SoK: Fully-homomorphic encryption in smart contracts

Daniel Aronoff[1], Adithya Bhat[2], Panagiotis Chatzigiannis[2], Mohsen Minaei[2], Srinivasan Raghuraman[1,2], Robert M. Townsend[1], and Nicolas Xuan-Yi Zhang[1]

[1] Massachusetts Institute of Technology, Cambridge, USA
daronoff@mit.edu, rtownsen@mit.edu, nxyzhang@mit.edu
[2] Visa Research, USA
aditbhat@visa.com, pchatzig@visa.com, mominaei@visa.com, srraghur@visa.com

Abstract. Blockchain technology and smart contracts have revolutionized digital transactions by enabling trustless and decentralized exchanges of value. However, the inherent blockchain transparency and immutability pose significant privacy challenges. On-chain data, while pseudonymous, is publicly visible and permanently recorded, potentially leading to 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.

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, which has been growing even further after recent works (e.g. Aronoff and Townsend [9]) have highlighted how smart contracts with those mechanisms can make novel protocols in economics possible. 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.

Motivation. Cryptography, distributed ledger technologies (DLTs), and fully homomorphic encryption (FHE) together enable resource allocation mechanisms previously infeasible. Smart contracts enforce pre-agreed terms, preventing ex-post renegotiation, selective privacy limits disclosure to address trust deficits, and FHE enables verification of correct execution without revealing inputs or outputs. This combination supports incentive-compatible truth-telling mechanisms [41]. We present three examples: (i) a model from Lee, Martin, Townsend (2024) [47] showing how a broker can use atomic smart contracts to match buyers and sellers without holding inventory; (ii) a self-reporting insurance mechanism from Townsend (1988) [74], extended with FHE [73] and implemented on DLT [75]; and (iii) a repo market extension that tokenizes MIT LEAD (2025) [46] based on Aronoff and Townsend (2025) [9]. These cases show how FHE on DLT can reduce frictions in financial intermediation by enabling multilateral coordination under limited commitment, incentivizing truthful preference revelation, increasing trade, creating new markets, and reducing intermediaries' inventory requirements. See Appendix A for detailed discussion on related works.

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.

On the SoK nature of this work. We emphasize that this paper is intended as a Systematization of Knowledge (SoK) contribution on the use of Fully Homomorphic Encryption (FHE) in smart contracts. While the research area is still in its early stages, our goal is to consolidate and critically analyze the existing, fragmented body of work (both academic and industrial) into a coherent structure. We systematically categorize existing approaches according to their cryptographic capabilities, integration models, and application domains, and identify the technical and practical challenges that remain unsolved. Our focus on representative implementations is motivated by their status as the only publicly accessible and usable FHE-for-smart-contract frameworks at the time of writing, enabling reproducible, implementation-driven comparisons. Beyond these case studies, we situate them in the broader design space, highlight missing functionalities (e.g., verifiable FHE, authenticated ciphertext data, privacy-preserving branching, communication cost considerations), and articulate open research questions. In doing so, this paper serves as a structured reference point for researchers and practitioners, fulfilling the central goal of an SoK: to map, clarify, and critically assess the state of the art in a rapidly emerging area.

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 [73,75,46], 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. In Sections 3.1 and 3.2 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) [31], which is associated with centralization, fairness and security concerns [48], and is generally considered a desireable and sought-after technology in blockchain applications [36] as well as in machine learning [58].

2 Background

We provide a brief overview of the three families of FHE schemes as discussed in [11], each optimized for a different plaintext algebra and gate set (we provide the fundamental background of FHE in appendix G).

1. **Residue-number (exact integer) schemes** such as **BFV** [18] and **BGV**. Based on the Ring-LWE assumption [49], they support exact modular arithmetic and SIMD packing, making them attractive for privacy-preserving integer workloads (e.g. balances). Noise grows with circuit depth, so periodic bootstrapping or careful parameter planning is required.

2. **Boolean/LWE schemes** such as **TFHE** [27]. Ciphertexts encrypt single bits (or small moduli) and offer nanosecond Boolean gates and fast bootstrapping, enabling unbounded depth and efficient encrypted comparisons or control-flow. They are preferred when smart contracts need many logical or branching operations.
3. **Approximate-arithmetic schemes** such as **CKKS**. These encode fixed- or floating-point vectors and allow high-throughput linear algebra with small, controllable error—ideal for private analytics and ML—but their approximate decoding is not suited to consensus-critical integer logic unless post-processed by rounding or accompanied by proofs of correct range.

Each family therefore targets a different corner of the design space: BFV/BGV for high-precision integer arithmetic, TFHE for fast Boolean circuits, and CKKS for high-dimensional approximate computation. We discuss how these schemes would fit in smart-contract settings later in our paper.

Smart contract FHE solutions. We defer detailed descriptions of Sunscreen and Zama to Appendix C, and here provide only a brief comparison. Sunscreen compiles high-level programs to efficient BFV-based FHE, making it well-suited for integer arithmetic in smart contracts. Zama, by contrast, builds on TFHE, offering richer data types, encrypted conditionals, and extensive programmability, but at higher cost for integer-heavy workloads. In our benchmarks, Sunscreen exhibited faster integer operations, while Zama's strengths lie in boolean logic, non-linear operations, and advanced features such as configurable decryption. Both approaches advance the feasibility of privacy-preserving, verifiable smart contract execution, but target different points in the performance–functionality trade-off.

3 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, comparing the performance of Sunscreen and Zama on trading protocols in two domains with significant economic impact (see Appendix E for some basic use-cases including a model of intermediation without inventories). Section 3.1 extends the basic model to an automated situation without intermediaries, and links that to insurance. Section 3.2 extends previous models with future repayment legs, and links that to repo and collateral markets.

3.1 Smart contracts to replace intermediaries, and links to the insurance industry

One can ask the question of how two parties A and C could negotiate directly without having intermediary B in the middle. This is similar to a problem asked in Townsend (1988) [74] and Townsend and Zhang (2020) [73] 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 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 [44]. 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 [74] and trust here (from the results of [74]) 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 1 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)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 pol-

icyholder 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 [74], 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, 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 1. Then, still at the first contracting period, the insurer needs to make a deposit(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 1), 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 biasthe final random bit r_3 (step 6). Finally, the smart contract through the homomorphic operations shown step 6 in Fig. 1 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.

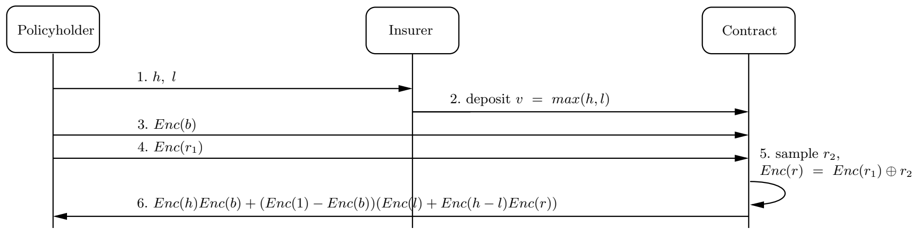

Policyholder
Insurer
Contract
1. $h,\ l$
2. deposit $v\ =\ max(h,l)$
3. $Enc(b)$
4. $Enc(r_1)$
5. sample r_2,
$Enc(r)\ =\ Enc(r_1) \oplus r_2$
6. $Enc(h)Enc(b) + (Enc(1) - Enc(b))(Enc(l) + Enc(h-l)Enc(r))$

Fig. 1. Policy holder - insurer protocol.

