DePTVM: Decentralized Pseudonym and Trust Value Management for Integrated Networks

Gao Liu, Zheng Yan, *Senior Member, IEEE*, Dongliang Wang, Haiguang Wang, *Senior Member, IEEE*, and Tieyan Li

***Abstract*—Evaluating and sharing user equipment (UE) trust across multiple network domains can greatly support security and trust management of future integrated heterogeneous networks. But the dilemma between identity privacy preservation and trust evaluation efficacy causes a big challenge in pseudonym and trust value management. Most existing approaches either rely on a trusted third party (TTP) and non-collusive parties, or deploy trusted execution environments (TEEs). They cannot be applied directly into a trustless heterogeneous network environment, where network domains do not trust with each other and it is hard to setup a fully trusted party. In this article, we propose DePTVM, a decentralized pseudonym and trust value management scheme for integrated heterogeneous networks, where different network operators jointly maintain a list of <pseudonym, trust value> pairs by employing verifiable shuffling and trust obfuscation based on blockchain in order to support anonymous trust evaluation and ensure pseudonym unlinkability. We analyze DePTVM with respect to correctness, unforgeability, anonymity and unlinkability, and evaluate its performance through simulations. Experimental results show that trust synchronization can be achieved across domains within 9 seconds with our experimental settings and the time taken by the most complex operation (i.e., verifiable shuffling) of operator agent increases linearly with the scale of maintained list. Analysis and experimental results imply DePTVM's potential in practical applications.**

***Index Terms*—Blockchain, identity privacy, integrated networks, trust value.**

I. Introduction

In 2022, the number of global mobile subscribers reaches 5.7 billion, where the number of 5 G subscribers exceeds 7 hundreds of millions. Beyond 5 G, 6 G is perceived as a large-scale heterogeneous network (LS-HetNet) [1]. It integrates different types of networks, namely mobile cellular networks, the Internet, satellite networks, marine networks and others to form a terrestrial-satellite-marine integrated heterogeneous network with global coverage. In such a network, multiple network domains enabled by different networking technologies do not trust each other. How to ensure network security and credibility, as well as user equipment (UE) privacy is a key issue to drive its wide usage. In LS-HetNet, serving networks need to authenticate accessing UEs before offering services. Although many cryptographic authentication techniques have been proposed to simultaneously ensure authentication and identity privacy [2], e.g., a root-key based method adopted by the 5 G standard, UE is only authenticated as trusted or distrusted. They cannot carry out fine-grained trust management and control.

As specified in international telecommunication union-telecommunication standardization sector (ITU-T), trust is expected to be embedded into 6G [3], which is defined as the willingness to accept a risk of an interaction [4], [5], [6], [7]. In 6 G, network access is expected to be offered anytime and anywhere in a seamless way even during UE roaming. Network access control on UE relies on its trustworthiness. According to the trust value of UE, defense can be arranged and relative services can be offered. This requires network operators to conduct comprehensive evaluation on UE trust in a cooperative way. Therefore, evaluating and sharing UEs' trust values in inter-domain networking becomes crucially important for managing the trust of integrated heterogeneous networks.

In a trust evaluation system, the network activities of a UE binding to its identity can be monitored for assessing its trust, but UE privacy could be impacted when UE uses a long-term identity [8]. Normally, UE pseudonym is applied and should be frequently updated to prevent an attacker from tracking UE activities. However, this update makes accurate trust evaluation hard to be achieved, and trust value linkage before and after pseudonym change could break pseudonym unlinkability, making pseudonym update useless [9]. Existing schemes rely on a trusted third party (TTP) [10], [11], [12], TEEs [9] or non-collusive parties [13] to overcome the dilemma of trust evaluation and pseudonym update. However, TTP and non-collusive parties may not exist in reality, and TEEs could suffer from some potential security vulnerabilities, e.g., unreasonable trust endorsement (namely trusting TEE providers). In addition, previous schemes focus on a single type of networks and fail to support integrated heterogeneous networks.

Actually, we are facing a number of challenges to manage UE pseudonyms and trust values in a trustless networking environment. First, deploying a TTP is not feasible in practice, since it

Manuscript received 26 June 2022; revised 7 January 2023; accepted 16 February 2023. Date of publication 20 February 2023; date of current version 12 January 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62072351; in part by the Key Research Project of Shaanxi Natural Science Foundation under Grant 2023-JC-ZD-35; in part by the open research project of ZheJiang Lab under Grant 2021PD0AB01; and in part by the 111 Project under Grant B16037. *(Corresponding author: Zheng Yan.)*

Gao Liu and Zheng Yan are with the State Key Laboratory on Integrated Services Networks and the School of Cyber Engineering, Xidian University, Xi'an 710126, China (e-mail: gaoliu9865@gmail.com; zhengyan.pz@gmail.com).

Dongliang Wang is with the School of Cyber Engineering, Xidian University, Xi'an 710126, China (e-mail: dlwang_1@stu.xidian.edu.cn).

Haiguang Wang and Tieyan Li are with the Huawei Technologies Co. Ltd., Singapore 138588 (e-mail: wang.haiguang.shieldlab@huawei.com; li.tieyan@huawei.com).

Digital Object Identifier 10.1109/TDSC.2023.3246799

is hard to find such a party that can be fully trusted by all network domains run by different operators. Second, TEE-capable devices may not be available anywhere for anonymous trust evaluation and management in inter-domain networking. Third, an attacker could analyze trust values or their ciphertexts to link UE pseudonyms in such schemes that are based on non-collusive parties [13]. Thus, most of existing works [9], [10], [13] cannot be directly applied into the integrated heterogeneous networks to satisfy practical requirements. In addition, trust values evaluated by different operators using different trust-related data are hard to be shared and synchronized across domains. Thus, how to efficiently ensure the consistency of evaluated trust values among domains while preserving UE identity privacy becomes a crucial issue. So far, the literature still lacks an effective scheme.

In this paper, we propose DePTVM, the first decentralized pseudonym and trust value management (DePTVM) scheme based on blockchain in integrated heterogeneous networks by employing verifiable shuffling and trust obfuscation. In order to preserve UE identity privacy, a number of operator agents (OAs) manage a list of UE <pseudonym, trust value > pairs and UE uses its pseudonym for network activities. In order to synchronize and share trust values across domains and ensure anonymity, blockchain consensus is implemented by embedding trust evaluation according to UE's pseudonym and relative behaviors. For solving the dilemma of trust evaluation and pseudonym update, OAs perform obfuscation over the trust values in the list of < pseudonym, trust value> pairs. In addition, they conduct backward and forward verifiable shuffling one by one to update pseudonyms by applying modular exponentiation (that uses UEs' public keys as bases and newly chosen random numbers as exponents) and secretly permuting rows of the list at the same time. Thus, the unlinkability of old and new pseudonyms can be ensured, and UEs' new pseudonyms inherit the obfuscated trust values of old pseudonyms. When UEs use new pseudonyms for networking, their trust can be evaluated and updated accordingly based on the new pseudonyms' inherited trust values and linked behaviors, thus trust evaluation efficacy can be ensured. Specifically, the main contributions of this paper are summarized as below.

1) We propose DePTVM, the first decentralized pseudonym and trust value management scheme based on blockchain to support anonymously evaluating and sharing UE trust across multiple network domains that do not trust with each other.
2) We integrate trust obfuscation with verifiable shuffling to ensure the unlinkability of pseudonyms before and after pseudonym update without relying on any TTPs or TEEs.
3) We formally analyze DePTVM with respect to correctness, unforgeability, anonymity and unlinkability.
4) We simulate DePTVM and conduct a number of experiments to evaluate its execution performance. Comparison with related work shows that DePTVM outperforms other related works.

The remainder of this paper is organized as follows. Section II reviews technical background and highly related work. In Section III, we specify problem statements by describing DePTVM's system model, security model, research assumptions and preliminaries. DePTVM design is described in detail in Section IV, followed by performance analysis and evaluation results in Section V and a discussion on DePTVM limitations in Section VI. Finally, Section VII concludes the whole paper.

TABLE I
COMPARISON OF DePTVM WITH RELATED WORKS

	TTP			TEE		Non-collusive parties	
	[10]	[11]	[12]	[9]	[17]	[13]	DePTVM
UT	●	●	●	●	○	●	●
IT	○	○	○	○	○	○	●
TS	●	●	●	●	●	○	●
Ef	●	●	●	●	●	○	●
Uk	●	●	●	●	-	○	●
De	○	○	○	○	●	●	●
UM	○	○	○	●	-	○	●

UT: UE trust evaluation; IT: inter-domain trust evaluation; TS: trust synchronization; Ef: efficiency; Uk: unlinkability; De: decentralization; UM: UE joining and revocation management; ●: supported; ○: not supported or considered; -: unable to evaluate.

II. BACKGROUND AND RELATED WORK

In this section, we review related works about blockchain technologies, decentralized identity systems, network trust evaluation and pseudonym and trust value management. We also point out their flaws that DePTVM aims to overcome, and compare DePTVM with related works in Table I to show its advance.

A. Background

1) Blockchain Technologies: Blockchain is a popular technique to achieve decentralized computation and storage [14]. It has attracted the attention of both industry and academia, and has many applications, e.g., data sharing. Blockchain's core is its consensus mechanism, which can be classified into three basic categories [9], namely proof of work (PoW), proof of stake (PoS), and byzantine faulty tolerant (BFT). PoW is considered as a meaningless task, wasting many computational resources in order to guarantee security. Meanwhile, it suffers from low efficiency due to long task completion time and low throughput, and faces a risk of centralization due to outsourceable tasks. PoS consumes almost no resources, but a malicious block creator could publish two blocks deliberately, thus causing forking, e.g., nothing-at-stake attack. Most of BFT based consensus mechanisms lack incentives for consensus nodes. In order to address specified problems of basic consensus mechanisms, some consensus mechanisms [14] have evolved from the basic ones. The most representative consensus mechanism appears in [15], [16], [17], which have solved main problems of current consensus mechanisms in an integrated way, namely forking, low efficiency, and centralization.

For consensus, a block creator belonging to a network domain proposes a block that includes computation results, and other consensus nodes should check the correctness of the results (e.g., through re-computation), which suggests the content of block is verified by and shared to consensus nodes located in other domains. In this paper, we deploy a blockchain consensus mechanism as described in our previous work [17] due to its superiority over others, in order to reach consensus on trust evaluation and synchronize trust values across multiple domains in an efficient and secure way.

