$, where r_3 is the Bernoulli random variable deciding if even a good state of the world will lead to high payout[4], and where b is sent by the policyholder with a value 1 if they are in a bad state of the world, and 0 if not.

This encoding of the terms into the smart contract is shown as step 1 on Figure 4. Then, still at the first contracting period, the insurer needs to make a deposit[5] (in the forms of ERC tokens for instance, or tokenized deposits, or wCBDC) to the smart contract that can accommodate the maximum payout possible corresponding to those 2 states of the world, i.e. max(h,l)=h (step 2 on Figure 4), to ensure that the smart contract can make the payment to match the loss reported by the policyholder at time 2.

At the second period, the policyholder sends to the smart contract an FHE encrypted dummy value b, corresponding to 1 if the policyholder is in the high loss state of the world, and to 0 if the policyholder is in the low loss state of the world (step 3), as well as a random bit r_1 (step 4) sampled uniformly. The contract then adds another layer of randomization through the sampling of r_2 (step 5), which can be an external randomness source, to ensure that neither the policyholder nor the insurer can bias[6] the final random bit r_3

[2] Appendix A.1 contains a description of the model in [66].

[3] We limit the policyholder to 2 states WLOG. Townsend (1988) [66] shows the state space can be an arbitrarily large number of discrete losses.

[4] Townsend and Zhang (2020)[67] shows how this random value should be drawn by the "central planner" under FHE encryption given a fixed probability distribution. Here this randomization is proxied via sampling of randomness generated by both parties, due to limitations of smart contracts not being able to generate randomness on their own. In general, the underlying distributions and the necessary operations can be picked to fulfill the requirements of [67].

[5] In the case of negative payouts such as insurance premium, the policyholder is required to make a deposit

[6] Townsend (1988) [66] designs a self-reporting insurance contract where policyholders have an incentive to truthfully report their loss state. A key element is the randomization of payouts, which blocks inference about policyholder loss reporting. This ensures the integrity of loss reporting by preventing policyholders from gaming the system with their own loss reporting.

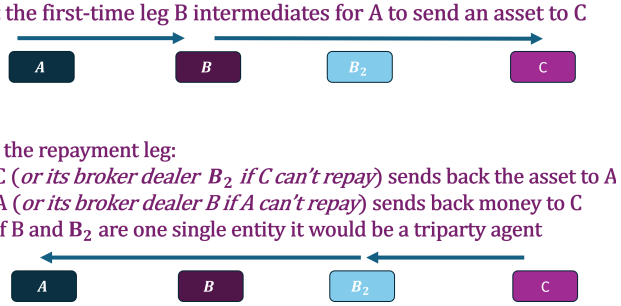
(step 6). Finally, the smart contract through the homomorphic operations shown step 6 in Fig. 4 computes the payout and transfers the corresponding amount to the policy holder. Note that the random probability that the low losses state to the world also leads to a payout h obscures to the outside observers if the high payout was indeed due to a high or low loss state of the policyholder.

The mechanism requires three things to make it viable; (i) privacy of the messages sent from agents to the mechanism operator, (ii) trust that the mechanism operator will not leak information on one agent's message to another agent and (iii) trust that the mechanism operator will implement the agreed upon algorithm to determine and send money to agents. These three requirements are met when using FHE-based smart contracts on a public blockchain: Encrypting messages solves (i). FHE computation on the ciphertext by the smart contract solves (ii). Verification of the smart contract computation on a public blockchain solves (iii) (contingent on trust in the integrity of the blockchain).

4.4 Adding a repayment leg: a repo trade with a coordination problem

If we add a future repayment leg (the "second-leg") to our model, so that A repurchases the asset, we are describing a repo market. The smart contract delineating the terms of sale of the asset for the first leg (from section I) can now also include a second transferFrom() (the standardized ERC20 function), this time to be activated at the second-leg. The second-leg introduces a complication. At least one of A or B is motivated to enter into a repo trade in order to use the financial object it acquires at the first-leg. This means that, say C, will have transferred the object and the second transferFrom() function in this smart contract would have nothing left to withdraw from C's account. One possible function of a broker-dealer in a repo trade is to ensure delivery of the object to its client at the second-leg. To bring the example closer to the empirical structure of repo market we introduce a second-broker-dealer, as in the economic model in Aronoff and Townsend (2022) [9] used for the FHE on DLT presented below, from MIT LEAD (2025) [40] (inspired similarly by Aronoff and Townsend (2022) [9]). A is then the client of B_1 and C is the client of B_2.

Figure 5: Illustration of the payment and the repayment legs, with different "insurers" - eg broker dealers - for each leg

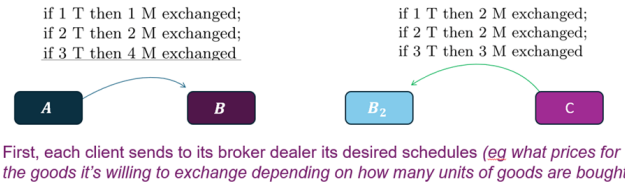


Protocol description. A is now a repo borrower who wants to sell a financial asset, T, in exchange for money, M, at the first-leg and who wants to repurchases the asset at a the second-leg.

So A owns the financial asset T and desires to sell it. C desires to purchase T in exchange for money M. A trades with its broker-dealer B and C trades with its broker B_2. The intermediated chain is depicted in Figure 8. We set as the objective to trade to maximize the volume of T at which the two broker-dealer price/volume pairs match. The protocol proceeds in the following sequential order:

Step 1, t_1 [off-chain, or in separate sets of on-chain smart contracts]: each client and its broker-dealer agree to a schedule of money, M, and volume of financial asset, T, denoted $\{M,T\}_{c,i}$ where $c = \{A, C\}$ indicates the client and $i \in \{1, ..., n\}$ indicates how many units M (resp. how many units T) the client is willing to trade for i units of T (resp. for i units of M). For simplicity and without loss of generality we assume both schedules have the same size n. Below we represent a visual example where n=3, with two examples of schedules from A and C. A client $c = \{A, C\}$ agrees to transact at any pair $\{M,T\} \in \{M,T\}_{c,i}$ that gets matched in the inter dealer market as in steps 2 to 4.

Figure 6: Steps 0 and 1

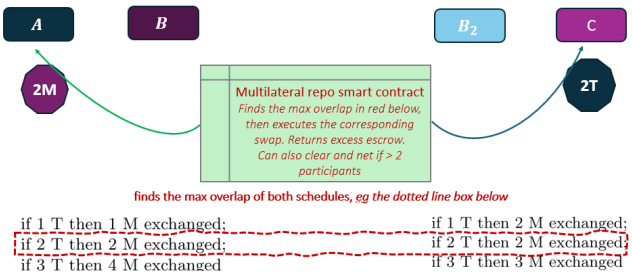


Step 2, t_2: [on-chain] Each broker-dealer then encrypts the schedule of its client and sends $Enc(pk, \{M,T\}_{c_d,i})$ $i \in \{1, ..., n\}$ to the smart contract. To ensure compliance, a broker-dealer may be required to send a deposit of the financial object it intends to trade into the smart contract escrow, and the client may be the source of the deposit object.

Step 3, t_3: [on-chain] the smart contract compares the encrypted schedules under FHE and selects from the two schedules the $\{M,T\}$ that matches between the two broker-dealers.

Step 4, t_4: [on-chain] B and B_2 send their requisite financial objects to the smart contract (if not sent already at Step 2) and the smart contract swaps the objects to A and C, and sends excess balances back to the broker-dealers if there are any.

