DeFiScanner: Spotting DeFi Attacks Exploiting Logic Vulnerabilities on Blockchain

Bin Wang, Xiaohan Yuan, Li Duan, Hongliang Ma, Bin Wang, Chunhua Su, and Wei Wang, *Member, IEEE*

***Abstract*— With the rapid development of decentralized financial (DeFi), the total value locked (TVL) in DeFi continues to increase. A big number of adversaries exploit logic vulnerabilities to attack DeFi applications for profit, such as flash loan attacks and price manipulation attacks. However, the current vulnerability detection tools for smart contracts cannot be directly used to detect the logic vulnerabilities generated by the combination of different protocols. How to characterize and detect DeFi attacks that exploited logic vulnerabilities is a big challenge. In this work, we propose a deep-learning-based attack detection system on DeFi, called DeFiScanner, in which we design a novel neural network that includes a global model, a local model, and a fusion model to characterize DeFi attacks. First, the unstructured emitted events are automatically and efficiently normalized. Second, the transaction-related features of normalized emitted events are enriched with the global model and the semantic features of emitted events are extracted with the local model. Finally, the transaction-related features and the semantic features of emitted events are fused efficiently with the fusion model to detect DeFi attacks. We collect a dataset that consists of 50 910 real-world DeFi transactions on Ethereum (ETH). The extensive experimental results demonstrate the effectiveness of DeFiScanner. The true positive rate (TPR) and the area under the receiver operating characteristic (ROC) curve of the system reach 0.91 and 0.97, respectively.**

***Index Terms*— Attacks detection, blockchain, decentralized finance, deep learning.**

I. Introduction

UNLIKE traditional finance, decentralized financial (DeFi) takes full advantage of the transparency and openness of a decentralized network (i.e., blockchain) to provide diverse financial services without relying on the third-party intermediaries. As of the end of May 2022, the total value locked (TVL) in DeFi applications reaches to $55 billion. With the rapid growth of DeFi applications on public blockchain ecosystems, such as Ethereum (ETH) [1], external attacks against DeFi applications have also emerged, which seriously threaten the financial security of DeFi participants. Many security issues related to DeFi have been reported, including pump-and-dump (P&D) scams [2], [3], front-running [4], [5], [6], and flash loan attacks [7], which have brought many security incidents [8], [9], [10], [11], [12]. Attackers compromise DeFi applications through code and logic vulnerabilities, causing huge losses to the DeFi ecosystem.

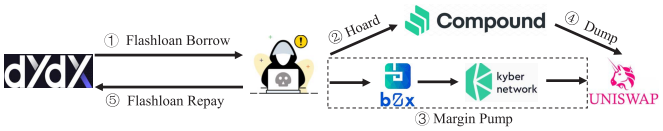


Fig. 1. Five composable DeFi protocols in bZx hack.

To illustrate the attacks, we take an example that took place in the bZx project [10]. The bZx is a DeFi project for lending and margin trading on Ethereum. The platform uses many other DeFi protocols to provide these services. Anyone can add funds to the fund pool of bZx to borrow other tokens and take advantage of long or short positions by trading other assets on margin. Due to the complex combination of protocols, attackers can exploit the dependence of bZx's oracle with other DeFi projects to manipulate the exchange rate of crypto assets and profit.

As shown in Fig. 1, the attack transaction (called external transaction) can be broken down into five steps (called internal transactions): flashloan borrow, hoard, margin pump, dump, and flashloan repay. First, the attacker takes advantage of the flashloan service of dYdX to borrow a large amount of ETH. Second, the attacker deposits parts of ETH into compound to borrow wrapped bitcoin (WBTC). After hoarding, the attacker uses the margin trade service of bZx to short ETH in favor of WBTC. Specifically, bZx forwards the order to KyberSwap to complete the transaction. KyberSwap consults its reserves and finds the best exchange rate. The price of WBTC in Uniswap is tripled by the attacker. Finally, the attacker swaps the WBTC borrowed from compound back for WETH in Uniswap to gain profits and repays the flashloan to dYdX.

The core step of arbitrage (i.e., profiting by buying and selling commodities at different prices) is that attackers

Manuscript received 2 September 2022; revised 24 October 2022 and 26 November 2022; accepted 29 November 2022. Date of publication 21 December 2022; date of current version 3 April 2024. This work was supported in part by the National Natural Science Foundation of China under Grant U21A20463 and Grant 61902021. *(Corresponding authors: Wei Wang; Bin Wang.)*

Bin Wang, Xiaohan Yuan, Li Duan, and Wei Wang are with the Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, China (e-mail: bin.wang@bjtu.edu.cn; xiaohan.yuan@bjtu.edu.cn; duanli@bjtu.edu.cn; wangwei1@bjtu.edu.cn).

Hongliang Ma is with the School of Information Science and Technology, Shihezi University, Shihezi 832003, China (e-mail: mhl_inf@shzu.edu.cn).

Bin Wang is with the Zhejiang Key Laboratory of Multi-Dimensional Perception Technology, Application and Cybersecurity, Hangzhou 310053, China (e-mail: bin_wang@zju.edu.cn).

Chunhua Su is with the Department of Computer Science and Engineering and the Division of Computer Science, University of Aizu, Aizuwakamatsu 965-8580, Japan (e-mail: chsu@u-aizu.ac.jp).

Digital Object Identifier 10.1109/TCSS.2022.3228122

successfully control the exchange rate of a crypto asset pair by exploiting the price dependencies of bZx.

The existing research work and tools [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28] mainly focus on privacy protection, malicious code detection, and detecting the vulnerabilities of smart contracts, such as the re-entrancy, timestamps dependency, and short address attack. Specifically, there have been a lot of works [29], [30], [31], [32], [33], [34], [35], [36], [37] on detecting abnormal behaviors, such as money laundering, frauding, and Ponzi scheme. However, on one hand, since many attacks against DeFi projects use the combinations of different DeFi protocols, these tools cannot be used to detect the logic vulnerabilities in DeFi projects as shown in Fig. 1. On the other hand, internal transactions contain high-level semantic features, e.g., a user deposits an amount of ETH into a decentralized exchange (DEX) to borrow WBTC. We cannot extract these features from the code level of smart contracts.

To detect the attacks against DeFi applications, we need to analyze internal transactions and extract semantic features of the combination of different DeFi protocols. There are two challenges in detecting attacks against DeFi applications. First, internal transactions are very complicated. Attackers exploit the logic vulnerabilities generated by the combinations of protocols to launch flash loan attacks for profit. The diverse and complex combinations between protocols bring great challenges to attack detection. External transactions usually contain a large number of internal transactions. In addition, since the data of internal transactions are unstructured, it is difficult to directly input them into a model for detection. We need to efficiently process large amounts of unstructured data and transform them into structured (semi-structured) data. Second, extracting the multilevel DeFi semantic information is difficult. Complex internal transactions often contain different levels of semantic information, such as external transaction level and internal transaction level. The external transaction level contains the block number, timestamp, and other global information of the transaction. High-level semantic information is contained at the internal transaction level. It usually takes a lot of human efforts to define features and obtain semantic information. We need to build a neural network that can extract effective features from different levels of semantic information and perform attack detection.

To address the above challenges, we propose a system, called DeFiscanner, to perform large-scale attack detection on DeFi applications with the deep learning methods. Considering the complexity of internal transactions, we do not directly process all the internal transactions. Instead, we use the events generated by internal transactions to extract high-level features. The events filter many unnecessary internal transactions. To automatically and efficiently process unstructured data, we adapt the word2vec [38] method to vectorize events and feed the vectorizes into a deep learning model. We also define a series of features from the account level and external transaction level guided by prior knowledge to enrich high-level features. With the designed neural network, the features of different levels are fused efficiently to detect attacks. To evaluate the effectiveness of DeFiScanner, we collect 50 910 real-world external transactions of DeFi applications from January 2021 to April 2022. We use seven models, namely, support vector machine (SVM) [39], k-means [40], autoencoder [41], deepautoencoder, long short-term memory (LSTM) [42], convolutional neural networks (CNN) [43], and LSTM-CNN to detect attack transactions.

We make the following contributions.

1) We propose a system called DeFiScanner for large-scale attack detection on DeFi applications with the deep learning methods. DeFiScanner can process unstructured emitted events generated by DeFi applications and fuse the multilevel features to extract high-level semantic features. To the best of our knowledge, our work is the first deep-learning-based system to detect DeFi attacks.
2) We design a neural network in DeFiScanner. It can fuse different kinds of features from external transactions and emitted events to accommodate multilevel features. It provides an effective fused method for deep-learning-based attack detection on DeFi.
3) We evaluate the effectiveness of DeFiScanner by collecting 50 910 real-world transactions on DeFi applications. The experimental results indicate that DeFiScanner can detect DeFi attacks effectively. The true positive rate (TPR) and the area under the receiver operating characteristic (ROC) curve of the system reach 91.11% and 0.97, respectively.

The remainder of this article is organized as follows. In Section II, we introduce the relevant works. We briefly introduce some concepts about Ethereum and DeFi applications in Section III. Next, we descript the details of DeFiScanner in Section IV. The setup of experiment is shown in Section V. In Section VI, we evaluate the performance of DeFiScanner. We conclude our work in Section VII.

II. Related Work

A. Ethereum DeFi Ecosystem

1) Order Book: Order book is the most common way to realize transactions on DeFi. The purpose of order book is to match the trading intentions of buyers and sellers. Some platforms such as Coinbase [44] and Binance [45] are implemented based on a centralized order book. Coinbase is the largest trading volume in the world and the first centralized exchange listed on Nasdaq [46]. dYdX [47] is a classic implementation of DEXs. Serum [48] implements a relatively special DEX. 0× Relayer [49] adopts the order matching method of the open order book, which continuously accepts and broadcasts orders with all 0 addresses. Waves Exchange [50] is a platform for exchanging stable currencies, which provides exchange services between Waves tokens and stable currencies that are anchored to the legal currencies of various countries. The exchange between the digital currency and the stablecoin is carried out through the first-in-first-out (FIFO) mechanism, and the stable currency maintains the stability of the legal currency based on the Neutrino [51] protocol. Injective [52] proposes an order matching mechanism based on frequent batch auctions (FBAs).

2) Automated Market Maker: The DEXs based on the automatic market maker (AMM) mechanism are currently the

most popular choice in the industry. Uniswap [53] is one of the classics among market maker mechanisms. Uniswap v3 [54] is an improvement of the Uniswap protocol. The main difference between them is the constant product function. Similar to Waves Exchange in the transaction order book, AMMs also have mechanisms to support stablecoin transactions, such as StableSwap [55]. SushiSwap [56] is a protocol almost the same as Uniswap, with the only difference being the structure of community governance. SushiSwap plundered the liquidity of Uniswap through a "vampire attack" [57] in August 2020. Raydium [58] is a mechanism that combines AMMs and transaction order books.

The relationship between DEXs is complex and an external transaction often involves a large number of DeFi protocols. With the combination of these protocols, it is easy to generate logical vulnerabilities that can be exploited by attackers.

B. Flash Loan Attacks

Due to the convenience of flash loans, attackers often use flash loans to boost profits, which brings serious financial security problems to the development of the DeFi ecosystem. Since the flash loan attack has just appeared in the past two years, there are only some initial research works about it. Wenkai et al. [59] conduct a systematic examination of the security issues of the DeFi ecosystem built on blockchain. Qin et al. [7] analyze the two initial flash loan attacks and redistribute the funds used in each step to maximize the revenue. Wang et al. [60] propose a flash loan transaction identification method to identify flash loan transactions on Uniswap, Aave, and dYdX. Cao et al. [61] propose a system to visualize capital flows of flash loan attacks. Wang et al. [62] implement a simple system to detect attack transactions on DeFi, which identified transactions with extremely high yields as attack transactions. Most of these works only introduce the principle of flash loan attacks and cannot automatically and accurately detect attacks on DeFi.

III. Background

We briefly introduce Ethereum transactions, cryptocurrencies, DeFi, and flash loan attacks in this section to better understand our work.

A. Ethereum Transactions

The accounts in Ethereum are divided into two types: external accounts and contract accounts. External accounts are created and remain unchanged when users join Ethereum. Each external account has a pair of public–private keys, the private key is used to sign transactions, and the public key is used to generate an Ethereum address. There are three types of transactions in Ethereum: one is a transfer transaction, which is the general exchange of funds between two accounts, and there is often no additional information in the transaction. The second is a contract creation transaction. Users can publish contracts to the blockchain through the transaction. The third is the contract invocation transaction. External accounts or other contracts can invocate certain functions in the contract with

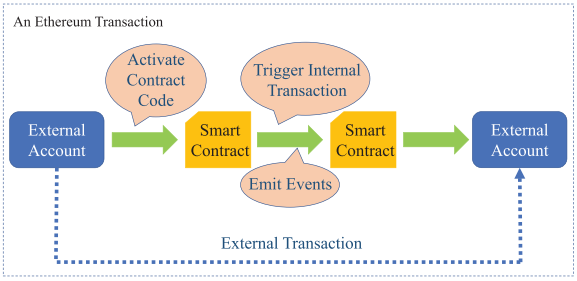


Fig. 2. Ethereum transaction.

this kind of transaction, as shown in Fig. 2. In this article, we refer to external transactions and internal transactions collectively as transactions.

Events are inheritable members of contracts. They are declared in the codes of smart contracts and can be emitted with the emit keywords. Events record the related parameters in the transaction logs. These logs are stored on the blockchain and can be accessed using the address until the contract appears on the blockchain.

B. Cryptocurrencies

In Ethereum, digital cryptocurrencies are divided into two categories: Ether (the native token) and ERC20 (Ethereum Request for Comment 20) token. Ether is generated along with Ethereum, and transaction fees in Ethereum have to be paid with Ether. ERC20 tokens are attached to Ethereum and created through smart contracts. DEXs can deploy a smart contract that conforms to the ERC20 standard to publish an ERC20 token. Since there are kinds of ERC20 tokens in Ethereum, we only introduce the stablecoins and liquidity provider (LP) tokens.

Stablecoin is designed to maintain the stability of the market, which relates to an anchored fiat currency (usually the U.S. dollar). Dai [63] of MarkerDAO is one of the notable implementations. It maintains price stability by over-collateralizing assets and triggering liquidations when the value of the collateralized assets falls below the collateral line. USD coin (USDC) [64] and tetherUS (USDT) [65] are also popular stablecoins. The stablecoins provide a valuable reference for transactions. The value of digital cryptocurrencies and other equity tokens can be linked with anchored fiat currency through stablecoins.

LP tokens are issued to crypto liquidity pools on DEXs with AMM protocols. Amounts of platforms on DeFi provide the service, such as Uniswap and Balancer. The Balancer is a popular DEX that distributes LP tokens to their LPs. LPs can sell LP tokens and use LP tokens to exchange for other tokens. The corresponding liquidities can be redeemed at any time.

C. Decentralized Finance

DeFi applications involve multilevel protocols. The most important are protocols of the on-chain asset exchange. The design of the transaction mechanism is related to the stability of complex applications. There are two main components

of DeFi, one is the stablecoins represented by Bitcoin and Ethereum, and the other is the smart contract that realizes trading, lending, and investment.

1) *Decentralized Exchange:* DEX is a peer-to-peer marketplace that allows trading cryptocurrencies directly between two parties. DEX is designed to solve problems inherent in a centralized exchange, such as centralized custody of assets, geographic restrictions, and asset selection. DEX uses AMMs instead of the traditional order books. Without matching individual buy and sell orders, users can deposit assets into a pool, and the price is determined based on the ratio between the assets in the pool. DEX also allows users to passively provide liquidity, market any asset on Ethereum, and provide traders with always-available liquidity.
2) *Lending:* DeFi protocols allow users to lend assets. Users must provide collateral assets that exceed the amount they borrowed. Similar to the mortgage loan in traditional finance, users can post various assets as collateral and borrow other crypto assets including stablecoins with DeFi protocols. Moreover, once the value of the collateral falls below a specified loan-to-value ratio, the collateral will be liquidated to ensure that the loan can be repaid.
3) *Flash Loan:* Flash loan is a new type of noncollateral loan and many DeFi apps have provided this kind of protocol [66], [67]. All the operations with the flash loan have to be accomplished in one transaction or one block. It allows users to borrow a considerable amount of capital without collateralizing assets (but paying extra fees). The loan should be repaid within a certain amount of time (about 13 s in Ethereum); otherwise, the transaction is reverted to ensure the safety of the fund, i.e., undoing all operations performed previously. The operations between borrowing and repaying are called circuits, and users can flexibly combine various DeFi protocols. But it has been abused by hackers to launch attacks [61].

D. Flash Loan Attacks and Price Manipulation

Generally, many DEXs require real-time information on the market price of assets to provide borrowing, lending, and redemption. To enable this functionality, DeFi protocols have introduced oracles, which report the prices of asset from real-world (off-chain) sources, such as on-chain DEXs (e.g., Uniswap, dYdX) and decentralized oracle networks (e.g., Chainlink).

In practice, flash loan attacks are often combined with price (or oracle) manipulation. An arbitrageur executes a strategy and makes a profit by lowering or increasing a specific asset. A simple successful flash loan attack involves the following steps, as shown in Fig. 3.

1) The attacker searches for DeFi protocols that support flash loans and borrows a large amount of token X as flash loans.
2) The attacker uses token X in exchange for token Y. Due to a large amount of exchange, the price of token X decreases, and the price of token Y increases.

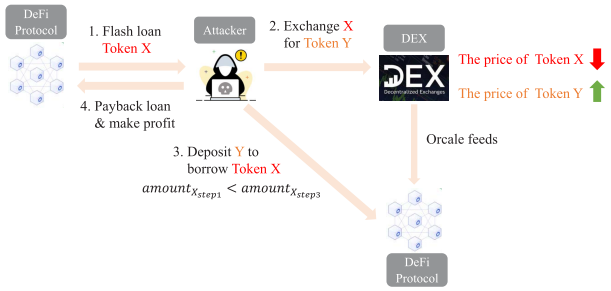


Fig. 3. Attacker manipulates the price of particular assets.

3) The attacker chooses another DeFi lending protocol as a target and deposits token Y as collateral to lend more token X. Note that the criterion for protocol selection is that the DeFi protocol uses the above DEX as its only oracle data source.
4) Ultimately, the attacker uses the borrowed token X to repay the previous flash loan. Due to the large amount of token X lent, the attacker can profit from the remaining token X.

Besides the above case on bZx, in the past two years, flash loan attacks have compromised the multiple DEXs resulting in a loss of $200 M.

IV. Detection Models

The target of our work is to design an attack detection system for DeFi applications, called DeFiScanner. It can automatically analyze the unstructured emitted events and output "attack" or "normal." In the DeFiScanner, we design a neural network that can extract efficient multilevel features and achieve good performance without consuming a lot of human effort. We describe the design of DeFiScanner in this section.

As shown in Fig. 4, DeFiScanner has two phases: the training (i.e., learning) phase and the detection phase.

1) The learning phase has five steps, as shown in Fig. 4(a).

Step 1: Extract the global information G, emitted events E, related accounts, balances, and transferred tokens with transaction hashes. The emitted events of a raw transaction are shown in Fig. 5. The global information include block number, timestamp, transaction cost, and gas used.

Step 2: Extract the global features for each transaction with global information. The global features have a total of 37 dimensions, denoted by $G(g_1, g_2, \ldots, g_{37})$, most of which are statistical features, such as the number of transfer functions, withdraw functions, and the related tokens. We show the details in Table I.

Step 3: Generate the ground-truth labels of training programs. To better evaluate the DeFiScanner, we label the ground truth of raw transactions. Specifically, we label each raw transaction as "0" (i.e., normal transaction) or "1" (i.e., attack). The ground-truth labels of raw transactions are available with the Etherscan [68] and the public information from the official websites of DEXs.

Step 4: Transform emitted events into vectors to extract semantic features. This step has two substeps, which are highlighted below.

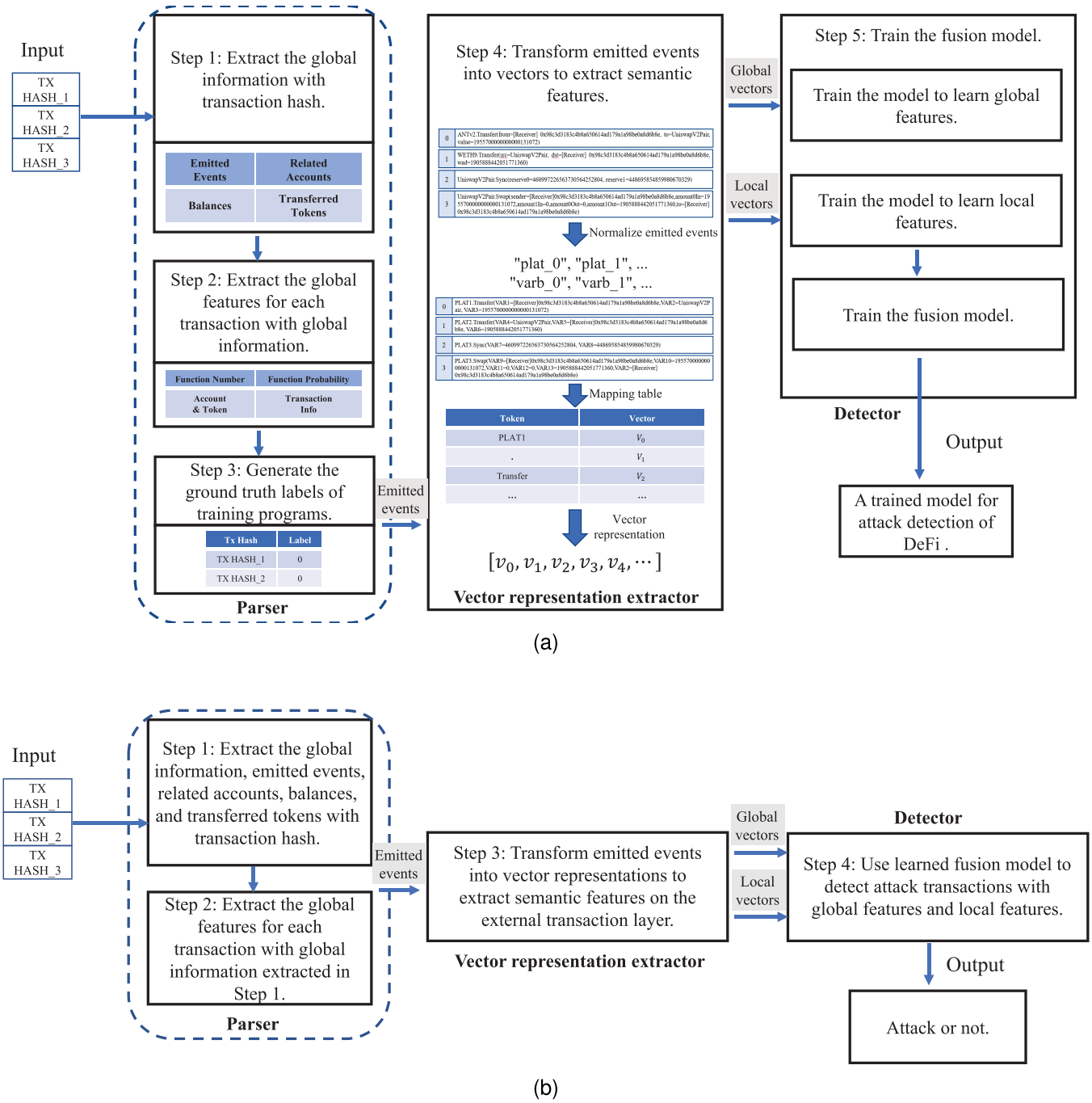


Fig. 4. Overview of DeFiScanner: the training phase generates patterns of DeFi normal transactions, and the detection phase uses learned patterns to determine whether a target transaction is an attack or not. (a) Training phase. It can be divided into three parts, including a parser, a vector representation extractor, and a detector. (b) Detection phase.

0	ANTv2.Transfer(from=[Receiver] 0x98c3d3183c4b8a650614ad179a1a98be0a8d6b8e, to=UniswapV2Pair, value=195570000000000131072)
1	WETH9.Transfer(src=UniswapV2Pair, dst=[Receiver] 0x98c3d3183c4b8a650614ad179a1a98be0a8d6b8e, wad=190588844205177136O)
2	UniswapV2Pair.Sync(reserve0=4609972265637305642528O4, reserve1=44869585485998067O329)
3	UniswapV2Pair.Swap(sender=[Receiver]0x98c3d3183c4b8a650614ad179a1a98be0a8d6b8e,amount0In=195570000000000131072,amount1In=0,amount0Out=0,amount1Out=190588844205177136O,to=[Receiver] 0x98c3d3183c4b8a650614ad179a1a98be0a8d6b8e)

Fig. 5. Emitted events of a raw transaction.

Transform emitted events into certain symbolic representations, denoted by $S_1 = \text{Map}(S)$. The purpose of this function is to filter some noise of internal transactions, as follows:

$$\text{Map}\left(\begin{pmatrix}\text{dYdX}\\ \text{Uniswap}\\ \vdots\end{pmatrix}, \begin{pmatrix}\text{from}\\ \text{to}\\ \vdots\end{pmatrix}\right) = \begin{pmatrix}\text{plat\_0}\\ \text{plat\_1}\\ \vdots\end{pmatrix}, \begin{pmatrix}\text{var\_0}\\ \text{var\_1}\\ \vdots\end{pmatrix}. \tag{1}$$

We observe that different transactions involve different DEX platforms, and the combinations of protocols are also different, (e.g., platform names, and variables), which may affect the effectiveness of extracted semantic features and

TABLE I
GLOBAL FEATURES OF ACCOUNT LEVEL AND EXTERNAL TRANSACTION LAYER

Function Number		Function Probability		Account & Token	Transaction Info
transferNum	withdrawNum	transferPr	withdrawPr	addressNum	txCostETH
flashloanNum	collectfeeNum	flashloanPr	collectfeePr	tokenNum	txCostUSD
depositNum	mintNum	depositPr	mintPr	amountTokenMax	gasUsed
borrowNum	collectNum	borrowPr	collectPr	senderMaxOutdegree	gasUsedGwei
approvalNum	rewardNum	approvalPr	rewardPr	senderMaxIndegree	
swapNum	liquidityNum	swapPr	liquidityPr		
unknownNum	sumNum	unknownPr	funEntropy		

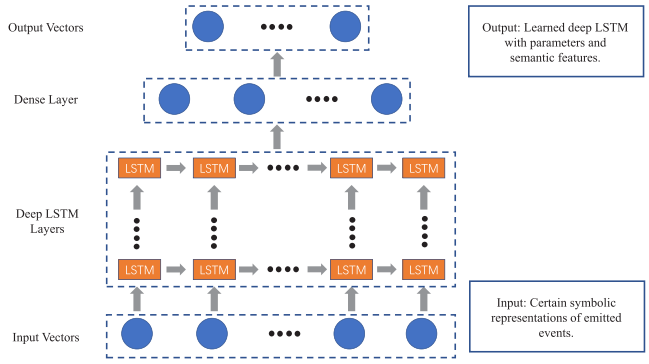


Fig. 6. LSTM neural network.