B. Related Work

1) Decentralized Identity Systems: Decentralized identity systems allow users to manage their own credentials based on their created identifiers or pseudonyms. A common use case is users authorize to release their personal credentials from devices to websites. For instance, Alice releases a digital credential from her university for proving her degree. Some standards and implementations [18], [19] aim to support such a function. However, they cannot support legacy compatibility, Sybil resistance, accountability and key recovery. To overcome the flaws, Maram et al. proposed a practical decentralized identity system called CanDID [20], which constructs users' credentials (including pseudonyms defined by the users themselves) through zero-knowledge proof. In detail, committee nodes verify pre-credentials that the users generate based on data with existing web services, and issue credentials through a threshold signature scheme. However, the assumption that less than 1/3 committee nodes (i.e., OAs in our work) collude is stronger than our assumption that at least one OA does not collude, especially in large-scale integrated networks. During pseudonym modification, each user can participate in pseudonym creation and thus suffer from additional burdens.

In addition, most of decentralized identity systems cannot be directly applied into our working scenario for replacing verifiable shuffling. UEs' pseudonyms should be modified periodically for unlinking new and old pseudonyms, and UEs' trust values are dynamically updated based on UEs' behaviors binding to their pseudonyms. When the UE's pseudonym is modified, it is challenging for a new pseudonym of UE to inherit the trust value of old pseudonym of this UE without breaking unlinkability in most of decentralized identity management systems.

2) Network Trust Evaluation: Trust evaluation models can be classified into three types, namely statistical models, reasoning models and machine learning models [21]. Statistical models usually adopt weighted sum to aggregate trust evidence (i.e., feedback) to generate trust values, e.g., weighted voting. All weighted votes on an entity or an event are summed up, where the weights of votes (i.e., feedback) are determined by the trust degrees of voters [22]. Reasoning models (e.g., subjective logic [21]) define trust as several dimensions, e.g., uncertainty, disbelief and belief. The main idea is to infer the trust value of an entity according to entity correlation and collected data. Machine Learning (ML) models can evaluate trust through classification by using supervised learning (e.g., decision tree), semi-supervised learning, unsupervised learning (e.g., k-means) and reinforcement learning [23]. Such models can help trust relation prediction and attack detection. At the same time, ML models can also be divided into direct and indirect evaluation. In direct evaluation, ML algorithms consider collected trust-related data as input for evaluation. If some essential data are missing, indirect evaluation adopts ML algorithms to preprocess or label data to help evaluation. However, towards network trust evaluation, the prior arts have not yet considered how to overcome the conflict between trust evaluation and privacy preservation without suffering from the single point of failure. Network trust can be evaluated based on the results of intrusion detection [17], which reflects whether a network or node is robust to offer expected services or behave as expected. In order to achieve privacy-preserving decentralized network trust evaluation, a decentralized intrusion detection framework named SeDID [17] was proposed, which is based on security-related data collection [24]. SeDID allows data collection nodes to sense and share security-related data, and then some detection nodes perform intrusion detection and network trust evaluation by analyzing collected data. However, SeDID does not take into account node trust evaluation and anonymous sharing across domains. In addition, it suffers from some flaws caused by adopting TEEs.

3) Pseudonym and Trust Value Management: Most of efforts to solve the conflict between trust evaluation and identity privacy preservation adopt TTP, TEEs, or non-collusive parties.

Some works employ TTP to manage trust values and update pseudonyms while ensuring pseudonym unlinkability. Androulaki et al. proposed a reputation system [10] for anonymous networks. It employs TTP (i.e., bank) to maintain reputation of each peer and address the conflict between trust evaluation and pseudonym unlinkability honestly and correctly. Christin et al. employed blind signature to periodically generate pseudonyms and achieve reputation transfer between old and new pseudonyms with unlinkability [11]. Participatory sensing outsources data collection, which requires privacy preservation and anonymity. Wang et al. employed the blind signature for creating a pseudonym in participatory sensing [12]. However, such blind signature based schemes heavily rely on the assumption that a centralized authority or TTP update pseudonyms and evaluate trust honestly and correctly. In addition, such a TTP cannot be setup in practice, especially in integrated heterogeneous networks.

TEEs can also be applied to solve the conflict. By leveraging Intel Software Guard Extension (SGX), Feng et al. [9] proposed a framework to evaluate and anonymously authenticate trust in a mobile crowdsourcing system. In order to ensure the unlinkability of pseudonyms, an SGX-enabled cloud server is employed to help altering the public/private keys of UEs and mixing new pseudonyms with faked ones. However, this design targets on mobile crowdsourcing, how to make use of it to support integrated heterogeneous networks requires additional investigation. On the other hand, SGX enclaves suffer from some flaws [25]. First, the enclaves are limited to execute large software codes, since SGX supports a limited memory size for the enclaves [26]. Although the execution of large software codes can be achieved by swapping SGX pages out and in, the necessity of integrity and confidentiality of pages causes heavy system burdens [27]. Second, an SGX platform is controlled by its owner, thus the owner can read, modify, block, and replay messages

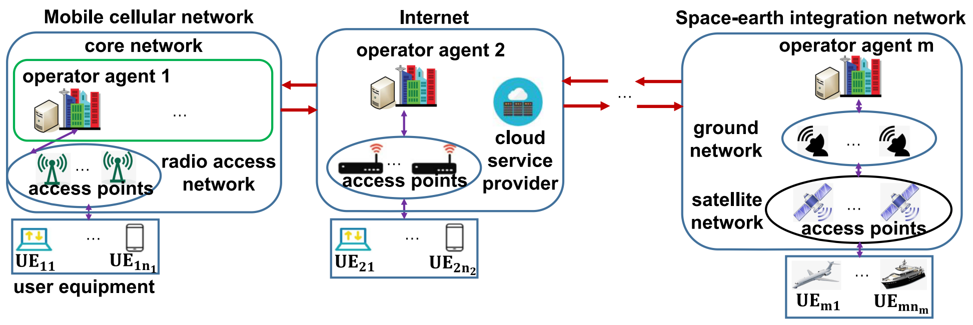


Fig. 1. System model.

from or sent to the enclave. Third, SGX could suffer from side channel attacks [28]. Fourth, if an SGX-enabled scheme relies on the reputation of a single SGX platform, the compromise of this platform could make the whole scheme unavailable. Most importantly, trust endorsement may not be reasonable, namely all entities in an SGX-enabled scheme should trust the SGX provider, which mostly cannot be satisfied in integrated networks run by different operators. Meanwhile, unreasonable trust endorsement makes TEE-based schemes suffer in practice.

Non-collusive parties can be deployed to cooperatively address the conflict by using verifiable shuffling [29], [30]. Zhai et al. [13] adopted the verifiable shuffling to manage a list of <pseudonym, trust value> pairs with such an assumption that several anonymity servers exist and at least one of them refuses to collude. But an attacker might track the pseudonyms of nodes by analyzing related trust values. Even though the authors introduced homomorphic encryption to protect trust values in order to avoid pseudonym tracking by analyzing and inferring distinct trust values, the ciphertexts of trust values could also be linked to tracked node pseudonyms and activities. In other words, this work cannot really ensure the unlinkability of pseudonyms. In addition, the requirement on non-collusive parties might not be rational in a network. After each round of trust evaluation, pseudonyms are updated, which seriously impacts efficiency since forward and backward shuffling, and proof creation and verification take much time. Meanwhile, frequently altering pseudonyms requires UEs to frequently extract their pseudonyms for network activities, which might be too complex for UEs. In integrated networks, some UEs might not carry out activities in some consecutive rounds of trust evaluation, but their trust values should be updated according to time decay. However, the work does not consider this, neither UE joining and revocation.

In summary, we compare DePTVM with the above three classes of related works in Table I in terms of the support on UE trust evaluation, inter-domain trust evaluation, trust synchronization, efficiency, unlinkability, decentralization, and UE joining and revocation management. We find that most existing schemes are not feasible to be directly applied into integrated heterogeneous networks to support inter-domain anonymous trust evaluation with synchronization, efficiency, unlinkability, decentralization, and UE joining and revocation management.

III. Problem Statements

A. System Model

Fig. 1 shows the system model of DePTVM, which mainly involves four types of entities, i.e., UEs, access points, cloud service provider, and operator agents. In integrated networks, each network connects through switches. Roaming UEs connect to serving networks through access points. The access point monitors the behaviors of connected UEs and provides trust related data. According to these data, different operator agents collaboratively evaluate the trust of each UE, in order to achieve trust management and control on UEs. We focus on inter-domain trust evaluation and sharing, and intra-domain one is its specific case (i.e., there is only one domain handled by one operator agent).

User Equipment (UE) uses its pseudonym to connect to access points for networking activities.

Access Points (APs) deployed by network operators sense cross-domain behaviors of connected UEs, and periodically share sensed behaviors (called trust related data) and a signature to a common cloud service provider.

Cloud Service Provider (CSP) offers a common data storage service to collect and verify trust related data from APs of different network domains, and allows the access to data with APs' signatures for inter-domain trust evaluation.

Operator Agent (OA) is deployed by its network operator, which is responsible for evaluating trust of UE based on trust related data from CSP and managing the list of <pseudonym, trust value> pairs of all UEs.