Figure 7: Steps 2,3 and 4



There are two key insights to be learned from this exercise. One insight that using FHE to compare the trading schedules ensures

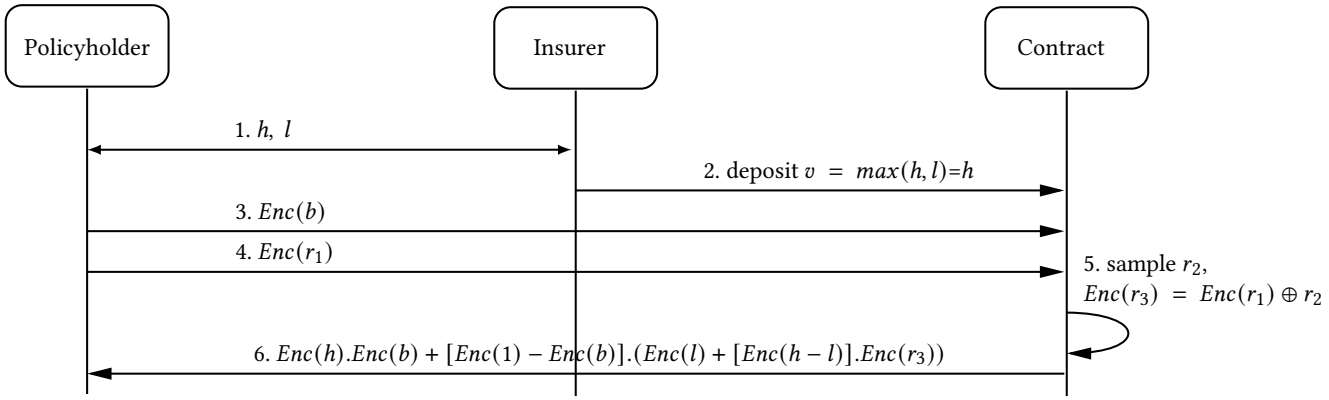


Figure 4: Policy holder - insurer protocol.

that broker-dealers can protect the privacy of their client information, which overcomes their reluctance to reveal their trading preferences. The other insight is that the smart contract encourages broker-dealer participation to solve the coordination problem and enables the attainment of a socially, higher desired trading volume.

5 EVALUATION

With the motivation for the two economic use cases described in sections 4.3 and 4.4 that utilize FHE and smart contracts, we perform a series of evaluation experiments using the solutions available today as discussed in section 3. We first perform microbenchmarks for all operations and all data types offered by both. Then, we benchmark the FHE smart contract operations costs for each use case and the FHE smart contract solution.

Environments. We used two environments for our experiments: (i) a CPU environment that uses a 16-inch Macbook Pro (2021) with 32 GB memory and 1 TB Hard Disk running Sequoia 15.1, and (ii) a GPU environment on AWS using `p2x.large` instance running Amazon Linux OS. We used the versions of the Rust libraries [65, 78] used in production by Sunscreen and Zama on their test networks. For Sunscreen, sometimes we use an *enhanced* version that adds more operations to the production version.[7] For Zama, we use the last version (v0.5), which compiles without errors in the GPU environment.

In our evaluations, we ignore the gas costs of the FHE operations in the respective systems and instead focus on the CPU running time as a measurement metric. We do this because the gas costs used in the existing test networks are arbitrary and unrelated to the CPU measurement costs. We used the Criterion Benchmark framework [28] to generate data points for our benchmarks.

5.1 Sunscreen Performance

Sunscreen takes a Rust function as input and creates a circuit after compiling it. They call these circuits 'applications'. The applications define the parameters from which the private and public keys are generated.

In their test network, they compile two functions: (i) a function that adds two encrypted 64-bit integers, and (ii) a function that multiplies two encrypted 64-bit integers. They expose these two functions to smart contracts via the precompiled contracts interface provided by Ethereum. We use these functions in our "production" version of the benchmarks.

We create a second version called the "enhanced" version that adds a subtraction function, a negation function, and scalar versions of the addition, subtraction, and multiplication functions. Using the compiled circuits, we can generate keys, encrypt inputs, run the circuit on encrypted inputs, and decrypt the outputs.

We evaluated the time consumed for all supported operations: key generation, encryption, addition, multiplication, subtraction, negation, and decryption. We benchmark the above across the six data types offered by Sunscreen: Signed, Rational, Fractional64, Fractional128, Fractional256, and Fractional512.

Micro Benchmarks. We present the results of our microbenchmarks in fig. 9. We observe that the key-generation operation takes the longest time. We also observe that the homomorphism property is usually very efficient for one operation, which is the addition operation in this case, and expensive in the other (multiplication). We observe that the Rational type is the most expensive operation. This is because this type supports division by encrypted ciphertexts, but as a result requires choosing larger parameters for the circuit, i.e., application, resulting in poorer performance. Finally, we observe that *enhancing* the application with more operations did not affect the performance by statistically significant amounts. Thus, we use this version of the circuit for the other parts of the benchmarks.

5.2 Zama Performance

Zama has a publicly available test network where $n = 1$ and $t = 0$ threshold servers implement the FHE operations. We used these system parameters for our experiments. Zama supports unsigned and signed integers of 8, 16, 32, 64, 128, and 256 bits, and we present microbenchmarks for them.[8] We also evaluated them in the CPU and GPU environments.

Micro Benchmarks. We present the results of our microbenchmarks in fig. 10. Like Sunscreen, we observe that the multiplication

[7] We are not sure why Sunscreen chose not to add these features on their test networks.

[8] In our benchmarks, the GPU version of the Zama library failed when running operations involving integers.

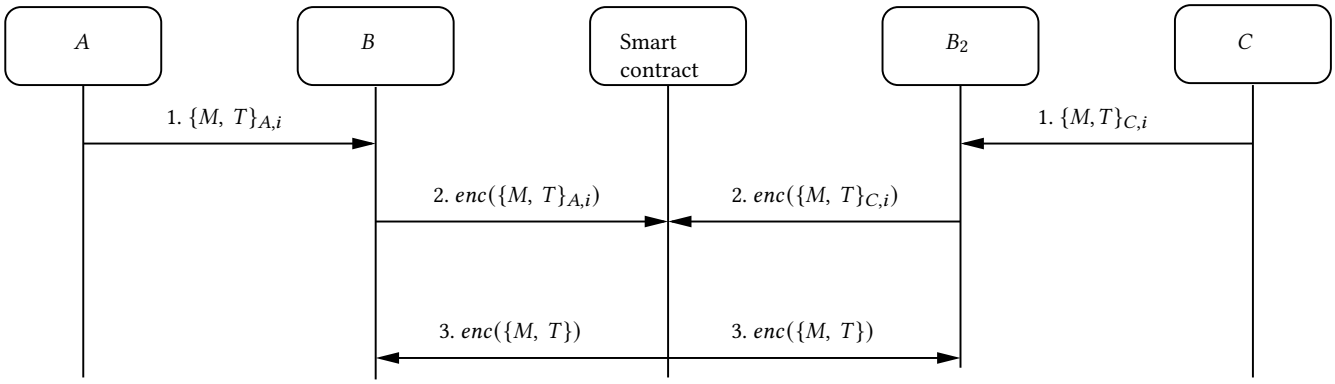


Figure 8: Intermediated Markets protocol. Note this figure represents the determination of contractual terms of trade, not the exchange of financial objects.