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).

3.2 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 standard 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 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 the 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, adapted from the intermediated trade model of Aronoff and Townsend (2025) [9] and applied to the FHE on DLT.A is then the client of B_1 and C is the client of B_2.

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 4. 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.

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

Fig. 2. Steps 0 and 1

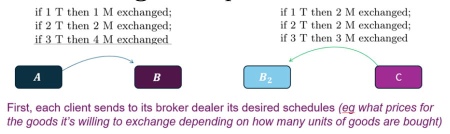
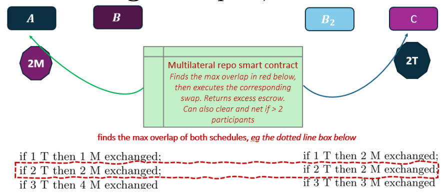

if 1 T then 1 M exchanged;
if 2 T then 2 M exchanged;
if 3 T then 4 M exchanged
if 1 T then 2 M exchanged;
if 2 T then 2 M exchanged;
if 3 T then 3 M exchanged
A
B
B_2
C
First, each client sends to its broker dealer its desired schedules *(eg what prices for the goods it's willing to exchange depending on how many units of goods are bought)*

Fig. 3. Steps 2,3 and 4


A
B
B_2
C
2M
2T
Multilateral repo smart contract
Finds the max overlap in red below, then executes the corresponding swap. Returns excess escrow. Can also clear and net if > 2 participants
finds the max overlap of both schedules, *eg the dotted line box below*
if 1 T then 1 M exchanged;
if 2 T then 2 M exchanged;
if 3 T then 4 M exchanged
if 1 T then 2 M exchanged;
if 2 T then 2 M exchanged;
if 3 T then 3 M exchanged

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 at Step 2) and the smart contract swaps the objects to A and C, sending excess balances back to the broker-dealers if there are any.

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

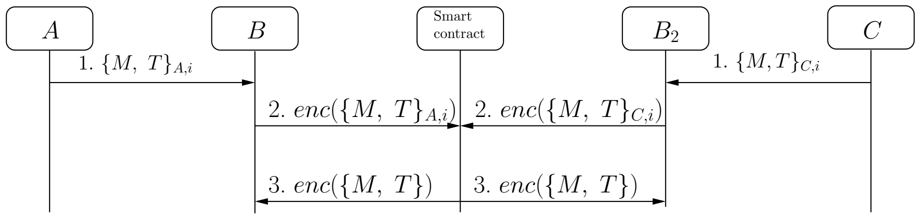


Fig. 4. Intermediated Markets protocol. Note this figure represents the determination of contractual terms of trade, not the exchange of financial objects.

4 Evaluation

We benchmark the two leading smart-contract–oriented FHE stacks available today, Sunscreen (BFV-based) and Zama (TFHE-based), to understand how current systems support the two economic applications developed earlier: unverifiable-loss insurance and improved coordination in intermediated markets. Our goal is not to exhaustively evaluate each library but to characterize the performance regimes most relevant for smart contract execution on DLTs.

Methodology and Environments. Experiments were executed in two settings: (i) a CPU environment on a 2021 Macbook Pro (32GB RAM), and (ii) a GPU environment (AWS `p2.xlarge`). We used production library versions from Sunscreen and Zama's publicly released test networks, plus an "enhanced" Sunscreen build that adds several integer operations not yet exposed on-chain. Because today's test-network gas charges are arbitrary and not correlated with computation costs, we focus on runtime and defer gas cost extrapolations to Appendix D. Benchmark data points were generated using the Criterion framework [32].

Sunscreen vs. Zama: Micro-Level Behavior. Both systems support addition, multiplication, key-generation, encryption, and decryption for encrypted integers, but they differ substantially in how they realize these operations. Sunscreen uses BFV, which works natively over integer rings and thus performs integer arithmetic efficiently. Zama uses TFHE over the torus, giving it an advantage for boolean logic, comparisons, and nonlinear functions.

Our microbenchmarks reveal several robust patterns. First, multiplication is consistently more expensive than addition in both systems, with Sunscreen maintaining a noticeable performance lead for 64-bit arithmetic. Second, key-generation and encryption dominate costs in both stacks, particularly for larger parameter sizes. Third, GPU acceleration materially improves Zama's performance only for large (128–256-bit) ciphertexts; for low-precision values the benefit is marginal. These trends align with our theoretical expectations: BFV's SIMD-friendly integer representation excels at arithmetic circuits, whereas TFHE's gate bootstrapping yields higher overhead for multi-bit integers but strong expressivity for control flow.

Implications for Smart Contract Execution. From the perspective of on-chain protocols, the relative behavior of both systems matters for incentive-compatible mechanism design. Many of our applications (particularly the unverifiable-loss insurance and the intermediated-market matching) require only a small number of public-key operations and a handful of arithmetic operations. Sunscreen's performance profile makes these workloads feasible in permissioned settings and even marginally viable on public Ethereum (Appendix D estimates roughly 4M gas for the unverifiable-loss contract).

By contrast, Zama's TFHE-based approach is well suited for comparisons (needed in the matching mechanism), but current runtimes remain too slow for permissionless deployment. For example, comparing roughly 100 encrypted (M, T) pairs requires approximately 25 seconds on a CPU—acceptable for a consortium chain but not for mainnet Ethereum.

High-Level Takeaways. Current FHE systems are already capable of executing small, economically meaningful smart contract logic end-to-end. Sunscreen offers competitive performance for integer-heavy computations; Zama provides richer programmability, support for comparisons, and configurable decryption mechanisms at the cost of slower arithmetic. For both systems, performance improves meaningfully with GPU acceleration only for higher bit-width ciphertexts. Full details including figures, tables, microbenchmarks, circuit configurations, and gas-cost extrapolations are provided in Appendix D.

5 Insights and Research Gaps

Having considered the FHE scheme families in section 2, the available FHE implementations in section 2, the use-cases in section 3 and our benchmarks in section 4, we now provide our overall findings and observations 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.

5.1 Choosing the appropriate FHE scheme for complex logic

As discussed in section 2, there are three major classes of FHE schemes are relevant (BFV/BGV, TFHE, and CKKS) with different trade-offs. In a smart contract setting, computations typically involve integers (token balances, counters) and conditional logic, which places specific demands on the FHE scheme. **Sunscreen**'s initial compiler chose BFV for its efficient integer operations, emphasizing performance on arithmetic tasks, as our benchmarks show. The downside is limited number of operations before noise grows too high (leveled FHE). Thus, BFV is well-suited for precise, batchable computations (e.g. financial transactions) but need careful parameter tuning for deep circuits [35]. They lack native support for comparisons or bit operations, which must be implemented via arithmetic circuits at a higher cost. On the other hand, **Zama** chose TFHE, which operates on bits or very small plaintext moduli (e.g. torus representation) and features fast bootstrapping to refresh ciphertext noise [21]. Boolean logic and comparisons are natural in TFHE, since an encrypted bit can represent a truth value and homomorphic gates (AND, XOR, multiplexers) can implement arbitrary control flow in encrypted form. Zama's FHE toolchain centers on a TFHE-based library (TFHE-rs) for exactly this reason: it supports a full range of operations including addition, multiplication, encrypted comparisons ($<$, $>$, $==$), and even encrypted conditional branching ("if-else") on secret values. Using TFHE, Zama provides 32-bit and even up to 256-bit encrypted integers with all basic ops and unlimited circuit depth via bootstrapping. The advantage of TFHE in blockchain contexts is that it can easily handle complex business logic or conditional rules in smart contracts (e.g. checking if Enc(balance) $>=$ Enc(amount) without revealing either) which is more challenging under BFV/BGV. The trade-off, however, is performance: TFHE evaluates arithmetic more slowly than BFV/BGV for equivalent bit-width, since it must compose many bit operations for multi-bit numbers. Recently, Sunscreen decided that since BFV is superior at bulk arithmetic but struggles with comparisons and branching, it would be more beneficial to plan a transition to a TFHE variant to cheaply support comparison operations and to remove the prior limit on circuit depth [72]. The rationale behind this decision is also supported by the results of our experiments from Section 4, where we observe significant variations between the available solutions for their corresponding data types. Eventually, while the current version of Sunscreen is more efficient, as 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.