Fig. 7. Example of illustrating emitted event normalization.

reduce the performance of DeFiScanner. This motivates us to normalize emitted events by mapping the same variables and platform names to constants. We rename different platform names and variables according to the order of their appearance (i.e., "plat_0," "plat_1," ..., "varb_0," "varb_1," ...). However, the functions and parameter values are not mapped because the function names are standard in the DeFi smart contracts, such as "Transfer," "Swap," and "FlashLoan." Then we encode symbolic emitted events and train a neural network, as shown in Figs. 6 and 7 gives an example of emitted events' normalization. As follows:

$$V = \text{word2vec}(S_1) = (v_1, v_2, \ldots, v_m). \tag{2}$$

The output h_t^l of the each layer in LSTM at time t can be denoted as

$$h_t^l = o_t^l * \tanh\left(C_t^l\right) \tag{3}$$

where o_t^l is the output gate, and C_t^l is the state of LSTM cell

$$o_t^l = \sigma\left(W_o \cdot \left[h_{t-1}^l \oplus V_t^l\right] + b_o\right) \tag{4}$$

and

$$\begin{aligned} C_t^l &= f_t^l * C_{t-1}^l + i_t^l * \tilde{C}_t^l \\ \tilde{C}_t^l &= \tanh\left(W_C \cdot \left[h_{t-1}^l \oplus V_t^l\right] + b_C\right) \\ i_t^l &= \sigma\left(W_i \cdot \left[h_{t-1}^l \oplus V_t^l\right] + b_i\right) \\ f_t^l &= \sigma\left(W_f \cdot \left[h_{t-1}^l \oplus V_t^l\right] + b_f\right) \end{aligned} \tag{5}$$

where $*$ denotes the elementwise multiplication, $\oplus$ denotes the concatenation of vectors, b_i is the bias of input gate, b_f is the bias of the forget gate, b_C is the bias of the cell, $\tanh x = (\sinh x / \cosh x) = (e^x - e^{-x}/e^x + e^{-x})$ is the hyperbolic tangent function, and $\sigma(x) = (1/1 + e^{-x})$ denotes the sigmoid function. $W_C = [w_{\text{hc}}^l, w_{\text{vc}}^l]$, $W_i = [w_{\text{hi}}^l, w_{\text{vi}}^l]$, and

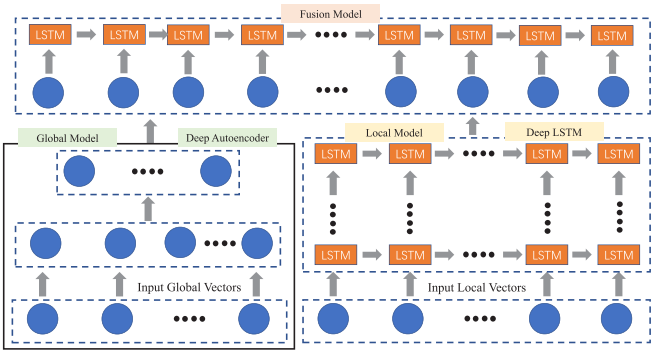


Fig. 8. Neural network architecture in DeFiScanner.

$W_f = [w_{\text{hf}}^l, w_{\text{vf}}^l]$ are the weight matrices of neural network. V_t^l denotes the input of each layer $l - 1 (l > 1)$ and also denotes the vectorized emitted events that inputted to the first layer $(l = 1)$. The output of the last layer is L. We divide the symbolic representation of the emitted events into a series of tokens with lexical analysis, including the platform names, the function names, the variables, and the values. But it results in a large corpus of tokens. With the idea of text mining [69], we consider using the word2vec to convert these tokens into vectors with a fixed length [70].

Step 5: Train the fusion model with global features and local features. Inspired by the multifeature fusion method in code vulnerability detection [71], we propose a new neural network architecture that uses the LSTM as a building block, as shown in Fig. 8. Because emitted events are time-dependent and LSTM is shown to be capable of extracting temporal features.

The network is able to extract two kinds of features: global features learned from Table I and local features learned from emitted events. Since global features and local features are extracted from different layers, we use a feature fusion layer to learn comprehensive features

$$f^f = M^g(G) \oplus M^l(L) \tag{6}$$

where f^f denotes the fusion features, M^g denotes the global model, and M^l denotes the local model.

The network consists of a deep autoencoder network and two LSTM networks. The deep autoencoder network is regarded as learning the global features, and the two LSTM networks are regarded as the local-feature learning model and the feature-fusion model, as shown in Fig. 8. To learn the local features from emitted events, the model uses preprocessing layers and deep LSTM layers. The preprocessing layer mainly filters "0" to fix the length of vectors, and the deep LSTM layer is mainly used to extract local features. We build the fusion model with a fusion layer, deep LSTM layers, and a dense layer. The global features and local features are fused with the fusion layer. The deep LSTM layers are mainly used to extract features at the semantic level. To detect attack transactions, we use the reconstruction error of the fused features as follows:

$$L\left(f_i{}^f, M^f\left(f_i{}^f\right)\right) = \sum_{i=1}^{n} \left\| f_i{}^f - M^f\left(f_i{}^f\right) \right\|_2^2 \tag{7}$$

where i denotes the number of samples used to train the fusion model.

2) As shown in Fig. 4(b), the detection phase contains the following steps.

Step 1: Extract the global information (similar to step 1 in the training phase).

Step 2: Extract the global features (similar to step 2 in the training phase).

Step 3: Transform emitted events into vectors (the same as Step 4 in the training phase).

Step 4: Use the learned fusion model to detect attack transactions with global features and local features.

V. Experiment Setup

A. Evaluation Dataset

We collect 50910 real-world transactions on DeFi that have been verified between January 2021 and April 2022. For the attacks in the dataset, we limit our collection to the DeFi attacks appearing on Ethereum. We obtain detailed data through the public transaction hashes of attacks and construct ground-truth labels by manually inspecting the files. Especially, the testing dataset contains some realistic complex transactions that have hundreds of emitted events. Since normal transactions account for a large proportion of all the transactions in the real environment, the proportion of positive and negative samples in the dataset is unbalanced. With the idea of anomaly detection, we train the fusion model with normal transactions to learn the distribution of normal transactions. To fit the distribution of real-world data, we set the ratio of normal transactions to attacks in the testing dataset to 10:1.

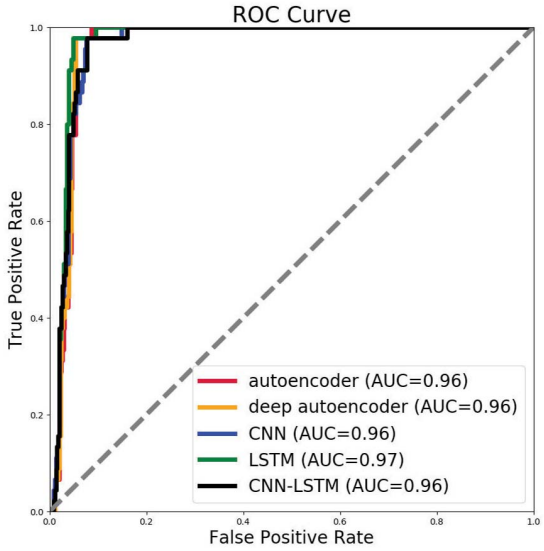


Fig. 9. ROC curve of the deep learning models.

In the dataset, the global information of external transactions and internal transactions are saved respectively. The global information of external transactions includes the gas used, the timestamp, and so on. For the global information of internal transactions, we collect the numbers of each function called, the amount of each token related, and the degree of each account. We also collect the corresponding emitted events of each transaction. Due to the transaction hashes being unique to each transaction, we only label the transaction hashes.

B. Training Phase

The training phase is divided into the following four steps.

Step 1: To generate global features from training data, we input the features shown in Table I into the global model.

Step 2: We identify and rename the noise to normalize emitted events, such as the names of platforms. Then we use word2vec to generate the vectors of local features. Each vector represents an emitted event with 50 vector dimensions. Let l_1, l_2, and l_3, respectively, denote the length of global features, local features, and fusion features. In the training phase, we tune l_1, l_2 and l_3 for different performances. We set $l_1 = 37$, $l_2 = 50$, and $l_3 = l_1 + l_2$. Since the length of the input is fixed by the local model, for emitted events with fewer words and the length of vectors is less than 50, we pad 0 at the end to fix the length of vectors to l_2. We abandon the tail of vectors to match length l_2 of which the length is more than 50.

Step 3: We get the representation of global features with the global model and extract semantic features (called local features) of emitted events with the local model.

Step 4: We continuously fine-tune the parameters of the model training phase (such as optimizer, learning rate, batch size, dropout, number of layers, number of neurons, and epoch). We also use expert knowledge to guide an exhaustive search to quickly find the best values of these hyperparameters. We first make the global model and the local model perform best by adjusting the parameters. Second, we freeze the

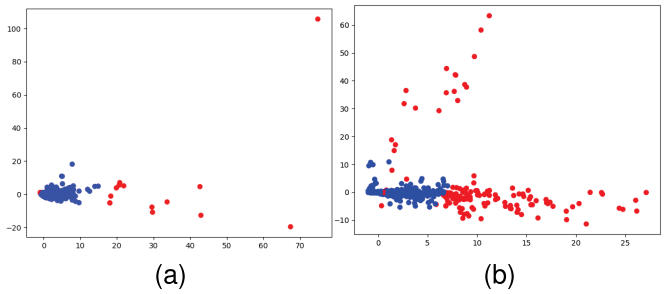


Fig. 10. Performance of k-means with global features and fusion features, respectively. (a) Global features. (b) Fusion features.

TABLE II
DETECTION PERFORMANCE COMPARISON OF SEVEN MODELS

Model	FPR	TPR	precision	F1-socre
SVM	98.63%	**99.45%**	6.70%	12.56%
K-means	19.37%	42.22%	31.67%	58.91%
Autoencoder	9.11%	97.78%	51.76%	67.69%
DeepAutoencoder	8.89%	97.78%	52.38%	68.22%
LSTM	**4.44%**	91.11%	**67.21%**	**77.36%**
CNN	6.44%	93.33%	59.15%	72.41%
LSTM-CNN	5.78%	91.11%	61.19%	73.21%

parameters of the global model and the local model to tune the hyperparameters of the fusion model. Finally, we get the attack detection model.

Unlike classification models that use a softmax activation function at the output layer and train the model with a categorical cross-entropy loss function, we use a sigmoid activation function at the output layer and train the fusion model with the reconstruction loss function.

C. Detection Phase

We perform four steps to detect attacks. The first three steps are consistent with the corresponding steps in the training phase. In step 4, the reconstruction loss is used to distinguish whether a transaction is an attack with the trained model.

D. Evaluation Metrics

We use the TPR or recall, false positive rate (FPR), precision (P), and F1-measure (F1) to evaluate DeFiScanner. The TPR or recall metric $\text{TPR} = (\text{TP}/\text{TP} + \text{FN})$ measures the ratio of true positive (TP) attacks to the entire population of samples that are attacks. The FPR metric $\text{FPR} = (\text{FP}/\text{FP} + \text{TN})$ measures the ratio of false positive (FP) attacks to the entire population of samples that are not attacks. The precision metric $\text{P} = (\text{TP}/\text{TP} + \text{FP})$ measures the correctness of the detected attacks. The F1-score metric $\text{F1} = (2 \cdot \text{P} \cdot \text{TPR}/\text{P} + \text{TPR})$ takes consideration of both precision and TPR. FP is the number of normal transactions detected as attacks, false negative (FN) is the number of samples for which real attacks are not detected, TP is the number of samples for which real attacks are detected, and true negative (TN) is the number of samples for which normal transactions are not detected.

It would be ideal that an attack detection system neither misses attacks (i.e., $\text{TPR} \approx 1$) nor triggers false alarms (i.e., $\text{FPR} \approx 0$ and $\text{P} \approx 1$).

VI. EXPERIMENTS AND RESULTS

Our experiments focus on answering the following two questions (RQs).

RQ1: Is DeFiScanner an effective attack detection system?

The attack detection systems for logic vulnerabilities on DeFi should detect both normal transactions and attacks. To answer this question, we conduct experiments involving normal transactions and attacks.

RQ2: Can DeFiScanner extract contextual information?

If the fusion model plays an important role in the system? To answer this question, we investigate the effectiveness of using global features without the fusion model.

A. Experiments for Answering RQ1

Table II summarizes the experimental results. We use seven models to detect attacks, namely, SVM, k-means, autoencoder, deep autoencoder, LSTM, CNN, and LSTM-CNN. These models are evaluated on the same testing dataset. k-means is a representative method of unsupervised learning. SVM is the representative method of supervised learning. For autoencoder, deep autoencoder, LSTM, CNN, and LSTM-CNN, these models are representatives of the self-supervised methods in deep learning. We observe that the FPR and TPR of k-means are 19.37%, and 42.22%, respectively. The precision and F1-score of k-means are 31.76%, and 58.91%, respectively. The FPR, TPR, precision, and F1-score of SVM are 98.63%, 99.45%, 6.70%, and 12.56%, respectively. Compared with SVM, k-means is 75% lower in terms of FPR and 55% lower in TPR. This can be explained that SVM cannot detect normal transactions and attacks at the same time, because both the FPR and TPR reach 90%, which means that regardless of whether the input is an attack or a normal transaction, the model judges it as an attack.

It can be seen that the self-supervised learning methods perform best in detecting both normal transactions and attack transactions simultaneously. However, autoencoder performs the worst among the self-supervised learning methods. The FPR, TPR, precision, and F1-score of autoencoder are 9.11%, 97.78%, 51.76%, and 67.69%, respectively. Compared with k-means, the autoencoder is 10.26% lower in terms of FPR, 55.56% higher in terms of TPR, 20.09% higher in terms of precision, and 8.78% higher in terms of F1-score. This may be due to the fact that the self-supervised learning model can learn the distribution of normal transactions, resulting in a smaller FPR and a larger TPR. Deep autoencoder, LSTM, CNN, and LSTM-CNN also perform well. Among the self-supervised learning methods, the LSTM model performs the best. The FPR of the LSTM is 4.67% lower than that of the autoencoder, which performs better on the TPR metric. But in practice, we will prioritize reducing the FPR. In addition, the accuracy and F1-score also show that the LSTM model is more effective than other self-supervised learning methods.

Due to the internal transactions containing specific logic and being executed sequentially, the emitted events have time dependencies among them. The emitted events of each external transaction can be viewed as time series data. It can be

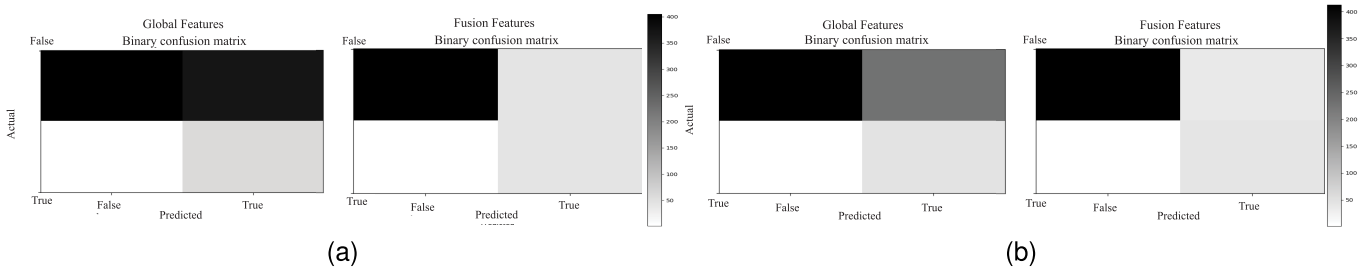


Fig. 11. Confusion matrix of (D)AE with global features and fusion features, respectively. (a) Autoencoder. (b) DeepAutoencoder.

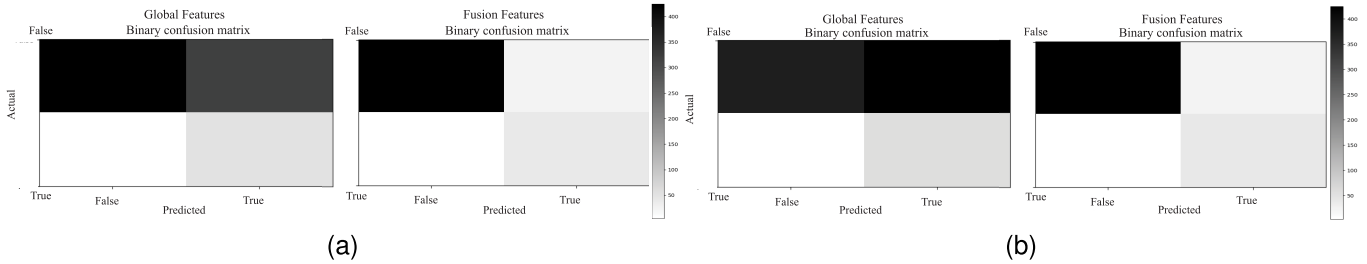


Fig. 12. Confusion matrix with global features and fusion features, respectively. (a) CNN. (b) LSTM.

explained why LSTM performs more effectively than other self-supervised learning models.

We further compare the ROC curves of all the models. The ROC curve can remain unchanged though the distribution of positive and negative samples in the testing set changes. So the ROC curve is an important metric when the samples in the dataset are imbalanced. The larger the area under the ROC curve, the better the detection effect of the model. Fig. 9 shows plots of the ROC curve of autoencoder, deep autoencoder, CNN, LSTM, and CNN-LSTM, respectively. The abscissa of the plane is the FPR and the ordinate is the TPR. We can observe that LSTM performs better than CNN and CNN-LSTM. The area under the ROC curve of LSTM reaches 0.97.

The ROC results also confirm that the emitted events are one kind of time series data. The performance is not good as using LSTM alone when we combine the CNN with LSTM in DeFiScanner.

Insight 1: The deep learning model that is trained with fusion features can perform well in attack transaction detection on DeFi.

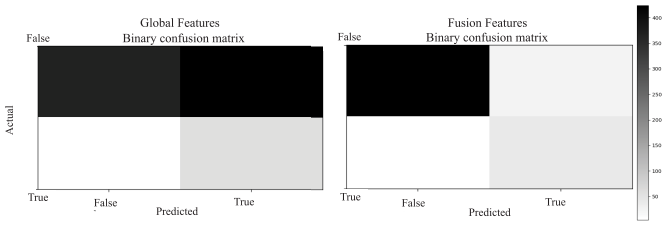


Fig. 13. Confusion matrix of CNN-LSTM with features and fusion features, respectively.

TABLE III
DETECTION PERFORMANCE COMPARISON OF SEVEN MODELS WITH GLOBAL FEATURES

Model	FPR	TPR	precision	F1-socre
SVM	98.63%	**99.99%**	3.67%	7.07%
K-means	**19.60%**	26.67%	2%	**37.21%**
Autoencoder	40.89%	84.44%	17.11%	28.46%
DeepAutoencoder	37.78%	84.44%	**18.27%**	30.04%
LSTM	56.89%	86.67%	13.22%	22.94%
CNN	63.33%	88.89%	12.31%	21.62%
LSTM-CNN	53.56%	84.44%	13.62%	23.46%

B. Experiments for Answering RQ2

To answer whether DeFiScanner is able to extract contextual information of events effectively, and evaluate the importance of the fusion model in the system, we conduct experiments with the global features to detect attacks.

Table III summarizes the experimental results. These models do not perform as well as models trained by fusion features. We observe that the FPR of LSTM trained by global features is 56.89%. Compared with LSTM trained with fusion features, the LSTM trained with global features has 52.45% higher in terms of FPR, 53.99% lower in terms of precision, and 54.42% lower in terms of F1-score.

Without the local features, the performance of DeFiScanner is significantly lower. We can conclude that there is much useful information in emitted events, and the features extracted from emitted events are important to DeFiScanner to realize attack detection. The local model in DeFiScanner can extract the key features to improve the performance.

Fig. 10 shows plots of the detection result of k-means with global features and the detection result of k-means with fusion features. The red points represent attacks and the blue points represent normal transactions of DeFi. It can be seen that when we fuse global features and local features, the model can effectively improve the performance of separating the attacks from the normal transactions.

Insight 2: DeFiScanner can effectively extract the high-level semantic features of internal transactions. The model trained

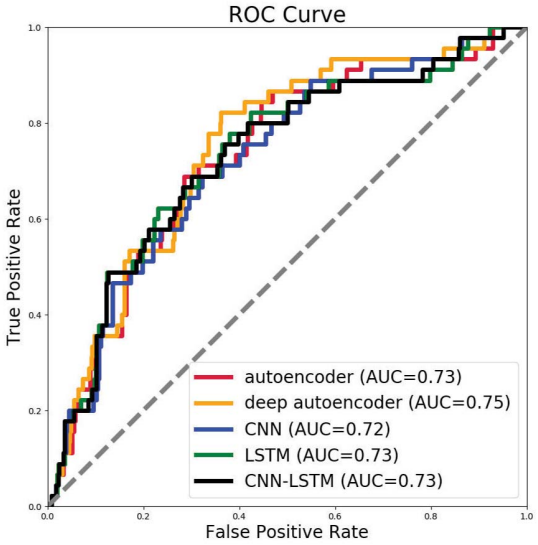


Fig. 14. ROC curve of the deep learning models without local features.

with global features cannot have a good performance in the task of DeFi attack transaction detection.

Finally, we compare the confusion matrices generated using global features and fusion features to detect attacks, as shown in Figs. 11–13. The confusion matrix is a specific matrix used to visualize the performance of an algorithm. Each column represents the predicted class, and each row represents the actual class. All the correct predictions are on the diagonal. The experimental results also validate the effectiveness of the DeFiScanner.

We also plot the ROC curves of these deep learning models trained by global features, respectively, as shown in Fig. 14. We can observe that the area under the ROC curve is 0.2 lower than the fusion models.

In a word, the local features and global features are all useful for detecting DeFi attacks, but the local features play a more important role in DeFiScanner than global features. The global features include block number, timestamp, transaction cost, the gas used, and the statistical features from the external transaction level. We often use these features to detect money laundering, frauding, and Ponzi schemes. Because the anomalies of these behaviors perform on the external transaction level and account level. Due to the profit of attacks reflected on the account level, and the complicated internal transactions often cause more transaction costs and the gas used, these global features can also assist DeFiScanner to detect DeFi attacks. The local features are extracted from the emitted events, and the detailed steps of attacks are the key to detecting DeFi attacks. The experimental results also demonstrate the discussion.

There exist limitations of DeFiScanner. First, DeFiScanner is limited to detecting attacks on DeFi based on the assumption of the availability for emitted events of raw transactions. Second, the present implementation of DeFiScanner only deals with DeFi attacks on the transaction layer. It cannot discover what codes cause the logic vulnerabilities or analyze the flow of funds stolen by attackers. We will focus on this in our future work.

VII. Conclusion

In this work, we are motivated to detect attacks on DeFi applications such as flash loan attacks and price manipulation attacks. We present the first deep-learning-based attack detection system on DeFi, named DeFiScanner. It processes the unstructured emitted events generated by DeFi applications with word2vec and uses the fusion model to extract various high-level semantics. First, the unstructured emitted events generated by DeFi applications are automatically processed with word2vec. Second, a series of features from the account level and external transaction level are extracted to enrich high-level features. Finally, the features of different levels are fused efficiently with the designed fusion model. We use the representative methods of an unsupervised learning, namely, k-means, a supervised classification algorithm, namely, SVM, and five self-supervised methods in deep learning, namely, autoencoder, deep autoencoder, CNN, LSTM, and CNN-LSTM, to conduct comparative experiments. Comprehensive experimental results show that DeFiScanner can extract high-level DeFi semantics with the deep learning models and perform well in the detection of DeFi attacks.

References

[1] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, Apr. 2017.

[2] J. Kamps and B. Kleinberg, "To the moon: Defining and detecting cryptocurrency pump-and-dumps," *Crime Sci.*, vol. 7, no. 1, pp. 1–18, Dec. 2018.

[3] J. Xu and B. Livshits, "The anatomy of a cryptocurrency pump-and-dump scheme," in *Proc. 28th USENIX Secur. Symp. (USENIX Secur.)*, 2019, pp. 1609–1625.

[4] P. Daian et al., "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 910–927.

[5] S. Eskandari, S. Moosavi, and J. Clark, "SoK: Transparent dishonesty: Front-running attacks on blockchain," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2019, pp. 170–189.

[6] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 428–445.

[7] K. Qin, L. Zhou, B. Livshits, and A. Gervais, "Attacking the DeFi ecosystem with flash loans for fun and profit," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2021, pp. 3–32.

[8] *88mph Incident*. Accessed: Nov. 2021. [Online]. Available: https://peckshield.medium.com/88mph-incident-root-cause-analysis-ce477e00a74d

[9] *Akropolis Incident*. Accessed: Nov. 2021. [Online]. Available: https://peckshield.medium.com/akropolis-incident-root-cause-analysis-c11ee59e05d4

[10] *BZX Hack I*. Accessed: Nov. 2021. [Online]. Available: https://peckshield.medium.com/bzx-hack-full-disclosure-with-detailed-profit-analysise6b1fa9b18fc

[11] *Cheese Bank Incident*. Accessed: Nov. 2021. [Online]. Available: https://peckshield.medium.com/cheese-bank-incident-root-cause-analysis-d076bf87a1e72020

[12] *Opyn Incident*. Accessed: Nov. 2021. [Online]. Available: https://peckshield.medium.com/opyn-hacks-root-cause-analysis-c65f3fe249db

[13] L. Cheng et al., "SCTSC: A semicentralized traffic signal control mode with attribute-based blockchain in IoVs," *IEEE Trans. Computat. Social Syst.*, vol. 6, no. 6, pp. 1373–1385, Dec. 2019.

[14] C. F. Torres, M. Baden, R. Norvill, B. B. F. Pontiveros, H. Jonker, and S. Mauw, "ÆGIS: Shielding vulnerable smart contracts against attacks," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 584–597.

[15] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "ContractWard: Automated vulnerability detection models for Ethereum smart contracts," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 1133–1144, Apr. 2021.

[16] L. Li et al., "CreditCoin: A privacy-preserving blockchain-based incentive announcement network for communications of smart vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 7, pp. 2204–2220, Jan. 2018.

[17] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, "Privacy risk analysis and mitigation of analytics libraries in the Android ecosystem," *IEEE Trans. Mobile Comput.*, vol. 19, no. 5, pp. 1184–1199, May 2020.

[18] W. Wang, Y. Shang, Y. He, Y. Li, and J. Liu, "BotMark: Automated botnet detection with hybrid analysis of flow-based and graph-based traffic behaviors," *Inf. Sci.*, vol. 511, no. 2, pp. 284–296, Feb. 2020.

[19] W. Wang, M. Zhao, and J. Wang, "Effective Android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *J. Ambient Intell. Hum. Comput.*, vol. 10, no. 8, pp. 3035–3043, 2019.

[20] W. Wang et al., "HGATE: Heterogeneous graph attention auto-encoders," *IEEE Trans. Knowl. Data Eng.*, early access, Dec. 28, 2021, doi: 10.1109/TKDE.2021.3138788.