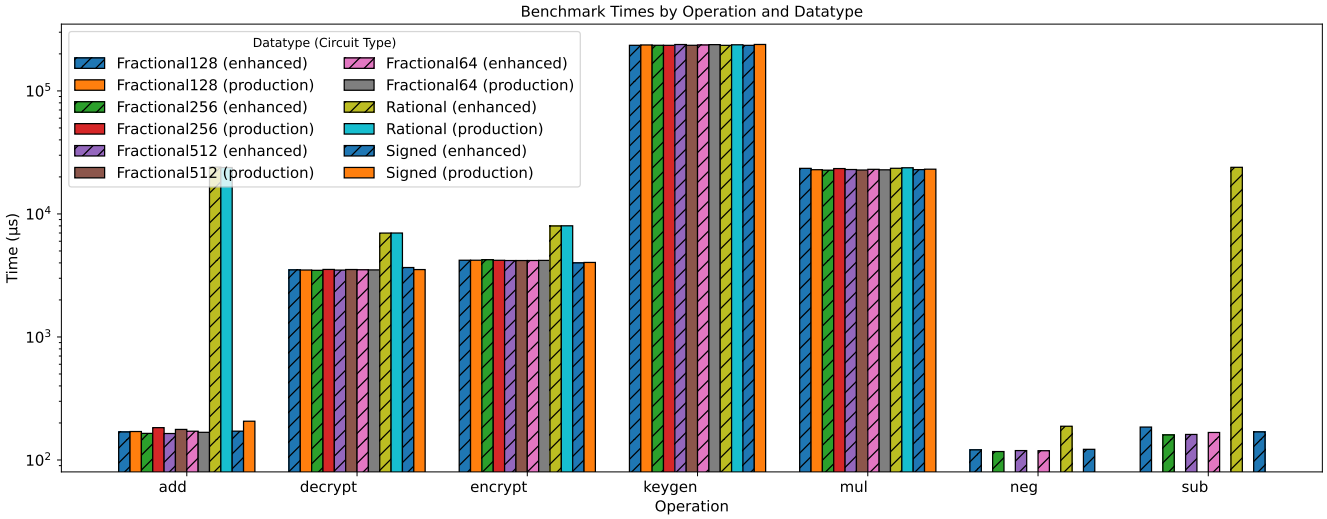


Figure 9: Microbenchmarks for all operations and all data types supported by Sunscreen

operation is more expensive than the addition operation. We observe that the cost of using larger bit sizes in data types increases the running times of operations. Another important observation is the effect of GPU acceleration; while GPU indeed provides substantial benefits for high-precision data types (roughly one order of magnitude benefit for u256 or i256), the benefits for low-precision data types is negligible, if any.

5.3 Comparison between Zama and Sunscreen

Both Zama and Sunscreen support operations over 64 bit-signed integers. Both of them also support key generation, encryption, decryption, addition, subtraction, negation, and multiplication over it. For the GPU environment, we used the unsigned 64-bit data type. We measured the performance of these common operations and presented our results in fig. 11.

We observe that the operations in Sunscreen are substantially cheaper when compared to the operations in Zama. This is due to the fact that Sunscreen uses BFV which is based on RLWE, working directly with integers; hence more efficient at integer arithmetic. Zama, on the other hand, uses TFHE, which while better at boolean operations and nonlinear operations, is less efficient when it comes to integer arithmetic and is hence less suitable with regards to our applications. We also see the impact of GPU acceleration on improving performance in general.

5.4 Unverifiable losses protocol

We now benchmark an FHE smart contract that insures unverifiable losses as discussed in section 4.3. While the most appropriate data type for this use-case would be an unsigned 8 bit integer (as it requires an encryption of a single bit), we benchmark other data types as well for a fair comparison (Sunscreen only supports 64 bits). We present our benchmark results in fig. 12. Although these overall indicate that the application is within practical limits for a permissioned blockchain (which typically has few validators controlled by

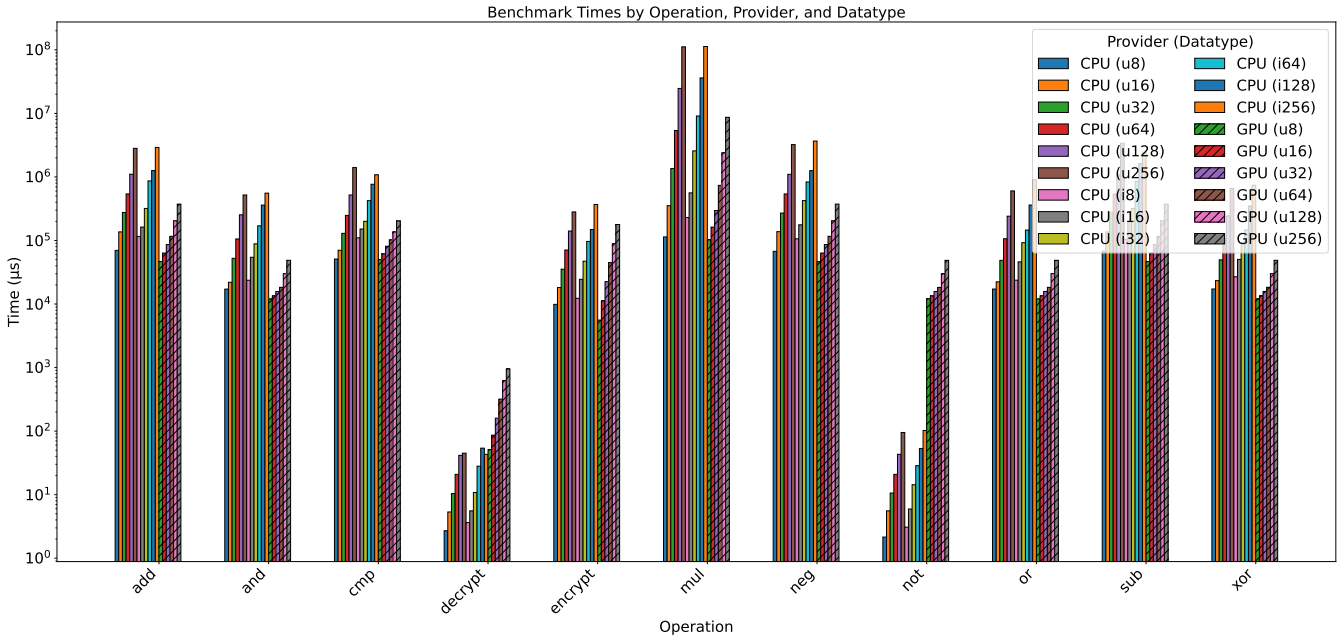


Figure 10: Microbenchmarks for all operations and all data-types supported by Zama