Insight 1 *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.*

The third class of FHE schemes, CKKS, is designed for arithmetic on real numbers by encoding floats and allowing slight precision error. While useful in domains like privacy-preserving machine learning or statistics, where a small numerical error is acceptable, non-deterministic approximation can be problematic in economic applications, which require exact integer results. As a CKKS ciphertexts decrypt to approximate values, two nodes might decrypt to values differing by a tiny ϵ, which is unacceptable for consensus. One could mitigate this by allocating large precision and rounding the result to an integer, however verifying the error stayed below the threshold is non-trivial without extra proofs. Still, CKKS could still be useful off-chain or in layer-2 solutions for heavy numeric computation, with the final result then converted to an exact form before feeding into on-chain logic. For completeness, we provide microbenchmarks for similar FHE operations in appendix H.

Insight 2 *Boolean-gate FHE schemes such as TFHE are steadily becoming the default choice for privacy-preserving smart-contract logic, not because their performance, but because they support unbounded depth and inexpensive encrypted comparisons or branching capabilities that integer-arithmetic schemes cannot match without costly circuit-depth planning or round-trip bootstrapping.*

Gap 1 *Hybrid, cross-scheme toolchains that combine the arithmetic efficiency of BFV with the branching expressiveness of TFHE are still missing for on-chain operations.*

Additionally, we focus on the branching methodology itself. Since encrypted smart contracts cannot observe secret-state conditions, every piece of control flow comparisons, multi-way branches or loops must be rewritten as a data-independent circuit. In practice, tool-chains like Zama's compile both sides of each branch and use TFHE's CMux gate [26] to select the correct result with an encrypted predicate bit. Similarly for loops, recent architecture work observes that "data-dependent operations (e.g., branching) are not supported in ciphertext, meaning all loops have compile-time known trip counts," forcing developers to unroll them to a public worst-case bound [89]. This transformation preserves privacy but inflates circuit width and depth, making comparisons and CMux trees a dominant performance cost for branch-heavy FHE smart-contract logic.

Gap 2 *Protocols are needed that let executors skip untaken branches or terminate loops on secret conditions without leakage would reduce run-time overhead. Also, sub-linear or hardware-accelerated comparison circuits (especially for 128–256-bit integers) are needed to keep encrypted DeFi and other branch-heavy workloads practical.*

5.2 Key management and decryption

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 3 *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 3 *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.*

Zama's new framework [16] advances key management beyond "secret-key-in-the-gateway" designs by providing a unified, audited protocol for BGV, BFV, and TFHE: an n-party MPC over Galois rings jointly generates the public key and splits the secret so that any subset of at most $t < n/3$ cannot decrypt, while honest parties always can; correctness is ensured via noise-flooding for BFV/BGV and a tailored adaptation for TFHE, and optional MPC-in-the-Head proofs certify ciphertext validity before computation. This removes the single point of failure, but still relies on a relatively small, semi-static off-chain committee, raising an unresolved governance issue: how are these MPC parties selected, granted authority, and deterred from colluding when the entire FHE ecosystem is at stake? Validator reputation alone is inadequate; a robust solution must embed open admission and slashing mechanisms that scale to permissionless blockchains without central trust. In addition, as Zama currently has a design target for $n \approx 13$ with highly-reputable validators, it is unclear how current MPC rounds and message complexity would scale to thousands of validators who join/leave unpredictably.

Gap 4 *Frameworks such as Zama's KMS might be cryptographically robust, however mechanisms such as slashing, staking or other reputation-based mechanisms that deter key-share collusion or other malicious behavior in an open network are missing. Formal incentive models are required to have a complete and secure system.*

Gap 5 *While combining "Proofs of Authority" with threshold MPC would avoid having a single centralized private key, question remains how proof of authority consensus validators would be selected, in a way that would be acceptable by all participants in permissionless settings. More research is required to make such approaches compatible with decentralized Proof of Stake blockchain ecosystems.*

5.3 Gas costs and scalability on public blockchains

From considering the use-cases and the benchmark results presented in section 4, and our rough estimates on the needed gas costs, 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. The same workloads can become viable when run in a dedicated roll-up, or an off-chain "coprocessor" (which is the approach followed by Zama), exposing only a low-cost precompile to mainnet, still under the corruption threshold assumptions.

Our gas-cost projections extrapolate runtimes from today's on-chain operations in the spirit of [22] and from Zama's preliminary fhEVM fee table [87], yet what the Ethereum community will ultimately judge as "reasonable" for FHE workloads, and how that standard will shift as libraries mature, remains uncertain, especially because validators equipped with GPU or FPGA accelerators could move the cost curve dramatically. At present the numbers are largely ad-hoc: Sunscreen and Zama each assign their own prices, while ciphertext sizes alone illustrate the challenge (e.g., $\approx$ 22.5 kB for a CKKS ciphertext, 2612 bytes for a Zama encrypted u64, and 372 bytes for a u8), implying roughly 40K gas just to pass a single u64 in calldata and about 544K gas to store it on chain. Standard tricks such as storing only a hash on L1 and requiring users to supply the full ciphertext in calldata, or replacing state with succinct commitments can help. Greater savings could be achieved by executing FHE logic in dedicated roll-ups or other layer-2s where storage and compute tariffs are set independently of main-net constraints.

Insight 4 *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 specialized 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 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 6 *Based on our benchmarks, FHE would not be scalable today in public permissionless smart contracts. While GPUs, FPGAs and ASICs could accelerate FHE computations, validators would need to adopt such hardware, potentially inducing 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.*

5.4 Malicious behavior

We also notice the lack of native, on-chain mechanisms for checking the correctness of FHE ciphertexts that users pass to smart contracts. In practice, existing implementations offload that burden off-chain: Sunscreen expects each ciphertext to be accompanied by a ZK proof, and Zama's fhEVM hands the proof to a gateway (or a small set of operator nodes) that verifies it in $\approx 1.5s$, then posts a signed attestation back on chain confirming the ciphertext is well-formed and within required bounds. While this design avoids the prohibitive gas cost of doing heavy lattice-based proof verification on chain, it re-introduces a trust assumption, as validators must now accept the gateway's signature and still leaves open the possibility that a malicious actor targets that off-chain layer or slips in malformed ciphertexts that never pass through it. A truly permissionless FHE stack therefore needs verifiable FHE primitives that allow validators to cryptographically validate ciphertext integrity without decryption. Such techniques could include zero-knowledge proofs, TEE attestations or homomorphic authenticity primitives such as homomorphic MACs and signatures [39,76]. Yet combining these tools with lattice-based FHE at blockchain scale remains largely unexplored, both in computational overhead and communication cost, making the design of efficient, fully verifiable FHE protocols a key open research problem for secure and reliable smart-contract deployments.