[21] Z. Wang, H. Jin, W. Dai, K.-K.-R. Choo, and D. Zou, "Ethereum smart contract security research: Survey and future research opportunities," *Frontiers Comput. Sci.*, vol. 15, no. 2, pp. 1–18, Apr. 2021.

[22] C. Wang et al., "Demystifying Ethereum account diversity: Observations, models and analysis," *Frontiers Comput. Sci.*, vol. 16, no. 4, pp. 1–12, Aug. 2022.

[23] Q. Zhenquan, J. Ye, J. Meng, B. Lu, and L. Wang, "Privacy-preserving blockchain-based federated learning for marine Internet of Things," *IEEE Trans. Computat. Social Syst.*, vol. 9, no. 1, pp. 159–173, Feb. 2021.

[24] X. Xu, Q. Liu, X. Zhang, J. Zhang, L. Qi, and W. Dou, "A blockchain-powered crowdsourcing method with privacy preservation in mobile environment," *IEEE Trans. Computat. Social Syst.*, vol. 6, no. 6, pp. 1407–1419, Dec. 2019.

[25] P. Li, J. Lai, and Y. Wu, "Accountable attribute-based authentication with fine-grained access control and its application to crowdsourcing," *Frontiers Comput. Sci.*, vol. 17, no. 1, pp. 1–14, Feb. 2023.

[26] P. Zhang and M. Zhou, "Security and trust in blockchains: Architecture, key technologies, and open issues," *IEEE Trans. Computat. Social Syst.*, vol. 7, no. 3, pp. 790–801, Jun. 2020.

[27] G. Cheng, Y. Chen, S. Deng, H. Gao, and J. Yin, "A blockchain-based mutual authentication scheme for collaborative edge computing," *IEEE Trans. Computat. Social Syst.*, vol. 9, no. 1, pp. 146–158, Feb. 2022.

[28] Y. Zhang, X. Xu, A. Liu, Q. Lu, L. Xu, and F. Tao, "Blockchain-based trust mechanism for IoT-based smart manufacturing system," *IEEE Trans. Computat. Social Syst.*, vol. 6, no. 6, pp. 1386–1394, Dec. 2019.

[29] Y. Xie, G. Liu, C. Yan, C. Jiang, M. Zhou, and M. Li, "Learning transactional behavioral representations for credit card fraud detection," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Oct. 5, 2022, doi: 10.1109/TNNLS.2022.3208967.

[30] Z. Li, M. Huang, G. Liu, and C. Jiang, "A hybrid method with dynamic weighted entropy for handling the problem of class imbalance with overlap in credit card fraud detection," *Expert Syst. Appl.*, vol. 175, Aug. 2021, Art. no. 114750.

[31] C. Yang, G. Liu, C. Yan, and C. Jiang, "A clustering-based flexible weighting method in AdaBoost and its application to transaction fraud detection," *Sci. China Inf. Sci.*, vol. 64, no. 12, pp. 1–11, Dec. 2021.

[32] L. Zheng, G. Liu, C. Yan, C. Jiang, and M. Li, "Improved TrAdaBoost and its application to transaction fraud detection," *IEEE Trans. Computat. Social Syst.*, vol. 7, no. 5, pp. 1304–1316, Oct. 2020.

[33] C. Jiang, J. Song, G. Liu, L. Zheng, and W. Luan, "Credit card fraud detection: A novel approach using aggregation strategy and feedback mechanism," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3637–3647, Oct. 2018.

[34] Q. Bai, C. Zhang, N. Liu, X. Chen, Y. Xu, and X. Wang, "Evolution of transaction pattern in Ethereum: A temporal graph perspective," *IEEE Trans. Computat. Social Syst.*, vol. 9, no. 3, pp. 851–866, Jun. 2022.

[35] D. Lin, J. Chen, J. Wu, and Z. Zheng, "Evolution of Ethereum transaction relationships: Toward understanding global driving factors from microscopic patterns," *IEEE Trans. Computat. Social Syst.*, vol. 9, no. 2, pp. 559–570, Apr. 2022.

[36] S. Li, Y. Yuan, J. J. Zhang, B. Buchanan, E. Liu, and R. Ramadoss, "Guest editorial special issue on blockchain-based secure and trusted computing for IoT," *IEEE Trans. Computat. Social Syst.*, vol. 6, no. 6, pp. 1369–1372, Dec. 2019.

[37] Y. Zhang, W. Yu, Z. Li, S. Raza, and H. Cao, "Detecting Ethereum Ponzi schemes based on improved LightGBM algorithm," *IEEE Trans. Computat. Social Syst.*, vol. 9, no. 2, pp. 624–637, Apr. 2022.

[38] J. Bhatta, D. Shrestha, S. Nepal, S. Pandey, and S. Koirala, "Efficient estimation of Nepali word representations in vector space," *J. Innov. Eng. Educ.*, vol. 3, no. 1, pp. 71–77, Mar. 2020.

[39] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1997.

[40] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI Mag.*, vol. 17, no. 3, p. 37, 1999.

[41] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[42] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[43] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and Cooperation in Neural Nets*. Cham, Switzerland: Springer, 1982, pp. 267–285.

[44] *Coinbase*. Accessed: Nov. 2021. [Online]. Available: https://www.coinbase.com/

[45] C. Busayatananphon and E. Boonchieng, "Financial technology DeFi protocol: A review," in *Proc. Joint Int. Conf. Digit. Arts, Media Technol. ECTI Northern Sect. Conf. Electr., Electron., Comput. Telecommun. Eng. (ECTI DAMT NCON)*, Jan. 2022, pp. 267–272.

[46] Oxford Analytica, "Coinbase listing will pave way for other exchanges," Emerald Expert Briefings, no. oxan-es, Dec. 2021.

[47] *dYdX Matching Specification*. Accessed: Nov. 2021. [Online]. Available: https://legacy-docs.dydx.exchange/

[48] *Serum White Paper*. Accessed: Nov. 2021. [Online]. Available: https://projectserum.com/

[49] W. Warren and A. Bandeali. *0x: An Open Protocol for Decentralized Exchange on the Ethereum Blockchain*. [Online]. Available: https://github.com/0xProject/whitepaper

[50] *Waves Exchange*. Accessed: Nov. 2021. [Online]. Available: https://docs.waves.exchange/en

[51] S. Ivanov and A. Pupyshev. *Neutrino Protocol White Paper*. Accessed: Nov. 2021. [Online]. Available: https://wp.neutrino.at/

[52] E. Chen and A. Chon. *Injective Protocol: A Collision Resistant Decentralized Exchange Protocol*. Accessed: Nov. 2021. [Online]. Available: https://coinpare.io/whitepaper/injective-protocol.pdf

[53] A. A. Aigner and G. Dhaliwal, "UNISWAP: Impermanent loss and risk profile of a liquidity provider," 2021, *arXiv:2106.14404*.

[54] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson, "Uniswap v3 core," Uniswap, New York, NY, USA, Tech. Rep., 2021.

[55] M. Egorov. (2019). *StableSwap-Efficient Mechanism for Stablecoin Liquidity*. [Online]. Available: https://medium.com/yield-protocol/introducing-ydai-43a727b96fc7

[56] *The Sushiswap Project*. Accessed: Nov. 2021. [Online]. Available: https://sushichef.medium.com/the-sushiswap-project-dd6eb80c6ba2

[57] D. Stone, "Trustless, privacy-preserving blockchain bridges," 2021, *arXiv:2102.04660*.

[58] Raydium Team. *Raydium Protocol Litepaper*. Accessed: Nov. 2021. [Online]. Available: https://raydium.io/Raydium-Litepaper.pdf

[59] W. Li, J. Bu, X. Li, and X. Chen, "Security analysis of DeFi: Vulnerabilities, attacks and advances," 2022, *arXiv:2205.09524*.

[60] D. Wang et al., "Towards a first step to understand flash loan and its applications in DeFi ecosystem," in *Proc. 9th Int. Workshop Secur. Blockchain Cloud Comput.*, 2021, pp. 23–28.

[61] Y. Cao, C. Zou, and X. Cheng, "Flashot: A snapshot of flash loan attack on DeFi ecosystem," 2021, *arXiv:2102.00626*.

[62] B. Wang et al., "BLOCKEYE: Hunting for DeFi attacks on blockchain," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng., Companion Proc. (ICSE-Companion)*, May 2021, pp. 17–20.

[63] M. Team. (2020). *The Maker Protocol: Makerdao's Mulfi-Collateral DAI (MCD) System*. [Online]. Available: https://makerdao.com/en/whitepaper/

[64] Centre. *Centre Whitepaper*. Accessed: Nov. 2021. [Online]. Available: https://www.centre.io/pdfs/centre-whitepaper.pdf

[65] Tether. (2016). *Fiat Currencies on the Bitcoin Blockchain*. Accessed: Nov. 2021. [Online]. Available: https://tether.to/wp-content/uploads/2016/06/TetherWhitePaper.pdf

[66] Aave. *Aave*. Accessed: Nov. 2021. [Online]. Available: https://aave.com/

[67] dYdX. *dYdX*. Accessed: Nov. 2021. [Online]. Available: https://dydx.exchange/

[68] Etherscan. *Etherscan*. Accessed: Nov. 2021. [Online]. Available: https://etherscan.io/

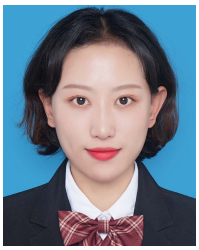
[69] L. Wolf, Y. Hanani, K. Bar, and N. Dershowitz, "Joint word2vec networks for bilingual semantic representations," *Int. J. Comput. Linguistics Appl.*, vol. 5, no. 1, pp. 27–42, 2014.

[70] G. Hinton, J. McClelland, and D. Rumelhart, *Distributed Representations. Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge, MA, USA: MIT Press, 1986, ch. 3, pp. 77–109.

[71] Z. Li et al., "VulDeePecker: A deep learning-based system for vulnerability detection," 2018, *arXiv:1801.01681*.

Bin Wang is currently pursuing the Ph.D. degree with Beijing Jiaotong University, Beijing, China.

His research interests include blockchain, data security, and machine learning.



Xiaohan Yuan is currently pursuing the Ph.D. degree with Beijing Jiaotong University, Beijing, China.

Her research interests include blockchain security and applications.

Li Duan received the Ph.D. degree in computer science and technology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2016.

She was a Research Fellow with Nanyang Technological University, Singapore, and the University of Science and Technology Beijing, Beijing. She is currently an Assistant Professor with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing. Her research interests are services computing and the Internet of Things, data security and privacy protection, and blockchain security and applications.

Dr. Duan received the National Natural Science Foundation of China, the Post-Doctoral Fund, and the Basic Scientific Research Project.

Hongliang Ma received the Ph.D. degree from Beijing Jiaotong University, Beijing, China, in 2018.

He is currently an Associate Professor with Shihezi University, Shihezi, China. His research interests mainly include artificial intelligence security, web security, and anomaly detection.

Bin Wang received the Ph.D. degree from the China National Digital Switching System Engineering and Technological Research and Development Center, Zhengzhou, China, in 2010.

He is currently a Professor of Zhejiang Key Laboratory of Multi-Dimensional Perception Technology, Application and Cybersecurity, Hangzhou, China, and Zhejiang University, Hangzhou. His research interests mainly include the Internet of Things security, cryptography, artificial intelligence security, and new network security architecture.

Chunhua Su received the B.S. degree from the Beijing Electronic and Science Institute, Beijing, China, in 2003, and the M.S. and Ph.D. degrees in computer science from the Faculty of Engineering, Kyushu University, Fukuoka, Japan, in 2006 and 2009, respectively.

He has worked as a Research Scientist with the Cryptography and Security Department, Institute for Infocomm Research, Singapore, from 2011 to 2013. From 2013 to 2016, he has worked as an Assistant Professor with the School of Information Science, Japan Advanced Institute of Science and Technology, Asahidai, Nomi-shi, Ishikawa. From 2016 to 2017, he worked as an Assistant Professor with the Graduate School of Engineering, Osaka University, Suita, Japan. He is currently working as a Senior Associate Professor with the Division of Computer Science, University of Aizu, Aizuwakamatsu, Japan. His research interests include cryptanalysis, cryptographic protocols, privacy-preserving technologies in data mining and the IoT security and privacy.

Wei Wang (Member, IEEE) received the Ph.D. degree from Xi'an Jiaotong University, Xi'an, China, in 2006.

He was a Post-Doctoral Researcher with the University of Trento, Trento, Italy, from 2005 to 2006. He was a Post-Doctoral Researcher with TELECOM Bretagne, Brest Cedex, France, and with INRIA, Paris, France, from 2007 to 2008. He was also a European ERCIM Fellow with the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, and with the Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Esch-sur-Alzette, Luxembourg, from 2009 to 2011. He visited INRIA, ETH, Switzerland, Zurich, NTNU, Trondheim, South-Trondelag, CNR, New York, America, New York University Polytechnic, New York, and King Abdullah University of Science and Technology, Thuwal, Saudi Arabia. He is a Full Professor with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China. He is an Elsevier highly cited Chinese researchers. He has authored or coauthored over 100 peer-reviewed papers in various journals and international conferences. His main research interests include mobile, computer, and network security.

Dr. Wang is an Editorial Board Member of *Computers & Security* and a Young AE of *Frontiers of Computer Science*.

Securing Deployed Smart Contracts and DeFi With Distributed TEE Cluster

Zecheng Li, Bin Xiao, *Senior Member, IEEE*, Songtao Guo, *Senior Member, IEEE*, and Yuanyuan Yang, *Fellow, IEEE*

Abstract—Smart contract technologies can be used to implement almost arbitrary business logic. They can revolutionize many businesses such as payments, insurance, and crowdfunding. The resulting birth of decentralized finance (DeFi) has gained significant momentum. Smart contracts and DeFi are now attractive targets for attacks. An important research question is how to protect deployed smart contracts and DeFi. Smart contracts cannot be modified once deployed, namely vulnerabilities cannot be fixed by patching. In this case, vulnerabilities in deployed contracts and DeFi might cause devastating consequences. In this paper, we put forward SolSaviour, a framework for protecting deployed smart contracts and DeFi. The core of SolSaviour is to build a smart contract protection mechanism based on democratic voting using a distributed trusted execution environment (TEE) cluster. Once a vulnerability in deployed contracts or DeFi is found, SolSaviour can destroy the defective contract and redeploy a patched contract via the distributed TEE cluster. Moreover, SolSaviour can migrate funds and state variables from the destroyed contract to the patched one. Compared with previous work, our approach can protect smart contracts and DeFi in a distributed manner, avoiding reliance on privileged users or trusted third parties. Our experiment results show that SolSaviour can protect smart contracts and complex DeFi protocols with feasible overhead.

Index Terms—Blockchain, DeFi security, decentralized finance (DeFi), smart contract, trusted execution environment (TEE)

1 Introduction

As the global market size of smart contract grows year by year, the security of smart contracts has raised huge concerns. Smart contracts and decentralized finance (DeFi) operate on a distributed blockchain environment. Their deployment and execution rely on the distributed consensus of the blockchain. Due to the tamper-proof feature of the blockchain, contracts and DeFi cannot be modified once deployed. Therefore, we cannot fix a deployed smart contract in the same way as patching a traditional application when the deployed contract has a vulnerability.

Vulnerabilities in deployed smart contracts and DeFi have caused significant losses in recent years due to the lack of proper protection methods. One of the most notorious incidents was the DAO hack [1], in which attackers exploited the reentrancy vulnerability in the DAO contract to steal ethers wantonly. During this attack, honest contract users can do nothing but to withdraw ethers to secure accounts as fast as possible. The DAO contract lost around 3.6 million ethers in this attack. In addition, with the increased popularity of DeFi applications, new types of attacks are appearing. In April 2022, the Fei protocol suffered from a reentrancy attack. The attacker discovered a reentrancy bug in Fei's collateral mechanism, meaning that it was possible to use the `fallback` function to release assets locked in the Fei contract when the loan arrived, and thus steal assets inside Fei. This attack caused around 28,380 ETH loss [2].

Spurred by these attacks, the community started to conduct research on detecting vulnerabilities before deployment. Many software analysis techniques are explored, including but not limited to symbolic execution [3], formal verification [4], static analysis [5], [6], dynamic analysis [7], and fuzzing testing [8]. However, these detection methods still have certain limitations. They cover limited types of vulnerabilities, have restricted detection efficiency (i.e., sensitive to some vulnerabilities, but insensitive to others), and suffer from possible false negative cases. We point out that for high-net-worth smart contracts, pre-deployment detection methods are not fully effective. It is possible that vulnerabilities may be discovered after deployment. Therefore, how to protect deployed smart contracts and DeFi remains a crucial problem.

Proxy pattern is a promising method to safeguard deployed smart contracts and DeFi. Using proxy pattern, a smart contract is separated into a proxy contract and an implementation contract. The proxy contract stores the address of implementation contract. Function calls to the proxy contract are forwarded to the implementation using `delegatecall` opcode, which executes the code of implementation contract with the data inside the proxy contract.

- *Zecheng Li and Bin Xiao are with the Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong. E-mail: {zechli, b.xiao}@polyu.edu.hk.*
- *Songtao Guo is with the College of Computer Science, Chongqing University, Chongqing 400044, China. E-mail: guosongtao@cqu.edu.cn.*
- *Yuanyuan Yang is with the Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794 USA. E-mail: yuanyuan.yang@stonybrook.edu.*

Manuscript received 26 June 2022; revised 11 November 2022; accepted 11 December 2022. Date of publication 27 December 2022; date of current version 13 January 2023.
This work was supported in part by Key-Area Research Development Program of Guangdong Provice under Grant 2020B0101090003, in part by HK RGC GRF under Grants PolyU 15209822 and PolyU 15217321.
(Corresponding author: Bin Xiao.)
Recommended for acceptance by Y. Yang.
Digital Object Identifier no. 10.1109/TPDS.2022.3232548

If a vulnerability is discovered in the implementation contract, the proxy pattern allows users to deploy a new patched implementation contract to replace the defective one. However, the proxy pattern still has some limitations, such as the proxy selector clashing problem, which can be exploited by attackers to steal assets inside contracts. In August 2022, a DeFi protocol Nomad was exploited and $190M assets were stolen [9]. Nomad bridge was deployed in proxy pattern and the vulnerability was introduced in an upgrade. In this case, though proxy pattern provides a possible solution to fix vulnerabilities in deployed smart contracts, it is still lack of the protection of inside assets. However, the asset in smart contracts and DeFi is, to some extent, the most important thing to protect.

In this paper, we focus on protecting deployed smart contracts and DeFi from external attacks or internal bugs, with a particular emphasis on securing inside assets. Considering that most high-net-worth contracts are multi-user scenarios, we implement the management of contracts in a democratic voting way. We propose the voteDestruct mechanism to allow contract participants (i.e., stakeholders) to vote on the contract's future. They can vote to lock, unlock, and destroy a potentially defective contract. The number of ethers (i.e., stake) they have deposited in the contract determines the weight of their votes. The more stake a stakeholder controls, the more weight its vote has. We also propose a distributed trusted execution environment (TEE) cluster to enable the management of contracts, such as update, destruction, and redeployment. The TEE cluster also takes charge of temporary asset escrow after destroying a defective smart contract and stake migration. By transferring the assets and state variables from defective contract to patched contract, TEE cluster protects the assets and guarantees the consistency of the contract state. For newly-deployed smart contracts, TEE cluster ensures that the data and internal stake distribution remains unchanged through state migration. For patched smart contracts, the TEE cluster deploys it and conducts the state migration to transfer all internal assets to the newly-deployed contract.

We achieve decentralized control of smart contracts and DeFi by multiple parties through a secure and principled combination of blockchain and trusted hardware. Assuming the integrity of the blockchain, users do not need to trust the validity, persistence, confidentiality, or correctness of smart contract creators, miners, or TEE nodes. SolSaviour thus can provide self-sustaining service even when some miners, contract creators, contract participants, or TEE nodes are unavailable.

In our preliminary conference version of this paper [10], we first present SolSaviour for repairing and recovering a defective smart contract. However, SolSaviour still has some drawbacks, such as external calls not being authenticated by the TEE cluster and no valid verification for generated patches. In this paper, we overcome these shortcomings and extend our work to the protection of DeFi. Some modules are added including a policy-based exception detector, an identity authentication module, and a patch tester. Complete API via which clients can invoke SolSaviour is provided. We also explore a method for redeploying a patched contract without TEE asset escrow. The main contributions of this paper are summarized as follow:

- We propose a voteDestruct mechanism to enable democratic voting on deployed smart contracts and DeFi, allowing stakeholders to make fair future decisions.
- We propose a distributed TEE cluster that allows contract stakeholders to replace the defective contract with a patched contract in a trusted manner. Our proposed TEE cluster can take charge of asset escrow and contract state migration. It also can verify the identity of message calls, preserve trusted execution of contract invocation, patching, and deployment. We also provide complete API of the TEE cluster.
- We give a thorough security analysis of SolSaviour in three aspects: balance security, correctness, and fairness.
- We collect smart contracts and DeFi protocols that were attacked in the past and use them to evaluate the effectiveness and performance of SolSaviour. Experiment results show that SolSaviour can effectively mitigate the loss caused by smart contract vulnerabilities with little overhead.

The remainder of this paper is organized as follows. Section 2 gives some background knowledge of this paper. In Section 3, we introduce the overview, workflow, building blocks, and API of SolSaviour. The detailed implementation is presented in Section 4. We analyze the security of SolSaviour in Section 5. We discuss potential improvement of SolSaviour in 6. The effectiveness and performance of SolSaviour are evaluated in Section 7. We present related work in Section 8 and conclude our work in Section 9.

2 Background

2.1 Smart Contract and DeFi

Smart Contract. A smart contract is a distributed computer program that runs in a blockchain environment. Supported by the underlying blockchain, smart contracts can store arbitrary state and perform arbitrary computation. The deployment and invocation of a smart contract is achieved by sending transactions to the blockchain. Message call transactions that conform to the internal logic of the contract are executed by miners and later included in the new blocks.

DeFi. DeFi applications are essentially smart contracts running on the blockchain. They can provide financial instruments that do not rely on third-party intermediaries, such as exchanges and banks. DeFi allows users to carry out financial services such as lending, investing in derivatives, and insuring. Users can store their money in a secure digital wallet and interact with DeFi through message call transactions. In DeFi, transactions are not executed through traditional centralized exchanges, but between participants and mediated via smart contracts. Since a DeFi application typically consists of multiple contracts, the risk of smart contract errors increases.

Internal State. The internal state indicates the values of contract variables and the stake distribution inside the contract. While migrating and upgrading contracts, the consistency of contract internal state should be maintained. People could recover the value of variables inside a smart

contract. The `getter` function can be used to acquire values of contract variables. The stake distribution can be recovered in a similar way as long as the contract explicitly define variables to store stake distribution. However, it is non-trivial to migrate the stake distribution from defective contracts to patched contracts, which requires actual transactions of funds. In this case, assets are transferred from defective contract account to patched contract account, in accordance with the stake distribution inside defective contract.

2.2 Defining Defects

Smart contracts and DeFi are subject to a wide variety of defects. Defective smart contracts can be divided into two categories: exploitable smart contracts and unexpected smart contacts (i.e., may have unexpected internal states). For the first type, either there are some problems within the contract implementation that create bugs (e.g., reentrancy vulnerability), or an attacker can exploit the contract internal logic to launch attacks (e.g., front running attack). Attackers can gain benefits that do not belong to themselves by exploiting these defects. For the second type, these defects may cause a smart contract to an unexpected state (e.g., a locked state). For example, a jackpot may never succeed because of a strictly equal operation [11]. Regardless of the defect, effective protection measures are needed to prevent potential asset loss.

2.3 Contract Protection

Generally, we define the defending methods of smart contracts and DeFi as repairing and recovering techniques. Repairing technique can alleviate the bugs in a smart contract, and recovering technique can save a contract from serious states.

Currently, almost all contract remediation techniques focus on repairing smart contracts before deployment. Some efforts identify vulnerabilities by statically analyzing the contract code and generating the appropriate patches. Other efforts, such as runtime validation, determine if a deployed contract is vulnerable and generating appropriate patches. The two efforts are similar in their inability to protect deployed smart contracts and DeFi. In addition, they cannot fix vulnerabilities that have not been detected. Nor can they protect the assets in deployed smart contracts.

One possible solution for protecting deployed contracts and DeFi is the proxy pattern, where the smart contract is separated into a proxy contract and a implementation contract. Once an error is exposed, a new implementation contract is deployed to replace the defective one. However, this approach is subject to the requirements of a trusted contract owner. That is, the contract developer sets its own address as a super user for future maintenance, which is a strong trust assumption.

2.4 Distributed TEE Cluster

TEE is a hardware-level trusted computing technology in which the CPU divides a portion of the memory area to ensure that internally loaded code and data are protected in terms of integrity and confidentiality. Integrity ensures that software outside the TEE cannot tamper with the inside code and data without authorization. Confidentiality implies that entities outside the TEE cannot acquire the information inside the TEE without permission. In Intel Software Protection Extensions (Intel SGX), these two attributes are achieved by hardware-level memory encryption, which isolates application-specific code and data in memory. TEE execution results can be verified in the form of remote attestation.

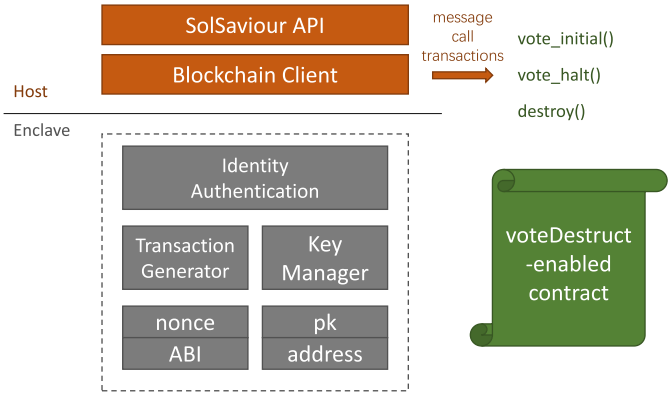


Fig. 1. The architecture of SolSaviour. Contract stakeholders can invoke the TEE cluster via SolSaviour API to generate message call transactions to instruct the voteDestruct-enabled contract.

Considering the availability and confidentiality problem of TEE, we propose distributed TEE cluster. Multiple TEE nodes work together in a distributed environment can avoid the service termination problem. We also propose a distributed signature scheme to avoid storing private keys in a single TEE node.

3 SolSaviour

3.1 What is SolSaviour

The architecture of SolSaviour is depicted in Fig. 1. SolSaviour consists of two core parts: a voteDestruct mechanism and a distributed TEE cluster. The voteDestruct mechanism is embedded in smart contracts. It allows smart contracts to be destroyed in the voting manner. TEE cluster can deploy a patched contract onto the blockchain and migrate all assets and stake distribution. Once a patched contract is deployed, stakeholders can continue to execute the contract without the vulnerability.

3.2 Workflow

First of all, stakeholders should prepare a TEE cluster for protecting smart contracts and DeFi. Contract stakeholders collect a cluster of SGX-capable computers and launch enclaves into them. Then, these TEE nodes corporate to establish a distributed TEE cluster following the bootstrapping process. After constructing the TEE cluster, users can invoke it to deploy a voteDestruct-enabled smart contract.