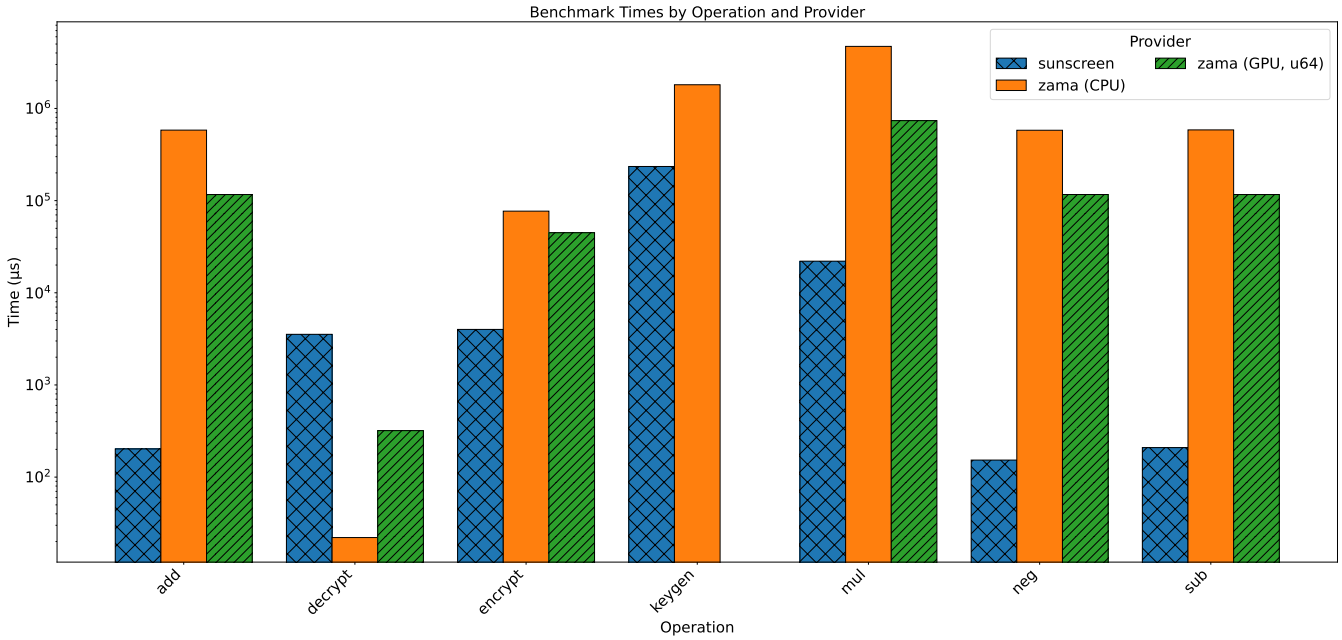


Figure 11: Comparing Zama and Sunscreen for common data type: Signed 64-bit integers and common operations.

the blockchain consortium), the smart contract computation for a permissionless blockchain is still considered expensive. In fact, we estimate Sunscreen would require about 4 million gas[9] (or about $80 based on today's gas fees and exchange rates), while Zama would exceed the maximum gas per Ethereum block (30 million), which would not be possible to deploy in a permissionless Ethereum blockchain as of today. We also observe that GPU acceleration in Zama is not offering any substantial benefits, as the use-case only

[9]This is based on a rough translation of computation time to gas costs given that one keccak256 hash takes about 0.8 microseconds on a typical Macbook and costs about 40k gas in Ethereum blockchain.

requires encrypting low-bit data types where GPU is not helpful, as discussed in section 5.2.

5.5 Coordination in an intermediated market protocol

We estimate the timing costs of the use-case discussed in Section 4.4, in particular, the cost of calculating and outputting the matches. For efficiency purposes, we let the broker-dealers A_d and B_d sort the $\{M, T\}_{c,i}$ pairs before encrypting them and submitting them to the contract. Note that it is reasonable to assume the broker-dealers will perform the sort honestly, as they do not have any incentive to do otherwise. Therefore, the contract would make FHE comparisons in a descending order, and would stop at the first match which would, as explained in Section 4.4, maximize the joint profit. For our benchmarking purposes, we take a sample of 100 sorted matches of type 32 bit unsigned integers in Zama (note comparison is only supported in Zama).[10] We find that comparisons for these parameters would takes about 25 seconds on average using CPU, which again might be feasible in a permissioned blockchain, but not scalable or deployable in a permissionless blockchain like public Ethereum (as of today).

Operation	Zama (CPU)	Zama (GPU)
Generate Orders	7.4777 s	4.4861 s
Match Orders	25.336 s	16.215 s

Table 1: Benchmarking the order matching application for Zama using encrypted 32-bit unsigned data type on CPU and GPU systems.

6 INSIGHTS AND RESEARCH GAPS

Having considered the available FHE implementations in Section 3, the use-cases in Section 4 and our benchmarks in Section 5, we now provide our overall findings in the form of research insights, as well a number of interesting research directions in the space of FHE used in smart contracts as a privacy tool, in the form of research gaps.

One of the first nuances with FHE operations in smart contracts is the issue of key issuance and management. All encryptions currently need to be performed under a "global" public key in order to make homomorphic computations possible. However, a single centralized party holding the corresponding private key would inherently defeat the whole of having decentralized computation with smart contracts. Therefore, a natural first approach is to distribute key shares among some of the blockchain validators.

Insight 1. *Both in Sunscreen and Zama, the validators hold the shares of the secret key for a global public key. Although a (n,t) threshold structure is described, both are implemented only for $n = 1$ and $t = 0$. It is unclear at this point if for $n > 1$ and $t > 0$ if it will be scalable and decentralizeable.*

Gap 1. *FHE additions and multiplications are performed under a single global key. Having protocols to perform these operations with a mixed key would make even more unique cases possible, and would not require distributing key shares to validators or other parties.*

While Zama's KMS works as an approach to avoid needing a single private key, the question remains how the MPC parties would first be selected and then granted the authority to keep those shares. Those parties/validators would likely have a much greater incentive to collude compared to a standard MPC wallet - here it is not just a wallet's assets at stake, but the whole FHE ecosystem, and having validator reputation as the single criterion is not sufficient to ensure security.

Gap 2. *While combining a "Proof of Authority" with threshold MPC would avoid having a single centralized private key, the question remains on how the validators in the proof of authority consensus would be selected, in a way that would be acceptable by all participants in a permissionless setting. More research is required to make such approaches compatible with a decentralized Proof of Stake blockchain ecosystem.*

We now consider the results of our experiments from Section 5, where we observe significant variations between the available solutions for their corresponding data types.

Insight 2. *For some data types, FHE operations are more efficient in Sunscreen than Zama. On the other hand, Zama has more rich data types and operations available for use.*

While Sunscreen is more efficient, because it does not support comparison, we could not use it to implement the coordination in intermediate market application. In general, the richness of the operation (or lack thereof) determines the number of supported applications.

From considering the use-cases and the benchmark results presented in section 5, and our rough estimates on the needed gas costs, we see that making the needed computation by smart contracts would be infeasible in a permissionless blockchain such as Ethereum. However these use-cases would still be feasible in a permissioned blockchain run by a few validators, where gas costs is not an issue.

Still, this is merely an estimate based on other computations and their gas costs as of today. It remains to be seen what the community will consider reasonable gas costs for such computations, and how these will evolve. However, validators utilizing specialized hardware suitable for FHE acceleration such as GPUs or FPGAs might change those costs dramatically.

Insight 3. *The gas costs in both Zama and Sunscreen are arbitrarily defined. Actual costs will depend on the community supply and demand after deployment. Potentially, FHE operations can be accelerated by special hardware and therefore gas costs can be improved.*

Introducing such hardware however in a permissioned blockchain is a double-edged sword. While such hardware would make FHE (and the desired use-cases we presented) possible, that would come at the cost of substantially raising the bar for the required hardware

[10] Our choice of 100 $\{M, T\}$ pairs was motivated by the observation that the dispersion of quoted rates in the US Treasuries repo market, and prices in the US Treasuries secondary market - which are priced in terms of yield - have historically been within this range, except during financial crises. Rates and yields are quoted at intervals ("ticks") of 1/32 of a percentage. 100 ticks exceeds 3%. The spread between the 1st percentile and the 90th percentile of daily repo rates reported to the Federal Reserve Bank of New York has not exceeded 0.4% from January 1, 2021 - November 27, 2024 [4].