B. Security Model

1) Adversary: UEs might behave maliciously and are not trusted, whose inter-domain trust should be evaluated and shared across domains for assisting other network entities to make decisions. AP is supposed to be semi-trusted, namely it follows a pre-defined protocol but is curious about each UE's sensitive information, e.g., UE's long-term public key and the linkage of pseudonyms or activities. The APs deployed by operators are like switches in software-defined networks and WiFi access points. They could be compromised by attackers and provide tampered or low-quality data. Some existing solutions can be applied to solve this problem. One solution is to remotely

Algorithm 1: Anonymity Server j's Verifiable Shuffling Operation $sh(L_{j-1}, g_{j-1}, e_j, z_j)$.

```
Input: $e_j$; $z_j$: the private key corresponding to $Z_j$; $L_{j-1}$;
    $g_{j-1} = g^{e_1 \cdots e_{j-1}}$ ($g_0 = g$); $pf_{j-1}$
Output: $L_j$; $g_j$; $pf_j$
1 if The received proof $pf_{j-1}$ has been verified as valid then
2     Implement modular exponentiation that considers
        elements in the first column of received list $L_{j-1}$ as
        bases and $e_j$ as exponents, $(g^{x_i e_1 \cdots e_{j-1}})^{e_j} = g^{x_i e_1 \cdots e_j}$ ;
3     Use $z_j$ to decrypt elements in the second column of $L_{j-1}$
        for obtaining $E_{Z_{j+1},R_{j+1}}(\cdots E_{Z_m,R_m}(TV_i))$;
4     Permute rows of the resulting list and create a new list $L_j$;
5     Compute $g_j = {g_{j-1}}^{e_j}$;
6     Create a proof $pf_j$ to attest the above operations are
        correct [13];
7     Return $L_j, g_j, pf_j$.
8 end
```

detect if AP platforms are compromised according to their status information [31]. Thus, the security status of AP platforms can be detected by their local operator agent, even other operator agents. Another possible solution is to directly assess the quality of collected data through data truth discovery methods [32], especially in 5 G ultra-dense networks. In such a scenario, UE's signals could be received by multiple APs, and thus these APs can sense the behaviors of UE and share collected data for assessing the quality of data through data truth discovery [33]. However, how to avoid or detect the misbehaviors of APs is out of the scope of this paper. Herein, based on the advance of current literature, we assume that APs can honestly follow a pre-defined protocol to offer data and the data have been pre-processed with data truth discovery. CSP cannot be fully trusted. It operates based on DePTVM design due to business profits, but it is curious on UEs' sensitive information and could behave maliciously. In DePTVM, CSP stores trust related data from APs and APs' signatures. OAs can verify extracted data with signatures and perform consensus on trust evaluation. CSP cannot forge the stored data and signatures from APs due to the unforgeability of secure digital signature. In addition, inconsistent data provision from CSP to OAs can be easily detected by OAs through monitoring or mutual checking, since these shared data should be signed by CSP. If any CSP's misbehaviors are discovered, its service provision is restricted, which makes CSP lose potential profits. Meanwhile, CSP can be incentivized through rewards to encourage its honest behaviors. Therefore, we assume CSP is semi-trusted (i.e., honest but curious) in terms of storage service provision in this paper. Since CSP may suffer from some single point of failure, external intrusion or internal attacks, we cannot fully depend on CSP to collect data and evaluate UE trust accordingly. That is solely using CSP for trust evaluation is not reliable. Therefore, we deploy CSP to provide storage services and employ blockchain consensus to ensure the correctness and trustworthiness of trust evaluation results. Similar to the assumption in [13], we suppose OA is not trusted by other domains' OAs, and at least one OA does not collude with other OAs. This assumption eliminates the case that all OAs collude to link UEs' $<$pseudonym, trust value $>$ pairs in both new and old destination lists, as well as $<$ public key, trust value ciphertext$>$ pairs in the initial list. Thus, anonymity and pseudonym unlinkability are ensured. In practice, collusion among all OAs is seldom due to business conflict and competition. Thereby, this assumption is reasonable. Note that this assumption is not related to the fault tolerance of a blockchain system.

2) Research Assumptions: Suppose each entity holds a pair of long-term private/public keys by securely registering at OAs. Each UE has its own private/public key pair $(x_i, y_i = g^{x_i})$, and its public key y_i is known by all OAs. Each AP has its private/public key pair for securely sharing its trust related data. Each OA holds its private/public key pair $(x_{OA_j}, y_{OA_j} = g^{x_{OA_j}})$, $j = 1, \ldots, m$, where m is the number of operator agents. OAs can synchronize their time through a handshake protocol and the time of APs can be synchronized with OAs' [34].

C. Preliminaries

1) Cryptographic Problems: We introduce three types of cryptographic problems that are involved in DePTVM.

Discrete Logarithm Problem. Suppose G is a cyclic multiplicative group with an order q, and g is a generator of G. For any given $A \in G$, it is infeasible to extract $a \in Z_q^*$ from $A = g^a$ in polynomial time.

Computational Diffie-Hellman Problem. Given $a, b \in Z_q^*$, extracting g^{ab} from g, g^a and g^b is computationally infeasible.

Decisional Diffie-Hellman Problem. Given g, g^a, g^b and T, an adversary $\mathcal{A}$ cannot determine $T = g^{ab}$, namely $\mathcal{A}$ cannot distinguish g^{ab} from a random number $Rand$ of G. If $|Pr\left[\mathcal{B}\left(g, g^a, g^b, T = g^{ab}\right) = 0\right] - Pr[\mathcal{B}(g, g^a, g^b, T = Rand) = 0]| \geq \varepsilon$ holds, an algorithm $\mathcal{B}$ that outputs a guess $b' \in \{0, 1\}$ has an advantage ε to solve this problem [35].

2) Verifiable Shuffling: Mix-net aims to make communications hard to trace with the help of m anonymity servers (AS) [36]. m anonymity servers take a list L_0 as input, and output a new list L_m, so that the pairs of L_0 cannot be linked to L_m's. Shuffling of mix-net consists of modular exponentiation, decryption and permutation. Fig. 2 gives a mix-net example with 2 anonymity servers and 3 UEs. In the input list L_0, the first column contains UE's long-term public key $y_i = g^{x_i}$, where x_i is UE_i's private key. The second column of the input list L_0 consists of UE_i's trust value ciphertext $E_{Z_1,R_1}(E_{Z_2,R_2}(TV_i))$, which is obtained by encrypting the trust value plaintext TV_i with anonymity servers' public key Z_j and random number R_j. In shuffling, AS_1 performs modular exponentiation that considers UEs' public keys $y_i = g^{x_i}$ in the first column of list L_0 as bases and AS_1's selected random number e_1 as exponent, $g^{x_i e_1}$, decrypts $E_{Z_1,R_1}(E_{Z_2,R_2}(TV_i))$ with its private key z_1 to obtain $E_{Z_2,R_2}(TV_i)$, and permutes rows of the resulting list. Then, it broadcasts its created list to AS_2 for shuffling, and AS_2 broadcasts the output list L_2. Herein, g_1 and g_2 are published by AS_1 and AS_2 for UE_i's pseudonym extraction. As a result, the first column of output list L_2 contains pseudonyms of all UEs, $pk_{\pi(i)} = g^{x_i e_1 e_2}$, and the second column consists of the relative trust value $TV_{\pi(i)}$. $\pi(i)$ is the location of UE_i's $<$pseudonym, trust value$>$ pair in L_2. UE_i can compute its pseudonym $pk_{\pi(i)} = {g_2}^{x_i}$ by using published g_2 and its private key x_i, and confirm $pk_{\pi(i)}$'s existence in L_2.

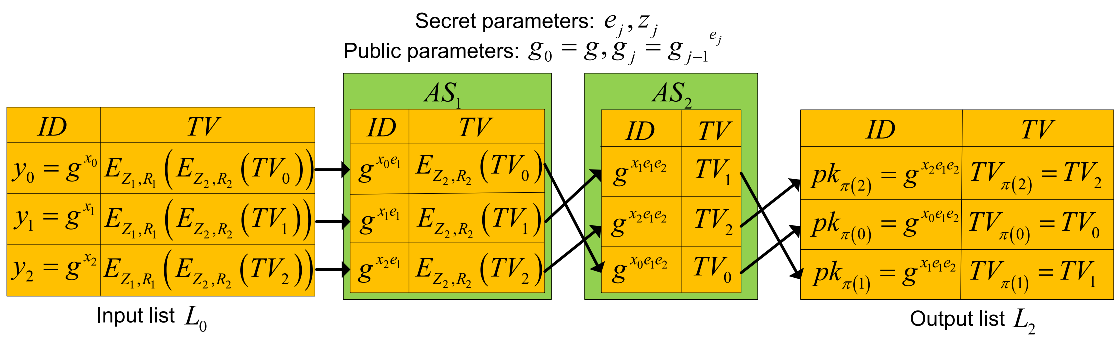


Fig. 2. A mix-net example with 2 anonymity servers and 3 UEs.

Algorithm 2: Block B_K Creation.

```
Input: $Re$: the computation result; $TD_{ti}$ $(ti = 1, \ldots, n_d)$: the data collected by the $ti$-th collection node and used for obtaining $Re$; $n_b$: the referenced number; $n_{bt}$: the threshold number of blocks that a consensus node created within the latest $n_b$ blocks of blockchain; $TAG$: the difficulty; $n_{tr}$: the threshold number of used data; $h_{K-1}$: the hash value of previous block; $(pk, sk)$: the public/private key pair of a block creator; $t$: the timestamp
Output: $B_K$: the $K$-th block; $sig_{sk}(B_K)$: the creator's signature on the content of block
1  if the creator has generated $n_{pk} (n_{pk} < n_{bt} < n_b)$ blocks within the latest $n_b$ blocks of blockchain and obtained the result $Re$ based on $TD_{ti}$ $(ti = 1, \ldots, n_d)$ and $(n_d > n_{tr})$, then
2      Compute $mr$, i.e., the root of Merkle tree constructed with $TD_{ti}$ $(ti = 1, \ldots, n_d)$;
3      for $t$ do
4          Compute a hash value $H_{pk} = H(K||h_{K-1}||mr||Re||pk||t)$;
5          if $H_{pk} \leq TAG \times w(n_b/n_{pk})$ then
6              Insert $K$, $h_{K-1}$, $mr$, $Re$, $pk$ and $t$ into the block $B_K$;
7              Sign on $B_K$ with the private key $sk$, $sig_{sk}(B_K)$;
8              End block creation;
9          end
10     end
11 end
12 Return $B_K$ and $sig_{sk}(B_K)$;
```