During the contract execution, an unknown bug may be disclosed. Then, stakeholders can check whether this exposed bug is a false positive. If not, they can invoke SolSaviour API to protect the deployed contract as shown in Fig. 2. The detailed workflow is summarized below:

Phase 1: Destroying the Defective Contract.

① Once identified a bug, stakeholders invoke the TEE cluster to lock the defective contract to prevent further attacks.

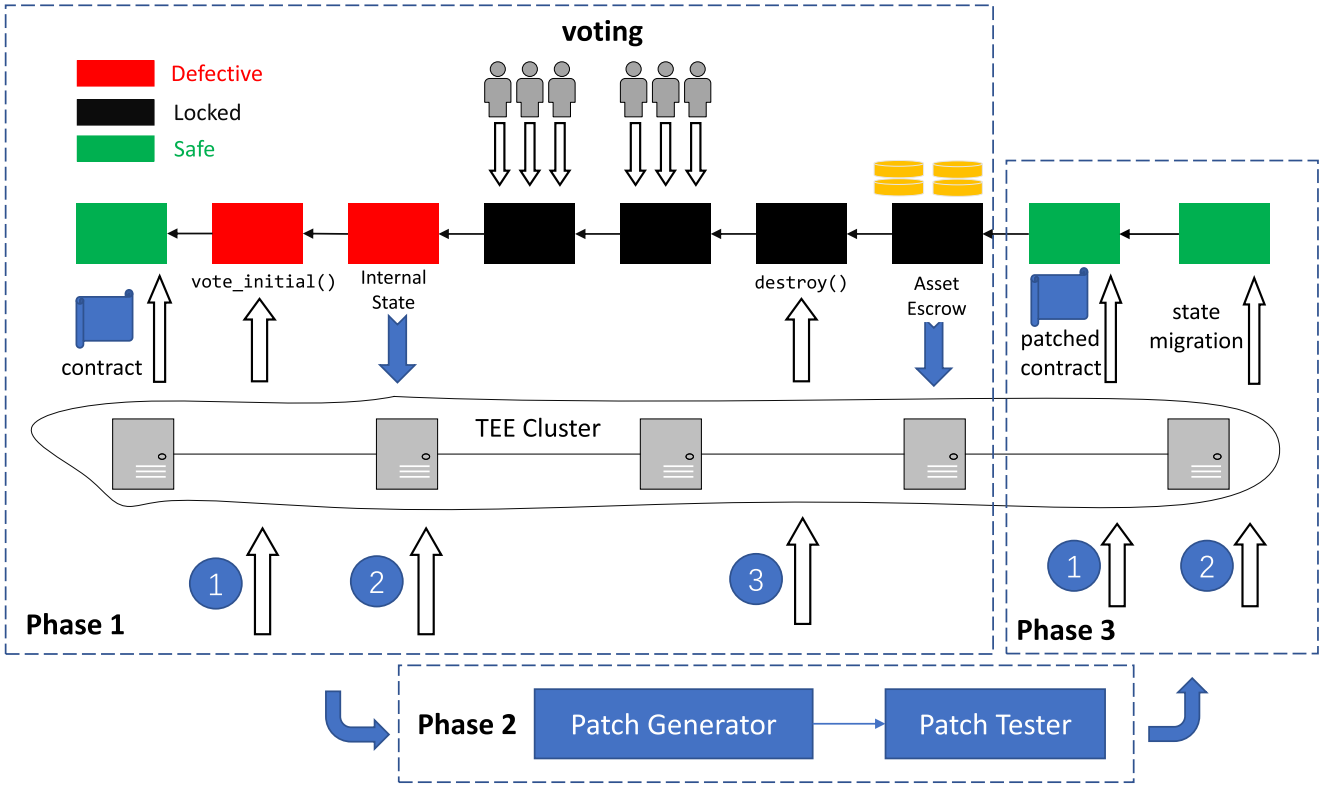


Fig. 2. The workflow of SolSaviour. In Phase 1, contract stakeholders can destroy a defective contract via democratic voting. In Phase 2, a patched contract is generated and tested. In Phase 3, patched contract is deployed to replace the defective one.

② Stakeholders invoke the TEE cluster to extract the internal state of the defective smart contract for future state migration. Specifically, it stores the values of state variables and the stake distribution at the time that the contract is locked.

③ Stakeholders can vote to destroy the defective smart contract via a cumulative voting way. They simply invoke the `vote()` function provided by the voteDestruct mechanism. After voting, stakeholders can invoke the TEE cluster to destroy the defective contract.

Phase 2: Preparing a Patched Contract Offline.

In this phase, stakeholders generate and test patches for the defective contract. Stakeholders can prepare a patch for the located vulnerability and integrate it with the original contract. After that, stakeholders leverage the patch tester to validate whether the vulnerability is resolved and whether the functionalities of the patched contract remain the same. Then, contract stakeholders can upload a patched contract into the TEE cluster.

Phase 3: Redeploying the Patched Contract.

① Stakeholders invoke the TEE cluster to deploy the patched contract prepared in Phase 2. The TEE cluster generates a contract creation transaction to deploy the patched contract.

② Stakeholders migrate the previously-extracted state as well as assets into the patched contract via TEE cluster. After destroying the defective smart contract, all inside assets are temporarily held by the TEE cluster. The temporarily-held assets are transferred into the deployed patched contract according to the stake distribution and previously-extracted values are written into the state variables of the newly-deployed patched contract.

3.3 Building Blocks

3.3.1 *Exception Detector*

For the detection of potential attacks in SolSaviour-protected smart contracts and DeFi, we propose a policy-based exception detector. Two exception detection strategies have been designed for SolSaviour. The first one is to alert when a contract withdrawal exceeds a certain percentage of the contract's total assets within a certain period of time. The exact parameters can be fine-tuned according to the security requirements. The second one is to alert when a contract withdrawal is made to an address that is not a contract stakeholder. Warning messages are published via event messages on the blockchain. These two strategies ensure that stakeholders are able to quickly perceive hazards and react when contracts are at risk of potential asset loss.

The implementation of exception detector is on the contract level. The warning messages exist on the blockchain network and are not actively communicated to the contract stakeholders, who therefore need to implement an active crawler to automatically and continuously monitor event messages. This way, when an exception occurs in a monitored contract, stakeholders can be aware of it and take actions as soon as possible.

3.3.2 *voteDestruct Mechanism*

Currently, a typical method to destroy a contract is to have a privileged destruction function that only specific users can invoke. This approach is not feasible to multi-user smart contract scenarios. For example, assets in a DeFi contract typically belong to different users. Stakeholders of a DeFi have the right to decide whether to destroy the smart

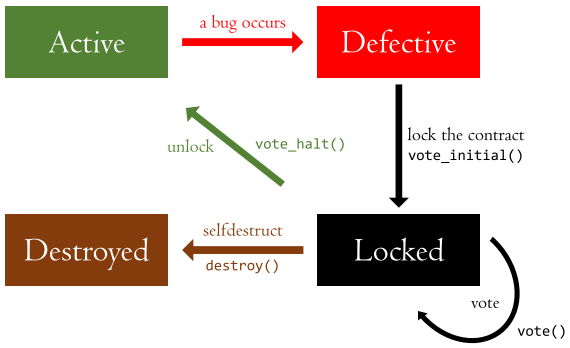


Fig. 3. The state diagram of a voteDestruct-enabled smart contract.

contract. In this case, we propose the voteDestruct mechanism to enable decentralized control of smart contracts and DeFi.

In voteDestruct-enabled contracts, stakeholders can vote on whether to destroy the smart contract and withdraw all internal funds. The voteDestruct mechanism is based on the contract stake distribution, which is recorded by state variables. The more stake a stakeholder controls, the greater its vote weight. Specifically, The voteDestruct is processed in three steps. The contract first forms the stake distribution during its execution. Then, once a vulnerability is exposed, stakeholders can invoke the contract via TEE cluster to vote. After that, the contract stakeholders can invoke the TEE cluster to destroy the defective contract if the support rate exceeds a predefined threshold.

Stake Distribution: Each depositing transaction will be recorded. Specifically, the address of stakeholder and amount of deposited assets are stored in a tuple.

Voting: During the voting process, the smart contract is locked so that no external users can deposit or withdraw assets. This ensures that the stake distribution remains constant throughout the voting. Once completed, the contract stores the percentage of stake on supporting and opposing, respectively. Then, stakeholders can determine whether this contract can be destroyed.

State Transition: The state transition of a voteDestruct-enabled smart contract is depicted in Fig. 3. A smart contract is initially in an active state. Then, an unknown bug is discovered and the contract enters a defective state. Stakeholders can now lock the defective contract using the `vote_initial()` function. For false positive cases, they could unlock the contract via `vote_halt()` function. For true bugs, stakeholders could vote on whether to destroy the defective contract. When the support rate exceeds the threshold, the user can call the `destroy()` function to destroy it.

Destroying: A contract can only be destroyed if the number of votes in favour of destruction exceeds the threshold. Stakeholders can instruct the TEE cluster to invoke the `destroy` function once the voting process completes.

3.3.3 TEE Cluster

Participants can monitor the blockchain for protocol deviations and respond appropriately. In SolSaviour, TEE act as the root of trust. The TEE cluster is independent of the blockchain and can assure faithful implementation. SolSaviour can provide safe destruction, redeployment, and state migration of defective smart contracts by employing the TEE cluster as an independent root of trust. Furthermore, the cluster architecture improves the system's overall failure tolerance. A single point of failure will not influence overall availability.

1: **procedure** BOOTSTRAPPING($\sigma_i, i \in [0, n-1]$)
2: load σ_i
3: broadcast $\{\text{sgx\_quote}_i, K_i\}$
4: verify sgx_quote_j
5: generate $\{K$,addr$\}$
6: **end procedure**
7: **procedure** IDENTITY AUTHENTICATION
8: on receiving a message call tx
9: require(st.map(msg.sender()).st_amount)
10: revert()
11: **end procedure**
12: **procedure** LOCKING DEFECTIVE CONTRACT
13: goto line 8, on receiving $addr$
14: $\text{Sign}_K(\text{tx.payload}(\text{addr}, \text{vote\_initial}))$
15: upload tx, broadcast(++nonce)
16: **end procedure**
17: **procedure** EXTRACTING INTERNAL STATE
18: goto line 8, verify_tx(tx,pub_1,pub_2)
19: d $\leftarrow$ create(tx)
20: mapping stake_dist(addr$\rightarrow$stake_amount)
21: stake_dist $\leftarrow$ addr.st_map_getter()
22: **end procedure**
23: **procedure** DESTROYING DEFECTIVE CONTRACT
24: goto line 8, on receiving $addr$
25: $\text{Sign}_K(\text{tx.payload}(\text{addr}, \text{destroy}))$
26: upload tx, broadcast(++nonce)
27: **end procedure**
28: **procedure** REDEPLOYING PATCHED CONTRACT
29: goto line 8, on receiving C_p
30: $\text{Sign}_K(\text{tx.payload}(C_p))$
31: upload tx, broadcast(++nonce)
32: **end procedure**
33: **procedure** MIGRATING TO PATCHED CONTRACT
34: goto line 8
35: tx.payload(addr_C_p, stake_dist), go to line 15
36: values[]$\leftarrow$stake_dist[]
37: balance[c_id_n]$\leftarrow$values[c_id_n]
38: **end procedure**

Fig. 4. The working logic of TEE cluster. The TEE cluster receives instructions from contract stakeholders and generates transactions to call smart contract functions.

The working logic of distributed TEE cluster is depicted in Fig. 4. Each enclave is denoted as σ_i. We use c to represent a potentially defective smart contract and C_p to denote a patched contract. To demonstrate the difference between intra-TEE cluster communication and TEE cluster-blockchain communication, we use "broadcast" to indicate broadcasting messages inside TEE cluster and "upload" to represent uploading transactions onto the blockchain.

Bootstrapping: Let $\mathbb{G}$ be an Elliptic curve group of order q with generator (base point) G. Each TEE node P_i has the information of $(\mathbb{G}, \text{G}, q)$. A TEE node P_i first chooses a random x_i from $\mathbb{Z}_q^*$ and computing $Q_i = x_i \cdot G$. Each TEE node stores $(\mathbb{G}, \text{G}, q, x)$. Then, each TEE node P_i broadcasts its

TABLE 1
SolSaviour API

SolSaviour API	Inputs	Outputs	API Description
Bootstrap			
new_address	N/A	$\top\|\bot$	Generate a new key pair as well as the blockchain address for the TEE cluster
Deployment			
new_contract	$\mathcal{C}_b$	tx_id	Deploys a compiled smart contract and returns a transaction id
patched_contract	$\mathcal{C}_b$	tx_id	Receives the bytecode-version smart contract, deploys it and returns a transaction id
Message Call			
lock	$\mathcal{C}_{addr}$	tx_id	Receives the address of contract and returns the id of transaction that invokes `vote_initial()`
unlock	$\mathcal{C}_{addr}$	tx_id	Receives the address of contract and returns the id of transaction that invokes `vote_halt()`
destroy	$\mathcal{C}_{addr}$	tx_id	Receives the address of contract and returns the id of transaction that invokes `destroy()`

generated x_i to P_j for every $j \in [n]\backslash\{j\}$. After receiving x_j ($j \in [n]\backslash\{j\}$) from other parties, P_i locally computes $x = \sum_{l=1}^{n} x_l$ and $Q = x \cdot G$. Each party P_i locally stores Q as the ECDSA public key.

Signing: A TEE node P_i locally generates k_i and ρ_i randomly. Then, P_i broadcasts k_i and ρ_i to P_j for every $j \in [n]\backslash\{j\}$. After receiving k_j and ρ_j ($j \in [n]\backslash\{j\}$). Each party can locally compute $k = \sum_{l=1}^{n} k_l$ and $\rho = \sum_{l=1}^{n} \rho_l$. Then, TEE node P_i computes $\tau = k \cdot \rho$ and $R = k \cdot G$. The TEE node P_i can now compute $R = (r_x, r_y)$ and $r = r_x \bmod q$. For a raw transaction m, node P_i generates $\beta = \rho \cdot (m + x \cdot r) \bmod q$. Then, P_i computes $s' = \tau^{-1} \cdot \beta \bmod q$ and $s = min(s, q - s)$. Finally, TEE node P_i outputs (r, s) as the ECDSA signature of the transaction.

Identity Authentication: Identity authentication is a crucial component in TEE cluster, which happens before each external call. Identity authentication restricts that only eligible stakeholders can invoke the TEE cluster. SolSaviour needs to verify that the identity of the object initiating the call is reliable, which reduces the likelihood of an attacker invoking the TEE cluster for dangerous operations, such as DoS attacks. TEE Cluster uses membership proof for identity authentication. It first extracts the address of the stakeholder that initiated the call, which in turn polls the membership proof integrated with the voteDestruct mechanism. Only if the membership proof is true, the TEE cluster determines that the invoking stakeholder is valid and proceeds.

Destroying: TEE cluster allows stakeholders to lock the defective contracts as shown in *line 12-16*. Once an error is detected, the stakeholder can call the TEE cluster to lock the contract. The TEE cluster generates a message call transaction to invoke the `vote_initial()` function and signs it. The signed transaction is uploaded to the blockchain and an incremented nonce is broadcasted inside the TEE cluster.

Before destroying the defective smart contract, the TEE cluster extracts the internal states from it as shown in *line 17-22*. Internal states include state variable values and the stake distribution of stakeholders at the time of locking. The TEE cluster can obtain internal states via `getter` functions and save them locally.

After completing the voting process, stakeholders can instruct the TEE cluster to invoke the `destroy()` function as shown in *line 23-27*. The TEE cluster generates a signed message call transaction destined to the address of the defective smart contract. Provided the amount of stake in favour of destruction exceeds a specified threshold, the contract is allowed to be destroyed.

Redeploying: Contract stakeholders first generate the patch offline. Then, they could invoke the TEE cluster to redeploy a patched smart contract as shown in *line 28-32*. After receiving the patched contract, the TEE cluster generates a contract creation transaction with compiled contract as payload, uploads the signed transaction onto the blockchain, and broadcasts an incremental nonce.

Based on the previously-extracted internal states, stakeholders can call the TEE cluster to migrate them to the patched contract as shown in *line 33-38*. For stake distribution and assets, TEE cluster generates signed message call transactions to invoke the patched contract. In this process, the TEE cluster ensures that states are consistent. The TEE cluster also guarantees the atomicity of execution, i.e. any intermediate state resulting from the call, and the need to provide an effective rollback mechanism in the event of a failed call, allowing the system to revert to the state before the call, eliminating the impact of intermediate state resulting from the call.

3.4 SolSaviour API

We summarize the API of SolSaviour in Table. 1. Contract stakeholders can invoke the functionalities provided by SolSaviour via these interfaces.

SolSaviour API includes 3 types: bootstrap, deployment, and message call. In the bootstrap type, stakeholders can invoke the `new_address` function to generate a new key pair for the agreed account address utilized in the TEE cluster. In the deployment type, there are two functions: `new_contract` and `patched_contract`. The former one is used to deploy a compiled new smart contract, and the later one is used to deploy a patched contract. Both functions return the id of the contract generation transaction that utilized to deploy the contract.

For message call type, stakeholders can invoke them to generate message call transactions to instruct the voteDestruct mechanism. Stakeholders should provide the address of the defective contract as inputs so that TEE cluster know which contract should call. SolSaviour provides 3 message

```
contract voteDestruct_sample {
    struct st_holder
    { uint key_index; uint st_amount; bool voted;}
    mapping(address => st_holder) public st_map;
    address public TEE_addr;
    uint public contract_stake; uint public support_stake;
    enum State {Active, Locked} State public state;

    modifier inState(State _state)
    { require(state == _state); _;}

    constructor() { TEE_addr = msg.sender;}

    function any_payable_function() inState(State.Active)
    public payable {
        st_map[msg.sender].st_amount += msg.value;
        contract_stake += msg.value;}

    function vote_initial() inState(State.Active) public {
        require (msg.sender == TEE_addr);
        state = State.Locked;}

    function vote_halt() inState(State.Locked) public {
        require (msg.sender == TEE_addr);
        state = State.Active;}

    function vote(bool choice) inState(State.Locked) public {
        require(!st_map[msg.sender].voted);
        st_map[msg.sender].voted = true;
        if (choice)
        { support_stake += st_map[msg.sender].st_amount;}}

    function destroy() public {
        require (msg.sender == TEE_addr);
        require (support_stake > (contract_stake * 2 / 3));
        selfdestruct(payable(TEE_addr));}}
```

Fig. 5. A voteDestruct-enabled contract sample.

calls: `lock`, `unlock`, and `destroy`. The `lock` call can invoke the `vote_initial` function in voteDestruct mechanism to lock a defective smart contract, `unlock` can invoke `vote_halt` function to unlock a falsely locked smart contract, and `destroy` function can invoke the `destroy` function in voteDestruct mechanism to destroy a defective smart contract and transfer all assets to TEE cluster. After receiving the invocation from stakeholders, the TEE cluster passes parameters to the transaction generator to generate a signed message call transaction, which is later uploaded onto the blockchain.

4 Implementation

4.1 Destroying

Currently, we can use `selfdestruct` to destroy deployed smart contracts and refund all inside assets. In SolSaviour, we introduce the voteDestruct mechanism to better leverage the `selfdestruct` opcode. A voteDestruct-enabled contract can only be destroyed if and only if most of the stakeholders vote to destroy it. Fig. 5 shows a sample of the voteDestruct mechanism. We emphasize that its implementation does not require new EVM opcodes. It is constructed based on pure Solidity language. Moreover, the voteDestruct mechanism can be implemented in different versions of Solidity with minor modifications.

In the life cycle of a smart contract, contract participants may deposit ethers before a bug is exposed. The voteDestruct mechanism records these participants as stakeholders `st_holder` and the amount of their deposited ethers as `st_amount`. Once the exception detector issues a warning message or a stakeholder notices that the contract has a potential vulnerability, all contract stakeholders can vote to lock the contract. If the support rate exceeds 1/3 (i.e., lock threshold), the contract enters a locked state and no external calls can be executed, except for calls that unlock or destroy the contract. During the locking phase, the contract stakeholders can analyze the exposed vulnerability and develop corresponding patches.

If most stakeholders think that this vulnerability is a false positive case, they can vote to unlock the smart contract. The voting threshold for unlocking a locked contract is the same as the threshold for locking it. If stakeholders think the vulnerability may lead to serious consequences, they could vote to destroy the defective contract. When the support rate exceeds 2/3 (i.e., destroy threshold), the vote is passed and the contract can be destroyed. All internal assets are transferred to the distributed TEE cluster for temporary escrow.

4.2 Patching

In SolSaviour, patches for defective smart contracts are provided by the contract stakeholders. This is because the main purpose of SolSaviour is to provide a framework for securing defective deployed smart contracts and DeFi, rather than providing a system that can automatically generate patches. Smart contract patches can be generated manually using existing tools such as sGuard [12] and SCRepair [13]. Once a patched contract is prepared, the patched contract should be tested thoroughly before deployment by replaying previous related transactions. This can test whether the patched contract functions well and has fixed all related bugs. We propose a patch tester to re-execute non-malicious history transactions on the patched contract and verifies whether the execution results of the old contract and the patched contract are consistent. Any execution discrepancies are scrutinised to determine whether the patch has caused the patched smart contract to function inconsistently with the defective contract. Detailed implementation is provided in Fig. 6.

4.3 Redeploying

4.3.1 *Redeploy a Patched Contract*

In this step, stakeholders can redeploy a patched contract and migrate the internal state from the defective contract to the patched one.

During redeployment, the TEE cluster takes charge of injecting the initial state of the patched contract. The TEE cluster injects a list of stakeholder addresses and the amount of their stakes to the patched contract, which indicates the amount of assets they deposited before contract destruction. Then, the TEE cluster generates a contract creation transaction for the patched contract and broadcast it onto the blockchain. For contract stakeholders, the internal state of the redeployed contract remains the same as the previous defective contract, but SolSaviour has already fixed the vulnerabilities.

```
static bytes C_patched[];
void patchGeneration(bytes C_defective, bytes& C_patched){
    C_patched[0] << sGuard(C_defective);
    C_patched[1] << SCRepair(C_defective);
    C_patched[3] << C_patched;}
bytes patchTest(bytes& C_patched[]){
    string Tx[];
    while(web3.eth.addr){
        Tx += web3.eth.addr.transaction;}
    uint length = Tx.length; uint index;
    uint success[] = 0;
    for(int i =0; i<3; i++){
        for(index=0, index<length; index++){
            if(C_patched[i].exe(Tx[index])){
                success[i]++;}}}
    if(success[0]>success[1]){
        if(success[0]>success[2]) return C_patched[0];
        else return(C_patched[2]);}
    else{
        if(success[1]>success[2]) return(C_patched[1]);
        else return(C_patched[2]);}}
```

Fig. 6. The implementation of patching a defective contract.

4.3.2 State Migration

For migrating states to the newly-patched smart contract, the TEE cluster first extracts required variables from the blockchain. Then, the TEE cluster modifies the patched smart contracts provided by the contract stakeholders. The purpose of this modification is to migrate the internal state from the old, defective contract to the new, patched smart contract. TEE cluster ensures that variable values in the patched contract are the same with before by initializing them. TEE cluster directly transfers all the escrow assets to the newly-deployed contract. Since the stake distribution has been injected by TEE cluster, the ownership of these assets is certain and consistent, as well as their corresponding voting rights. Detailed implementation is provided in Fig. 7.

```
void stateVariableGetter(string addr){
    struct stateVariable{
        string name;
        bytes value;};
    stateVariable states[]; uint index;
    uint length = addr.ABI.length;
    for (index=0, index<length; index++){
        states[index].name = addr.ABI[index].name;
        states[index].value = addr.ABI[index].value;}}
void stakeDistribution(string addr){
    struct stakeDist{
        string addr;
        uint amount;};
    stakeDist stakes[]; uint index;
    uint length = addr.st_map.length;
    for (index=0; index<length; index++){
        stakes[index].addr = addr.st_map[index].key;
        stakes[index].amount = addr.st_map[index].st_amount;}}
void stateMigration(string addrFrom[], string addrTo[],
↪  uint256 values[]){
    require(addrFrom.length == values.length);
    uint256 length = values.length;
    uint i;
    for (i=0; i<length; i++){
        balances[addrTo[i]] = values[i];
        emit Transfer(0x0, addrTo[i], values[i]);}}
```

Fig. 7. The implementation of recovering a patched contract.

5 Security Analysis

5.1 Threat Model

Our threat model considers the security of SolSaviour from following three perspectives.

Host. We assume hosts are potentially malicious. They may delay messages between the blockchain and its hosted enclaves for a period of time. We also assume an adversary $\mathcal{A}$ that can corrupt up to t of n hosts in the TEE cluster.

Enclave. We assume the attested execution result of enclave is trusted, which means the malicious host cannot tamper with the code and data inside enclaves. However, we assume the enclave may suffer from some confidentiality problem.

Stakeholder. We assume the stakeholders are rational and potentially malicious. If there are benefits, they may deviate from the protocol execution and try to steal funds that belong to others. Stakeholders are greedy, if a smart contract is under attack, they will steal assets that belong to others.

5.2 Threat Analysis

In this section, we analyze the security of SolSaviour in three aspects: balance security, correctness, and fairness.

5.2.1 Balance Security

The balance security of SolSaviour is twofold. First, honest stakeholders won't lose assets except necessary transaction fees as long as stakeholders behave honestly. Second, the stake distribution remains the same in the old defective contract and new patched contract.

First, we consider the case that a smart contract is in the locked state. If contract stakeholders cannot reach an agreement on a patched contract, the defective smart contract remains locked and reject all external calls except ones from the TEE cluster. In this case, no assets can be stolen by attackers. If stakeholders agree on a patched contract, the TEE cluster proceeds to conduct the state migration.

We then prove that the assets security is preserved during the state migration process. Before migrating assets into new patched contract, assets are held in a blockchain account controlled by the TEE cluster. As assumed before, attackers cannot control more than t of n TEE nodes. In this case, attackers cannot withdraw assets from the new patched contract with legitimate signature so that cannot steal assets. Moreover, there is no way for attackers to tamper with the code inside TEE cluster since the execution logic of the enclave is fixed once it has been encapsulated. As a result, attackers are unable to alter the TEE cluster's state migration logic and transfer assets to their accounts.

Finally, we discuss the consistency of stake distribution between the defective contract and patched contract. The history of defective smart contracts is publicly available on the blockchain. The stake distribution a defective contract can be determined by looking up its transaction history. Additionally, the TEE cluster can crawl the contract history to ascertain the internal variable values for the defective contract. In this way, TEE cluster can ensure a safe migration of contract internal state from the old vulnerable smart contract to the new patched one. After state migration, due

to the tamper-evident nature of on-chain data, assets cannot be stolen once they are transferred into a new contract.

5.2.2 Correctness

Intuitively, correctness states that all honest parties can output correct results, namely successfully patching and transmitting the assets from defective smart contracts to the patched contracts. For the proof of correctness, we divide it into two parts: bootstrapping of the TEE cluster and destruction of a defective contract.