Malicious parties beyond the users themselves also need to be considered, as encrypted values might be altered during off-chain protocols where blockchain-based integrity guarantees do not apply. For instance in Zama, computation is offloaded to FHE coprocessors. If those parties are corrupted above the threshold t, the FHE results can still be tampered with by an adversary.

Gap 7 *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.*

In the case of Zama, trusted execution environments and synchronous networks are assumed in order to ensure the correctness and robustness of their MPC protocol. However, Ethereum for instance assumes malicious players and partially synchronous networks in order to uphold the security guarantees of their blockchain. For the end-user, both assumptions need to be *compatible*, i.e., non-conflicting, in order for the end-to-end protocol to be secure. An example of conflicting assumptions is semi-honest and rational players in a system.

Gap 8 *Formal analysis and proofs are needed to ensure compatibility of consensus assumptions and FHE assumptions for bootstrapping, MPC etc. to ensure end-to-end system security.*

6 Conclusion

In this paper, we examined the intersection of fully homomorphic encryption (FHE) and smart contracts, systematizing existing approaches for enabling privacy-preserving computation on blockchains. Our analysis highlights the need for such mechanisms, particularly for advanced economic use-cases requiring secure and confidential decentralized computations. We reviewed the current FHE-based frameworks for smart contracts, assessed their ability to preserve privacy while retaining automation, decentralization, and security, and identified key challenges such as key management, suitable FHE schemes for blockchain settings, and the high gas costs of FHE operations on permissionless networks.

Our study highlights open research directions, including efficient branching, mixed-key protocols, validator selection in Proof of Authority settings, mitigating centralization risks from specialized hardware, verifiable encrypted data, and reconciling FHE with blockchain consensus assumptions. By addressing these challenges, future work can advance scalable, privacy-preserving smart contracts and unlock novel decentralized applications.

References

1. Erc-1440: Security token standard (2018), `https://github.com/ethereum/EIPs/issues/1440`
2. Erc-2020: Reversible transactions (2019), `https://github.com/ethereum/EIPs/issues/2020`
3. Bme digital bond (2023), `https://www.iberclear.es/docs/docsSubidos/Paper-Digital-Bond-20july.pdf`
4. Abbe, E.A., Khandani, A.E., Lo, A.W.: Privacy-preserving methods for sharing financial risk exposures. American Economic Review **102**(3), 65–70 (2012). https://doi.org/10.1257/aer.102.3.65
5. Abidin, A., Aly, A., Cleemput, S., Mustafa, M.A.: An mpc-based privacy-preserving protocol for a local electricity trading market. In: Foresti, S., Persiano, G. (eds.) Cryptology and Network Security. pp. 615–625. Springer International Publishing, Cham (2016)
6. Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., et al.: Openfhe: Open-source fully homomorphic encryption library. In: proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography. pp. 53–63 (2022)
7. Almashaqbeh, G., Solomon, R.: 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. pp. 124–139. IEEE (2022). https://doi.org/10.1109/EUROSP53844.2022.00016, `https://doi.org/10.1109/EuroSP53844.2022.00016`
8. Angelis, S.D., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A., Sassone, V.: PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain. In: Ferrari, E., Baldi, M., Baldoni, R. (eds.) Proceedings of the Second Italian Conference on Cyber Security, Milan, Italy, February 6th - to - 9th, 2018. CEUR Workshop Proceedings, vol. 2058. CEUR-WS.org (2018), `https://ceur-ws.org/Vol-2058/paper-06.pdf`

9. Aronoff, D., Townsend, R.M.: A smart-contract to resolve multiple equilibrium in intermediated trade (2025), https://arxiv.org/abs/2505.22940
10. Asharov, G., Balch, T.H., Polychroniadou, A., Veloso, M.: Privacy-preserving dark pools. In: Seghrouchni, A.E.F., Sukthankar, G., An, B., Yorke-Smith, N. (eds.) Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020. pp. 1747–1749. International Foundation for Autonomous Agents and Multiagent Systems (2020). https://doi.org/10.5555/3398761.3398969, https://dl.acm.org/doi/10.5555/3398761.3398969
11. Badawi, A.A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., R. V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., Zucca, V.: OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Report 2022/915 (2022), https://eprint.iacr.org/2022/915
12. Bartoletti, M., yu Chiang, J.H., Lluch-Lafuente, A.: Maximizing extractable value from automated market makers. In: Eyal, I., Garay, J.A. (eds.) FC 2022. LNCS, vol. 13411, pp. 3–19. Springer, Cham (May 2022). https://doi.org/10.1007/978-3-031-18283-9_1
13. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press (May 2014). https://doi.org/10.1109/SP.2014.36
14. Bogdanov, D., Kamm, L., Kubo, B., Rebane, R., Sokk, V., Talviste, R.: Students and taxes: a privacy-preserving study using secure computation. Proceedings on Privacy Enhancing Technologies **2016**(3), 117–135 (2016). https://doi.org/10.1515/popets-2016-0019
15. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J.: Secure multiparty computation goes live pp. 325–343 (2009). https://doi.org/10.1007/978-3-642-03549-4_20
16. Bootland, C., Cong, K., Demmler, D., Frederiksen, T.K., Libert, B., Orfila, J.B., Rotaru, D., Smart, N.P., Tanguy, T., Tap, S., Walter, M.: Threshold (fully) homomorphic encryption. Cryptology ePrint Archive, Paper 2025/699 (2025), https://eprint.iacr.org/2025/699
17. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: ZEXE: Enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy. pp. 947–964. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00050
18. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Berlin, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_50
19. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277 (2011), https://eprint.iacr.org/2011/277
20. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 423–443. Springer, Cham (Feb 2020). https://doi.org/10.1007/978-3-030-51280-4_23

21. Caliber: Exploring fully homomorphic encryption: From theory to real-world applications. Mirror.xyz (Aug 2024), https://mirror.xyz/calibervb.eth/PHokST-E6j4G4EV6mYcOH67RnqQRDtMAlU4t7YQelPw, blog post
22. Cardozo, A.S., Williamson, Z.: EIP-1108: Reduce alt_bn128 precompile gas costs. Standards Track – Core 1108, Ethereum Improvement Proposals (May 2018), https://eips.ethereum.org/EIPS/eip-1108, final
23. de Castro, L., Lo, A.W., Reynolds, T., Susan, F., Vaikuntanathan, V., Weitzner, D., Zhang, N.: Scram: A Platform for Securely Measuring Cyber Risk. Harvard Data Science Review **2**(3) (sep 16 2020), https://hdsr.mitpress.mit.edu/pub/gylaxji4
24. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 409–437. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_15
25. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Advances in cryptology–ASIACRYPT 2017: 23rd international conference on the theory and applications of cryptology and information security, Hong kong, China, December 3-7, 2017, proceedings, part i 23. pp. 409–437. Springer (2017)
26. Chillotti, I.: TFHE Deep Dive – Part III: Key switching and leveled multiplications (2022), https://www.zama.ai/post/tfhe-deep-dive-part-3
27. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 377–408. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_14
28. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. Cryptology ePrint Archive, Report 2021/091 (2021), https://eprint.iacr.org/2021/091
29. Dahl, M., Danjou, C., Demmler, D., Frederiksen, T., Ivanov, P., Joye, M., Rotaru, D., Smart, N., Thibault, L.: fhevm confidential evm smart contracts using fully homomorphic encryption. Tech. rep. (2024), https://github.com/zama-ai/fhevm/blob/main/fhevm-whitepaper.pdf
30. 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. Cryptology ePrint Archive, Report 2023/815 (2023), https://eprint.iacr.org/2023/815
31. 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. pp. 910–927. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00040
32. Developers, C.: Criterion: A statistics-driven microbenchmarking library for rust (2023), https://github.com/bheisler/criterion.rs
33. Diamond, B.E.: Many-out-of-many proofs and applications to anonymous zether. In: 2021 IEEE Symposium on Security and Privacy. pp. 1800–1817. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00026
34. Ethereum: Eip-1724: Ethereum improvement proposal - 1724 (2023), https://github.com/ethereum/EIPs/issues/1724
35. Fhenix Team: Cracking the code: Overcoming challenges of on-chain fhe (part 1). Fhenix Blog (Jan 2024), https://www.fhenix.io/cracking-the-code-overcoming-challenges-of-on-chain-fhe-part-1, blog post