In order to guarantee the correctness of shuffling, many verifiable shuffling schemes were proposed [13], [29]. In these schemes, an anonymity server usually should generate a zero-knowledge proof of shuffling such that anyone can check whether shuffling (i.e., modular exponentiation, decryption and permutation) is performed correctly. *Algorithm 1* details AS_j's verifiable shuffling operation, which takes the results from the $(j-1)$-th server as input and outputs results that is also the input of $(j+1)$-th server's shuffling. After AS_j's verifiable shuffling operation, it broadcasts its generated results to AS_{j+1} for verifiable shuffling. The result of AS_j's verifiable shuffling includes a proof pf_j for convincing anyone that shuffling is performed correctly. The proof is constructed according to the specific encryption algorithm such as ElGamal encryption scheme. To be specific, Zhai et al. [13] provided the source code of prototype[1], which has achieved *Algorithm 1* by heavily relying on an open-source Go library of cryptographic primitives that include verifiable Neff shuffling [29], and we can also directly use this library for the proof. If m anonymity servers completed shuffling and proof verification, the element in the first column of the published output list L_m is UE_i's pseudonym, $pk_{\pi(i)} = g^{x_i e_1 \cdots e_m}$, and the element $TV_{\pi(i)}$ in the second column is UE_i's trust value plaintext. UE_i computes its pseudonym $pk_{\pi(i)} = {g_m}^{x_i}$ by using published $g_m = g^{e_1 \cdots e_m}$ and its private key x_i, and uses $pk_{\pi(i)}$ for network activities, since the ElGamal encryption and digital signature can be executed. With respect to ElGamal digital signature, UE_i can use its private key to sign a message and anyone can verify its signature with its pseudonym. To be specific, UE_i can use x_i to sign a message mes with a randomly selected number k, $(r = {g_m}^k, s = (H(mes) + x_i r)k^{-1})$. With UE_i's pseudonym $pk_{\pi(i)}$, anyone can verify the signature by checking ${g_m}^{\frac{H(mes)}{s}} {pk_{\pi(i)}}^{\frac{r}{s}} = r$. In terms of ElGamal encryption, anyone can use UE_i's pseudonym to encrypt a message and UE_i can decrypt the ciphertext by using its private key. An entity computes the ciphertext of mes with a random number k, $(C_1, C_2) = ({g_m}^k, mes * {pk_{\pi(i)}}^k)$. With the private key x_i, UE_i can obtain the plaintext $mes = C_2 * {C_1}^{-x_i}$.

3) Blockchain Consensus: DePTVM applies an efficient and secure consensus mechanism proposed in our previous work [17] to synchronize and share trust values in a trustless context. The consensus mechanism includes block creation, block winner selection and incentive execution. The incentive mechanism can be designed according to specific scenarios. In this paper, operators can set funding to provide awards to participants. Some consensus node proposes a block including its computation results, and other consensus nodes verify the block for reaching consensus on the results. In our scenario, a consensus node can insert its trust evaluation results into a block and publishes this block. If other consensus nodes receive the block, they can check the correctness of the results (e.g., by re-computing). Thus, the process of trust evaluation can be verified in a decentralized and public way, and the computation results confirmed with consensus should be trustworthy.

Herein, we briefly review the process of consensus. A consensus node performs *Algorithm 2* to create a valid block. The block B_K contains K, h_{K-1}, mr, Re, pk and t. If n_d is large, $TD_{ti}, ti = 1, \ldots, n_d$ is suggested to store at the CSP, which can be accessed by consensus nodes in order to reduce their storage burdens. $w(n_b/n_{pk})$ is the weight to adjust the difficulty TAG.

[1]https://github.com/anonyreputation/anonCred

TABLE II
NOTATIONS

Symbols	Descriptions
UE_i	The i-th UE.
(x_i, y_i)	The private/public key pair of UE_i.
OA_j	The j-th operator agent.
(x_{OA_j}, y_{OA_j})	The private/public key pair of OA_j.
n_d	The number of APs whose trust related data are used for evaluation.
TV_i	The trust value of UE_i.
(z_j, Z_j)	The private/public key pair chosen by OA_j.
$E_{Z,R}(mes)$	The ciphertext of the message mes with the public key Z and the random number R.
$sig_{sk}(mes)$	The signature on the message mes with the private key sk.
$\pi(i)$	The location of UE_i's pseudonym in the destination list L_m.
$pk_{\pi(i)}$	UE_i's pseudonym in L_m.
$TV^{old}_{\pi(i)}$	The latest trust value of UE_i appearing in the blockchain.

The larger n_b/n_{pk} is, the bigger the weight, the easier to create a block. When a consensus node succeeds in creating a valid block without receiving other valid blocks, it publishes its created block for other consensus nodes' acceptance. After receiving the block and accessing data from the CSP, a consensus node performs a block verification procedure similar to *Algorithm 2*. It is possible that multiple valid blocks could be created at the same time, and consensus nodes could receive different valid blocks within a short period due to network latency. In order to ensure the block with the highest priority can be received by the majority of consensus nodes, each consensus node sets a time window θ to receive other valid blocks after receiving the first valid block. Through comparison, the block with the earliest t, the smallest n_{pk}, the largest n_d, and the smallest H_{pk} can finally win to become the next block on the blockchain.

IV. DePTVM Design

In this section, we introduce DePTVM's design in detail. We give a design overview and then describe main functions of DePTVM. For simplicity of presentation, we provide the notations used in the rest of this paper in Table II.

A. Overview

DePTVM consists of a number of functions: genesis block generation, trust value update and consensus, and list L_m maintenance, which are run in sequence. The procedure of DePTVM is shown in Fig. 3. In genesis block generation, operator agents of m different domains collaboratively generate a list L_m of n UE <pseudonym, trust value> pairs, and insert this list into the genesis block. UEs can compute and extract their pseudonyms by accessing blockchain, and use these pseudonyms for network activities. In trust value update and consensus, APs monitor their connected UEs, and periodically share collected trust related data to the common CSP (e.g., every 10 minutes). Then, the operator agents perform trust evaluation based on sufficient trust

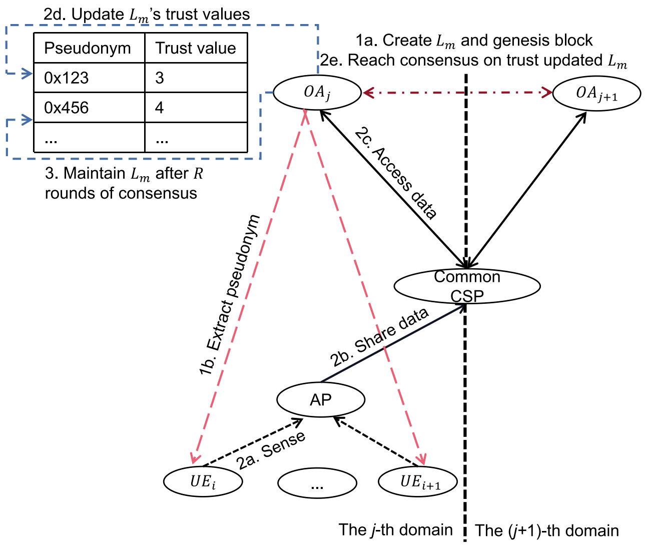


Fig. 3. The procedure of DePTVM.

related data retrieved from CSP, update trust values in L_m, and reach consensus on the trust updated L_m. After R rounds of trust value update and consensus, the agents should maintain L_m for updating pseudonyms with the assurance of pseudonym unlinkability, which is called list L_m maintenance.

DePTVM uses a blockchain as a decentralized ledger to record the list L_m. The blockchain of DePTVM is a kind of consortium blockchain. Operator agents create and publish blocks for consensus in inter-domain networking by themselves. A consensus node (i.e., operator agent) proposes a block that includes a trust updated list for the acceptance of other consensus nodes, which helps reaching consensus on trust evaluation and synchronizing trust values across multiple domains.

In order to save the operator agents' storage space, the blockchain of DePTVM records the root of Merkle tree generated with trust related data collected by APs, instead of recording these data directly. CSP provides the storage space to store all the trust related data. DePTVM adopts trust obfuscation and verifiable shuffling to solve the conflict of trust evaluation and pseudonym update. It allows operator agents to maintain the list L_m of <pseudonym, trust value> pairs. The agents first obfuscate trust values in L_m. Then, they cooperatively perform backward and forward shuffling one by one to update pseudonyms by executing modular exponentiation (that regards UEs' public keys as bases and newly chosen random numbers as exponents) and simultaneously permuting rows of the list secretly. Therefore, pseudonym unlinakbility is guaranteed, and the new pseudonyms of UEs inherit the obfuscated trust values of old ones. If UEs adopt new pseudonyms for network activities, the new pseudonyms' inherited trust values and linked behaviors can be used to compute a new trust value, thus ensuring the efficacy of evaluation.

B. Genesis Block Generation

Initial List L_0 Creation. m OAs create the list of <pseudonym, trust value> pairs involving a total of n UEs with $(x_i, y_i = g^{x_i})$, $i = 1, \ldots, n$. m OAs collaboratively use their selected public key Z_j and random number R_j to encrypt the initial trust value of each UE (e.g., $TV_i = 0.01$), $E_{Z_1,R_1}(\ldots E_{Z_m,R_m}(TV_i))$. In order to ensure the ciphertext of $TV_i = 0.01$ is correct as expected, each OA is asked to publish a proof that attests its encryption operation is correct. This can be achieved by using verifiable encryption shuffling [29]. After ensuring $E_{Z_1,R_1}(\ldots E_{Z_m,R_m}(TV_i))$ is correct, OA_1 constructs a list L_0 that consists of $< y_i, E_{Z_1,R_1}(\ldots E_{Z_m,R_m}(TV_i)) >$, $i = 1, \ldots, n$.

Destination List L_m Creation. According to *Algorithm 1*, OA_1 selects an ephemeral number e_1, and uses e_1 and z_1 (i.e., the private key corresponding to Z_1) to perform $sh(L_0, g_0, e_1, z_1)$ for obtaining the result L_1, g_1, and pf_1 [13], where $g_0 = g$. OA_1 broadcasts $\{L_1, g_1, pf_1\}$ with a signature $sig_{x_{OA_1}}(L_1, g_1, pf_1)$. After receiving the results from OA_1, OA_2 verifies the signature and proof pf_1, and performs $sh(L_1, g_1, e_2, z_2)$ to obtain the result L_2, g_2, and pf_2. Similarly, OA_2 broadcasts its result with a signature, and so on, until the above process goes to OA_m. OA_m verifies the received signature and proof, and performs $sh(L_{m-1}, g_{m-1}, e_m, z_m)$ to obtain L_m that contains $< pk_{\pi(i)}, TV_{\pi(i)} >$, $i = 1, \ldots, n$, $g_m = g^{e_1 \cdots e_m}$ and pf_m. At last, OA_m publishes $\{L_m, g_m, pf_m\}$ with a signature $sig_{x_{OA_m}}(L_m, g_m, pf_m)$ for public verification on its shuffling. Anyone (including OAs) can check the correctness of shuffling. Since at least one OA does not collude with others (as we assumed), an attacker cannot link the pairs of L_0 to L_m's.