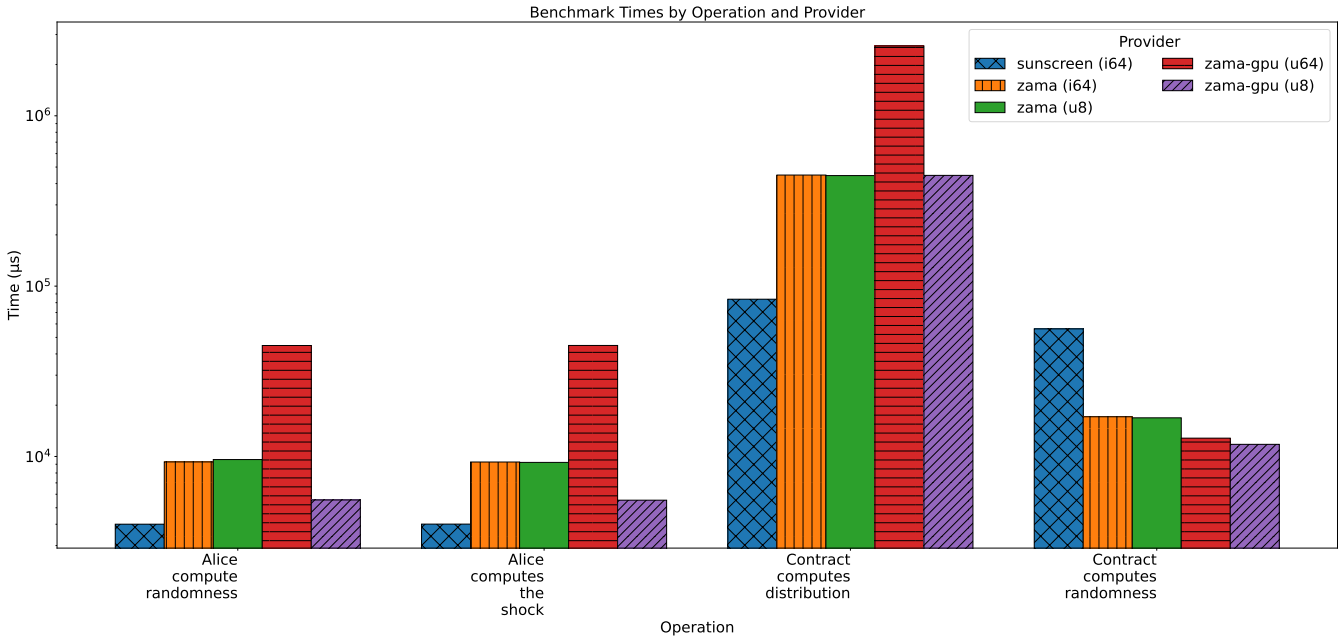


Figure 12: Benchmarking the operations for the unverifiable losses protocol.

from the validators (as of today, more than 1 million validators maintain the Ethereum blockchain). This would make participation in the proof-of-stake consensus even harder for the typical user, as joining the validator pool would require not only staking Ethereum coins, but also a substantial invenstment in hardware (a validator today requires relatively basic hardware). This would inadvertently lead to a higher degree of centralization, more energy usage from the Ethereum ecosystem, and eventually defeating the purpose of the proof-of-stake consensus algorithm.

Gap 3. *Based on our benchmarks, FHE would not be scalable today in smart contracts. While GPUs, FPGAs and ASICs could accelerate FHE computations, validators would need to adopt such hardware, potentially leading into more centralization in the system and defeating the original purpose of Proof of Stake. More research is required to accelerate FHE in smart contracts without inducing such centralization. At this point, it is unclear if true decentralization is possible, i.e, implement FHE smart contracts in a permissionless setting.*

We also notice the absence of mechanisms to verify the correctness and integrity of submitted FHE ciphertexts to smart contracts. Existing solutions like Sunscreen and Zama operate under the implicit assumption that users provide well-formed ciphertexts representing valid plaintext values. However, in adversarial settings, malicious actors could submit malformed or invalid ciphertexts—such as random data or ciphertexts encoding erroneous values, with the goal to manipulate the contract's execution or produce invalid results. This highlights the need for verifiable FHE schemes that allow smart contracts to ascertain the validity of ciphertexts without compromising their encrypted nature. There exist potential approaches to address this issue, such as zero-knowledge proofs (ZKPs) to enable users to prove certain properties about their ciphertexts (e.g., correct formation or bounds on the plaintext) without revealing any sensitive information, leveraging trusted execution environments (TEEs), or potentially integrating other cryptographic tools such as homomorphic Message Authentication Codes (MACs) and homomorphic signatures [33, 69]. Nevertheless, the combination of FHE and such tools poses challenges in terms of computational and communication overhead, and the extent of this overhead remains largely unexplored. Developing efficient verifiable FHE protocols is thus an open research area critical to ensuring secure and reliable use of FHE in smart contract applications.

Gap 4. *Further research is needed to design homomorphic authentication schemes that minimize computational overhead and are fully compatible with existing FHE systems to enable secure and scalable verification of encrypted data in decentralized applications.*

7 CONCLUSION

In this paper, we explored the intersection of fully-homomorphic encryption (FHE) and smart contracts towards the support of privacy-preserving computation in blockchains. Our systematization highlights the critical need for privacy-preserving mechanisms in smart contracts, especially for advanced economic use-cases that require performing decentralized, secure and confidential computations. Through our comprehensive systematization of currently available FHE-based approaches in smart contracts, we examined the potential of these methods to maintain the privacy of sensitive information while retaining the benefits of smart contracts, such as automation, decentralization, and security.

We then identified several key insights and research gaps that pave the way for future work in this domain. Key insights include

the challenges of key issuance and management in FHE operations, the varying efficiencies of current FHE solutions, and the significant gas costs associated with FHE computations on permissionless blockchains like Ethereum. These findings underscore the complexity of integrating FHE into smart contracts and the need for further innovation to achieve scalable and decentralized privacy-preserving solutions. We also identified several research gaps, such as the need for protocols to perform FHE operations with mixed keys, the challenge of selecting validators in a Proof of Authority consensus, and the potential centralization risks posed by specialized hardware for FHE computations.

Overall, our study provides a foundation for future research to enhance privacy in blockchain smart contracts. By addressing these research gaps and continuing to innovate in the field of FHE, we can move closer to realizing the full potential of privacy-preserving smart contracts in decentralized applications and making advanced use-cases a reality.

DISCLAIMERS

Case studies, comparisons, statistics, research and recommendations are provided "AS IS" and intended for informational purposes only and should not be relied upon for operational, marketing, legal, technical, tax, financial or other advice. Visa Inc. neither makes any warranty or representation as to the completeness or accuracy of the information within this document, nor assumes any liability or responsibility that may result from reliance on such information. The Information contained herein is not intended as investment or legal advice, and readers are encouraged to seek the advice of a competent professional where such advice is required. These materials and best practice recommendations are provided for informational purposes only and should not be relied upon for marketing, legal, regulatory or other advice. Recommended marketing materials should be independently evaluated in light of your specific business needs and any applicable laws and regulations. Visa is not responsible for your use of the marketing materials, best practice recommendations, or other information, including errors of any kind, contained in this document. All trademarks are the property of their respective owners, are used for identification purposes only, and do not necessarily imply product endorsement or affiliation with Visa.

REFERENCES

[1] 2018. ERC-1440: Security Token Standard. https://github.com/ethereum/EIPs/issues/1440

[2] 2019. ERC-2020: Reversible Transactions. https://github.com/ethereum/EIPs/issues/2020

[3] 2023. BME Digital Bond. https://www.iberclear.es/docs/docsSubidos/Paper-Digital-Bond-20july.pdf

[4] 2023. Secured Overnight Financing Rate Data. Technical Report. Federal Reserve Bank of New York. https://www.newyorkfed.org/markets/reference-rates/sofr

[5] Emmanuel A Abbe, Amir E Khandani, and Andrew W Lo. 2012. Privacy-preserving methods for sharing financial risk exposures. American Economic Review 102, 3 (2012), 65–70. https://doi.org/10.1257/aer.102.3.65

[6] Aysajan Abidin, Abdelrahaman Aly, Sara Cleemput, and Mustafa A. Mustafa. 2016. An MPC-Based Privacy-Preserving Protocol for a Local Electricity Trading Market. In Cryptology and Network Security, Sara Foresti and Giuseppe Persiano (Eds.). Springer International Publishing, Cham, 615–625.