First, we prove that the bootstrapping process of TEE cluster satisfies correctness with threshold t, which indicates that TEE nodes can reach an agreement on a key when there are at most t corrupted parties among n TEE nodes. During the bootstrapping process, once a node U_i receives other node's K_j, it can verify it. If the check fails for index j, U_i can broadcast a complaint against node U_j. If more than t nodes complain about a node U_j, that node is recognized as disqualified. Each node stores a node set `QUAL` for all qualified nodes. In this case, they generate the key based on nodes inside `QUAL`. As all honest nodes construct identical `QUAL`, they can generate the same key and derive the same blockchain address. Thus, we can show that TEE cluster can correctly complete the bootstrapping process for a given attacker threshold.

Second, we prove that the destruction of defective smart contracts satisfies correctness. We consider rational stakeholders, that means they behave maliciously if they think their behaviour is more profitable. In this case, we analyze the choice of rational stakeholders in two scenarios. When a smart contract is exposed to a vulnerability or under attack, stakeholders have two choices, attacking or locking. As stakeholders have some assets inside defective smart contracts, the best way is to lock contract as soon as possible. In the scenario that a defective contract is successfully locked, the best way is also to prepare a patched contract for deployment as soon as possible. This is because stakeholders have assets inside locked smart contracts, which do no output interest. Only when smart contracts are active, stakeholders can use their assets for arbitrage or other financial services.

5.2.3 Fairness

Fairness indicates that our proposed voteDestruct mechanism is fair. All stakeholders cannot have more power than the amount of their controlled stake. For the proof of fairness, even if an attacker holds a certain amount of stake in the defective contract and has the right to vote, it cannot maliciously manipulate the voting result or prevent the destruction of the defective smart contract.

We first analyze the case where there are some malicious stakeholders. As malicious stakeholders are profit oriented, what they want is to steal the assets from honest stakeholders. However, honest stakeholders can always safely exit a smart contract as long as their cumulative stake amount exceeds the specified destroy threshold. In SolSaviour, a smart contract has three states: active (but potentially defective), locked, and destroyed.

In locked state, a contract is protected by blockchain miners that reject all function calls except those initiated from the TEE cluster. In this case, malicious stakeholders cannot steal assets. In destroyed state, assets in a contract are held by the TEE cluster, so that malicious stakeholders cannot profit either. The only chance for malicious stakeholders to profit is during the active (but potentially defective) state. In SolSaviour, the threshold of required stake amount to lock a contract is $1/3$. Only when the amount of stake held by malicious stakeholders exceeds $2/3$, they can prevent the contract from entering the locked state.

During the active (but potentially defective) state, when a hidden vulnerability is exposed, malicious stakeholders can try to prevent the contract from entering the locked state and exploit the vulnerability. However, due to the unknown feature of the vulnerability, it may cause the contract to a deadlock state, which is also unprofitable for malicious stakeholders. Therefore, it is also unprofitable for malicious stakeholders to prevent locking defective contract.

6 Discussion

6.1 Limitations and Security Risks

In this section, we discuss the limitations and security risks of SolSaviour. One of the main limitations of SolSaviour is that it can only protect contracts that have integrated the voteDestruct mechanism. Due to the tamper-proof feature of the blockchain, SolSaviour cannot provide the defence mechanism for active smart contracts that have already been deployed. As TEE is a technology still under development, there may be unknown vulnerabilities. TEE is therefore at risk and newly discovered TEE vulnerabilities could compromise the security of the entire system and the in-contract assets.

6.2 Recovering Patched Contract without TEE Escrow

6.2.1 Implementation

Considering the risks to the security of assets temporarily held in the TEE cluster, we introduce a new way to recover a defective smart contract to a patched contract directly.

Setup Phase. In this way, SolSaviour first locks a potentially defective contract to prevent it from further attacks. Then, stakeholders develop a patched smart contract and provide it to the TEE cluster. The TEE cluster first deploys the patched contract onto the blockchain. After deploying the patched smart contract, the voteDestruct mechanism could set the parameter of the `destroy` function to the address of the patched contract. In this way, the assets are directly transferred from the defective smart contract to the patched contract without interaction of the TEE cluster. In this way, even the serious problem such as key leakage of the account controlled by the TEE cluster, the assets are still under protection.

Recovering Phase. Unlike reverting to a TEE cluster, this approach eschews the use of TEE and therefore its state transfer behaviour cannot be implemented through TEE. We then need to implement complex state migration logic in the smart contract. Specifically, we need to store the complete state variables of the smart contract and the take distribution of the assets stored in the take holders, and we need to have transfer logic that acts on top of these state variables. We consider using a bridging contract to accomplish this

step, i.e. deploying a smart contract dedicated to state migration, in which the state variables and take distribution associated with the defective smart contract are stored, and in the constructor of the patched contract, allowing it to read and write the state variables in the contract. In this way, we can achieve state transfer for smart contracts without the need for TEE intervention. In contrast, this approach requires significant gas consumption to maintain the various operations and data stores. The cost required to implement contract recovery on the public chain is significantly higher due to the addition of bridging contracts and the corresponding storage of state variables. However, the advantages of this approach are clear: by avoiding temporary asset hosting of TEE clusters, this approach significantly reduces the attack surface of the system and eliminates the risk of security issues that may arise from the TEE itself.

6.2.2 Comparison

In summary, there are advantages and disadvantages to both recovering to the TEE cluster and recovering to the patched contract, the former being more gas efficient but less secure than the latter, and the latter being secure, but the cost of protecting the smart contract is greatly increased by the high gas consumption it entails. The latter is secure, but the high gas consumption associated with it can add significantly to the cost of protecting smart contracts. Therefore, when faced with a specific smart contract, the contract developer needs to make a trade-off between security and cost and choose the best method to protect the smart contract.

7 Experiment

In our prototype of SolSaviour, the voteDestruct mechanism is implemented in Solidity and the TEE cluster is implemented based on Intel SGX with around 2000 LOC. Four nodes are set up in the TEE cluster. The experiments are conducted in two aspects: effectiveness and performance.

7.1 Dataset Collection

To accurately evaluate the effectiveness and performance of SolSaviour, we collected ordinary and DeFi contracts that were exposed to vulnerabilities. Some contracts have experienced real attacks that have caused asset losses, some have not been exploited but their defects have been confirmed. The reason we chose these contracts is that we want to prove the effectiveness of SolSaviour with the deduction of loss for these contracts when they are under the protection of SolSaviour.

Our collected contracts are the DAO [1], PoWH Coin [20], 1st [18] and 2nd [19] Parity Multisig Wallet, King of Ether [14], Bancor Exchange [21], GovernMental [15], and Rubixi [16]. We also collected DeFi contracts that were exposed to some severe bugs such as SushiSwap [22], ENS Name Wrapper [17], Fei [2], and Uniswap [23]. We list these contracts in Table. 2, accompanying with contract vulnerability type and caused damage as well as losses in monetary form (if so).

For our collected contracts, we also prepare corresponding voteDestruct-enabled contracts and patched contracts. The voteDestruct mechanism is injected on the source code level. As collected contracts are written in different versions of Solidity, we make minor modifications to make our voteDestruct mechanism compatible in all versions of Solidity. For patched contract, our collected contracts are also patched manually by modifying the code. We also ensure that the compiled voteDestruct-enabled contracts and patched contracts following the same version of Solidity as original contracts.

SolSaviour is then applied to these generated comparative smart contracts so that we could verify the effectiveness and performance by validating the results. We can verify whether the voteDestruct mechanism is effective and that the patched smart contract redeployed by SolSaviour fixed vulnerability.

7.2 Effectiveness

The effectiveness of SolSaviour is evaluated from two perspectives: qualitative and quantitative.

For qualitative part, we check whether we can leverage SolSaviour to safely exit from all collected defective contracts, refund locked assets back to stakeholders, and redeploy a patched contract. To test whether SolSaviour can recover a buggy smart contract, we generate a large and representative evaluation dataset by collecting transactions sent to the collected contracts from the Ethereum. Replaying those transactions and observing outcomes can check the functionality and defence of patched contracts. Specifically, we test whether SolSaviour can successfully destroy a defective smart contract with voteDestruct mechanism and redeploy a patched one with TEE cluster. In addition, we test whether the TEE cluster can successfully migrate the previous state to the new contract to ensure the state consistency.

For the quantitative part, we set up two contract instances for each collected defective contract: an original contract instance and a SolSaviour-protected instance. We compare the loss between the original one and SolSaviour-protected one. We think the effectiveness of SolSaviour is reflected in the loss deduction when a vulnerable smart contract is protected by SolSaviour. That's the reason we compare the loss of smart contracts in difference settings. For the original one, we also record the loss when taking traditional defence measures and doing nothing. We test to what extent can SolSaviour save loss when facing different vulnerabilities.

7.2.1 Qualitative

We evaluate the effectiveness of SolSaviour in three aspects: successful state migration, identical functionalities, and successful defence. For each contract, we use Ganache to simulate 10 accounts, who play the role of contract stakeholders and each has deposited 100 ethers. Then, a random stakeholder initializes the `vote_initial` and provides a patched contract to the TEE cluster. In our experiments, we omit the security assumption of potential malicious stakeholders and assume all of them will vote to destroy the defective contract. Once the voting completes, the TEE cluster destroys the defective contract, redeploys a patched one, and conducts state migration.

By checking the patched contracts deployed by the TEE cluster, we can evaluate whether state migration successes. A successful state migration means the internal states of

TABLE 2
The List of Contracts That Have Been Attacked or Exposed to Serious Bugs

Contract	Address	Vulnerability Type	Caused Damage
King of Ether [14]	0x2464d1d97f8D0180CFaD67BdB19bc30ccA69DdA0	Unchecked Return Values	Ownership Loss
GovernMental [15]	0xF45717552f12Ef7cb65e95476F217Ea008167Ae3	Timestamp Dependence	DoS
Rubixi [16]	0xe82719202e5965Cf5D9B6673B7503a3b92DE20be	Bad Constructor	Ownership Loss
ENS Name Wrapper [17]	0x00000000000C2E074eC69A0dFb2997BA6C7d2e1e	Access Control	Domain Ownership Loss
1st Parity Multisig [18]	0x863DF6BFa4469f3ead0bE8f9F2AAE51c91A907b4	Delegatecall	153,037 ETH Loss ($31M)
2nd Parity Multisig [19]	0x863DF6BFa4469f3ead0bE8f9F2AAE51c91A907b4	Denial of Service	513,774.16 ETH Locked ($300M)
The DAO [1]	0xBB9bc244D798123fDe783fCc1C72d3Bb8C189413	Reentrancy	3.6M ETH Loss ($150M)
PoWH Coin [20]	0xA7CA36F7273D4d38fc2aEC5A454C497F86728a7A	Integer Underflow	866 ETH Loss ($800k)
Bancor Exchange [21]	0x1F573D6Fb3F13d689FF844B4cE37794d79a7FF1C	Front Running	Economic Earns ($150)
SushiSwap [22]	0x6B3595068778DD592e39A122f4f5a5cF09C90fE2	Supply Chain Attack	864.8 ETH
Fei [2]	0x956F47F50A910163D8BF957Cf5846D573E7f87CA	Price Manipulation	60k ETH at risk
Uniswap [23]	0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984	ERC777 Reentrancy Exploit	$320M

buggy contract and patched contract are identical. Not only the stake distribution, but also the ownership. We check this by letting each stakeholder withdraw their previously-deposited ethers. We found that stakeholders can withdraw their assets successfully from all contracts. Then, we check the functionality and defence of patched contracts by replaying collected transactions. We compare the execution results of patched contract with buggy contract's history state transition.

7.2.2 Quantitative

We quantitatively evaluate the effectiveness of SolSaviour by attacking and recovering defective contracts with SolSaviour simultaneously. Then, we evaluate to what extent can SolSaviour reduce loss. Since different contracts are tested in different scenarios, where the amount of loss are different, we use a two-tuple (loss, percentage) to show results.

For the DAO contract, we simulate a scenario, where the defective deployed contract contains 100 ethers. Then, we start to attack and recover it at the same time. Attackers can arbitrarily withdraw ethers until honest stakeholders lock the contract. Then, we tried to refund all locked ethers to stakeholders and calculate the loss. Similar steps are conducted to evaluate the loss when using SolSaviour. For the PoWH coin contract, since the real attack transactions are limited, we simply replay these attack transactions and check the execution results. We also test the loss when doing nothing and taking traditional defence. Each contract are tested 5 times and the average loss is recorded. The results are listed in Table. 3.

7.3 Performance

7.3.1 Contract Size Increase

In this section, we evaluate the additional code required to use SolSaviour. On Ethereum, deploying smart contracts consumes gases, which are proportionally to the size of the deployed contract. In SolSaviour, as the voteDestruct mechanism is implemented inside contracts, extra codes are introduced. The manually-generated patch may also increases the code of contracts. Moreover, the method on recovering patched contract without TEE also introduces extra codes in the contract. Results are summarized in Fig. 8. The code size of the original version of collected defective contracts are listed as the basis. Each subplot has six bars. The left three bars represent the size of contract when recovering with TEE, while the right three bars represent the size of contract when recovering without TEE. In each subplot with three bars, from left to right are the size of the original contract, the size voteDestruct-enabled contract and the size of the patched contract.

However, since patches are generated manually, it is impossible to determine the performance of the system in terms of the number of lines of code added by the patch. The only additional code introduced by the system is the

TABLE 3
Comparison of Losses/Damages in an Attack Against Our Collected Contracts

Contract	Actual	Traditional	SolSaviour
King of Ether [14]	Lose Onwership	No Mitigation	Fix
GovernMental [15]	Lose Ownership	No Mitigation	Fix
Rubixi [16]	Lose Ownership	No Mitigation	Fix
ENS Name Wrapper [17]	Lose Ownership	No Mitigation	Fix
1st Parity Multisig [18]	100	100	0
2nd Parity Multisig [19]	100	100	0
The DAO [1]	100	48.6	6.5
PoWH Coin [20]	100	100	0
Bancor Exchange [21]	100	69.6	0
SushiSwap [22]	100	100	3.3
Fei [2]	100	61.2	2.3
Uniswap [23]	100	54.3	4.6

"Actual" indicates that no action is taken; "Traditional" indicates that traditional defence methods are used; "SolSaviour" indicates that the contract is maintained by SolSaviour.

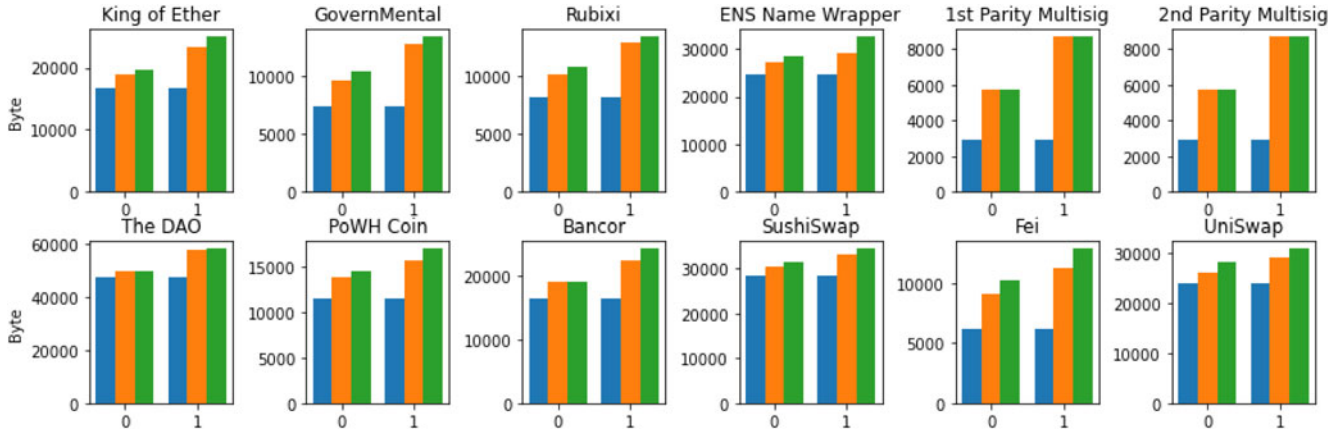


Fig. 8. Contract size increased in SolSaviour. The left three bars represent using TEE cluster for asset escrow while the right three bars using patched contract. Blue bar indicates original contract, orange bar indicates voteDestruct-enabled contract, and green bar indicates patched contract.

voteDestruct framework that makes defective contracts have the ability to iterative upgrades under SolSaviour. In this case, we compare the size of compiled contract. We found that voteDestruct mechanism introduces limited size to the original contract. These code size increases are worth compared to the security enhancement that SolSaviour brings. For Parity Miltisig contract, we note that injecting voteDestruct mechanism naturally resolves the vulnerability so that the patched contract and voteDestruct-enabled contract have the same size.

7.3.2 Gas Consumption

In this section, we evaluate the additional gas consumption incurred by SolSaviour, which mainly arises from two aspects: the voteDestruct mechanism and the redeployment of the patched contract. We also evaluate the gas consumption on redeploying a patched contract without TEE asset escrow. Results are summarized in Fig. 9. The gas consumption to deploy the original version of collected defective contracts are evaluated as the basis. Each subplot has six bars. The left three bars represent the gas consumption of original contract, voteDestruct-enabled contract, and patched contract respectively when letting TEE conduct asset escrow. By contrast, the right three bars represent the gas consumption when recovering without TEE asset escrow.

For voteDestruct mechanism, the evaluation is conducted by deploying our prepared voteDestruct-enabled contracts. From the results, we can see the voteDestruct mechanism introduced minimal gas overhead. The gas cost are mainly introduced by additional storage of state variables and corresponding logic. Storing data on Ethereum is expensive, which leads to a lot of gases to be consumed. For the redeployment of patched contract, the extra gas consumption are mainly introduced by the patcg. As the gas consumption depends on the contract size, namely the size of original contract plus the size of the patch as well as the voteDestruct mechanism for future protection. As shown in the results, the overhead introduced by the patch is not stable. This is because different contracts require different type of patches.

7.3.3 TEE Cluster Overhead

In SolSaviour, state migration and asset escrow are conducted by TEE cluster. We therefore evaluate the overhead introduced by TEE cluster. We build a Ethereum private network with four nodes (i.e., node A, B, C, and D), each is installed with an Ethereum endpoint. We record the number of blocks mined by them in one day. Then, we initialize the enclave in one node and monitor the blocks mined by each node. After that, we sequentially initialize enclaves in

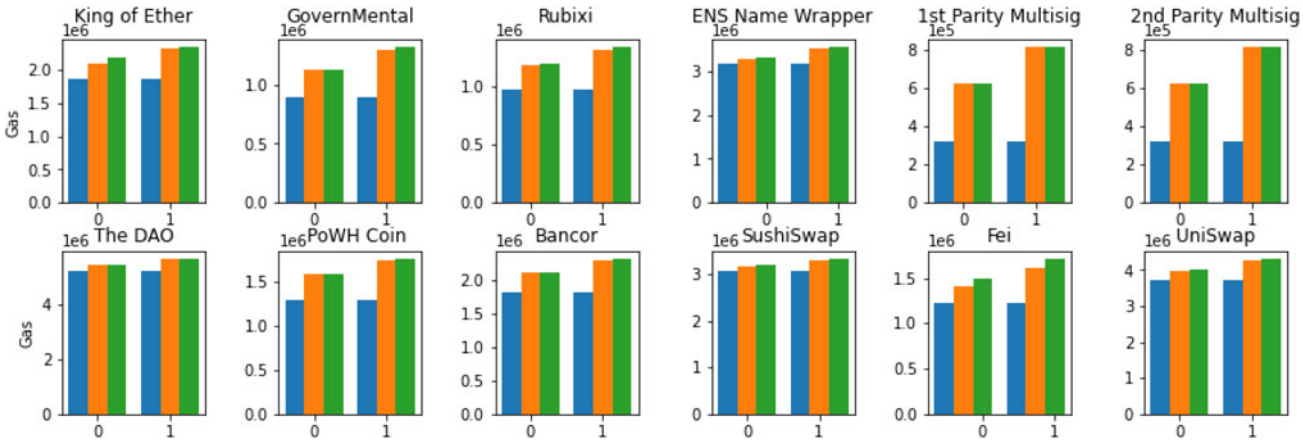


Fig. 9. Gas consumption increased in SolSaviour. The left three bars represent using TEE cluster for asset escrow while the right three bars using patched contract. Blue bar indicates original contract, orange bar indicates voteDestruct-enabled contract, and green bar indicates patched contract.

TABLE 4
The Overhead of TEE Cluster, Which is Counted in the Number of Blocks Mined by Different Combinations of TEE Nodes

Node		A	AB	ABC	ABCD
All	5780	5685(-1.6%)	5507(-4.7%)	5469(-5.3%)	5391(-6.7%)
A	1451	1368(-5.7%)	1312(-9.5%)	1340(-7.6%)	1341(-7.6%)
B	1446	1439	1325(-8.3%)	1339(-7.4%)	1335(-7.7%)
C	1438	1441	1439	1351(-6.1%)	1349(-6.2%)
D	1445	1437	1431	1439	1366(-5.5%)

the other three nodes and add them to the TEE cluster. During this time, we continuously monitor the number of blocks mined by each node. The mining difficulty remains the same during this experiment. We summarize the results in Table. 4. As we can see, for nodes without TEE cluster, they can mine around 1440 blocks per day, which satisfies the Ethereum blockchain generation speed, namely a block per 15 seconds. For nodes with TEE cluster, the mining rate is slightly affected. The impact is greatest when only half nodes participate the TEE cluster, and tends to become smaller when all nodes initialize TEE.

8 Related Work

In this section, we first present work on detecting contract vulnerabilities, protecting deployed contracts, and generating patches for vulnerable contracts. Then, we introduce recent work on investigating the security of DeFi. Finally, we present work on combining blockchain and TEE.

8.1 Smart Contract Vulnerabilities

Most work on smart contract vulnerability detection relies on techniques used in traditional software analysis. Luu et al., proposed Oyente [3] based on symbolic execution to automate the reentrancy bug detection. After that, lots of symbolic execution tools are proposed such as Osiris [24], teEther [25], Maian [26], EthBMC [27], and Zeus [28]. Feist et al., proposed Slither [5] to analyze the contract on source code level. Tsankov et al., presented Security [7] to analyze the contract on bytecode level. Brent et al., proposed Ethainter [6] to conduct information flow analysis to reveal composite vulnerabilities. Chen et al., proposed SODA [29], which accepts user-defined vulnerability pattern. Pan et al., proposed ReDefender [30], which detects reentrancy vulnerabilities with fuzz testing. Furthermore, the semantics of Solidity was formalized [31]. However, these proposed detection methods still have limitations, such as the inability to identify unknown vulnerabilities. Therefore, there has also been some work focused on protecting deployed smart contracts.

For contract defence, Rodler et al., proposed *Sereum* [32] to defend against reentrancy exploits through analyzing the EVM execution traces. Ferreira et al., proposed ÆGIS [33] to defend deployed contracts against known attacks. Specifically, it records a number of known contract attack execution traces through a proposed domain-specific language and integrates within the EVM to revert the execution of transactions that match the attack traces. Ellul et al., proposed a runtime verification mechanism [34] to ensure that violating party provides insurance for correct behavior. Li et al., proposed Solythesis [35] to address the high overhead in runtime validation.

For contract patch, Yu et al., proposed SCRepair [13] to detect and repair bugs in smart contracts before deployment. Zhang et al., proposed SMARTSHIELD [36], which leverages bytecode rewriting techniques to fix contract vulnerabilities. Bytecode rewriting technique was also used in the EVMPatch [37] to repair defective contracts.

8.2 DeFi Vulnerabilities

As DeFi has evolved greatly, more work has been carried out to investigate its security. Qin et al., explored attack vectors that leverage the Flash Loan in [38]. They quantitatively show how transaction atomicity affects arbitrage revenue. Zhou et al., proposed sandwich attack to exploit the information that may change the asset price [39]. Zhou et al., worked on crafting profitable transactions across the intertwined DeFi platforms automatically [40]. They investigated two methods that focus on arbitrage and complicated DeFi setting, respectively. Qin et al., analyzed the danger of blockchain extractable value (BEV) quantitatively [41]. They deduced the amount of possible profits for the attacker in the case of sandwich attacks, liquidations, and decentralized exchange arbitrage.

8.3 Smart Contract and TEE

There is a range of work focused on offloading the execution of smart contracts into the TEE, which enables the implementation of private and high-performance smart contracts. Cheng et al., proposed Ekiden [42], whose execution of smart contracts is deployed inside enclaves. With the enclave's public key, users can encrypt the input data in a confidential message call transaction. Das et al., proposed FastKitten [43] to enable the execution of complex, high-performance smart contracts on Bitcoin. In FastKitten, enclaves take charge of executing smart contracts and generating state transitions, which are recorded by the Bitcoin. FastKitten can extend its work to execute smart contracts on more blockchains which were designed to only support naive transactions. Liu et al., proposed Saber [44], a parallel and asynchronous execution paradigm of smart contracts boosted by TEE.

In addition, the combination of blockchain and TEE shows promise in many other areas. Zhang et al., proposed Town Crier to address the problem that smart contracts running on the blockchain cannot access information in a trusted way [45]. Zhang et al., proposed REM [46], a resource efficient mining algorithm that works on useful computation workloads. Considering the problem that traditional PoW consensus algorithms consume lots of power, REM converts the traditional meaningless hash computation into meaningful computation workload, accompanied by a SGX-based verification mechanism. Lind et al., proposed Teechain [47], a layer-2 payment network that can process off-chain transactions asynchronously with TEE. Li et al., proposed Pistis [48], which leverages smart contract and TEE to issue trusted and authorized certificates to address the problem of rogue certificates. Matetic et al., proposed Bite [49], a SGX-based lightweight node, to address

the privacy issue in lightweight nodes. In BITE, full nodes leverage SGX enclaves to process privacy preserving requests from lightweight clients.

9 CONCLUSION

In this paper, we present SolSaviour that can protect deployed but defective smart contracts and DeFi protocols. Compared with existing work that requires a trusted third party to redeploy patched contract and can only migrate contract data, SolSaviour can achieve effective migration of contract assets and does not require the involvement of a trusted third party. SolSaviour enables the offline patching of defective smart contracts through the decentralized control provided by voteDestruct mechanism and the state migration provided by TEE cluster. For all collected contracts and DeFi, our experiment results demonstrate that SolSaviour can effectively repair and recover all of them with affordable overhead.

REFERENCES

[1] V. Buterin, "Critical update re: Dao vulnerability," 2016. [Online]. Available: https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/

[2] R. Behnke, "Explained: The Fei protocol hack (April 2022)," 2022. [Online]. Available: https://halborn.com/explained-the-fei-protocol-hack-april-2022/

[3] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 254–269.

[4] K. Bhargavan et al., "Formal verification of smart contracts: Short paper," in *Proc. ACM Workshop Program. Lang. Anal. Secur.*, 2016, pp. 91–96.

[5] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proc. IEEE/ACM 2nd Int. Workshop Emerg. Trends Softw. Eng. Blockchain*, 2019, pp. 8–15.