After verifying the correctness of shuffling, OAs package the intermediate results and corresponding verification information of L_0 and L_m creation (i.e., each OA's broadcast result and signature), L_m, and g_m into a block, which is locally stored as the genesis block. The intermediate results and relative verification information published on the blockchain can help verifying or auditing the creation of genesis block publicly. The creation of L_m and g_m belongs to off-chain computation. If only the final off-chain computation results (i.e., L_m and g_m) are recorded on the blockchain, an observer cannot publicly verify or audit them later on, since the observer does not know execution traces of the off-chain computation. Thus, intermediate results and verification information of off-chain computation should be recorded on the blockchain in order to allow public verifications or auditing. Similarly, public verifications on the following trust evaluation and list maintenance can be supported by recording their intermediate results and verification information on the blockchain. UE_i can compute its pseudonym $pk_{\pi(i)} = {g_m}^{x_i}$ with its private key x_i and g_m obtained from the blockchain, and checks whether $pk_{\pi(i)}$ exists in L_m by accessing the blockchain maintained by all OAs. If $pk_{\pi(i)}$ exists, UE_i can use this pseudonym for network activities according to the ElGamal encryption and digital signature. It is worth noting that when UEs carry out network activities, some anonymous tools (e.g., disposable IP and MAC addresses) can be adopted with pseudonyms in order to hide UEs' local information for avoiding linking their pseudonyms [11].

C. Trust Value Update and Consensus

Trust Value Update. OAs should evaluate and update the trust of UE for some consecutive periods. It is worth noting that AP_{ti}'s collected trust related data TD_{ti} should include behavior sets of all UEs monitored by AP_{ti}. Based on each UE's relative trust related data (i.e., behavior set) that APs provide and CSP stores and its trust value recorded in the newest updated L_m, OAs compute a new trust value according to a trust evaluation algorithm [9]. For example, behavior patterns $P = \{P_N, P_A\}$ can be used to evaluate the trust of monitored UEs. P_N is a normal behavior pattern set and P_A is an abnormal behavior pattern set. As for UE_i's behavior set $Be = \{Be_1, \ldots, Be_l\}$, if I_N behaviors match the patterns in P_N and I_A behaviors match the patterns in P_A, its trust can be computed as

$$TV_{\pi(i)} = \frac{1}{e^{-|K-u_{\pi(i)}|/\tau} + 1} \times \frac{I_N - \tilde{k} \times I_A}{I_N + \tilde{k} \times I_A} + \frac{e^{-|K-u_{\pi(i)}|/\tau}}{e^{-|K-u_{\pi(i)}|/\tau} + 1} \times TV_{\pi(i)}^{old},$$

where $u_{\pi(i)}$ is the serial number of the latest block in the blockchain that alters UE_i's trust value. K is the block serial number of current trust evaluation, τ is a parameter to control time decay, $\tilde{k}$ ($\tilde{k} > 1$) is a parameter to adjust the weight of the number of abnormal behaviors and $TV_{\pi(i)}^{old}$ is the latest trust value of UE_i appearing in the blockchain. Then, OAs replace $TV_{\pi(i)}^{old}$ in the newly updated L_m with $TV_{\pi(i)}$.

Trust Value Consensus. OAs act as consensus nodes to perform blockchain consensus in order to synchronize and share L_m across multiple domains in an efficient and secure way. If OAs have updated the list L_m based on trust related data $TD_{ti}, ti = 1, \ldots, n_d$, they consider L_m, P, τ, and $\tilde{k}$ as the computation result Re and start to perform blockchain consensus. After the consensus, the next block can be confirmed by OAs. The updated list L_m stored at blockchain helps fine-grained management on UEs according to their new trust values. Meanwhile, anyone can access TD_{ti}, P, τ, and $\tilde{k}$, and then verify L_m published on the blockchain, thus enabling public verification on trust value update.

D. List L_m Maintenance

After some rounds of consensus (e.g., R rounds), the old list should be maintained by updating each UE's pseudonym to prevent an attacker from tracking UE's activities for a long time. The reason that we do not maintain the list after each round of consensus (i.e., $R = 1$) for achieving strong unlinkability of pseudonyms is to guarantee the efficiency of the whole network system.

Unique List Confirmation. In the blockchain system, consensus nodes might backup different blocks due to possible forking after R rounds of consensus, and different blocks might include different lists. In order to ensure the consistency of lists waiting to be maintained, each OA should publish its latest block backup of blockchain (i.e., the backed block or vote v_j), and determine the unique block that the majority of consensus nodes accept. Thus, the old list L_m in the unique block should be maintained by all OAs.

Trust Obfuscation. If only pseudonyms of L_m are updated, it is possible for an attacker to link two pseudonyms in old and new lists through analysis on trust values, which violates the privacy of UEs and the goal of list maintenance. The probability that a pseudonym in the old list has the same trust value with others is high [13], such that tracking UE's pseudonyms through analysis on trust values for a long time is hard to be successful. However, we still consider to increase such the probability through trust obfuscation for improving the unlinkability to an expected degree.

Before trust obfuscation, each UE's trust value should be reassessed and updated by all OAs due to time decay as

$$TV_{\pi(i)} = \frac{e^{-|K-u_{\pi(i)}|/\tau}}{e^{-|K-u_{\pi(i)}|/\tau}+1} \times TV^{old}_{\pi(i)},$$

where $TV^{old}_{\pi(i)}$ is UE_i's trust value in the old list. According to UEs' trust values, OAs can determine the revocation list L_{revo} of pseudonym and trust value pairs. For example, the trust value of some UEs could be reduced to 0 due to time decay or abnormal behaviors, thus these UEs' pseudonym and trust value pairs could be revoked. After determining L_{revo}, OAs can delete L_{revo} from the trust updated list.

After updating each UE's trust value according to time decay and deleting L_{revo}, OAs should perform trust obfuscation to improve the unlinkability of old and new pseudonyms. Suppose a trust value ranges from 0 to 1. We divide the interval [0, 1] into d sub-intervals, where d is an integer. $N_{TV} = 1/d$, $d \times N_{TV} \leq 1$, and $1 - d \times N_{TV} < N_{TV}$. The sub-intervals are defined as $[(c-1) \times N_{TV}, c \times N_{TV})$, $c = 1, \ldots, d-1$ and $[(d-1) \times N_{TV}, 1]$. In order to avoid trust promotion and achieve trust obfuscation, OAs compute

$$TV_{\pi(i)} = \begin{cases} (c-1) \times N_{TV}, & TV_{\pi(i)} \in [(c-1) \times N_{TV}, \\ & c \times N_{TV}) \\ (d-1) \times N_{TV}, & TV_{\pi(i)} \in [(d-1) \times N_{TV}, \\ & 1] \end{cases}.$$

Therefore, after trust obfuscation, the probability that a pseudonym has the same trust value with others increases. In addition, multiple rounds of maintenance make the probability that an attacker always succeeds in tracking a UE tend to 0 based on our theoretical analysis as described below.

In order to quantify the capability of trust obfuscation, we consider the number N_{min} of pseudonyms whose trust values belong to a sub-interval with the worst anonymity after trust obfuscation. In this case, there are the fewest pseudonyms whose trust values located in this sub-interval. Suppose N_{th0} is the minimum unit of trust value. When $d < 1/N_{th0}$ decreases, N_{min} increases and the anonymity is enhanced. As long as we ensure $N_{min} > 1$ by adjusting d, the anonymity can be achieved. According to K-anonymity, a pseudonym can be tracked with an expected probability $p = 1/N_{min}$ in an extreme case after trust obfuscation. We can set a threshold p_{th} of the maximum probability in advance that an attacker can track a pseudonym after each round of maintenance, and derive an appropriate d for maintenance. If UE's trust value always belongs to a sub-interval with the worst anonymity after maintenance, the probability of an attacker to always track UE's pseudonyms is less than $p_0 = {p_{th}}^T$ after T rounds of maintenance. When p_{th} is small enough, p_0 tends to 0, which suggests it is hard to break the unlinkability of pseudonyms or activities.

Algorithm 3: Interval Division Determination.

```
Input: $TV_{\pi(i)}, i = 1, \cdots, n$; $d_0$; $N_{th0}$; $p_{th}$
Output: $d$
1 for $D, d_0 < D < 1/N_{th0}$ do
2     Obtain $N_{min}$ when $[0, 1]$ is divided into $D$ sub-intervals ;
3     if $1/N_{min} > p_{th}$ then
4         $d = D - 1$ ;
5         Return $d$ ;
6     end
7 end
```

According to the above analysis, *Algorithm 3* shows how to determine d, when giving the threshold d_0 of the number of sub-intervals and the threshold p_{th} of probability of an attacker to track a UE according to its trust value in a sub-interval with the worst anonymity. p_{th} could be a public parameter negotiated by OAs.

Before trust obfuscation, each OA determines a unique list waiting to be maintained. Once the unique list is confirmed, each OA performs trust obfuscation in a parallel way based a common list and thus obtains a trust-obfuscated list with OAs' potential consensus. The potential consensus guarantees the correctness of obfuscation. Otherwise, the backward and forward shuffling in the following *Pseudonym Update* cannot pass public verifications with consensus.

While the system is running, new UEs may attempt to join the system. After OAs agree with new UE participation, OAs construct new UEs' list L_{join} of pseudonym and trust value pairs by performing *Initial List* L_0 *Creation* and *Destination List* L_m *Creation* based on these UEs' public keys. It is noted that OA_j adopts the random number e'_j in forward shuffling of the previous round of maintenance while performing forward shuffling, and the creation of L_{join} could be completed before this round of maintenance. Then, each OA inserts L_{join} at the end of trust-obfuscated list for obtaining a new one.