[7] Ghada Almashaqbeh and Ravital Solomon. 2022. SoK: Privacy-Preserving Computing in the Blockchain Era. In 7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022. IEEE, 124–139. https://doi.org/10.1109/EUROSP53844.2022.00016

[8] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. 2018. PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain. In Proceedings of the Second Italian Conference on Cyber Security, Milan, Italy, February 6th - to - 9th, 2018 (CEUR Workshop Proceedings, Vol. 2058), Elena Ferrari, Marco Baldi, and Roberto Baldoni (Eds.). CEUR-WS.org. https://ceur-ws.org/Vol-2058/paper-06.pdf

[9] Daniel Aronoff and Robert M. Townsend. 2022. Essays on Incentive Designs to Improve Market Performance. Ph. D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA. https://dspace.mit.edu/handle/1721.1/147218 Chapter 1: Coordination Problems, Leverage Constraints and Smart Contract Solutions for the Repo Market.

[10] Gilad Asharov, Tucker Hybinette Balch, Antigoni Polychroniadou, and Manuela Veloso. 2020. Privacy-Preserving Dark Pools. In Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020, Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An, and Neil Yorke-Smith (Eds.). International Foundation for Autonomous Agents and Multiagent Systems, 1747–1749. https://doi.org/10.5555/3398761.3398969

[11] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R. V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Report 2022/915. https://eprint.iacr.org/2022/915

[12] Massimo Bartoletti, James Hsin yu Chiang, and Alberto Lluch-Lafuente. 2022. Maximizing Extractable Value from Automated Market Makers. In FC 2022 (LNCS, Vol. 13411), Ittay Eyal and Juan A. Garay (Eds.). Springer, Cham, 3–19. https://doi.org/10.1007/978-3-031-18283-9_1

[13] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In 2014 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, 459–474. https://doi.org/10.1109/SP.2014.36

[14] Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, and Riivo Talviste. 2016. Students and taxes: a privacy-preserving study using secure computation. Proceedings on Privacy Enhancing Technologies 2016, 3 (2016), 117–135. https://doi.org/10.1515/popets-2016-0019

[15] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, and Jakob Pagter. 2009. Secure multiparty computation goes live. (2009), 325–343. https://doi.org/10.1007/978-3-642-03549-4_20

[16] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. 2020. ZEXE: Enabling Decentralized Private Computation. In 2020 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, 947–964. https://doi.org/10.1109/SP40000.2020.00050

[17] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In CRYPTO 2012 (LNCS, Vol. 7417), Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Berlin, Heidelberg, 868–886. https://doi.org/10.1007/978-3-642-32009-5_50

[18] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2011. Fully Homomorphic Encryption without Bootstrapping. Cryptology ePrint Archive, Report 2011/277. https://eprint.iacr.org/2011/277

[19] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards Privacy in a Smart Contract World. In FC 2020 (LNCS, Vol. 12059), Joseph Bonneau and Nadia Heninger (Eds.). Springer, Cham, 423–443. https://doi.org/10.1007/978-3-030-51280-4_23

[20] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In ASIACRYPT 2017, Part I (LNCS, Vol. 10624), Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, Cham, 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

[21] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2017. Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE. In ASIACRYPT 2017, Part I (LNCS, Vol. 10624), Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, Cham, 377–408. https://doi.org/10.1007/978-3-319-70694-8_14

[22] Ilaria Chillotti, Marc Joye, and Pascal Paillier. 2021. Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks. Cryptology ePrint Archive, Report 2021/091. https://eprint.iacr.org/2021/091

[23] Mariana Botelho da Gama, John Cartlidge, Antigoni Polychroniadou, Nigel P. Smart, and Younes Talibi Alaoui. 2022. Kicking-the-Bucket: Fast Privacy-Preserving Trading Using Buckets. In FC 2022 (LNCS, Vol. 13411), Ittay Eyal and Juan A. Garay (Eds.). Springer, Cham, 20–37. https://doi.org/10.1007/978-3-031-18283-9_2

[24] Morten Dahl, Clement Danjou, Daniel Demmler, Tore Frederiksen, Petar Ivanov, Marc Joye, Dragos Rotaru, Nigel Smart, and Louis Thibault. 2024. fhEVM Confidential EVM Smart Contracts using Fully Homomorphic Encryption. Technical Report. https://github.com/zama-ai/fhevm/blob/main/fhevm-whitepaper.pdf

[25] Morten Dahl, Daniel Demmler, Sarah El Kazdadi, Arthur Meyre, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P. Smart, Samuel Tap, and Michael Walter. 2023. Noah's Ark: Efficient Threshold-FHE Using Noise Flooding. Cryptology ePrint Archive, Report 2023/815. https://eprint.iacr.org/2023/815

[26] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In 2020 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, 910–927. https://doi.org/10.1109/SP40000.2020.00040

[27] Leo de Castro, Andrew W. Lo, Taylor Reynolds, Fransisca Susan, Vinod Vaikuntanathan, Daniel Weitzner, and Nicolas Zhang. 2020. SCRAM: A Platform for Securely Measuring Cyber Risk. Harvard Data Science Review 2, 3 (sep 16 2020). https://hdsr.mitpress.mit.edu/pub/gylaxji4.

[28] Criterion Developers. 2023. Criterion: A Statistics-Driven Microbenchmarking Library for Rust. https://github.com/bheisler/criterion.rs

[29] Benjamin E. Diamond. 2021. Many-out-of-Many Proofs and Applications to Anonymous Zether. In 2021 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, 1800–1817. https://doi.org/10.1109/SP40001.2021.00026

[30] Ethereum. 2023. EIP-1724: Ethereum Improvement Proposal - 1724. https://github.com/ethereum/EIPs/issues/1724

[31] Cardano Foundation. 2023. CIP-0041: Cardano Improvement Proposal - 415. https://github.com/cardano-foundation/CIPs/issues/415.

[32] Serhan Gener, Parker Newton, Daniel Tan, Silas Richelson, Guy Lemieux, and Philip Brisk. 2021. An fpga-based programmable vector engine for fast fully homomorphic encryption over the torus. In SPSL: Secure and Private Systems for Machine Learning (ISCA Workshop).

[33] Rosario Gennaro and Daniel Wichs. 2013. Fully Homomorphic Message Authenticators. In ASIACRYPT 2013, Part II (LNCS, Vol. 8270), Kazue Sako and Palash Sarkar (Eds.). Springer, Berlin, Heidelberg, 301–320. https://doi.org/10.1007/978-3-642-42045-0_16

[34] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In 41st ACM STOC, Michael Mitzenmacher (Ed.). ACM Press, 169–178. https://doi.org/10.1145/1536414.1536440

[35] Milton Harris and Robert M. Townsend. 1985. Allocation Mechanisms, Asymmetric Information and the 'Revelation Principle'. Palgrave Macmillan UK, London, 379–394. https://doi.org/10.1007/978-1-349-06876-0_11

[36] Xiao Hu, Minghao Li, Jing Tian, and Zhongfeng Wang. 2022. Efficient Homomorphic Convolution Designs on FPGA for Secure Inference. IEEE Trans. Very Large Scale Integr. Syst. 30, 11 (2022), 1691–1704. https://doi.org/10.1109/TVLSI.2022.3197895

[37] Liina Kamm, Dan Bogdanov, Sven Laur, and Jaak Vilo. 2013. A new way to protect privacy in large-scale genome-wide association studies. Bioinformatics 29, 7 (2013), 886–893. https://doi.org/10.1093/bioinformatics/btt066

[38] Alexander Karaivanov, Benoît Mojon, Luiz A. Pereira da Silva, and Robert M. Townsend. 2023. Financial Intermediation and Technology: What's Old, What's New? BIS Papers 139. Bank for International Settlements. https://www.bis.org/publ/bppdf/bispap139.pdf

[39] Andrei Lapets, Nikolaj Volgushev, Azer Bestavros, Frederick Jansen, and Mayank Varia. 2016. Secure MPC for analytics as a web application. 2016 IEEE Cybersecurity Development (SecDev) (2016), 73–74. https://doi.org/10.1109/SecDev.2016.027

[40] MIT LEAD. 2025. Benefits of blockchain technologies for the dealer system.