36. Foundation, C.: Cip-0041: Cardano improvement proposal - 415. https://github.com/cardano-foundation/CIPs/issues/415 (2023)
37. da Gama, M.B., Cartlidge, J., Polychroniadou, A., Smart, N.P., Alaoui, Y.T.: Kicking-the-bucket: Fast privacy-preserving trading using buckets. In: Eyal, I., Garay, J.A. (eds.) FC 2022. LNCS, vol. 13411, pp. 20–37. Springer, Cham (May 2022). https://doi.org/10.1007/978-3-031-18283-9_2
38. Gener, S., Newton, P., Tan, D., Richelson, S., Lemieux, G., Brisk, P.: 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) (2021)
39. Gennaro, R., Wichs, D.: Fully homomorphic message authenticators. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 301–320. Springer, Berlin, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42045-0_16
40. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 169–178. ACM Press (May / Jun 2009). https://doi.org/10.1145/1536414.1536440
41. Harris, M., Townsend, R.M.: Allocation Mechanisms, Asymmetric Information and the 'Revelation Principle', pp. 379–394. Palgrave Macmillan UK, London (1985). https://doi.org/10.1007/978-1-349-06876-0_11, https://doi.org/10.1007/978-1-349-06876-0_11
42. Hu, X., Li, M., Tian, J., Wang, Z.: Efficient homomorphic convolution designs on FPGA for secure inference. IEEE Trans. Very Large Scale Integr. Syst. **30**(11), 1691–1704 (2022). https://doi.org/10.1109/TVLSI.2022.3197895, https://doi.org/10.1109/TVLSI.2022.3197895
43. Kamm, L., Bogdanov, D., Laur, S., Vilo, J.: A new way to protect privacy in large-scale genome-wide association studies. Bioinformatics **29**(7), 886–893 (2013). https://doi.org/10.1093/bioinformatics/btt066
44. Karaivanov, A., Mojon, B., da Silva, L.A.P., Townsend, R.M.: Digital safety nets: a roadmap. BIS Papers 139, Bank for International Settlements (2023), https://www.bis.org/publ/bppdf/bispap139.pdf
45. Lapets, A., Volgushev, N., Bestavros, A., Jansen, F., Varia, M.: Secure mpc for analytics as a web application. 2016 IEEE Cybersecurity Development (SecDev) pp. 73–74 (2016). https://doi.org/10.1109/SecDev.2016.027
46. LEAD, M.: Benefits of blockchain technologies for the dealer system (2025)
47. Lee, M.J., Martin, A., Townsend, R.M.: Optimal design of tokenized markets. Staff Report 1121, Federal Reserve Bank of New York (September 2024). https://doi.org/10.59576/sr.1121, https://www.newyorkfed.org/research/staff_reports/sr1121.html
48. Levin, I.: Fhenix — bringing end-to-end encryption onchain (2023), https://medium.com/colliderventures/fhenix-brining-end-to-end-encryption-into-crypto-6650d1e12aa5
49. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Berlin, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_1
50. Madathil, V., Scafuro, A.: PriFHEte: Achieving full-privacy in account-based cryptocurrencies is possible. Cryptology ePrint Archive, Report 2023/710 (2023), https://eprint.iacr.org/2023/710

51. Mert, A.C., Öztürk, E., Savas, E.: Design and implementation of encryption/decryption architectures for BFV homomorphic encryption scheme. IEEE Trans. Very Large Scale Integr. Syst. **28**(2), 353–362 (2020). https://doi.org/10.1109/TVLSI.2019.2943127, https://doi.org/10.1109/TVLSI.2019.2943127
52. Nam, K., Oh, H., Moon, H., Paek, Y.: Accelerating n-bit operations over TFHE on commodity CPU-FPGA. In: Mitra, T., Young, E.F.Y., Xiong, J. (eds.) Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2022, San Diego, California, USA, 30 October 2022 - 3 November 2022. pp. 98:1–98:9. ACM (2022). https://doi.org/10.1145/3508352.3549413, https://doi.org/10.1145/3508352.3549413
53. News, B.: Jpmorgan launches blockchain settlement in blackrock-barclays trade (2023), https://www.bloomberg.com/news/articles/2023-10-11/jpmorgan-jpm-launches-blockchain-settlement-in-blackrock-barclays-trade
54. Polychroniadou, A., Asharov, G., Diamond, B.E., Balch, T., Buehler, H., Hua, R., Gu, S., Gimler, G., Veloso, M.: Prime match: A privacy-preserving inventory matching system. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 6417–6434. USENIX Association (Aug 2023)
55. Pont, D.D., Bertels, J., Turan, F., Beirendonck, M.V., Verbauwhede, I.: 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. pp. 1–8. IEEE (2024). https://doi.org/10.1109/ARITH61463.2024.00011, https://doi.org/10.1109/ARITH61463.2024.00011
56. Qi, H., Xu, M., Yu, D., Cheng, X.: Sok: Privacy-preserving smart contract. Cryptology ePrint Archive, Paper 2023/1226 (2023), https://eprint.iacr.org/2023/1226, https://eprint.iacr.org/2023/1226
57. Radomski, W., Cooke, A., Castonguay, P., Therien, J., Binet, E., Sandford, R.: Erc-1155 multi token standard (2018), https://github.com/ethereum/EIPs/issues/1155
58. Research, A.M.L.: Combining machine learning and homomorphic encryption in the apple ecosystem (2024), https://machinelearning.apple.com/research/homomorphic-encryption
59. Research, M.: Microsoft seal: A homomorphic encryption library (2018), https://github.com/microsoft/SEAL
60. Riazi, M.S., Laine, K., Pelton, B., Dai, W.: HEAX: an architecture for computing on encrypted data. In: Larus, J.R., Ceze, L., Strauss, K. (eds.) ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020. pp. 1295–1309. ACM (2020). https://doi.org/10.1145/3373376.3378523, https://doi.org/10.1145/3373376.3378523
61. Routledge, Bryan Zetlin-Jones, A.: Currency stability using blockchain technology (2022), https://ideas.repec.org/a/eee/dyncon/v142y2022ics0165188921000907.html
62. Solana: Confidential token: Deep dive on encryption (2023), https://spl.solana.com/confidential-token/deep-dive/encryption
63. Solomon, R.: Smart contracts from fully homomorphic encryption (2021), https://ethresear.ch/t/smart-contracts-from-fully-homomorphic-encryption/9465
64. Solomon, R., Weber, R., Almashaqbeh, G.: 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.