Pseudonym Update. After trust obfuscation, OAs reversely perform *Destination List* L_m *Creation* of *Genesis Block Generation* on the trust-obfuscated list cooperatively, obtaining a list L_0^{new} containing each UE's pair of public key and corresponding trust value ciphertext. In order to update each UE's pseudonym, OAs choose a new ephemeral number e_j^{new}, and then perform *Destination List* L_m *Creation* for obtaining $g_m^{new} = g^{e_1^{new} \ldots e_m^{new}}$ and a new list L_m^{new}, which consists of new pseudonyms and obfuscated trust values.

New List Attachment. When each OA obtains the new list L_m^{new}, it tries to insert the list into the blockchain. In detail, OA packages the hash value h_{K-1} of the latest block, $v_j, j = 1, \ldots, m$, τ, L_{revo}, L_{join}, p_{th}, d, L_m^{new}, g_m^{new}, the intermediate results and corresponding verification information of L_{join} and L_m^{new} creation into a block, which is considered by OA as the next block of the blockchain, but has no serial numbers. After L_m^{new} is announced on the blockchain, UEs can use their private key x_i and new published g_m^{new} to compute their new pseudonyms, and check whether these pseudonyms are recorded on the blockchain. Finally, they use new pseudonyms to carry out network activities. When UE_i conceals its local information (e.g., IP and MAC addresses) and accesses the

DePTVM blockchain with its newly computed pseudonym to check whether this pseudonym exists in the new list, the query includes the new pseudonym but does not contain sensitive data in our scheme, the blockchain could return a result (i.e., ”yes” or ”no”), and the blockchain access of UE_i can be regarded as a network activity based on UE_i's pseudonym. In addition, protecting the association between a query and its corresponding result is not necessary, since such an association is public on the blockchain and is not sensitive. In the next round of maintenance, UE_i computes another new pseudonym and uses this pseudonym to access the blockchain for checking its existence, and this new pseudonym is unlinked with the aforementioned new pseudonym, which implies that UE_i's queries are unlinked. Similarly, when UE_i conceals its local information and attempts to obtain the same parameter g_m from blockchain for computing its pseudonym, its query does not contain any sensitive data since the query only requests g_m. The result of query is the same parameter g_m, thus protecting the relation between a query and g_m is not needed. One UE's queries cannot be linked together before and after pseudonym update. In addition, the two types of queries from the same UE to the blockchain for obtaining g_m or checking the existence of its pseudonym cannot be linked, and the association between the above two types of UE queries cannot imply any sensitive information, thus it is not necessary to protect such the association. Therefore, employing some tools like private membership testing to further hide the access pattern of UE_i becomes unnecessary during this pseudonym transition phase. The involved intermediate results and relative verification information can provide public verification on list maintenance.

V. Analysis and Performance Evaluation

In this section, we analyze the properties of DePTVM, and evaluate its performance through simulation.

A. Performance Analysis

In this part, we discuss that DePTVM can ensure correctness, unforgeability, anonymity and unlinkability.

1) Correctness: Definition 1 (Correctness). The majority of consensus nodes reach consensus on the update of pseudonym and trust value after trust evaluation or list maintenance.

Theorem 1. If the fault tolerance of blockchain is $\eta > 50\%$, DePTVM can ensure its correctness.

Proof 1. The consensus on the trust-updated list after *Trust Value Update* is similar to the confirmation of the unique list after R rounds of consensus, which suggests voting on the lists. For simplicity, we only consider the correctness of trust value update. We adopt the proof by contradiction. Assume DePTVM cannot ensure the correctness, namely it accepts a forged trust-updated list.

There exist two lists, L and L', which represent correct and forged trust-updated lists respectively. DePTVM accepts L', thus the majority of consensus nodes accept L', and the proportion of these malicious consensus nodes is $\chi > 50\%$. In blockchain, L and L' should be inserted into B and B' respectively for consensus. Honest consensus nodes verify the contents of received blocks, thus they only accept and store the valid B. But the malicious consensus nodes accept and store B'. In other words, $\chi > 50\%$ consensus nodes accept B', and $\eta = 1 - \chi < 50\%$ consensus nodes accept the correct B, which contradicts the fault tolerance of adopted blockchain consensus. Therefore, if the fault tolerance of blockchain is $\eta > 50\%$, the correctness can be guaranteed. $\square$

2) Unforgeability: Unforgeability: DePTVM can prevent an attacker from forging the message of any entities.

DePTVM employs the ElGamal digital signature to authenticate data sources. For simplicity, we consider an entity can use its private key x and random number k to sign the message mes, $\left(r = {g_m}^k, s = (H(mes) + xr)\,k^{-1}\right)$. Any entities can verify ${g_m}^{\frac{H(mes)}{s}} pk^{\frac{r}{s}} = r$ based on the pseudonym $pk = {g_m}^x$ corresponding to the private key x. In order to simply a proof, we consider $g_m = g$. First, we design a *Forgeability Game* according to the forging attack, and define the advantage of an adversary $\mathcal{A}$ to succeed in attacking. Then, a simulator $\mathcal{B}$ employs $\mathcal{A}$ to solve the discrete logarithm problem in the *Unforgeability Proof.* If $\mathcal{A}$'s advantage is not negligible, we can infer that the advantage of $\mathcal{B}$ to solve the problem is not negligible, which contradicts with the assumption that the problem is hard. Thus, $\mathcal{A}$'s advantage is negligible.

(1) Forgeablity Game

We construct a game under adaptive chosen-message attacks. This game has two players, namely adversary $\mathcal{A}$ and challenger $\mathcal{C}$. In addition, the game should satisfy the following requirements: (1) $\mathcal{A}$ does not request the hash value of the same message. (2) If $\mathcal{A}$ requests the signature on mes^*, it has requested the hash value of mes^*. (3) If $\mathcal{A}$ has forged the pair $(mes^*, sign(mes^*))$ of message and signature, it has requested the hash value of mes^*. The game's process is described as follows.

1) *Initialization:* $\mathcal{C}$ sets a digital signature algorithm $sign(mes)$ and a hash function $H(mes)$. Then, it creates a private/public key pair $(x, y = g^x)$, and sends y to $\mathcal{A}$.
2) *Query 1:* $\mathcal{A}$ requests the hash value of mes, and $\mathcal{C}$ returns $H(mes)$.
3) *Query 2:* $\mathcal{A}$ requests the signature on mes, and $\mathcal{C}$ returns $sign(mes)$.
4) *Challenge:* $\mathcal{A}$ forges a pair of message and signature, $(mes^*, sign(mes^*)) \leftarrow \mathcal{F}(y)$, and returns $(mes^*$, $sign(mes^*))$ to $\mathcal{C}$. $\mathcal{A}$ has not requested the signature on mes^*, but has obtained the hash value of mes^*. $\mathcal{C}$ computes $Verify(y, mes^*, sign(mes^*))$. $Verify(y, mes^*$, $sign(mes^*)) = 1$ suggests the forged signature is valid. Otherwise, $Verify(y,\ mes^*,\ sign(mes^*)) = 0$ means the signature is not valid.

Therefore, the advantage of $\mathcal{A}$ for existentially forging a signature is

$$AdvSig_{\mathcal{A}} = Pr\,[Verify(y, mes^*, sign(mes^*)) = 1 : (x, y) \leftarrow KeyGen, (mes^*, sign(mes^*)) \leftarrow \mathcal{F}(y)].$$

Definition 2. If the polynomial-time adversary's advantage $AdvSig_A$ is negligible, DePTVM can guarantee unforgeability.

(2) Unforgeablity Proof

Theorem 2. If there exists the discrete logarithm problem, no polynomial-time adversaries can break the unforgeability with a non-negligible advantage.

Proof 2. We suppose $\mathcal{A}$ can win the above game with a non-negligible advantage $AdvSig_{\mathcal{A}} = \varepsilon$, thus we construct a simulator $\mathcal{B}$, which can solve the discrete logarithm problem with a non-negligible advantage $\left(1 - \frac{1}{q_H}\right)^{q_H} \frac{\varepsilon}{q_H}$ (i.e., $\mathcal{B}$ can use $\mathcal{A}$ to solve $x = \frac{sk-h^*}{r}$, which satisfies $g^{\frac{h^*}{s}} y^{\frac{r}{s}} = r$). $\mathcal{A}$ is allowed to request hash values of at most q_H messages, $\mathcal{B}$ randomly determines $h^* = H(mes^*)$, the corresponding message mes^* has been used for requesting a hash value and no signatures. $\mathcal{B}$ acts as $\mathcal{C}$.

1) *Initialization:* $\mathcal{B}$ sends y to $\mathcal{A}$. $\mathcal{B}$ randomly determines $j \in [1, q_H]$ for identifying a forged signature corresponding to $\mathcal{A}$'s j-th hash value query mes^*.
2) *Query 1:* $\mathcal{B}$ sets an empty list $H^{list} = \emptyset$ for storing (mes_i, r_i, s_i, h_i), where $h_i = H(mes_i)$ and $g^{\frac{h_i}{s_i}} y^{\frac{r_i}{s_i}} = r_i$. If $\mathcal{A}$ requests the hash value of the i-th message mes_i, $\mathcal{B}$ performs the following operations.
   a) If $i = j$, $\mathcal{B}$ returns h^*;
   b) If $i \neq j$, $\mathcal{B}$ selects random numbers (r_i, s_i), which satisfy $g^{\frac{h_i}{s_i}} y^{\frac{r_i}{s_i}} = r_i$. $\mathcal{B}$ stores (mes_i, r_i, s_i, h_i), and sends h_i to $\mathcal{A}$.
3) *Query 2:* If $\mathcal{A}$ has requested the hash value of mes_i and now requests its signature, $\mathcal{B}$ performs the following operations.
   a) If $i = j$, $\mathcal{B}$ breaks off;
   b) If $i \neq j$, $\mathcal{B}$ extracts (mes_i, r_i, s_i, h_i) in H^{list}, and sends (r_i, s_i) to $\mathcal{A}$.
4) *Challenge:* $\mathcal{A}$ forges and sends a pair (mes, r, s) of message and signature to $\mathcal{B}$. If $mes \neq mes^*$, $\mathcal{B}$ breaks off. If $mes = mes^*$ and verifications hold, $\mathcal{B}$ outputs (r, s), which suggests $\mathcal{B}$ succeeds in solving the discrete logarithm problem.

The event that $\mathcal{B}$ outputs (r, s) relies on the following events.