[41] Michael Junho Lee, Antoine Martin, and Robert M. Townsend. 2024. Optimal Design of Tokenized Markets. Staff Report 1121. Federal Reserve Bank of New York. https://doi.org/10.59576/sr.1121

[42] Idan Levin. 2023. Fhenix — Bringing end-to-end encryption onchain. https://medium.com/colliderventures/fhenix-brining-end-to-end-encryption-into-crypto-6650d1e12aa5

[43] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In EUROCRYPT 2010 (LNCS, Vol. 6110), Henri Gilbert (Ed.). Springer, Berlin, Heidelberg, 1–23. https://doi.org/10.1007/978-3-642-13190-5_1

[44] Ahmet Can Mert, Erdinç Öztürk, and Erkay Savas. 2020. Design and Implementation of Encryption/Decryption Architectures for BFV Homomorphic Encryption Scheme. IEEE Trans. Very Large Scale Integr. Syst. 28, 2 (2020), 353–362. https://doi.org/10.1109/TVLSI.2019.2943127

[45] Kevin Nam, Hyunyoung Oh, Hyungon Moon, and Yunheung Paek. 2022. Accelerating N-Bit Operations over TFHE on Commodity CPU-FPGA. In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2022, San Diego, California, USA, 30 October 2022 - 3 November 2022, Tulika Mitra, Evangeline F. Y. Young, and Jinjun Xiong (Eds.). ACM, 98:1–98:9. https://doi.org/10.1145/3508352.3549413

[46] Bloomberg News. 2023. JPMorgan Launches Blockchain Settlement in BlackRock-Barclays Trade. https://www.bloomberg.com/news/articles/2023-10-11/jpmorgan-jpm-launches-blockchain-settlement-in-blackrock-barclays-trade

[47] Antigoni Polychroniadou, Gilad Asharov, Benjamin E. Diamond, Tucker Balch, Hans Buehler, Richard Hua, Suwen Gu, Greg Gimler, and Manuela Veloso. 2023. Prime Match: A Privacy-Preserving Inventory Matching System. In USENIX Security 2023, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 6417–6434.

[48] David Du Pont, Jonas Bertels, Furkan Turan, Michiel Van Beirendonck, and Ingrid Verbauwhede. 2024. Hardware Acceleration of the Prime-Factor and Rader NTT for BGV Fully Homomorphic Encryption. In 31st IEEE Symposium on Computer Arithmetic, ARITH 2024, Malaga, Spain, June 10-12, 2024. IEEE, 1–8. https://doi.org/10.1109/ARITH61463.2024.00011

[49] Huayi Qi, Minghui Xu, Dongxiao Yu, and Xiuzhen Cheng. 2023. SoK: Privacy-Preserving Smart Contract. Cryptology ePrint Archive, Paper 2023/1226. https://eprint.iacr.org/2023/1226 https://eprint.iacr.org/2023/1226.

[50] Witek Radomski, Andrew Cooke, Philippe Castonguay, James Therien, Eric Binet, and Ronan Sandford. 2018. ERC-1155 Multi Token Standard. https://github.com/ethereum/EIPs/issues/1155

[51] Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. In 37th ACM STOC, Harold N. Gabow and Ronald Fagin (Eds.). ACM Press, 84–93. https://doi.org/10.1145/1060590.1060603

[52] Apple Machine Learning Research. 2024. Combining Machine Learning and Homomorphic Encryption in the Apple Ecosystem. https://machinelearning.apple.com/research/homomorphic-encryption

[53] Microsoft Research. 2018. Microsoft SEAL: A Homomorphic Encryption Library. https://github.com/microsoft/SEAL

[54] M. Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. 2020. HEAX: An Architecture for Computing on Encrypted Data. In ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020, James R. Larus, Luis Ceze, and Karin Strauss (Eds.). ACM, 1295–1309. https://doi.org/10.1145/3373376.3378523

[55] Ariel Routledge, Bryan Zetlin-Jones. 2022. Currency stability using blockchain technology. https://ideas.repec.org/a/eee/dyncon/v142y2022ics0165188921000907.html

[56] Solana. 2023. Confidential Token: Deep Dive on Encryption. https://spl.solana.com/confidential-token/deep-dive/encryption

[57] Ravital Solomon. 2021. Smart Contracts from Fully Homomorphic Encryption. https://ethresear.ch/t/smart-contracts-from-fully-homomorphic-encryption/9465

[58] Ravital Solomon, Rick Weber, and Ghada Almashaqbeh. 2023. smartFHE: Privacy-Preserving Smart Contracts from Fully Homomorphic Encryption. In 8th IEEE European Symposium on Security and Privacy, EuroS&P 2023, Delft, Netherlands, July 3-7, 2023. IEEE, 309–331. https://doi.org/10.1109/EUROSP57164.2023.00027

[59] Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin T. Vechev. 2022. ZeeStar: Private Smart Contracts by Homomorphic Encryption and Zero-knowledge Proofs. In 2022 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, 179–197. https://doi.org/10.1109/SP46214.2022.9833732

[60] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin T. Vechev. 2019. zkay: Specifying and Enforcing Data Privacy in Smart Contracts. In ACM CCS 2019, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 1759–1776. https://doi.org/10.1145/3319535.3363222

[61] Yang Su, Bailong Yang, Chen Yang, and Luogeng Tian. 2020. FPGA-Based Hardware Accelerator for Leveled Ring-LWE Fully Homomorphic Encryption. IEEE Access 8 (2020), 168008–168025. https://doi.org/10.1109/ACCESS.2020.3023255

[62] Yang Su, Bai-Long Yang, Chen Yang, Zepeng Yang, and Yi-Wei Liu. 2022. A Highly Unified Reconfigurable Multicore Architecture to Speed Up NTT/INTT for Homomorphic Polynomial Multiplication. IEEE Trans. Very Large Scale Integr. Syst. 30, 8 (2022), 993–1006. https://doi.org/10.1109/TVLSI.2022.3166355

[63] Yang Su, Bai-Long Yang, Chen Yang, and Songyin Zhao. 2022. ReMCA: A Reconfigurable Multi-Core Architecture for Full RNS Variant of BFV Homomorphic Evaluation. IEEE Trans. Circuits Syst. I Regul. Pap. 69, 7 (2022), 2857–2870. https://doi.org/10.1109/TCSI.2022.3163970

[64] Sunscreen. 2024. revm - Rust Ethereum Virtual Machine. https://github.com/Sunscreen-tech/revm

[65] Sunscreen. 2024. Sunscreen FHE compiler. https://github.com/Sunscreen-tech/Sunscreen

[66] Robert M. Townsend. 1988. Information Constrained Insurance: The Revelation Principle Extended. Journal of Monetary Economics 21 (1988), 411–450. https://doi.org/10.1016/0304-3932(88)90038-4

[67] Robert M Townsend and Nicolas Zhang. 2020. Innovative financial designs utilizing homomorphic encryption and multiparty computation.