pp. 309–331. IEEE (2023). https://doi.org/10.1109/EUROSP57164.2023.00027, https://doi.org/10.1109/EuroSP57164.2023.00027
65. Steffen, S., Bichsel, B., Baumgartner, R., Vechev, M.T.: ZeeStar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In: 2022 IEEE Symposium on Security and Privacy. pp. 179–197. IEEE Computer Society Press (May 2022). https://doi.org/10.1109/SP46214.2022.9833732
66. Steffen, S., Bichsel, B., Gersbach, M., Melchior, N., Tsankov, P., Vechev, M.T.: zkay: Specifying and enforcing data privacy in smart contracts. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 1759–1776. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3363222
67. Su, Y., Yang, B., Yang, C., Yang, Z., Liu, Y.: A highly unified reconfigurable multicore architecture to speed up NTT/INTT for homomorphic polynomial multiplication. IEEE Trans. Very Large Scale Integr. Syst. **30**(8), 993–1006 (2022). https://doi.org/10.1109/TVLSI.2022.3166355, https://doi.org/10.1109/TVLSI.2022.3166355
68. Su, Y., Yang, B., Yang, C., Zhao, S.: Remca: A reconfigurable multi-core architecture for full RNS variant of BFV homomorphic evaluation. IEEE Trans. Circuits Syst. I Regul. Pap. **69**(7), 2857–2870 (2022). https://doi.org/10.1109/TCSI.2022.3163970, https://doi.org/10.1109/TCSI.2022.3163970
69. Su, Y., Yang, B., Yang, C., Tian, L.: Fpga-based hardware accelerator for leveled ring-lwe fully homomorphic encryption. IEEE Access **8**, 168008–168025 (2020). https://doi.org/10.1109/ACCESS.2020.3023255, https://doi.org/10.1109/ACCESS.2020.3023255
70. Sunscreen: revm - rust ethereum virtual machine (2024), https://github.com/Sunscreen-tech/revm
71. Sunscreen: Sunscreen fhe compiler (2024), https://github.com/Sunscreen-tech/Sunscreen
72. Sunscreen Team: Sunscreen Documentation (May 2025), online documentation
73. Townsend, R., Zhang, N.: Innovative financial designs utilizing homomorphic encryption and multiparty computation (2020)
74. Townsend, R.M.: Information constrained insurance: The revelation principle extended. Journal of Monetary Economics **21**, 411–450 (1988), https://doi.org/10.1016/0304-3932(88)90038-4
75. Townsend, R.M., Zhang, N.X.: Technologies that replace a central planner. AEA Papers and Proceedings **113**, 257–62 (May 2023). https://doi.org/10.1257/pandp.20231031, https://www.aeaweb.org/articles?id=10.1257/pandp.20231031
76. Traverso, G., Demirel, D., Buchmann, J.: Homomorphic signature schemes - A survey. Cryptology ePrint Archive, Report 2015/653 (2015), https://eprint.iacr.org/2015/653
77. Viand, A., Jattke, P., Hithnawi, A.: SoK: Fully homomorphic encryption compilers. In: 2021 IEEE Symposium on Security and Privacy. pp. 1092–1108. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00068
78. Vogelsteller, F., Buterin, V.: Erc-20 token standard (2015), https://github.com/ethereum/EIPs/issues/20
79. Wu, S., Chen, K., Shieh, M.: 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. pp. 2242–2245. IEEE (2022). https://doi.org/10.1109/ISCAS48785.2022.9937536, https://doi.org/10.1109/ISCAS48785.2022.9937536

80. Yang, Y., Kannan, R., Prasanna, V.K.: 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. pp. 61–70. IEEE (2024). https://doi.org/10.1109/ASAP61560.2024.00023, https://doi.org/10.1109/ASAP61560.2024.00023
81. Yang, Y., Zhang, H., Fan, S., Lu, H., Zhang, M., Li, X.: Poseidon: Practical homomorphic encryption accelerator. In: IEEE International Symposium on High-Performance Computer Architecture, HPCA 2023, Montreal, QC, Canada, February 25 - March 1, 2023. pp. 870–881. IEEE (2023). https://doi.org/10.1109/HPCA56546.2023.10070984, https://doi.org/10.1109/HPCA56546.2023.10070984
82. Zama: Concrete ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists (2022), https://github.com/zama-ai/concrete-ml
83. Zama: Evmos (2024), https://github.com/zama-ai/evmos
84. Zama: fhevm (2024), https://github.com/zama-ai/fhevm
85. Zama: Tfhe-rs (2024), https://github.com/zama-ai/tfhe-rs
86. Zama: Threshold key management service (2024), zama.ai/fhe-multi-party-key-management-service
87. Zama Team: Gas estimation in fhEVM smart contracts (Feb 2025), online documentation, last updated February 2025
88. Zhang, J., Cheng, X., Yang, L., Hu, J., Liu, X., Chen, K.: Sok: Fully homomorphic encryption accelerators. ACM Comput. Surv. **56**(12), 316:1–316:32 (2024). https://doi.org/10.1145/3676955, https://doi.org/10.1145/3676955
89. Zhang, Z., Liu, Y., Zhang, Y., Chen, Z., Zhao, J., Feng, X., Cui, H., Xue, J.: Qiwu: Exploiting ciphertext-level simd parallelism in homomorphic encryption programs. In: Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization. p. 523–537. CGO '25, Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3696443.3708917, https://doi.org/10.1145/3696443.3708917
90. Zyskind, G., Erez, Y., Langer, T., Grossman, I., Bondarevsky, L.: Fhe-rollups: Scaling confidential smart contracts on ethereum and beyond. Tech. rep. (2024), https://www.fhenix.io/wp-content/uploads/2024/04/FHE_Rollups_Paper_Final-20241404.pdf
91. Şah Özcan, A., Savaş, E.: HEonGPU: a GPU-based fully homomorphic encryption library 1.0. Cryptology ePrint Archive, Paper 2024/1543 (2024), https://eprint.iacr.org/2024/1543

A Related works

Works in the Computer Science field. A recent SoK paper [56] 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 [64] 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 [66] which extends Solidity smart contracts by data privacy annotations using additive homomorphic encryption and non-interactive zero-knowledge (NIZK) proofs, and ZeeStar [65], which provides a compiler to enable instantiation of privacy preserving smart contracts without substantial expertise, however it also only supports additive homomorphisms. Zexe [17] 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. Also, [50] shows how account-based cryptocurrencies can attain both confidential values and sender/receiver anonymity by updating every account's ciphertext in each transfer, thereby demonstrating a ledger-layer use of FHE for full-privacy payments that complements contract-level FHE systems discussed above. Note that anonymity is out of scope in our work.

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 [88] 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 [69,55,79] for the BGV FHE scheme [19], [51,68,67,42] for the BFV scheme [18], [60,80,81] for CKKS [24] and [38,52] for THFE [27]. 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 [77], 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 [91]. 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 [37,54].

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 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 [43]; a simulation of a decentralized and privacy-preserving local electricity trading market [5]; an analysis of the gender wage gap in Boston using data from a large set of Boston employers [45]; use of FHE-MPC to collect financial risks [4] and cybersecurity data [23].

B Smart contract operations

One of the common standards in Ethereum smart contracts is ERC-20 [78], 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 privacy of the sender and receiver, nor of the amount. Since all transactions and contract interactions are publicly recorded on the blockchain, sender and receiver 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 [20], 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 [33].

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 [57,1,2], 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.