E_1: $\mathcal{B}$ does not break off in signature queries;

E_2: $\mathcal{A}$'s forged (mes, r, s) is valid;

E_3: The event E_2 occurs, and the message mes is mes^* (i.e., the j-th hash value query).

We can get $Pr[E_1] = \left(1 - \frac{1}{q_H}\right)^{q_H}$, $Pr\left[E_2|E_1\right] = \varepsilon$ and $Pr\left[E_3|E_1E_2\right] = Pr\left[mes = mes^*|E_1E_2\right] = \frac{1}{q_H}$. The advantage of $\mathcal{B}$ is

$$AdvSig_{\mathcal{B}} = Pr[E_1]Pr[E_2|E_1]Pr[E_3|E_1E_2] = \left(1 - \frac{1}{q_H}\right)^{q_H} \frac{\varepsilon}{q_H}.$$

Thus, $\mathcal{B}$ can solve the discrete logarithm problem with a non-negligible advantage, which contradicts with the assumption that the problem is hard. Therefore, $\mathcal{A}$'s advantage ε is negligible, and DePTVM can ensure the unforgeability. □

3) Anonymity and Unlinkability: Anonymity and Unlinkability. DePTVM can ensure both anonymity and unlinkability at the same time. Namely, no one can know the real identifiers of UEs when they carry out network activities by using pseudonyms, and anyone cannot link their old and new pseudonyms for tracking their activities.

DePTVM can achieve anonymity, since UEs use their pseudonyms that are periodically updated as their identifiers for networking instead of their long-term public key. As discussed in *Trust Obfuscation*, we can determine d in order to constrain the ability of an attacker to track pseudonyms of UE according to its trust value in a sub-interval with the worst anonymity. We design a *Linkability Game* based on linking attacks, and define the advantage of an adversary $\mathcal{A}$ to successfully attack. Then, a simulator $\mathcal{B}$ adopts $\mathcal{A}$ to solve the decisional Diffie-Hellman problem in the *Linkability Proof.* If $\mathcal{A}$'s advantage is not negligible, we can infer that the advantage of $\mathcal{B}$ to solve the problem is not negligible, which conflicts with the assumption that the problem is hard. Thus, $\mathcal{A}$'s advantage is negligible.

(1) Linkability Game

We consider m OAs as $\mathcal{OA}$. The *pseudonym update* can be described as follows. $\mathcal{OA}$ performs backward and forward shuffling based on a trust obfuscated list L_{old}, and obtains a new trust obfuscated list L_{new}. For simplicity, we suppose L_{old} and L_{new} have the same trust value. We consider the unlinkability of pairs of L_{old} and L_{new}, namely we cannot distinguish whether new and old pairs of pseudonym and trust value before and after pseudonym update belong to the same UE. Based on this thought, we design a game that has two players (i.e., adversary $\mathcal{A}$ and challenger $\mathcal{C}$). This game's process is described as follows.

1) *Initialization:* $\mathcal{C}$ simulates $\mathcal{OA}$ to create L_{old}. $\mathcal{C}$ determines UE_0's public key g^{x_0} and old pseudonym $g^{x_0 E}$, and sends g^{x_0}, $g_m = g^E$ and the pair $< g^{x_0 E}, TV >$ of pseudonym and trust value to $\mathcal{A}$.
2) *Challenge:* $\mathcal{A}$ challenges $\mathcal{C}$. $\mathcal{C}$ randomly selects $b \in \{0, 1\}$, and sends UE_b's new pseudonym and trust value pair $< g^{x_b \tilde{E}}, TV >$ and $g_m^{new} = g^{\tilde{E}}$ to $\mathcal{A}$.
3) *Guess:* $\mathcal{A}$ guesses $b' \in \{0, 1\}$. If $b' = b$, $\mathcal{A}$ wins this game. The advantage of $\mathcal{A}$ to win the game is defined as $Adv_{\mathcal{A}} = |Pr[b' = b] - 1/2|$.

Definition 3. If the adversary's advantage $Adv_{\mathcal{A}}$ is negligible, DePTVM can guarantee the unlinkability of new and old pairs of pseudonym and trust value or pseudonyms.

(2) Unlinkability Proof

Theorem 3. If there exists the decisional Diffie-Hellman problem, no polynomial-time adversaries can break the unlinkability through linking new and old pairs of pseudonym and trust value with a non-negligible advantage.

Proof 3. We adopt the proof by contradiction. We assume the adversary $\mathcal{A}$ wins the above game with a non-negligible advantage $\varepsilon = Adv_{\mathcal{A}}$, and construct a simulator $\mathcal{B}$, which breaks the decisional Diffie-Hellman problem with an advantage $\varepsilon/2$. $\mathcal{C}$ simulates $\mathcal{OA}$ to generate L_{old} and $g_m = g^E$. If $b = 0$, $\mathcal{C}$ sets $T = g^{x_0(\tilde{E}-E)}$. Otherwise, $\mathcal{C}$ sets T as a random number $T = Rand$. Then, $\mathcal{C}$ sends $(g, g^{x_0}, g^{\tilde{E}-E}, T, L_{old}, g_m, g_m^{new} = g^{\tilde{E}})$ to $\mathcal{B}$. $\mathcal{B}$ plays as $\mathcal{C}$.

1) *Initialization:* $\mathcal{B}$ sends g^{x_0}, g_m and $< g^{x_0 E}, TV >$ to $\mathcal{A}$.

2) *Challenge:* $\mathcal{A}$ challenges $\mathcal{B}$. $\mathcal{B}$ randomly selects $b' \in \{0, 1\}$. $\mathcal{B}$ sends $UE_{b'}$'s new pair $< g^{x_{b'}E}T, TV >$ of pseudonym and trust value and $g_m^{new} = g^{\tilde{E}}$ to $\mathcal{A}$.
3) *Guess:* $\mathcal{A}$ guesses $\hat{b} \in \{0, 1\}$. If $b' = \hat{b}$, $\mathcal{B}$ outputs 0 for guessing $T = g^{x_0(\tilde{E}-E)}$. Otherwise, $\mathcal{B}$ outputs 1 for guessing T is a random number.

If $\mathcal{B}$ adopts $T = g^{x_0(\tilde{E}-E)}$, $< g^{x_{b'}E}T, TV >$ is UE_0's new pair of pseudonym and trust value, and $\mathcal{A}$'s advantage is ε. Thus, $Pr[\mathcal{B}(g, g^{x_0}, g^{\tilde{E}-E}, T = g^{x_0(\tilde{E}-E)}) = 0] = \frac{1}{2} + \varepsilon$.

If $\mathcal{B}$ uses $T = Rand$, $\mathcal{A}$ considers $g^{x_{b'}E}T$ as random. The probability that $b' \neq \hat{b}$ is 1/2, thus $Pr[\mathcal{B}(g, g^{x_0},\ g^{\tilde{E}-E},\ T = Rand) = 0] = \frac{1}{2}$.

As a result, $\mathcal{B}$'s advantage is

$$\begin{aligned} Adv_{\mathcal{B}} &= \frac{1}{2} Pr\left[\mathcal{B}\left(g, g^{x_0}, g^{\tilde{E}-E}, T = g^{x_0(\tilde{E}-E)}\right) = 0\right] \\ &\quad + \frac{1}{2} Pr\left[\mathcal{B}\left(g, g^{x_0}, g^{\tilde{E}-E}, T = Rand\right) = 0\right] - \frac{1}{2} \\ &= \frac{1}{2} \times \left(\frac{1}{2} + \varepsilon\right) + \frac{1}{2} \times \frac{1}{2} - \frac{1}{2} = \frac{\varepsilon}{2}. \end{aligned}$$

This contradicts with the assumption that the decisional Diffie-Hellman problem is hard, which suggests $\mathcal{A}$'s advantage is negligible. Therefore, DePTVM can prevent adversaries from breaking the unlinkability of new and old pairs of pseudonym and trust value. □

B. Performance Evaluation

In this part, we evaluate DePTVM's performance through simulations with respect to block creation time, task completion time, throughput, memory usage, shuffling time, and trust anonymity.

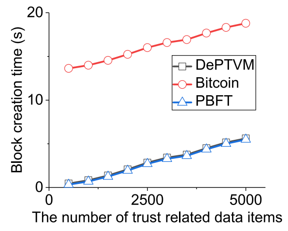
1) Metrics: We take following metrics for evaluation. (1) Block creation time: the average time that DePTVM takes to create a block; (2) Task completion time: the average time from the time point when APs share their trust related data to the time point when the next block is confirmed; (3) Throughput: the average number of UEs whose monitored trust related data are used for the next block creation per second. (4) Memory usage: the maximum memory usage of a consensus node regarding the whole system process. (5) Shuffling time: the average time that OA takes to perform forward and backward shuffling. (6) Trust anonymity: the trust anonymity of a sub-interval with the fewest trust values.

2) Experimental Settings: DePTVM was simulated using the Go programming language (Golang) in a virtual machine running Ubuntu 20.04.2 LTS with 4 cores and 4 GB RAM, hosted on a laptop that runs Windows 10 with Intel Core i5-9300H CPU @2.4 GHz and 8 GB RAM, adopting VMware workstation. We simulated 16 operator agents as consensus nodes, 500 access points as collection nodes, and one CSP. Each access point monitors 10 UEs and collects their trust related data. Thus, 5000 UEs exist, and their public key and initial trust value set as 0.01 are used to create the list L_m. Data collection consumes negligible time for the powerful CSP, thus we set its time as 0 s. The speed of downloading data from CSP is set as 10 MB/s. We set the block size as 160B, thus the block propagation time is negligible. In order to obtain trust values for assessing the capability of trust obfuscation, we used learning vector quantization (LVQ) clustering to simply classify the sampled instances of the CIC-IDS2017 dataset [37] for obtaining patterns and adopted some instances as trust related data of monitored UEs. These patterns were employed to match data of UEs for computing each UE's trust value. The trust values of UEs roughly conform to Gaussian distribution [15], especially in integrated networks with a large number of UEs. We used the evaluated trust values to assess the capability of trust obfuscation. In addition, we set $R = 10$ for ensuring the expected unlinkability capacity and high efficiency. Under the same experimental settings, we test the performance of Bitcoin and PBFT based blockchain for comparison. Because Proof of Useful Work (PoUW) and tree and Directed Acyclic Graph (DAG) based consensus mechanisms heavily rely on PoW [17], and there exist three basic types of consensus mechanisms, i.e., PoW, BFT and PoS. The consensus mechanism that DePTVM adopts depends on PoS, thus we employ Bitcoin and PBFT based blockchain for comparison.