[68] Robert M. Townsend and Nicolas X. Zhang. 2023. Technologies That Replace a Central Planner. AEA Papers and Proceedings 113 (May 2023), 257–62. https://doi.org/10.1257/pandp.20231031

[69] Giulia Traverso, Denise Demirel, and Johannes Buchmann. 2015. Homomorphic Signature Schemes - A survey. Cryptology ePrint Archive, Report 2015/653. https://eprint.iacr.org/2015/653

[70] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. 2021. SoK: Fully Homomorphic Encryption Compilers. In 2021 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, 1092–1108. https://doi.org/10.1109/SP40001.2021.00068

[71] Fabian Vogelsteller and Vitalik Buterin. 2015. ERC-20 Token Standard. https://github.com/ethereum/EIPs/issues/20

[72] Shi-Yong Wu, Kuan-Yu Chen, and Ming-Der Shieh. 2022. Efficient VLSI Architecture of Bluestein's FFT for Fully Homomorphic Encryption. In IEEE International Symposium on Circuits and Systems, ISCAS 2022, Austin, TX, USA, May 27 - June 1, 2022. IEEE, 2242–2245. https://doi.org/10.1109/ISCAS48785.2022.9937536
[73] Yang Yang, Rajgopal Kannan, and Viktor K. Prasanna. 2024. A Framework for Generating Accelerators for Homomorphic Encryption Operations on FPGAs. In 35th IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2024, Hong Kong, July 24-26, 2024. IEEE, 61–70. https://doi.org/10.1109/ASAP61560.2024.00023
[74] Yinghao Yang, Huaizhi Zhang, Shengyu Fan, Hang Lu, Mingzhe Zhang, and Xiaowei Li. 2023. Poseidon: Practical Homomorphic Encryption Accelerator. In IEEE International Symposium on High-Performance Computer Architecture, HPCA 2023, Montreal, QC, Canada, February 25 - March 1, 2023. IEEE, 870–881. https://doi.org/10.1109/HPCA56546.2023.10070984
[75] Zama. 2022. Concrete ML: a Privacy-Preserving Machine Learning Library using Fully Homomorphic Encryption for Data Scientists. https://github.com/zama-ai/concrete-ml.
[76] Zama. 2024. Evmos. https://github.com/zama-ai/evmos
[77] Zama. 2024. fhEVM. https://github.com/zama-ai/fhevm
[78] Zama. 2024. TFHE-rs. https://github.com/zama-ai/tfhe-rs
[79] Zama. 2024. Threshold Key Management Service. zama.ai/fhe-multi-party-key-management-service
[80] Junxue Zhang, Xiaodian Cheng, Liu Yang, Jinbin Hu, Ximeng Liu, and Kai Chen. 2024. SoK: Fully Homomorphic Encryption Accelerators. ACM Comput. Surv. 56, 12 (2024), 316:1–316:32. https://doi.org/10.1145/3676955
[81] Guy Zyskind, Yonatan Erez, Tom Langer, Itzik Grossman, and Lior Bondarevsky. 2024. FHE-Rollups: Scaling Confidential Smart Contracts on Ethereum and Beyond. Technical Report. https://www.fhenix.io/wp-content/uploads/2024/04/FHE_Rollups_Paper_Final-20241404.pdf
[82] Ali Şah Özcan and Erkay Savaş. 2024. HEonGPU: a GPU-based Fully Homomorphic Encryption Library 1.0. Cryptology ePrint Archive, Paper 2024/1543. https://eprint.iacr.org/2024/1543

issue

A ECONOMIC MOTIVATIONS FOR FHE SMART CONTRACT APPLICATIONS

In this appendix we provide additional motivation and context for the latter two mechanisms presented in Section 4. Each mechanism addresses a circumstance where agents have private information which they are reluctant to reveal truthfully to a counterparty who could make strategic use of the disclosures to increase its payoff at the expense of the disclosing party. The trust deficit is overcome by using FHE on a public blockchain to prevent leakage of information while enabling verification of the computation on the data. Information leakage is prevented by computation on encrypted data. Verification of computation is enabled by the immutable record on the blockchain of the smart contract operation. The examples in Sections 4.3 and 4.4 are derived from mechanisms that contain incentives for agents to send truthful reports of the information they are instructed to submit, provided they are assured that their data remain private and the data transformations that determine resource allocations are correct. The truthful reporting enables each mechanism to achieve a Pareto improvement in resource allocation compared to the situation where leakage occurs. The protocols are examples of how application of FHE, distributed ledgers, smart contracts and relevant mechanism design can improve welfare.

A.1 Private loss insurance smart contract - general model

Here we discuss the generalization of of the toy example in Section 4.3. The general protocol applies to an insurance market where policyholders (in the case of private insurance) and citizens (in the case of public insurance) experience idiosyncratic shocks that are not common knowledge. The model in Townsend (1988) [66], from which the example in the text is derived, has the following features. There are two agents, a and b two dates, t_0 and t_1 and a smart contract, denoted the "Contract". At each date a and b experience private shocks θ_t^a and θ_t^b. The joint distribution of shocks $\{\theta_0^a, \theta_0^b, \theta_1^a, \theta_1^b\}$ is over a finite set Ω which is common knowledge. The joint distribution has two salient properties. One is that an agent is not able to infer with certainty the counterparty's date 0 shock based on its own shocks; e.g. for agent a, $Pr(\theta^a_{t=0}|\theta^b_{t=0}, \theta^b_{t=1}) > 0$ for every possible value of θ^a. It is also supposed that date 1 shocks cannot be inferred with certainty from date 0 values. That is $Pr(\theta_1^a, \theta_1^b|\theta_0^a, \theta_0^b) > 0$ over the set Ω.[11] The Contract allocates resources c_t (which is common knowledge) to agents in each period. The resources can be premiums in the case of private insurance and endowments in the case of public insurance. Agents communicate with the Contract via a message space M_0^i at date t_0 for each agent ($i = a, b$) and a message space $M_1^i(m_0^i, c_0)$. The key idea is that the message sent by agent a at date t_1 cannot be known to agent b at date 1 (and vice versa). At date 1 each agent knows only its own past message and its payout at date t_0. The Contract computes payouts based solely on the messages it receives from the agents, and does not receive any other information related to the true state of the agents.

An insurance contract that achieved a social welfare optimum would condition payouts on the realized shocks. But since the shocks are private (or costly to verify) the achievement of the social welfare optimum would require each agent to truthfully reveal the value of the shock to its counterparty or to an operator of the insurance mechanism. That, in turn, requires the agents be given an incentive to truthfully report their shock, because an agent will report whatever shock value maximizes its payout. This can prevent a viable market from coming into existence. Townsend (1988) [66] overcomes this limitation with a payout algorithm that creates an incentive for truthful reporting. Agents are not paid the amount of their claims, but rather are paid amounts that are functions of the collective (i.e. agent and counterparty) claims sent in the current and past time period plus a random variable. The precise formula in [66] (a) prevents inference of the counterparty's message, which is a necessary condition to incentivize truth revelation and (b) is calibrated to incentivize an agent to send a truthful message to the Contract. The resulting payout to an agent is correlated with its realized shock value in expectation, which is what enables an insurance market to exist. However, the randomized payout function induces an imperfect correlation between payout and loss. Consequently, the Contract achieves a Pareto improvement versus no self-reporting insurance, but it does not achieve a social optimum.

[11] The example in the text sets degenerate shocks for b at date 0 and a at date t_1. See Townsend (1988) [JME] for details.