[6] L. Brent, N. Grech, S. Lagouvardos, B. Scholz, and Y. Smaragdakis, "Ethainter: A smart contract security analyzer for composite vulnerabilities," in *Proc. 41st ACM SIGPLAN Conf. Program. Lang. Des. Implementation*, 2020, pp. 454–469.

[7] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 67–82.

[8] T. D. Nguyen, L. H. Pham, J. Sun, Y. Lin, and Q. T. Minh, "sFuzz: An efficient adaptive fuzzer for solidity smart contracts," in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng.*, 2020, pp. 778–788.

[9] R. Behnke, "The Nomad bridge hack: A deeper dive," 2022. [Online]. Available: https://halborn.com/the-nomad-bridge-hack-a-deeper-dive/

[10] Z. Li, Y. Zhou, S. Guo, and B. Xiao, "SolSaviour: A defending framework for deployed defective smart contracts," in *Proc. Annu. Comput. Secur. Appl. Conf.*, 2021, pp. 748–760.

[11] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "Defining smart contract defects on ethereum," *IEEE Trans. Softw. Eng.*, vol. 48, no. 1, pp. 327–345, Jan. 2022.

[12] J. Feist, G. Grieco, and A. Groce, "sGUARD: Towards fixing vulnerable smart contracts automatically," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 1215–1229.

[13] X. L. Yu, O. Al-Bataineh, D. Lo, and A. Roychoudhury, "Smart contract repair," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 4, pp. 1–32, 2020.

[14] K. Team, "King of ether throne post-mortem investigation," 2016. [Online]. Available: https://www.kingoftheether.com/postmortem

[15] u/ethererik, "GovernMental's 1100 ETH jackpot payout is stuck because it uses too much gas," 2016. [Online]. Available: https://www.reddit.com/r/ethereum/comments/4ghzhv/governmentals_1100_eth_jackpot_payout_is_stuck/

[16] Katatsuki, "Re: Hi! My name is Rubixi. I'm a new Ethereum Doubler. Now my new home - Rubixi.tk," 2016. [Online]. Available: https://bitcointalk.org/index.php?topic=1400536.60

[17] samczsun, "The dangers of surprising code," 2021. [Online]. Available: https://samczsun.com/the-dangers-of-surprising-code/

[18] H. Qureshi, "A hacker stole 31m of ether - how it happened, and what it means for ethereum," 2017. [Online]. Available: https://www.freecodecamp.org/news/a-hacker-stole-31m-of-ether-how-it-happened-and-what-it-means-for-ethereum-9e5dc29e33ce/

[19] A. Akentiev, "Parity multisig hacked. Again," 2017. [Online]. Available: https://medium.com/chain-cloud-company-blog/parity-multisig-hack-again-b46771eaa838

[20] E. Banisadr, "How 800k Evaporated from the PoWH coin ponzi scheme overnight," 2018. [Online]. Available: https://medium.com/@ebanisadr/how-800k-evaporated-from-the-powh-coin-ponzi-scheme-overnight-1b025c33b530

[21] I. Bogatyy, "Implementing ethereum trading front-runs on the bancor exchange in Python," 2017. [Online]. Available: https://hackernoon.com/front-running-bancor-in-150-lines-of-python-with-ethereum-api-d5e2bfd0d798

[22] samczsun, "Two rights might make a wrong," 2021. [Online]. Available: https://samczsun.com/two-rights-might-make-a-wrong/

[23] PeckShield, "Uniswap/Lendf.me hacks: Root cause and loss analysis," 2020. [Online]. Available: https://peckshield.medium.com/uniswap-lendf-me-hacks-root-cause-and-loss-analysis-50f3263dcc09

[24] C. F. Torres, J. Schütte, and R. State, "Osiris: Hunting for integer bugs in ethereum smart contracts," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, 2018, pp. 664–676.

[25] J. Krupp and C. Rossow, "teEther: Gnawing at ethereum to automatically exploit smart contracts," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1317–1333.

[26] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, 2018, pp. 653–663.

[27] J. Frank, C. Aschermann, and T. Holz, "EthBMC: A bounded model checker for smart contracts," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 2757–2774.

[28] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: Analyzing safety of smart contracts," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–12.

[29] T. Chen et al., "Soda: A generic online detection framework for smart contracts," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–17.

[30] Z. Pan, T. Hu, C. Qian, and B. Li, "Redefender: A tool for detecting reentrancy vulnerabilities in smart contracts effectively," in *Proc. IEEE 21st Int. Conf. Softw. Qual., Rel. Secur.*, 2021, pp. 915–925.

[31] J. Jiao, S. Kan, S.-W. Lin, D. Sanan, Y. Liu, and J. Sun, "Semantic understanding of smart contracts: Executable operational semantics of solidity," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 1695–1712.

[32] M. Rodler, W. Li, G. O. Karame, and L. Davi, "Sereum: Protecting existing smart contracts against re-entrancy attacks," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

[33] C. Ferreira Torres, M. Baden, R. Norvill, B. B. Fiz Pontiveros, H. Jonker, and S. Mauw, "Ægis: Shielding vulnerable smart contracts against attacks," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur.*, 2020, pp. 584–597.

[34] J. Ellul and G. J. Pace, "Runtime verification of ethereum smart contracts," in *Proc. 14th Eur. Dependable Comput. Conf.*, 2018, pp. 158–163.

[35] A. Li, J. A. Choi, and F. Long, "Securing smart contract with runtime validation," in *Proc. 41st ACM SIGPLAN Conf. Program. Lang. Des. Implementation*, 2020, pp. 438–453.

[36] Y. Zhang, S. Ma, J. Li, K. Li, S. Nepal, and D. Gu, "Smartshield: Automatic smart contract protection made easy," in *Proc. IEEE 27th Int. Conf. Softw. Anal., Evol. Reengineering*, 2020, pp. 23–34.

[37] M. Rodler, W. Li, G. O. Karame, and L. Davi, "EVMPatch: Timely and automated patching of ethereum smart contracts," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 1289–1306.

[38] K. Qin, L. Zhou, B. Livshits, and A. Gervais, "Attacking the defi ecosystem with flash loans for fun and profit," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, 2021, pp. 3–32.

[39] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 428–445.

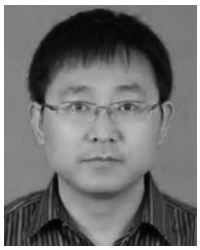
[40] L. Zhou, K. Qin, A. Cully, B. Livshits, and A. Gervais, "On the just-in-time discovery of profit-generating transactions in defi protocols," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 919–936.

[41] K. Qin, L. Zhou, and A. Gervais, "Quantifying blockchain extractable value: How dark is the forest?," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 198–214.

[42] R. Cheng et al., "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2019, pp. 185–200.
[43] P. Das et al., "Fastkitten: Practical smart contracts on bitcoin," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 801–818.
[44] J. Liu, P. Li, R. Cheng, N. Asokan, and D. Song, "Parallel and asynchronous smart contract execution," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 5, pp. 1097–1108, May 2022.
[45] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 270–282.
[46] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. Van Renesse, "REM: Resource-efficient mining for blockchains," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1427–1444.
[47] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: A secure payment network with asynchronous blockchain access," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, 2019, pp. 63–79.
[48] Z. Li, H. Wu, L. H. Lao, S. Guo, Y. Yang, and B. Xiao, "Pistis: Issuing trusted and authorized certificates with distributed ledger and TEE," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 7, pp. 1636–1649, Jul. 2022.
[49] S. Matetic, K. Wüst, M. Schneider, K. Kostiainen, G. Karame, and S. Capkun, "Bite: Bitcoin lightweight client privacy using trusted execution," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 783–800.

Zecheng Li received the BEng degree from the School of Information Science and Technology, Southeast University, Nanjing, China, in 2017 and the PhD degree from the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, in 2022. His research interest lies in the network security, blockchain security, and smart contract security.

Bin Xiao (Senior Member, IEEE) received the BSc and MSc degrees in electronics engineering from Fudan University, China, and the PhD degree in computer science from University of Texas at Dallas, USA. He is a full professor with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. After his PhD graduation, he joined the Department of Computing of the Hong Kong Polytechnic University as an assistant professor. His research interests include AI and network security, data privacy, and blockchain systems. He published more than 180 technical papers in international top journals and conferences. Currently, he is the associate editor for *IEEE Internet of Things Journal*, *IEEE Transactions on Cloud Computing*, and *IEEE Transactions on Network Science and Engineering*. He is the vice chair of IEEE ComSoc CISTC committee. He has been the symposium track co-chair of IEEE ICC 2020, ICC 2018 and Globecom 2017, and the general chair of IEEE SECON 2018. He is the member of ACM and CCF.



Songtao Guo (Senior Member, IEEE) received the BS, MS and PhD degrees in computer software and theory from the Chongqing University, Chongqing, China, in 1999, 2003 and 2008, respectively. At present, he is a full professor with Chongqing University, China. His research interests include wireless sensor networks, wireless ad hoc networks and parallel and distributed computing. He has published more than 80 scientific papers in leading refereed journals and conferences. He has received many research grants as a principal investigator from the National Science Foundation of China and Chongqing and the Postdoctoral Science Foundation of China.

Yuanyuan Yang (Fellow, IEEE) received the BEng and MS degrees in computer science and engineering from Tsinghua University, Beijing, China, and the MSE and PhD degrees in computer science from Johns Hopkins University, Baltimore, Maryland. She is a SUNY distinguished professor of computer engineering and computer science and the associate dean for Academic Affairs in the College of Engineering and Applied Sciences with Stony Brook University, New York. Her research interests include data center networks, cloud computing and wireless networks. She has published more than 380 papers in major journals and refereed conference proceedings and holds seven US patents in these areas. She is currently the associate editor-in-chief for *IEEE Transactions on Cloud Computing*. She has served as an associate editor-in-chief and an associate editor for *IEEE Transactions on Computers* and associate editor for *IEEE Transactions on Parallel and Distributed Systems*. She has also served as a general chair, program chair, or vice chair for several major conferences and a program committee member for numerous conferences.

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 5, 2021, accepted February 22, 2021, date of publication March 9, 2021, date of current version March 18, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3064896

PETchain: A Blockchain-Based Privacy Enhancing Technology

IBRAHIM TARIQ JAVED[1], FARES ALHARBI[2], TIZIANA MARGARIA[1], NOEL CRESPI[3], AND KASHIF NASEER QURESHI[4]

[1]Lero—Science Foundation Ireland Research Centre for Software, University of Limerick, V94 T9PX Limerick, Ireland
[2]Computer Science Department, Shaqra University, Riyadh 11961, Saudi Arabia
[3]Institut Polytechnique de Paris, Telecom SudParis, 91011 Evry, France
[4]Department of Computer Science, Bahria University, Islamabad 44000, Pakistan

Corresponding author: Ibrahim Tariq Javed (ibrahimtariq.javed@lero.ie)

This work is partly supported with the financial support of the Science Foundation Ireland grant 13/RC/2094_P2 and partly funded from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 754489.

ABSTRACT With the increasing use of smart devices and sensors, enormous amounts of data are being generated continuously. The data is commonly stored in centralized cloud platforms and consumed by different services. The data is indeed a valuable resource for many service providers who provide advanced features and utilities to their subscribers. However, user data include personal and sensitive information which can be misused in many ways. There is no way for a subscriber to confirm that their service provider is compliant with data privacy regulations. The existing privacy enhancing techniques such as anonymization and differential privacy substantially reduce data usability while ensuring privacy. Therefore, it remains essential to provide a feasible solution that allows service providers to take advantage of user data while guaranteeing their privacy. In this paper, we present PETchain: a novel privacy enhancing technology using blockchain and smartcontract. In PETchain, data is stored securely in a distributed manner and processed in a user-selected trusted execution environment. Users deploy the smartcontract that allows them to decide whether and how their data can be exploited by service providers. The feasibility and performance of PETchain are presented by implementing PETchain over a consortium Ethereum blockchain.

INDEX TERMS Blockchain, Ethereum, privacy enhancing technology, privacy preservation, smartcontract.

I. INTRODUCTION

The rapid development in the fields of IoT, social networks, and cloud computing has led to the increased generation, storage, and processing of personalized data. In this new era of big data, both public and private service providers are collecting large amounts of data from their users to provide them with enhanced services and features. However, most of the collected contains private and confidential information that can be easily abused. Service providers focus on providing strong authentication, integrity, and confidentiality solutions but generally under-look user's privacy while collecting, storing, and processing their personal data [1]. Once a service provider acquires the data it can easily misuse or distribute it without user consent. Thus, users must completely trust their service providers and have little or no control over their data. Furthermore, the users are unable to define and implement access control for their data, to decide who and when can access their data, and how can they process it.

The General Data Protection Regulation (GDPR)[1] came into act on 25 May 2018 in the European Union. The GDPR contains provisions and requirements to protect users' rights related to their data. This includes the user's right to remain informed about their data being gathered, processed, and distributed by their service provider. It also provides the user with the right to have their data updated or even deleted forever [2]. However, users, for now, do not have any ability to check whether their service providers are GDPR compliant or not. Users have to completely trust their service providers with their data if they wish to use their services. Furthermore, if any investigation on a privacy breach is conducted, the supervisory authority does not have any reliable or auditable logs to inspect. Authorities must rely on the logs maintained by the service providers themselves.

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Imran Tariq.

[1]https://gdpr-info.eu/

To protect and improve user privacy several privacy-enhancing technologies (PET) have been introduced in the literature. The most widely adopted include homomorphic encryption [3], anonymization [4], and differential privacy [3]. Homomorphic encryption has a significant computational overhead for data processing which makes it incompatible with most of the current client-server applications. On the other hand, anonymization and differential privacy reduce the originality of the data, thus limiting the service providers' ability to derive value from user data. Recently, the utilization of blockchain technology is considered to be a promising solution to preserve user privacy due to its widely recognized accuracy and integrity features [1]. DIN SPEC 4997 [5] describes technical and organizational measures for data protection as well as requirements based on GDPR. It further presents architectural blueprints to illustrate how to use blockchain to improve data privacy. Therefore, in this paper we propose PETchain: a user-centric PET solution that utilizes blockchain technology along with smartcontracts, InterPlanetary File System (IPFS), and Trusted Execution Environment (TEE). PETchain allows users to securely store their data in a distributed manner and implement their data access policy in an accountable and auditable manner. Thus, PETchain is suitable to ensure privacy in data management and sharing applications. It further allows service providers to provide services to their users without keeping a copy of their data.

The major contributions of this paper are as follows:

1) A novel privacy enhancing technology, ''PETchain'', that allows service providers to utilize user data while guaranteeing their privacy. PETchain allows users to securely store and maintain their data in IPFS. Data owners are given complete control over their data by deciding who can access, process, and utilize their data. The data is processed in a user-selected isolated secure environment, assuring the confidentiality and integrity of the data. Logs of each data access are maintained in an immutable and auditable manner in the blockchain.
2) A smartcontract that allows users to define their access control policy. The PETchain smartcontract consists of several functions. With the set_identifier function the user stores data identifiers of the data files uploaded over IPFS, whereas with the set_authorization function users authorize service providers to utilize their data. Get_identifier is the only function that can be called by the service provider to request data access. Moreover, with the destroy_smartcontract and pause_smartcontract functions the users can pause and remove their smartcontract at any time.
3) An implementation of PETchain over a consortium blockchain. We choose to use a consortium blockchain as it maintains user control and privacy while having reduced cost and high throughput when compared to the public blockchain. Furthermore, unlike a private blockchain network, it does not consolidate power to a single party and distributes it across different organizations. We analyze the performance of PETchain on a consortium blockchain by using four performance metrics: transaction GAS cost, transactions per second, number of lost blocks, and propagation delay. Different parameters such as sealers, block-time, and block-gas-limit were analyzed to achieve the best possible results for our consortium blockchain. We choose to implement our solution using the Ethereum platform as it facilitates developers to deploy their smart contracts in a secure and simple manner. However, PETchain can be implemented over any consortium blockchain platform that supports the deployment of smart contacts.

The rest of the paper is structured as follows: the related work is described in Section II highlighting existing PET solutions. In section III, we provide an overview of the blockchain technology, describing types and advantages of blockchains, consensus agreement, and smartcontracts. In section IV, we introduce and detail the PETchain solution to ensure the privacy of users in a holistic manner. In Section V, we describe how we implemented our solution on Ethereum and provide its performance analysis. Lastly, we offer our conclusions in Section VI.

II. RELATED WORK

In this section, we detail the related work by presenting different PET solutions from the literature. As illustrated in Figure 1, this includes anonymization, pseudonymization, homomorphic encryption, differential privacy, data summarization, and decentralized learning. We describe the benefits and limitations of each PET solution. We further present a few prototypes that utilize blockchain technology to improve user privacy.

Anonymization and pseudonymization techniques are used to remove personal information from the dataset allowing data owners to remain unidentifiable [6]. Anonymization is completely irreversible whereas pseudonymization allows the data owner to be re-identified by using some additional information. The three most common techniques are K-anonymization [4], L-diversity [7] and T-closeness [8]. K-anonymity is achieved using suppression and generalization methods until each row of the dataset becomes identical to at least $k - 1$ other rows. L-diversity is based on k-anonymity to address homogeneity and background knowledge attacks [9], whereas T-closeness [8] reduces attribute disclosure by decreasing the granularity of the data. These techniques are being widely adopted to achieve user privacy. However, they consistently reduce the originality of the information by decreasing the accuracy and precision. This highly affects the utility of the data. Higher levels of anonymization enhance privacy but make data unproductive and ineffective.

Homomorphic encryption allows procedures to be performed on encrypted information, guaranteeing the same output as if the operations were performed on the clear data. This property enhances privacy by enabling applications

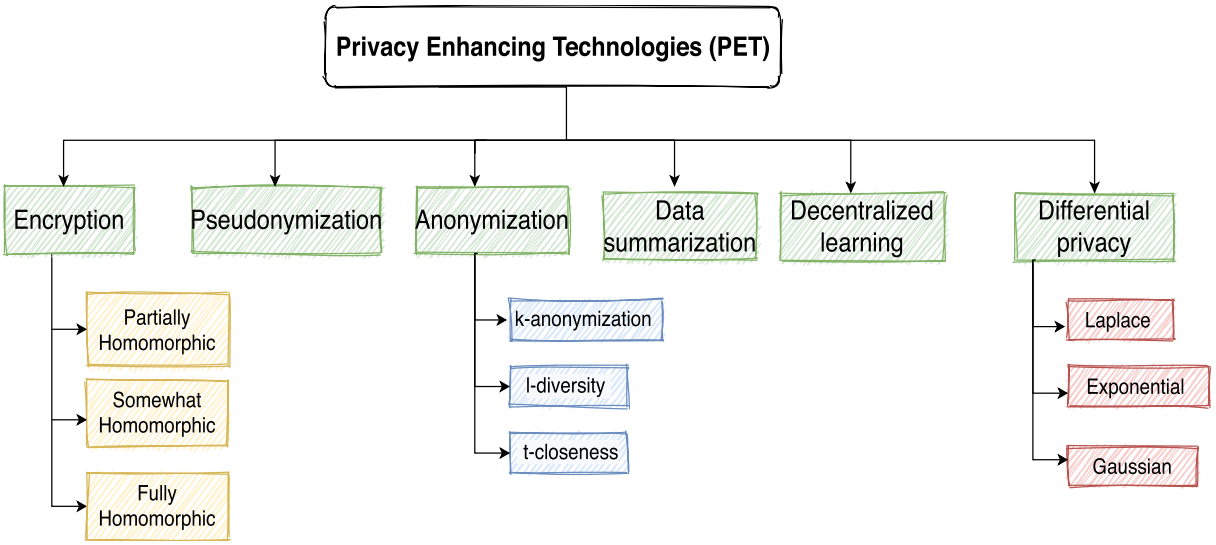


FIGURE 1. Taxonomy of privacy enhancing technologies.

to process, store, and share encrypted data rather than the original data. Homomorphic encryption can be further categorized into partial, somewhat, and total homomorphic encryption. Partial homomorphic encryption can only support one operation, whereas somewhat homomorphic encryption can support two types of operations together. However, both partial and somewhat homomorphic encryption techniques can only support basic operations types such as addition and multiplication. For this reason, they don't apply to scenarios that require advanced analytical computations to process the data. On the other hand, total homomorphic encryption supports a vast variety of analytical operations on encrypted data. However, this comes at the expense of computational latency: homomorphic encryption is impractically slow as it has a large computation overhead when applying operation on encrypted data. It usually takes 2-5 seconds per operation, which is incredibly slow for most practical uses [3]. Therefore, unless the computational overhead is substantially decreased, homomorphic encryption algorithms remain counterproductive for data-hungry applications.

Decentralized learning [10] allows sensitive data to be offloaded to end-user devices where the processing can be done. This enhances privacy by eliminating the risk of exposure of sensitive data, but it overburdens the user devices. Therefore, decentralized learning also remains impractical for applications that require immense computations similar to homomorphic encryption. Moreover, decentralized learning is also prone to attacks that expose intermediate results and leak information about user data.

Differential privacy techniques allow to collect and share aggregate information about users while maintaining the privacy of individual users. This is done by adding statistical noise to each user's data before it is shared [11]. The aggregate information from the dataset can then be extracted by deducting the noise from it. However, information regarding a particular individual cannot be derived. Three major strategies are used in differential privacy: Laplace, Exponential, and Gaussian [12]. The Laplace and Gaussian strategies are typically utilized for numerical datasets, while exponential techniques are used for non-numerical datasets. Differential privacy is considered to be an effective method to gather aggregate user information while preserving the privacy of individuals. However, privacy is achieved at the expense of data accuracy. The amount of noise added highly impacts the precision and accuracy of data. The trade-off between data precision and privacy is controlled using an operator referred to as epsilon [13]. Concealing sensitive information requires a high value of epsilon, which affects the usefulness of the information. Furthermore, differential privacy is only applicable to scenarios where aggregate information is useful and cannot be used for personalized services. Similarly, data summarization techniques [14] are also not applicable to personalized services as they create summarized variants of data sets that hide the actual information of individuals.

After the success of Bitcoin [15], researchers started paying more attention to the underlying blockchain technology. Due to its cryptographic security, immutability, and accountability features blockchain is useful in many non-financial fields. Over the last few years, researchers started using blockchain technology as a decentralized access control moderator in various areas where access control policies are defined by using smartcontracts. FairAcces [16] provides a secure access control mechanism for IoT applications. A proof of concept is presented using Raspberry PI and local blockchain to grant, delegate, and revoke access. In [17] smartcontracts are used to provide attribute-based access control for cloud storage systems. A role-based access control is presented in [18] in which user relationships are publicly visible on the blockchain. [19] presents a user-driven access control framework for the decentralized online social network which allows user to define their privacy policies. The main advantage of these access control mechanisms is auditability,

as blockchain handles each transaction in a decentralized, transparent, and immutable manner. However, these mechanisms are designed to provide access control in a particular scenario where privacy is not considered their primary objective.

Researchers have also used blockchain technology to decentralize traditional data management systems. For instance, in [20] a blockchain-based data management solution ensures GDPR compliance and avoids security violations. An architectural blueprint for personal data management using blockchain is presented in [5]. [21] presents an IoT storage system that allows to manage and trade data generated by IoT devices securely and efficiently. [22] provides a decentralized personal data management system where user-defined access control and regulations are embedded into the blockchain using smartcontracts. For industry 4.0 applications, [23] introduces a distributed resource management framework based on smartcontract technology. Whereas a privacy preservation solution for Medical IoT applications is proposed in [24] to protect individuals' rights related to personal data. Most of these works are considered to be prototypes and lack technical analysis and implementation. The conceptual and architectural groundwork of theoretical nature has been presented but a complete implementation of a privacy preservation solution is still missing. However, the major drawback of decentralized data management and storage systems is that whenever a service provider gains access to user data they retain a copy of the user's data. This leads to privacy concerns as service providers can use the data to extract critical information or distribute it to a third party without user consent. Therefore, in the current decentralized data management systems users still have to trust their service providers to properly utilize their data when providing them with services.

We observed that a comprehensive approach to resolve privacy using blockchain technology with proper implementation and performance analysis is still missing in the literature. Therefore, in this paper we utilize existing concepts to propose a new PET solution that holistically addresses privacy by using blockchain and smartcontract technologies. It aims to cover three aspects of privacy protection: i) control over data, ii) access control and iii) auditability. A user-centric approach is adopted to ensure user control over data, allowing the user to be in charge of what data is collected, where it is stored, and how it is processed. Access control refers to the policies that the user can implement to specify by whom and when they want their data to be accessed or shared. Auditability is ensured by maintaining immutable logs for each data access so that a supervisory authority has accurate and authentic data records to investigate upon.

III. BLOCKCHAIN OVERVIEW

Blockchain technology has recently attained a lot of attention as it is used in various intuitive applications. Blockchain is a distributed database maintaining a scalable list of data records. Each block of information carries transaction data, a timestamp and an encrypted hash value of the previously linked block [25]. Linking these blocks using the hash function makes the chain of data grow in the database. The concept came up in 2008 by Satoshi Nakamoto, who devised the first blockchain database by solving the double-spending problem using the Hashcash method. Blockchain was further adopted as a technique to implement the cryptocurrency bitcoin [15]. In bitcoin, blocks serve as a public ledger that keeps all the financial transactions. Recently, blockchain has been integrated with other business domains and fields of application to enhance their security and privacy. Besides, smartcontracts deployed over blockchain promise to improve agreements between parties without third-party intervention. The following subsections will present a further overview of some terminology of blockchain.

A. WORKING OF BLOCKCHAIN

A blockchain consists of many blocks that themselves consist of hashed batches of transactions. A transaction contains information that needs to be shared securely. Transactions are grouped inside a block based on the size of the block. The block is sent over the network in a synchronized manner that allows each node to store a copy of the blockchain. The synchronization requires an agreement between nodes which is called consensus. The working of a blockchain can be summarized in the following phases: transactions creation, transaction submission and broadcast phase, block validation, block inclusion into the blockchain. Figure 2 shows stepwise how blockchain works. In the first step, an entity requests a transaction that contains useful information. The block representing the transaction is then broadcast to all nodes in a network using a Peer-to-peer protocol. The network of nodes validates the transaction with the help of a known algorithm. Once the transaction is validated a new block containing the transaction is added to the blockchain.

Once the transaction is submitted it needs to be authenticated. This is done using a pair of cryptographic public and private keys. The public key is used to create a digital identity whereas the private key is used to provide a digital signature. The digital signature provided with the transaction is used to authenticate the user [26]. However, before the transaction is added to a block it also needs to be approved or validated. This decision is made using a consensus protocol [27]. The validators investigate the blocks and check that they meet all the requirements. Once this is completed, the transaction is marked as valid, grouped, and published. The validation process is the key that allows a new block to be added to the blockchain. Once the validation is agreed by all nodes, the agreement has been reached for the block following the consensus protocol.