C Smart contract FHE solutions

C.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 [59] and uses the BFV-FHE scheme [19]. Its core cryptographic library [71] (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) [70] with adds two additional opcodes: `FHE_ADD` and `FHE_MULTIPLY`. The SmartFHE framework [63,64] 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 purposes, we treat SmartFHE in a fashion similar to that of Sunscreen.

C.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 [85] which uses Fully Homomorphic Encryption over the Torus (TFHE) encryption scheme [27], and includes:
  - A codec to encrypt and decrypt 32-bit numbers
  - API to run FHE operations in Rust
- An SDK [84,29] to develop FHE enabled smart contracts in Solidity
- An SDK to develop FHE programs written in Python [82]
- An Ethereum client [83] 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 [28].

A vital feature also is "Configurable Decryption", which users can instantiate with threshold [30], centralized, or on a "Key Management System" (KMS) based decryption [86]. 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 [30] 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 [90] 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.

D Evaluation Details

With the motivation for the two economic use cases described in sections 3.1 and 3.2 that utilize FHE and smart contracts, we perform a series of evaluation experiments using the solutions available today as discussed in section 2. 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: (i) CPU environment — 16-inch Macbook Pro (2021), 32GB RAM, 1TB HDD, Sequoia 15.1. (ii) GPU environment — AWS `p2x.large` instance, Amazon Linux OS.

Libraries: Sunscreen and Zama production test-network versions; Sunscreen "enhanced" version added operations absent from testnet; Zama v0.5 compiled for GPU. Criterion Benchmark framework [32] used. Gas costs ignored; see section 5.3.

D.1 Sunscreen Performance

Sunscreen compiles Rust functions into FHE circuits ("applications"), generating keys, encrypting inputs, executing circuits, and decrypting outputs. Production benchmarks used addition and multiplication for encrypted 64-bit integers; enhanced version added subtraction, negation, and scalar variants. Operations benchmarked for all six data types: Signed, Rational, Fractional64, Fractional128, Fractional256, Fractional512.

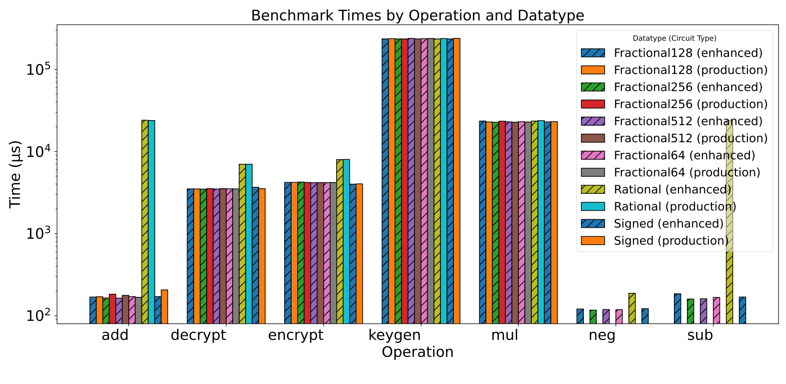


Fig. 5. Microbenchmarks for all operations and all data types supported by Sunscreen

Key generation was slowest; addition fastest, multiplication expensive; Rational type most costly due to division by encrypted ciphertexts requiring larger parameters. Enhanced circuits showed no significant performance change.

D.2 Zama Performance

Zama testnet ($n = 1$, $t = 0$ threshold servers) supports unsigned/signed integers (8–256 bits). Benchmarks run on CPU and GPU.

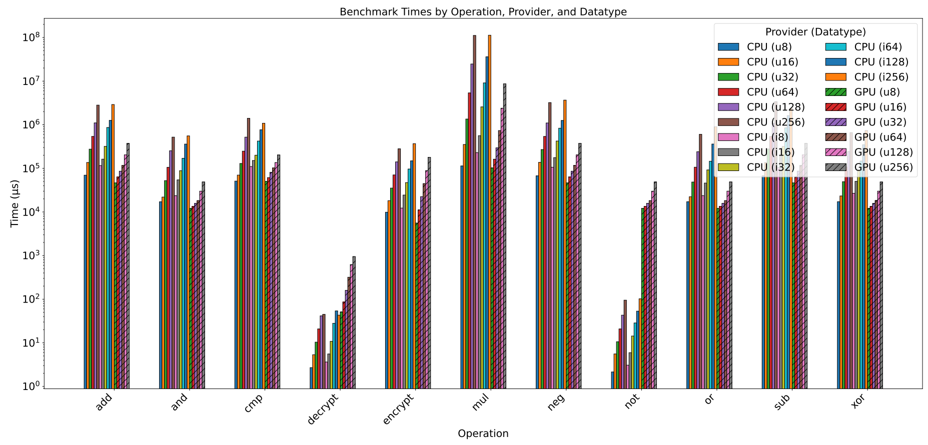


Fig. 6. Microbenchmarks for all operations and all data-types supported by Zama

Multiplication slower than addition; larger bit-widths increase runtime. GPU acceleration gave significant gains for high-precision types (u256/i256), minimal for low-precision.

D.3 Comparison: Zama vs Sunscreen

Both support signed 64-bit integers and common ops (keygen, encrypt, decrypt, add, sub, neg, mul). GPU used unsigned 64-bit type.

Sunscreen markedly faster for integer arithmetic due to BFV's RLWE structure; Zama's TFHE excels at boolean/non-linear ops but is less integer-efficient. GPU improved Zama performance overall.

D.4 Unverifiable Losses Protocol

Benchmarks for unsigned 8-bit integers (ideal for single-bit encryption) and other types (Sunscreen supports only 64-bit). See fig. 8.

Feasible in permissioned chains; costly for permissionless Ethereum. Sunscreen 4M gas ($80), Zama exceeds 30M gas limit. GPU acceleration negligible for low-bit types.

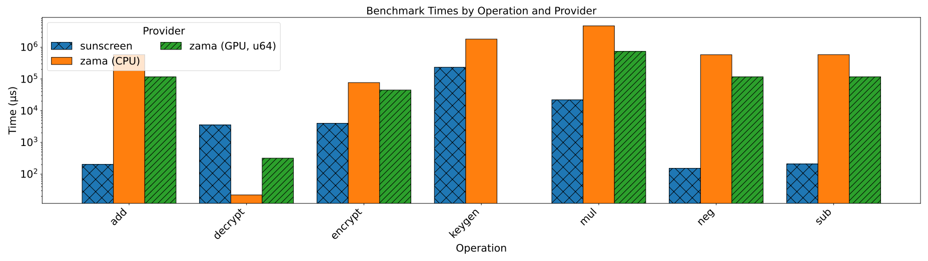


Fig. 7. Comparing Zama and Sunscreen for common data type: Signed 64-bit integers and common operations.

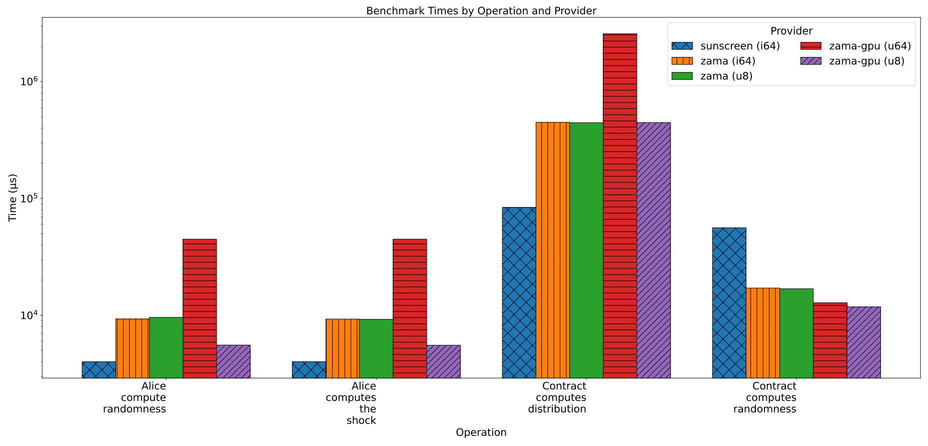


Fig. 8. Benchmarking the operations for the unverifiable losses protocol.