B. TYPES OF BLOCKCHAIN

There are four types of blockchain architecture: public, private, consortium, and hybrid. All these types have a peer-to-peer network consisting of nodes that can create and validate transactions. However, there are differences in terms

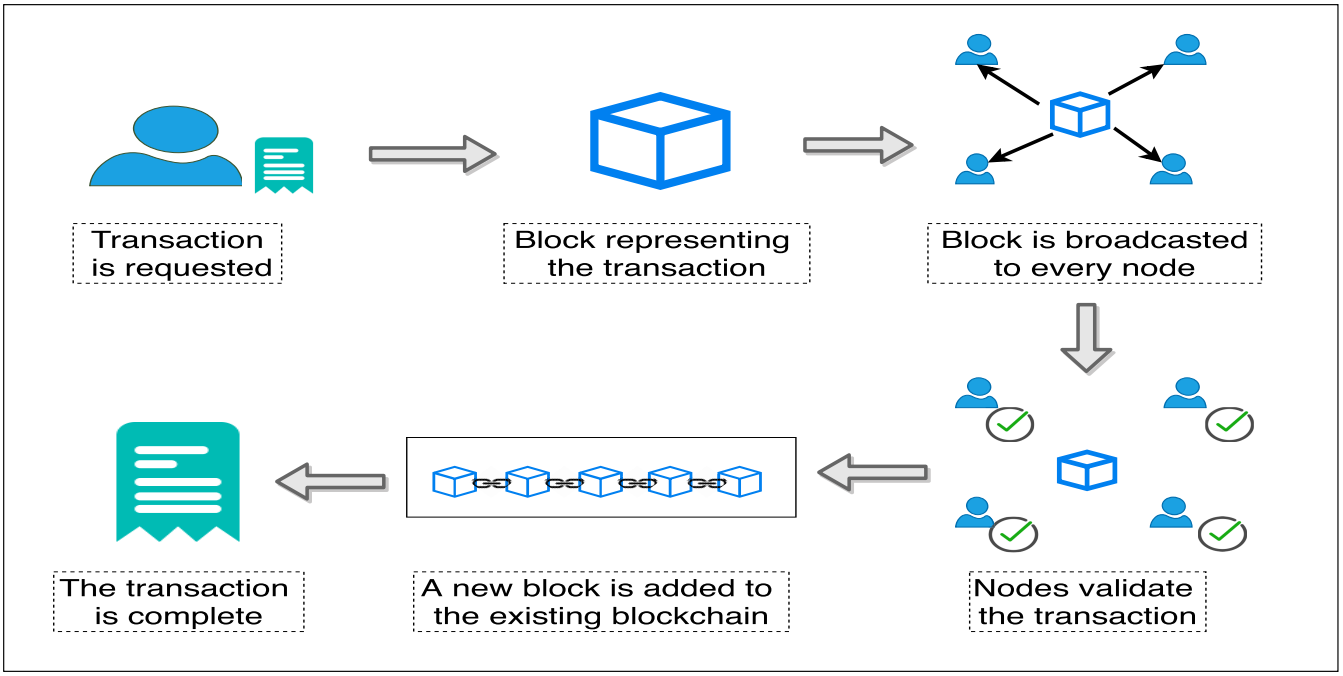


FIGURE 2. Working of blockchain.

of accessibility and availability between different types of blockchain.

1) PUBLIC BLOCKCHAIN

A public blockchain is an open-source platform that allows anyone to sign-in and access without any permission. All its participating nodes are given the authority to verify transactions and validate blocks. Nodes conduct a mining process to reach a consensus agreement for including a block to the blockchain. The permissionless and non-restrictive nature of public blockchain makes it suitable for cryptocurrencies such as Bitcoin [15].

2) PRIVATE BLOCKCHAIN

A private blockchain consists of a closed network owned by an entity or organization and is restricted to specific users. A new user can be added only if the owner of the private blockchain allows it. The creation of new blocks and adding transactions are restricted, and permission is only given to controlling nodes set up by the owner of the blockchain. The restrictions due to inaccessibility and authorization increase the level of security and privacy. Therefore, this characteristic of private blockchains makes them favorable for use in financial institutions and banks.

3) CONSORTIUM BLOCKCHAIN

A consortium blockchain is a community blockchain that allows more than one organization to be involved in managing a private blockchain [28]. In a consortium blockchain, the authority is shared among a group of entities, so that consortium blockchains are considered semi-decentralized. Transactions can only be verified by a set of consortium nodes. Thus, the consortium blockchain awaits has a low transaction cost associate with it and does not face scalability problems as compared to public blockchains.

4) HYBRID BLOCKCHAIN

Hybrid blockchain is a mixture of public and private blockchain and takes advantage of both characteristics simultaneously [29]. The hybrid blockchain is customizable as it allows users to decide who can participate in the blockchain and which transactions can be made public. Thus, its hybrid architecture ensures privacy by taking advantage of the restricted access of private blockchain while maintaining integrity, transparency, and security as in a public blockchain.

C. CONSENSUS AGREEMENT

Consensus is defined as a general common interest allowing participants of a multi-agent system to agree on a specific target to be accomplished. In the blockchain, consensus agreement is used to define rules and regulations that allow different nodes in the network to reach a settlement on which transaction is valid and can be included in a new block of the blockchain [30]. Various consensus protocols exist, and three have gained popularity: Proof of Work (PoW), Proof of Stake (PoS), and Proof of Authority (PoA).

1) PROOF OF WORK

The PoW protocol is used during mining to provide validation for new blocks. In this protocol, the nodes compete with each other to receive a reward by confirming that the transaction received is accurate. The process includes a hard mathematical puzzle to be solved by mining nodes. The nodes try to solve a puzzle using a trial and error method. Once a mining node finds a solution it communicates it with the

rest of the nodes. A consensus is achieved when the majority of the nodes agree that the miner has solved the problem. Once consensus is achieved the transaction is added to the blockchain and the miner receives a reward in the form of transaction fees. It is impossible to fake a consensus as it requires hacking at least 50% of nodes in the network. The PoW is a very efficient protocol, however, it consumes a lot of power and time due to its high computation cost to solve a mathematical puzzle.

2) PROOF-OF-STAKE

The PoS has a different methodology to generate consensus: it uses a pseudo-random function to select a miner that is allowed to create the new block based on the node's wealth. The stake or wealth of the node determines the chances for a node to become a validator for the next block. The higher the stake of the node, the higher its chances to become a validator. PoS resolves the high computational cost of the PoW protocol. However, a node's wealth is not always appropriate for the selection since the creation of new blocks may be taken over by a single wealthy node as done in centralized systems [31]. This may result in the blocks being managed unfairly. Different methods such as randomized block selection and coin age selection are introduced so that PoS does not favor only the wealthiest nodes.

3) PROOF-OF-AUTHORITY

In PoA the validators stake their identity instead of their wealth. The person who wants to validate a transaction needs to first confirm its identity. The validation performed is linked to the identity and stored over the blockchain. This means that the validators are staking their reputation. To become an authorized validator, the validator needs to confirm its real identity. When a transaction has been validated the identity of the validator is confirmed on-chain in a certain way by an approved protocol. The identity is only staked by a small group of validators, improving the efficiency and security of the consensus agreement. PoA does not require high computational cost as PoW or a huge amount of wealth to stake as PoS. However, the PoA is only applicable to private and consortium blockchain networks.

D. WHY BLOCKCHAIN

The revolution of blockchain technology has recently reached efficient and successful integration with different applications. This success comes from fundamental features that we summarise here as follows:

- Decentralization: Blockchain distributes information to all nodes available on its network instead of keeping the information in a central entity. This makes it more secure, as no single entity can take control of the network and tamper with information.
- Immutability: Data inside a blockchain cannot be altered since there are many copies of data on different nodes, which makes tampering almost impossible. Furthermore, the use of cryptography does not allow anyone to intrude and alter the data saved on the blockchain. Any modification of data needs to be agreed upon by a majority of nodes.
- Integrity, Authenticity, and Non-Repudiation: When a transaction is made the data is guaranteed against any modifications by verifying the content against its transmitted hash. The user's private key is used to digitally sign the transaction, which assures the authenticity and non-repudiation of each message.
- Auditability and Traceability: Each transaction is recorded in a verifiable and permanent manner. In a blockchain, all operations are traceable and available to each participant. These features facilitate accountability and audit.

E. SMARTCONTRACT

We conclude the section of blockchain overview discussing smartcontracts since they are relevant to our work. A smartcontract is a method used to make, negotiate, and verify agreements electronically. The method is built by coding a computer program to allow participants to transfer anything of value under predefined conditions. Smartcontracts are considered as a replacement for the traditional contracts facilitated by third parties. It simply removes the third party by translating an agreement to a computer code that is executed over the blockchain. The code is automated and keeps track of the agreement's terms and conditions. Thus, smartcontacts are self-verifying, self-enforcing, and tamper-proof. The smartcontract is considered to be the third generation of the blockchain revolution. Blockchain has drastically gained in popularity from the smartcontract capability. Businesses that aim to simplify, speed up, and reduce their costs are leveraging smartcontracts. Fields that are currently benefiting from smartcontacts include healthcare, merchandising, real-estate, e-governance, IoT, etc.

IV. PETchain FRAMEWORK

In this section, we present PETchain: a privacy enhancing technology that utilizes blockchain and smartcontracts to ensure the privacy of users in a holistic manner. We start by describing the system actors and presenting the system model of PETchain. Next, we detail how PETchain utilizes distributed storage and a trusted executor to preserve user privacy. Finally, we describe PETchain smartcontracts and their various functions.

A. SYSTEM MODEL

The PETchain framework consists of five major entities as described below:

- **User**: An individual or organization that owns data and has the right to allow who can access and process it.
- **Service Provider**: An online public or private organization that is established to deliver various applications to their consumers. They provide services and features to their subscribers by acquiring and processing their data.
- **Distributed Storage**: A peer-to-peer distributed open-source file-sharing system that allows data to be

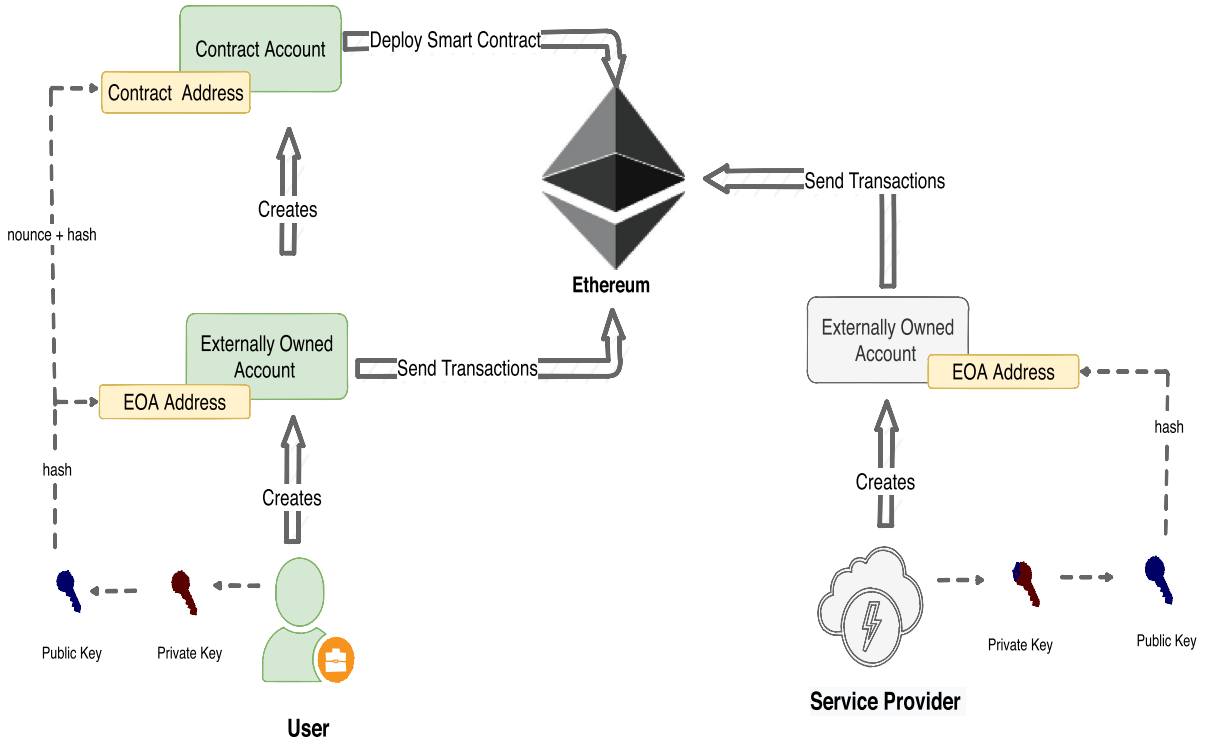


FIGURE 3. Ethereum accounts and addresses.

stored, maintained, and distributed in a fast and secure manner.

- **Trusted Executor**: A trusted individual or organization that provides a secure isolated environment that allows code to be executed over some data while guaranteeing the confidentiality and integrity of the data as well as the code there loaded.
- **Blockchain**: A peer-to-peer distributed platform that supports decentralized applications. It should supports smartcontract code execution and storage over the distributed network such as Ethereum.

In the PETchain framework, all users and service providers are obliged to register themselves to the Ethereum network. Figure 3 shows two types of accounts and addresses over the Ethereum blockchain. As shown in the figure, each entity registers an externally owned account (EOA) represented by an EOA address. Users also create a contract account to deploy their smartcontract having a unique contract address. PETchain exploits several encryption keys presented in Table 1. Each entity operating over Ethereum generates a pair of public key P_u^{eth} and private key P_r^{eth}. The P_u^{eth} is used to generate a unique 20-byte EOA account address that identifies it over Ethereum blockchain. Each message sent by the user over the PETchain platform is digitally signed by their P_r^{eth} for maintaining the authenticity and integrity of each message. K_D is a symmetric key generated by the user to encrypt data before uploading it over the distributed storage. The uploaded data is uniquely identified by its data identifier. The user also shares a secret key K_{TE} with the trusted executor. Users further encrypt K_D using K_{TE} to produce

TABLE 1.

Symbol	Representation
P_u^{eth}	Private key for ethereum
P_r^{eth}	Public key for ethereum
K_D	Symmetric key to encrypt data
K_{TE}	Symmetric key shared between user and trusted executor
K_D^E	Encrypted K_D using K_{TE}

K_D^E and store it over the smartcontract along with the data identifier.

The system model of PETchain is presented in Figure 4. There are two kinds of messages over PETchain: API calls, and contract transactions. The API messages are used by entities to interact with each other, whereas contract transactions are sent to the smartcontract over the Ethereum network. To completely understand how the PETchain system model works we describe the several steps involved:

1) *Publish smartcontract:* The user defines its access control over its smartcontract. The smartcontract contains the list of service providers authorized to utilize user data.
2) *Store data:* The user encrypts its data using E_D and uploads it over the distributed storage system.
3) *Update data identifier:* The user updates the data identifier of the uploaded data over its smartcontract.
4) *Request data:* The service provider requests data by calling a function of smartcontract. If the service provider belongs to the user's authorized list it will be returned with the data identifier and K_D^E. It also receives the URL of the user's trusted executor.

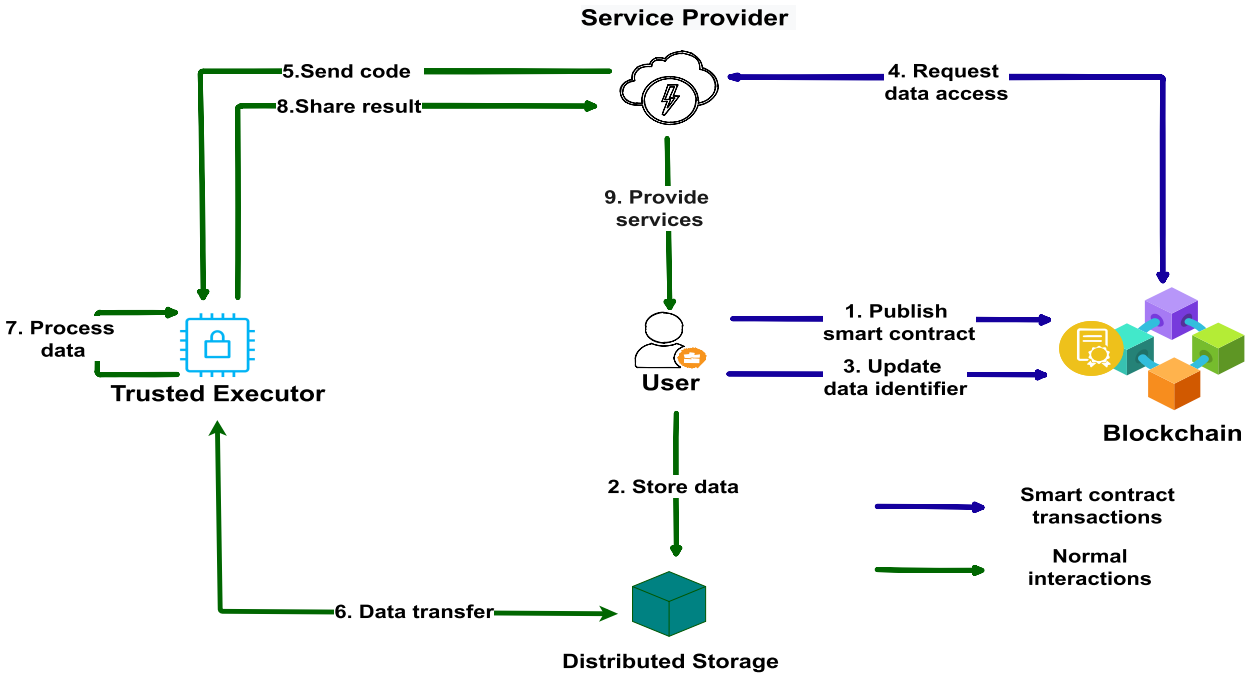


FIGURE 4. PETchain system model.

5) *Send code:* The service provider sends the code to be executed over user data and the data identifier to the trusted executor.
6) *Data transfer:* The trusted executor uses the data identifier to download the data from the distributed storage.
7) *Process data:* The data is processed in an isolated secure environment using the code received from the service provider. Trusted executor decrypts K_D^E to get K_D, which is then used to decrypt the downloaded data before executing the code.
8) *Share data:* The results of the processed data are shared with the service provider.
9) *Provide services:* The service provider can use the results to provide its services to the user.

PETchain supports a user-centric architecture that aims to give complete control to the user who owns the data. Users can maintain a list of authorized service providers (EOA addresses) over its smartcontract that are allowed to utilize their data. PETchain ensures the security of the data by allowing users to store the encrypted versions of their data on the distributed storage. Only the trusted executors will be able to decrypt and process user data. This is done in a secure isolated environment to ensure user privacy. Thus, PETchain allows service providers to provide users with services without obtaining their data. Moreover, with the use of blockchain PETchain keeps the logs of all data access in an auditable and immutable manner.

B. InterPlanetary FILE SYSTEM

PETchain uses a distributed data storage system instead of centralized cloud storage. In distributed data storage there is no single point of failure as data is scattered and stored across different peers. Moreover, faster access to data is achieved as peers can directly share content, unlike centralized storage which has high bandwidth requirements. PETchain uses InterPlanetary File System (IPFS)[2] as its distributed data storage system. IPFS defines a peer-to-peer protocol that allows data to be accessible through distributed peers [32]. IPFS is considered to be faster, secure, and reliable compared to cloud storage services. It also aims to resolve the data redundancy issue of the web. This is mainly because IPFS uses content addressing rather than location addressing to identify data stored over the network. The IPFS uses the hash of the data file as its identifier, which always remains unique to the data. A newer version of data is uploaded in case the data is updated or changed. If a particular data file is requested then the data identifier is used to locate the file across different peers. A node on IPFS whose stored hash matches with the content identifier will return the data file to the requester.

In PETchain, the user encrypts and digitally signs the data before uploading it over the IPFS network. Figure 5 illustrates the different steps a data file goes through before being uploaded over the IPFS. The user starts by selecting the data file required to be uploaded. The data file is then encrypted using a symmetric encryption algorithm such as AES and symmetric key K_d. Thus, K_d is required by anyone who wants to access and process the data. In the following step, the data file is digitally signed by the user. The user uses the private key P_r^{eth} of thee Ethereum's EOA to sign the data file. The public key P_u^{eth} can then be used to verify that the data file belongs to the same user that is registered on Ethereum and owns the smartcontract. The hash of the digitally signed data

[2]https://ipfs.io/

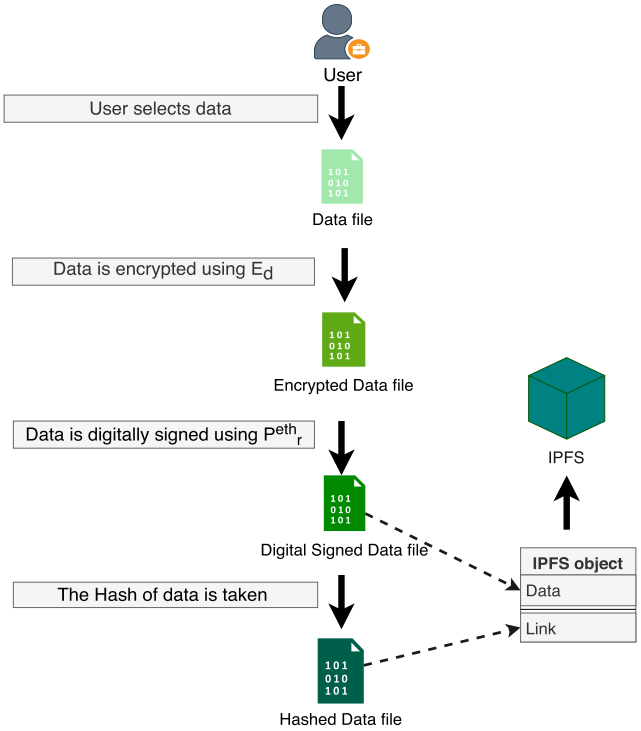


FIGURE 5. Data storage on IPFS.

file is taken using an SHA 256 algorithm. The hash acts as the data identifier. The user then uploads the data over IPFS in the form of an IPFS object that consists of the data field and link field. The data field consists of a data file whereas the link consists of the data identifier. IPFS data field can only accommodate 256 Kb of data. Therefore, if the data file is larger than 256 KB, it is broken down into several IPFS objects and stored over different peers in the network. The main object consists of the links to each IPFS object which can be used to locate other objects and recover the data file.

C. TRUSTED EXECUTION ENVIRONMENT

In PETchain, a trusted execution environment (TEE) is used to process user's data. Users can implement their TEE or use a third party trusted executor. A TEE is a processing environment that allows the authenticity, integrity, and confidentiality of its data and code. The basic building blocks of TEE are presented in Figure 6. TEE allows service providers to process the user's encrypted data in a secure environment without having access to the data. The trusted executor has access to both user's data and the service provider's code. However, the architecture of TEE makes it unfeasible for anyone to have a copy of the data. Once the code is executed over the data the service provider is informed about the results of the execution. In PETchain, we choose GrapheneOS [3] which is an open-source, security-hardened TEE. GrapheneOS implemented over Intel SGX hardware provides adequate performance. The latency on Graphene-SGX for

[3]https://grapheneos.org/

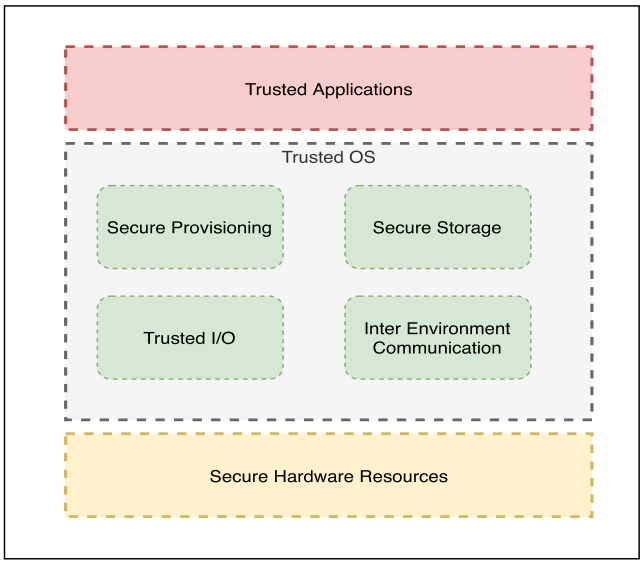


FIGURE 6. Basic building blocks of TEE.

Algorithm 1: set_identifier(*data_id*, *SP_address*, *data_key*, *TE_url*)

```
if msg.sender == owner then
    Map SP_address to data_id; Map SP_address to
     data_key; Map SP_address to TE_url; return true;
else
    return false;
end
```

opening a 64 KB file is 383μs, reading a 64KB file takes 0.5μs whereas forking a 16MB process takes 0.8s [33].

D. PETchain SMARTCONTRACT

We describe now the PETchain smartcontract and its functions. The PETchain smartcontract is written and compiled in solidity. The contract is deployed and owned by the user. When the contract is deployed the user's EOA address is recorded as the owner of the smartcontract. The smartcontract is then used to store and retrieve data identifiers. The variables used in the smartcontract to store information include i) *data_id*: contains data identifier, ii) *data_key*: contains encrypted key for the data, iii) *SP_address*: contains the address of the service provider, iv) *TE_url*: contains the URL of the trusted executor, and v) *Status*: indicates whether the service provider is trusted or not. We use mapping value type in solidity to relate and store information due to its lower computational cost compared to arrays [34]. The mapping stores information in a virtual hash table having each potential key mapped to a value.

The PETchain consists of five functions. The algorithms of these functions are presented in Algorithms 1-5 and explained as follows:

1) **set_identifier**: This function allows the user to upload the data identifiers securely over the smartcontract.

Algorithm 2: set_authorization(*SP_address*, *status)*

```
if msg.sender == owner then
    map status to SP_address;
    return true;
else
    return false;
end
```

Algorithm 3: get_identifier()

```
if Status of msg.sender is trusted then
    get data_identifier, get data_key and TE_url using
      msg.sender address
    returndata_id, data_key, TE_url;
else
    return false;
end
```

Algorithm 4: destroy_smartcontract()

```
if msg.sender == owner then
    destroy owner's smartcontract
    return true;
else
    return false;
end
```

Algorithm 5: pause_smartcontract()

```
if msg.sender == owner then
    Pause owner's smartcontract
    return true;
else
    return false;
end
```

The EOA address of the service provider is sent as an argument of the function along with the data identifier, encrypted key, and URL of the trusted executor. The address is mapped to the data identifier, encrypted key, and URL of the trusted executor to be stored over the smartcontract. The function allows only the owner of smartcontract to upload the data identifier. The same function can be used to change the data identifier if the user's data is updated.

2) **set_authorization**: This function allows the user to maintain a list of trusted service providers. Only the owner of the smartcontract can call this function. Its arguments require the EOA address of the service provider and the status, indicating whether it is trusted or untrusted. This function can also be used to change the status of the service provider.
3) **get_identifier**: This function is used by the service provider to retrieve data identifier. When the service provider calls it, its EOA address is recorded. Using the address, a check is performed whether the service provider belongs to the list of trusted service providers. If the service provider is listed as trusted, the function will return the corresponding data identifier, encrypted key, and URL of the trusted executor. If the address is indicated as untrusted or not listed, the requester will be notified that it does not have the authority to access the user's data identifier.
4) **destroy_smartcontract**: the owner is able to destroy the smartcontract. Once the smartcontract is destroyed its functions can never be called again.
5) **pause_smartcontract**: the owner is able to pause and unpause the contract.