D.5 Coordination in Intermediated Market Protocol

Benchmarked matching encrypted $\{M, T\}_{c,i}$ pairs (100 samples, u32) using Zama (comparison only in Zama). Sorting done off-chain by broker-dealers before encryption.

Comparisons averaged 25s CPU — feasible for permissioned, impractical for permissionless Ethereum.

Gas costs in Sunscreen and Zama are arbitrarily set; actual feasible costs depend on validator hardware and consensus rules. GPU/FPGA acceleration could enable these workloads but risks raising validator hardware requirements, increasing centralization, and undermining PoS accessibility.

E Basic applications

Fig. 9. B and C using the smart contract negotiated between A and B

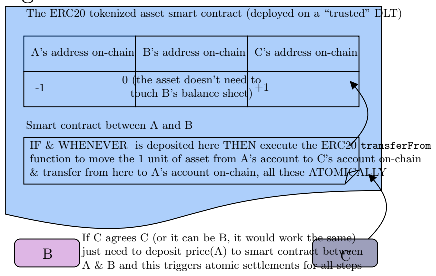
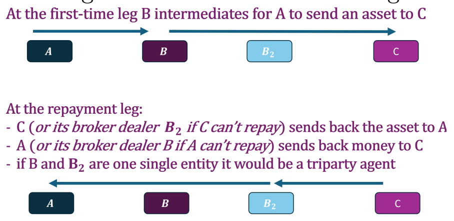

The ERC20 tokenized asset smart contract (deployed on a "trusted" DLT)
A's address on-chain
B's address on-chain
C's address on-chain
-1
0 (the asset doesn't need to touch B's balance sheet)
+1
Smart contract between A and B
IF & WHENEVER is deposited here THEN execute the ERC20 transferFrom function to move the 1 unit of asset from A's account to C's account on-chain & transfer from here to A's account on-chain, all these ATOMICALLY
B
C
If C agrees C (or it can be B, it would work the same) just need to deposit price(A) to smart contract between A & B and this triggers atomic settlements for all steps

Fig. 10. Illustration of the payment and the repayment legs, with different "insurers" - eg broker dealers - for each leg


At the first-time leg B intermediates for A to send an asset to C
A
B
B_2
C
At the repayment leg:
- C (or its broker dealer B_2 if C can't repay) sends back the asset to A
- A (or its broker dealer B if A can't repay) sends back money to C
- if B and B_2 are one single entity it would be a triparty agent
A
B
B_2
C

After reviewing the basics of FHE applied to exchanges of tokenized assets in Section E.1. Section E.2 presents a first and very general model of financial intermediation.

E.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 [78,57]. 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 [62,34], 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 [53], dark pools in securities trading [10,37,54], or information flows to deal with crisis contagion [61].

E.2 A model of intermediation without inventories

Lee, Martin, Townsend (2024) [47] 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 risk of loss by lowering the price it is willing to purchase from A. This will reduce the volume of trade.[3]

Building on Lee, Martin, Townsend (2024), [46] show 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. [46] 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 11. A and B meet. They can agree on conditions for B to potentially sell to C later, but

[3] 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.

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 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 12: 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 9:

Fig. 11. Initial allocation before any trade.

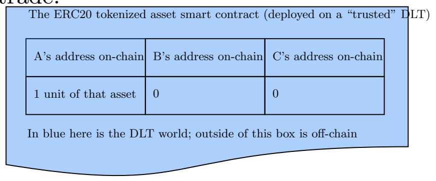
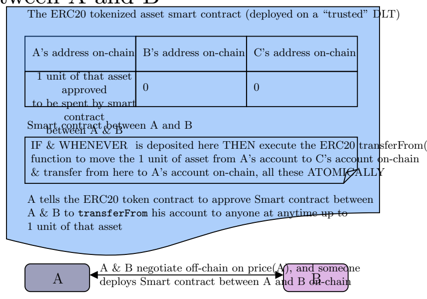

The ERC20 tokenized asset smart contract (deployed on a "trusted" DLT)
A's address on-chain
B's address on-chain
C's address on-chain
1 unit of that asset
0
0
In blue here is the DLT world; outside of this box is off-chain

Fig. 12. Smart contract negotiated between A and B


The ERC20 tokenized asset smart contract (deployed on a "trusted" DLT)
A's address on-chain
B's address on-chain
C's address on-chain
1 unit of that asset approved to be spent by smart contract between A & B
0
0
Smart contract between A and B
IF & WHENEVER is deposited here THEN execute the ERC20 transferFrom() function to move the 1 unit of asset from A's account to C's account on-chain & transfer from here to A's account on-chain, all these ATOMICALLY
A tells the ERC20 token contract to approve Smart contract between A & B to transferFrom his account to anyone at anytime up to 1 unit of that asset
A
B
A & B negotiate off-chain on price(A), and someone deploys Smart contract between A and B on-chain

F Economic Motivations for FHE Smart Contract Applications

In this appendix we provide additional motivation and context for the latter two mechanisms presented in Section 3. 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 3.1 and 3.2 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.

F.1 Private loss insurance smart contract - general model

Here we discuss the generalization of of the toy example in Section 3.1. 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) [74], 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, $Pr(\theta_{t=0}^a | \theta_{t=0}^b, \theta_{t=1}^b) > 0$ 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.[4] 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) [74] 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 [74] (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

[4] 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.

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.

G Cryptographic primitives - Fully Homomorphic encryption

We now provide a basic background on FHE as a cryptographic primitive.

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 [40] 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.

H CKKS for Blockchains

In this section we evaluate the CKKS [25] FHE scheme for blockchain adoption. In a nutshell, CKKS allows approximate homomorphic operations over encrypted floating point numbers. However, CKKS has the following limitations:

- CKKS supports only approximate arithmetic, which is not suitable for applications that require exact results.
- CKKS does not support circuits of unbounded depth.

To evaluate CKKS, we implemented a microbenchmark that measures the time taken for encryption, addition, multiplication, and decryption operations. We used the OpenFHE library [6] with the CKKS scheme. OpenFHE is a C++ library that provides a high-level interface for FHE schemes, including CKKS. It automatically handles the underlying complexities of CKKS, such as scaling and noise management. We used the default parameters provided by OpenFHE for a precision of 59 bits, which are suitable for most applications. We ran the microbenchmark in the CPU environment. The results of the microbenchmark are shown in table 1.

Table 1. CKKS Microbenchmarks

Operation	**Time**
Encryption	13.47 ms
Addition	0.528 ms
Multiplication	12.71 ms
Decryption	17.42 ms

We observed that CKKS is relatively fast for addition and multiplication operations, but encryption and decryption are significantly slower. Between addition and multiplication, multiplication is much slower, as expected. The cost of the addition and multiplication operations are comparable to the BFV FHE schemes.

Acknowledgments

The authors used generative AI-based tools to revise the text, improve flow and correct any typos, grammatical errors, and awkward phrasing.