A sample code of PETchain **set_identifier** and **get_identifier** function are provided as follows.

```
function set_identifier(bytes32 _dataid, address
    _address,  bytes32 _key, string memory _url)
   public
        {
        require (msg.sender == owner,"You are not
   the owner");
          keymapping[_address]=_key;
         urlmapping[_address]=_url;
         addressmapping[_address]=_dataid;
        }
```

```
        function get_identifier() public view
   returns(bytes32, bytes32, string memory)
        {
        require(paused == false,"contract is
   paused by owner");
        require (msg.sender == owner || keccak256(
   bytes(authmapping[msg.sender])) == keccak256("
   trusted"),"policy doesnot allow to access");
        return(addressmapping[msg.sender],
   keymapping[msg.sender],urlmapping[msg.sender])
   ;
        }
```

V. IMPLEMENTATION

In this section, we describe how we implemented PETchain on a consortium blockchain to analyze its feasibility and performance. A consortium blockchain is more suitable in terms of ensuring privacy bcause the information contained in the blocks is only visible to the members of the consortium and not to the entire public. To implement our solution we deployed an Ethereum PoA blockchain consisting of five nodes and executed PETchain smartcontract over it. Currently, Ethereum allows two consensus protocols: PoW and PoA. As we choose to implement PETchain over a consortium blockchain we selected the PoA consensus. We performed various experiments to select the appropriate configuration to achieve the best possible performance. The three critical parameters for this PoA blockchain are:

1) Sealers: Sealers are nodes that have the authority to validate a transaction and include it on a block. A block can only be added to the blockchain if it is validated by at least 51% of the sealers present in the network. Initially, sealers are configured in the blockchain genesis block. New sealers can be added anytime if 51% of the sealers present in the network allow it.

2) Block-time: Block-time is the time between two consecutive blocks. It is configured inside the blockchain genesis block. After each interval of block-time, a new block is added to the blockchain. However, the actual block-time always varies from the configured block time due to various delays caused by the network during synchronization.
3) Block-gas-limit: Block-gas-limit is the maximum amount of gas that can be collected from transactions in each block. It is set in *wei* where one *wei* is equivalent to $10^{-}18$ ETH. The gas limit determines the size of the block and the number of transactions that can fit inside a block, which is dependent on the GAS cost of each transaction.

A. EXPERIMENTAL SETUP

In our experimental setup, we used the AWS cloud service to deploy five EC2 virtual machines. We ran Ubuntu Server 18.04 LTS (HVM) having 1 GB RAM and 20 GB storage in each virtual machine. The virtual machines were configured to run a Go Ethereum (geth[4]) client to initialize Ethereum nodes. For each node, at least one externally owned account was created using a public-private key pair. The public key is further used to generate a 20-byte public address. For example Alice creates an externally owned account and has a public address of "$0 \times 1f4993692CDaD87DEc7227650B7aA557EcE6E5b1$". She can further use this address to deploy her smartcontract and sends transactions.

The complete code of PETchain smartcontract is available in the github repository.[5]

To generate a genesis block for our blockchain we used Puppeth.[6] The genesis consists of a simple JSON file that contains the configuration parameters and thresholds. Listing 1 shows a sample genesis file that consists of all necessary parameters required to set up a PoA blockchain. The algorithm used for PoA consensus in Ethereum is Clique. The *chainid* for a Clique network can be any number other than 1,2,3,4,42 and 62 which are associated with the main and test networks. The *difficulty* field is set to zero in clique as no mining is required. The *gasLimit* field is used to set the block-gas-limit whereas the *Epoch Period* is for configuring the block-time. The *extraData* field is used to add the addresses of accounts that are selected as sealers at blockchain initialization.

We used geth to initialize a node in each virtual machine. Each node is configured with the same genesis block. To initialize the blockchain the enode address of nodes are provided. Enode address is a URL that consists of the 512 bit public key and the IP address of nodes. This allows nodes to discover their peers. After initializing the blockchain we

```
{
  "config": {
    "chainId": 2006,
    "clique": {
      "period": 5,
      "epoch": 30000
    }
  },
  "nonce": "0x0",
  "timestamp": "0x5f195492",
  "extraData": "da84a025a783dd6c97
  df6208f4e33ece781847fb",
  "gasLimit": "0x47b760",
  "difficulty": "0x0",
  "coinbase": "0x0000000000000000
  0000000000000000000000",
  "alloc": {
    "da84a025a783dd6c97df6208f4e33
    ece781847fb": {
      "balance": "0x20000000000000
      0000000000000000000000000000
      00000000000000000000"
    }
  },
  "number": "0x0",
  "gasUsed": "0x0",
  "parentHash": "0x000000000000000
  0000000000000000000000000000000
  00000000000000000"
}
```

Listing 1. JSON genesis.

used Remix [7] and Metamask [8] wallet to deploy PETchain contracts. The EOA generated using geth were imported to the Metamask to deploy smartcontract. The remix is linked to Metamask using injected web3 provider whereas the Metamask wallet is connected to our clique network using RPC URL.

B. PERFORMANCE EVALUATION

The performance of PETchain over Ethereum blockchain is analyzed using the following metrics:

1) **Transaction GAS Cost** (*TGS*)**:** The GAS required to execute a smartcontract transaction in the network.
2) **Transaction Per Second** (*TPS*)**:** The number of transactions that can be executed in the network in each second.
3) **Lost blocks:** The number of blocks lost due to the delay caused in broadcasting the signed block by the sealer.

[4]https://github.com/ethereum/go-ethereum/wiki/geth
[5]https://github.com/ibrahimtariqjaved/PETchain
[6]https://github.com/puppeth?q=&type=&language
[7]https://remix.ethereum.org/
[8]https://metamask.io/ wallet

TABLE 2. Transaction GAS cost for PETchain.

No	Contract Transaction	TGS
1	**PETchain deployment**	1296004
2	**set_identifier**	90819
3	**set_authorization**	46655
4	**get_identifier**	30299
5	**pause_smartcontract**	28508
6	**destroy_smartcontract**	14134

4) **Propagation Delay:** The amount of time it takes to propagate a block to the entire network.

In the PoA network, users do not have to pay miners the cost of executing their transactions. However, *TGS* remains a good metric to check the computational efficiency of a smartcontract. Lower *TGS* implies that less computational power is required for execution which enhances the overall throughput of the network. We computed *TGS* required for the completion of different transactions over PETchain. This includes the deployment of smartcontract and executing functions such as set_identifier, set_authorization, get_identifier, pause_smartcontract, destroy_smartcontract. We made efforts to keep the GAS cost as low as possible. We were able to decrease the GAS cost substantially by using mapping instead of arrays to store information [34]. Table 2 enumerates the amount of GAS required to complete different transactions over the blockchain. We can observe that a user requires approximately 1300, 000 GAS to deploy a PETchain smartcontract over the Ethereum PoA network. This cost is required only once when the smartcontract is deployed. On the other hand, the functions require very little GAS. This shows the computational efficiency of our smartcontract. The set_identifier function requires the highest GAS, around 90, 000, as it maps three different values including data_key, data_id, and TE_URL to SP_address as presented in Algorithm 1.

The public Ethereum network supports $12 - 15$ *TPS* [35]. However, with the consortium network we were able to achieve a much higher *TPS*. We ran our blockchain for one hour to compute *TPS* using the following equation:

$$TPS = \frac{Gas\ Limit}{TGS * Block - time}$$

where block-gas-limit and block-time are configured in the genesis file. However, the actual block-time varies due to network delays and synchronization issues. Therefore, we measured the average block-time by running the blockchain for one hour. The measured vs. configured block-time is presented in Figure 7. The measured block-time is then used to compute the *TPS* for three different block-gas-limits (400000, 1000000and 2000000) as shown in Figure 8. It can be observed that high *TPS* is achieved with increased block-gas-limit. This is because more transactions can be accommodated into a block by having a high block-gas-limit. On the other hand, the *TPS* decreases rapidly by increasing the block-time. To achieve a high *TPS* a low block-time and high block-gas-limit can be selected. However, selecting an appropriate block-time is crucial as it causes delays in the network as seen in further experiments.

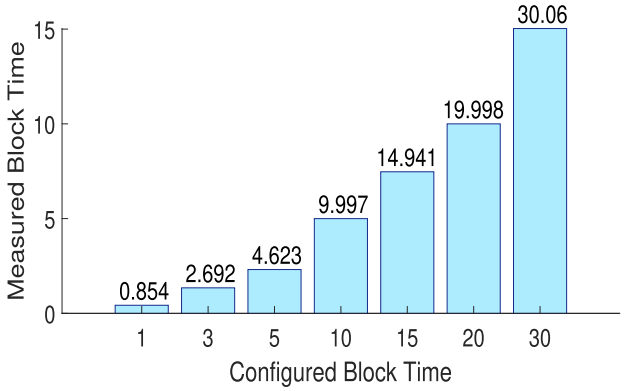
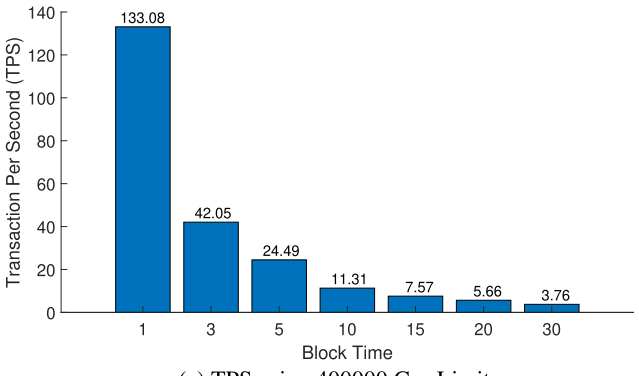
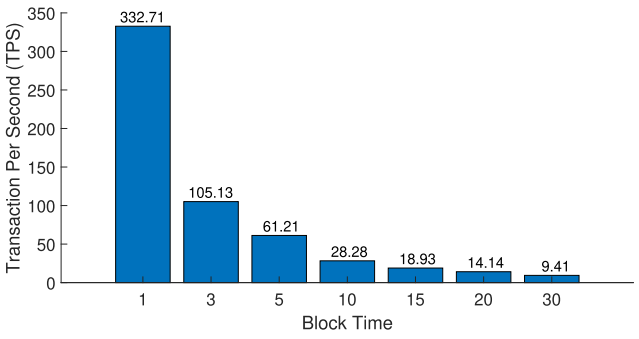


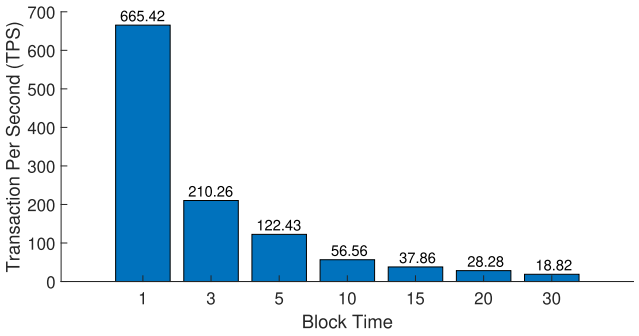
Figure 7. Measured vs. Configured block time.



(a) TPS using 400000 Gas Limit



(b) TPS using 10000000 Gas Limit



(c) TPS using 20000000 Gas Limit

Figure 8. Transaction Per Second (TPS) as a function of block-time.

When a sealer delays its signed block in broadcasting, the backup sealers propose a new block. This causes the block to be lost. The number of lost blocks causes a delay in adding

TABLE 3. Lost blocks in terms of a) Blocktime and b) Sealers.

Block-Time	Lost Blocks
1	845
3	269
5	45
10	17
15	23
20s	3
30s	10

(a) Block-time

Sealers	Lost Blocks
1	0
2	0
3	45
4	32
5	67

(b) Sealers

TABLE 4. Propagation time in terms of a) Blocktime and b) Sealers.

Block-Time	Propagation Time (μ sec)
1	327.84
3	352.54
5	392.74
10	254.32
15	321.1
20	271.2
30	302.28

(a) Effect of Block-time on Propagation Time

Sealers	Propagation Time (μ sec)
1	250.86
2	280.64
3	392.74
4	515.98
5	632.74

(b) Lost Blocks

new blocks to the blockchain, that affects the overall latency of the network. Therefore, we tried to observe the amount of lost blocks in a one hour period concerning block-time and sealers in the network. Table 3a shows the effect of block-time on lost blocks. We observed that lost blocks have a direct relationship with the block-time. A sharp decrease in lost blocks is observed when the block-time is increased. This is because the sealer gets more time to receive, validate, and sign the transactions inside a block. Therefore, for PETchain it is appropriate to select a block-time between $5-10$ to reduce the number of lost blocks. Moreover, Table 3b shows the effect of the number of sealers over lost blocks. We observed 0 lost blocks for 1 and 2 sealers. However, as the number of sealers increases the number of lost blocks also increases. This is due to the synchronization issues between sealers, as at-least 51% sealers need to verify the block before it can be part of the blockchain. Therefore we recommend not to use more than half of the nodes as sealers in the network.

We further calculated the propagation delay in the network. This represents the time required for a block to be available to each node in the network. We present the effect of block-time and sealer to propagation delay in Figure 4. From Table 4a we observe that the propagation delay is strongly dependent on the number of sealers. More synchronization issues were observed when the number of sealers were added to the network, and this was the reason behind higher propagation delay. The propagation delay in the 5 node network having 1 sealer was observed to be 250.86. By increasing the sealers to 5 the propagation delay increased to 632.74. However, we could not observe any relationship between block-time and propagation delay as seen in 4b.

The performance analysis allowed us to characterize PETchain at different configuration settings. We used different amounts of sealers, block-time, and block-gas-limit to see the effect on our performance metrics. We observed that high *TPS* can be simply be achieved by lowering the block-time. However, block-time was strongly related to the number of lost blocks, which affects the overall performance of the network. Therefore, we propose to use a block-time of around 10 seconds. This allows to achieve a high *TPS* and minimize the delays in the network caused by lost blocks. We were able to achieve a *TPS* of around 60 for a gas limit of 20000000. This is almost four times higher than that of the public Ethereum network [35]. To minimize the propagation delay we propose to use only half of the nodes as sealers. Thus, in a 5 node network no more than $2-3$ sealers should be used.

At last we present a comparison of existing Privacy enhancing techniques with PETchain as shown in Table 5. The comparison is conducted concerning features including, data utility, accuracy, complexity, and overheads. Anonymization, differential privacy, and data summarization ensure privacy by reducing the originality of the data. Therefore, these techniques damage the utility and usefulness of the data. Moreover, they cannot guarantee the accuracy of results of data analytic applied. On the other hand, homomorphic encryption, decentralized learning, and PETchain maintain data utility and accuracy of results. However, homomorphic encryption techniques are fairly complex to implement and present a very high computational overhead since operations

TABLE 5. Comparison of privacy enhancing technologies.

Privacy Enhancing Technologies (PET)	Damage to Data Utility	Accuracy of Results	Very complex to apply	High computational overhead	Overburdens user device
Anonymization	Yes	No	No	No	No
Differential Privacy	Yes	No	No	No	No
Data summarization	Yes	No	No	No	No
Homomorphic Encryption	No	Yes	Yes	Yes	No
Decentralized Learning	No	No	No	Yes	Yes
PETchain	No	Yes	No	No	No

are conducted over encrypted data. Decentralized learning also presents high computational overhead as machine learning algorithms are applied on individual devices in a distributed manner before the results can be used by the system model. Decentralized learning also overburdens user devices as computations are offloaded and conducted over the user device. On the contrary, PETchain neither overburdens user device nor has a high computational overhead. However, proper experimental analysis is required to compare each PET in terms of actual latency observed when a particular operation is applied over data.

VI. CONCLUSION

In this paper, we proposed a novel privacy enhancing technology based on blockchain to manage and exploit user data. The proposed technique aims to address user privacy holistically. In our approach, users can define their access control policy by deploying their smartcontract. Users upload encrypted data to IPFS and store its hash to their smartcontract. The authorized service providers can access the hash but can only decrypt and process the data in an isolated execution environment. This allows service providers to process user data without acquiring nor misusing the data. To the best of our knowledge, this is the first-ever implementation of PET using blockchain technology that holistically addresses privacy. In our approach, the users select a trusted execution environment where their data can be processed in a privacy enabled manner. Moreover, they can store their data independently in a distributed manner using IPFS. The proposed solution has been implemented on a consortium blockchain due to its privacy, cost, and performance advantages over the public blockchain. The performance of PETchain is analyzed using the Ethereum platform. It was observed that a high TPS can be achieved by having low block-time and a high gas-limit. However, lower block-time was observed to have a high amount of lost blocks, decreasing the performance of the blockchain. Therefore, a bock-time of 10 is proposed that allows a TPS of around 60. Moreover, a lower number of sealers is recommended, to reduce propagation delay in the network. For our future work, we aim to analyze and improve PETchain by checking its compatibility with GDPR.

ACKNOWLEDGMENT

The authors would like to acknowledge Shaqra University, Saudi Arabia, for their support in this work.

REFERENCES

[1] S.-C. Cha, T.-Y. Hsu, Y. Xiang, and K.-H. Yeh, "Privacy enhancing technologies in the Internet of Things: Perspectives and challenges," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2159–2187, Apr. 2019.

[2] P. Voigt and A. Von dem Bussche, "The EU general data protection regulation (GDPR)," in *A Practical Guide*, 1st ed. Cham, Switzerland: Springer, 2017.

[3] H. Zhou and G. Wornell, "Efficient homomorphic encryption on integer vectors and its applications," in *Proc. Inf. Theory Appl. Workshop (ITA)*, Feb. 2014, pp. 1–9.

[4] L. Sweeney, "K-anonymity: A model for protecting privacy," *Int. J. Uncertainty, Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 557–570, Oct. 2002, doi: 10.1142/S0218488502001648.

[5] *Privacy by Blockchain Design: A Standardised Model for Processing Personal Data Using Blockchain Technology*, Standard DIN ISO 4997:2020-04, 2020. [Online]. Available: https://www.beuth.de/de/technische-regel/din-spec-4997/321277504

[6] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in Internet-of-Things," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1250–1258, Oct. 2017.

[7] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, "L-diversity: Privacy beyond k-anonymity," in *Proc. 22nd Int. Conf. Data Eng. (ICDE)*, Mar. 2006, p. 24.

[8] N. Li, T. Li, and S. Venkatasubramanian, "T-closeness: Privacy beyond k-anonymity and l-diversity," in *Proc. IEEE 23rd Int. Conf. Data Eng.*, Apr. 2007, pp. 106–115.

[9] A.-E.-E.-A. Hussien, N. Hamza, and H. A. Hefny, "Attacks on anonymization-based privacy-preserving: A survey for data mining and data publishing," *J. Inf. Secur.*, vol. 4, no. 2, pp. 101–112, 2013.

[10] B. Jeon, S. M. Ferdous, M. R. Rahman, and A. Walid, "Privacy-preserving decentralized aggregation for federated learning," 2020, *arXiv:2012.07183*. [Online]. Available: https://arxiv.org/abs/2012.07183

[11] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, nos. 3–4, pp. 211–407, 2014.

[12] M. U. Hassan, M. H. Rehmani, and J. Chen, "Differential privacy in blockchain technology: A futuristic approach," *J. Parallel Distrib. Comput.*, vol. 145, pp. 50–74, Nov. 2019.

[13] E. ElSalamouny and S. Gambs, "Differential privacy models for location-based services," *Trans. Data Privacy*, vol. 9, no. 1, pp. 15–48, Apr. 2016.

[14] B. Mirzasoleiman, M. Zadimoghaddam, and A. Karbasi, "Fast distributed submodular cover: Public-private data summarization," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, in Neural Information Processing Systems. Red Hook, NY, USA: Curran Associates, 2016, pp. 3601–3609.

[15] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," White Paper, 2008, vol. 4. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[16] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "FairAccess: A new blockchain-based access control framework for the Internet of Things: FairAccess: A new access control framework for IoT," *Secur. Commun. Netw.*, vol. 9, no. 18, pp. 5943–5964, Dec. 2016.

[17] S. Wang, X. Wang, and Y. Zhang, "A secure cloud storage framework with access control based on blockchain," *IEEE Access*, vol. 7, pp. 112713–112725, 2019.

[18] M. U. Rahman, B. Guidi, F. Baiardi, and L. Ricci, "Context-aware and dynamic role-based access control using blockchain," in *Advanced Information Networking and Applications*, L. Barolli, F. Amato, F. Moscato, T. Enokido, and M. Takizawa, Eds. Cham, Switzerland: Springer, 2020, pp. 1449–1460.

[19] M. U. Rahman, F. Baiardi, B. Guidi, and L. Ricci, "Protecting personal data using smart contracts," in *Internet and Distributed Computing Systems*, R. Montella, A. Ciaramella, G. Fortino, A. Guerrieri, and A. Liotta, Eds. Cham, Switzerland: Springer, 2019, pp. 21–32.

[20] N. B. Truong, K. Sun, G. M. Lee, and Y. Guo, "GDPR-compliant personal data management: A blockchain-based solution," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1746–1761, 2020.

[21] R. Li, T. Song, B. Mei, H. Li, X. Cheng, and L. Sun, "Blockchain for large-scale Internet of Things data storage and protection," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 762–771, Sep. 2019.

[22] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *Proc. IEEE Secur. Privacy Workshops*, May 2015, pp. 180–184.

[23] A. Lahbib, K. Toumi, A. Laouiti, and S. Martin, "DRMF: A distributed resource management framework for industry 4.0 environments," in *Proc. IEEE 18th Int. Symp. Netw. Comput. Appl. (NCA)*, Sep. 2019, pp. 1–9.

[24] B. Alamri, I. T. Javed, and T. Margaria, "Preserving patients' privacy in medical IoT using blockchain," in *Edge Computing*. New York, NY, USA: Springer, 2020, pp. 103–110.

[25] M. Belotti, N. Božić, G. Pujolle, and S. Secci, "A vademecum on blockchain technologies: When, which, and how," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3796–3838, Jul. 2019.

[26] T. Salman, M. Zolanvari, A. Erbad, R. Jain, and M. Samaka, "Security services using blockchains: A state of the art survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 858–880, 1st Quart., 2019.

[27] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, "A taxonomy of blockchain-based systems for architecture design," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Apr. 2017, pp. 243–252.

[28] O. Dib, K.-L. Brousmiche, A. Durand, E. Thea, and B. Hamida, "Consortium blockchains: Overview, applications and challenges," *Int. J. Adv. Telecommun.*, vol. 11, nos. 1–2, pp. 51–64, 2018.
[29] D Team. *Blockchains Tutorials*. Accessed: Aug. 10, 2020. [Online]. Available: https://data-flair.training/blogs/types-of-blockchain/
[30] A. Ellervee, R. Matulevi, and N. Mayer, "A comprehensive reference model for blockchain-based distributed ledger technology," in *Proc. CEUR Workshop*, 1979, pp. 320–333.
[31] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, in Computer and Communications Security. New York, NY, USA: Association for Computing Machinery, 2016, pp. 3–16, doi: 10.1145/2976749.2978341.
[32] J. Benet, "IPFS-content addressed, versioned, P2P file system," 2014, *arXiv:1407.3561*. [Online]. Available: https://arXiv.org/abs/1407.3561
[33] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A practical library OS for unmodified applications on SGX," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, Santa Clara, CA, USA, Jul. 2017, pp. 645–658. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation
[34] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, Apr. 2014.
[35] *The Ethereum Blockchain Explorer*. Accessed: Feb. 1, 2021. [Online]. Available: https://etherscan.io/

IBRAHIM TARIQ JAVED received the Ph.D. degree in computer science and telecommunication from the Institut Mines-Telecom, Telecom SudParis, France. He is currently working as a Postdoctoral Researcher with the Lero—Science Foundation Ireland Research Centre for Software, University of Limerick, Ireland. He is also an Assistant Professor with the Department of Computer Science, Bahria University, Pakistan. He has several years of research and teaching experience at various academic and research institutes. His research interests include blockchain, privacy, identity and trust management, peer-to-peer communication, and nano networks.

FARES ALHARBI received the Ph.D. degree in computer systems engineering from the Queensland University of Technology, Brisbane, QLD, Australia, in 2019. He is currently an Assistant Professor with Shaqra University, Riyadh, Saudi Arabia. His research interests include network management, cloud computing, data center optimization, data management, and resource scheduling of distributed systems, and identity management and privacy.

TIZIANA MARGARIA is currently the Chair of Software Engineering and the Head of the Department of Computer Science and Information Systems, University of Limerick. She also heads the Lero Committee on International Relations Development. In Lero, she heads research projects on scientific workflows, in particular for data analytics, on model-driven service-oriented software design for evolving systems, and holistic HW/SW cyber-security. She is a Fellow of the Irish Computer Society and the Society for Design and Process Science (SDPS). She is a Vice President of the European Association of Software Science and Technology (EASST), a Past President of the ERCIM Working Group on Formal Methods for Industrial Critical Systems (FMICS), a Steering Committee Member of ETAPS and the European Joint Conferences on Theory and Practice of Software, a Managing Editor of STTT, the *International Journal on Software Tools for Technology Transfer* (Springer), and a Co-Founder of TACAS and ISoLA series of conferences. In the European Society for Emergency Medicine (EuSEM), she co-chairs the Special Interest Group on Technology and Processes of Care in the Emergency Care (SIG-TPCEC).

NOEL CRESPI received the master's degree from the University of Orsay (Paris 11) and the University of Kent, U.K., the Diplome d'ingenieur degree from Telecom ParisTech, and the Ph.D. and Habilitation degrees from Paris VI University (Paris-Sorbonne). Since 1993, he has been with CLIP, Bouygues Telecom, and then at Orange Labs, in 1995. He took leading roles in the creation of new services with the successful conception and launch of Orange prepaid service, and in standardization (from rapporteur ship of IN standard to coordination of all mobile standards activities for Orange). In 1999, he joined Nortel Networks as a Telephony Program Manager, architecting core network products for EMEA region. In 2002, he joined the Institut Mines-Telecom, where he is currently a Professor and the Program Director leading the Service Architecture Laboratory. He coordinates the standardization activities for Institut Mines-Telecom at ITU-T, ETSI, and 3GPP. He is also an Adjunct Professor with KAIST, an Affiliate Professor with Concordia University, and a Guest Researcher with the University of Goettingen. He is the Scientific Director of the French-Korean Laboratory ILLUMINE. His current research interests include service architectures, softwarization, social networks, and the Internet of Things/services.

KASHIF NASEER QURESHI received the Ph.D. degree in computer communications (networks) from Universiti Teknologi Malaysia, in 2016. He is currently working as an Associate Professor with the Department of Computer Science, Bahria University, Islamabad. He is an Experienced Cybersecurity and Information Technology Researcher with more than ten years of experience at leading academic institutes in Malaysia and Pakistan. He has worked as a Key Researcher on several projects sponsored by the Ministry of Education Malaysia (MOE) and the Higher Education Commission Pakistan. His current research interests include information security, network, and wireless communication, and the Internet of Things.