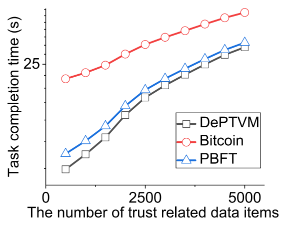
In terms of experimental execution, we first adopted the CIC-IDS2017 dataset to learn patterns with the LVQ clustering, and loaded the patterns into operator agents. Each system entity generates its own public/private keys. Based on UEs' public keys, operator agents collaboratively create a list L_m that contains these UEs' pseudonyms and initial trust values. Access points periodically use instances of CIC-IDS2017 dataset as the trust related data items of their monitored UEs, and share them with CSP. CSP verifies and stores all the provided data items. Through multiple threads, operator agents act as consensus nodes to obtain trust related data items from CSP. They verify the received data items, use the patterns to match the data items, and compute trust values of UEs based on matching results. Then, they update the relative trust values in their local list L_m, and generate and publish a block for consensus. After $R = 10$ rounds of consensus, operator agents collaboratively maintain the list L_m with trust obfuscation and verifiable shuffling.

3) Experimental Results: Block Creation Time. As depicted in Fig. 4(a), we show the block creation time changed with the number of trust related data items. The time increases with the number increasing, since the larger number suggests the consensus node should verify and match more data items. In addition, the block creation time of DePTVM is similar to PBFT's, and is shorter than Bitcoin's. Because Bitcoin creates a block with the basic difficulty TAG, DePTVM adopts low difficulty and PBFT has no difficulty.

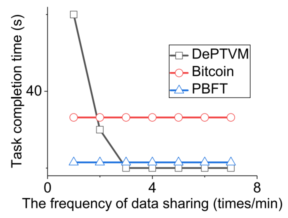
Task Completion Time. Fig. 4(b) presents the task completion time with the different number of trust related data items when the frequency of trust related data sharing is set as 7.26 times/min and $n_b = 0$. When the number increases, the task completion time increases since more data items should be processed. DePTVM's task completion time is shorter than Bicoin's and PBFT's, because DePTVM and Bitcoin adopt low difficulty and basic difficulty, respectively. Although PBFT creates a block with no difficulty, its two rounds of subsequent voting take some time.



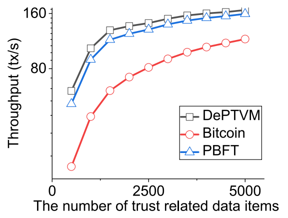
(a) Block creation time with regard to the number of data items



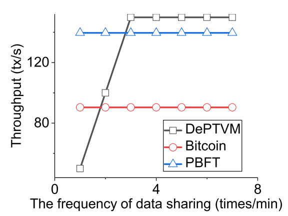
(b) Task completion time with regard to the number of data items



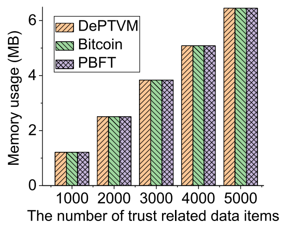
(c) Task completion time with regard to the frequency of data sharing



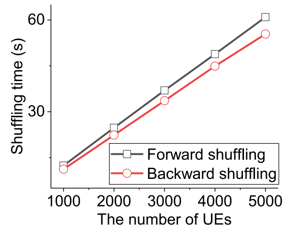
(d) Throughput with regard to the number of data items



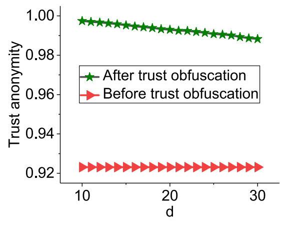
(e) Throughput with regard to the frequency of data sharing



(f) Memory usage of a consensus node with regard to the number of data items



(g) Shuffling time with regard to the number of UEs



(h) Trust Anonymity with regard to d

Fig. 4. Experimental results.

Fig. 4(c) shows the task completion time with the frequency change of trust related data sharing when we set the number of trust related data items as 3000 and $n_b = 0$. We observe the time is reduced with the increase of data sharing frequency when it is less than 3 times/min. If the frequency is larger than 3 times/min, the task completion time remains unchanged since it is larger than the time interval of data sharing. In addition, the task completion time of Bitcoin and PBFT remains invariable, because they generally start to create a new block immediately after generating a block. When the frequency is larger than 3 times/min, DePTVM's task completion time is shorter than Bitcoin's and PBFT's due to DePTVM's low difficulty of block creation.

In Table III, we present the task completion time with different n_b if the frequency of trust related data sharing is fixed as 7.26 times/min, the number of trust related data items is 1000, and $n_{bt} = 1$. The completion time increases when n_b increases, because fewer consensus nodes with different difficulties of block creation participate in block mining.

TABLE III
TASK COMPLETION TIME AND THROUGHPUT WITH REGARD TO n_b

n_b	0	3	7	11	15
TCT (s)	8.27	11.84	14.81	18.74	21.44
TH (tx/s)	60.46	42.23	33.76	26.68	23.32

TCT: task completion time; TH: throughput.

Throughput. Fig. 4(d) shows the throughput with the different number of trust related data items if the frequency of trust related data sharing is 7.26 times/min and $n_b = 0$. If the number increases, the throughput increases, since more data items are inserted into a block. Furthermore, DePTVM's throughput is larger than Bitcoin's and PBFT's, since DePTVM's task completion time is shorter compared to Bitcoin and PBFT.

Fig. 4(e) shows the throughput with the frequency change of trust related data sharing when the number of trust related data items is set as 3000 and $n_b = 0$. We observe the throughput increases with the increase of data sharing frequency if the frequency is less than 3 times/min. When the frequency is larger than 3 times/min, the throughput remains unchanged, because the task completion time is longer than the time interval of data sharing. Moreover, the throughput of Bitcoin and PBFT remains unchanged, because they generally start to create new blocks immediately when they succeed in creating blocks. If the frequency exceeds 3 times/min, DePTVM's throughput is larger than Bitcoin's and PBFT's due to DePTVM's low difficulty of block creation.

Table III presents the throughput with the different n_b when the frequency of trust related data sharing is 7.26 times/min, the number of trust related data items is 1000, and $n_{bt} = 1$. The throughput is reduced with the increase of n_b, since fewer consensus nodes that have different difficulties of block creation take part in block mining.

Memory Usage. Fig. 4(f) shows the maximum memory usage of a consensus node during system execution when the number of trust related data items is changed. We observe that the memory usage increases with the increase of the number of data items, because the bigger number requests a consensus node to process more trust related data. In addition, the memory usage of a consensus node is almost the same in DePTVM, Bitcoin and PBFT, since their processes that occupy the maximum memory are similar, i.e., processing trust-related data. Although PBFT involves two rounds of subsequent voting, voting uses a negligible amount of memory space compared to processing trust-related data. During consensus, consensus nodes load and process massive trust-related data from CSP in a parallel way and then store their evaluation results locally. They can package and verify blocks based on their local results. During block creation, solving a puzzle and signing occupy a negligible memory space. During block verification, consensus nodes compare their local results with that in a received block without loading and processing massive trust-related data. Thus, when each consensus node processes massive trust-related data for obtaining an evaluation result, its memory usage reaches a maximum value and such a memory consumption is almost the same no matter with the applied consensus mechanisms, e.g., PoW, PBFT and the consensus mechanism used in DePTVM.

Pseudonym Update Time. Fig. 4(g) depicts the shuffling time with the number change of UEs. The time increases with the increase of the number, since the bigger number requests consensus nodes to update more UEs' pseudonyms.

Trust Anonymity. Fig. 4(h) presents the trust anonymity with the change of d. We can see that the trust anonymity becomes worse with the increase of d after trust obfuscation. The bigger d suggests that each sub-interval is smaller and has fewer trust values, thus the anonymity becomes worse. However, trust anonymity after trust obfuscation is still stronger than that before obfuscation.

VI. Discussion

DePTVM provides a decentralized pseudonym and trust value management scheme based on blockchain in a trustless environment. However, DePTVM also has its limitations, which motivate our future research.

Pattern Creation. DePTVM adopts behavior patterns for trust evaluation as an example, but how to create such patterns should be investigated. In addition, patterns should be confidential and shared among OAs for evaluation. Once the patterns are disclosed, malicious UEs could evade detection to achieve malicious purposes and maintain high trust by adjusting their behaviors according to the released patterns. Meanwhile, public verifications on pattern creation should be considered for auditing. Thus, how to secretly generate and share behavior patterns among OAs for evaluation and support the public verification is a good direction of our future work.

Efficient Trust Evaluation. In DePTVM, consensus nodes access data from CSP for trust evaluation, and insert evaluation results into the blockchain. However, when the consensus nodes deal with massive data, it is hard to ensure DePTVM's high efficiency. Thus, how to achieve efficient trust evaluation when facing massive data should be further studied.

Prototype System. DePTVM is simulated in experiments, which adopts the LVQ clustering to illustrate trust evaluation. However, the LVQ clustering may not be the best method for trust evaluation. Therefore, DePTVM should consider main factors affecting UE trust and integrate corresponding methods to guarantee high evaluation accuracy. In the future, we will develop and improve the prototype of DePTVM, which can integrate different trust evaluation methods for accurate evaluation and include a data acquisition tool towards practical deployment and further experiment.

VII. Conclusion

In this article, we proposed DePTVM, a decentralized UE pseudonym and trust value management scheme based on blockchain for integrated heterogeneous networks. By employing trust obfuscation and verifiable shuffling, our scheme can ensure the unlinkability of new and old pseudonyms. Trust values are synchronized and shared across multiple domains through blockchain consensus. Serious analysis shows DePTVM's efficacy in terms of correctness, unforgeability, anonymity and unlinkability. Simulation based performance evaluation further proves its efficacy.

References

[1] Y. Xiao, G. Shi, and M. Krunz, "Towards ubiquitous AI in 6G with federated learning," 2020, *arXiv:2004.13563*.

[2] M. Khan et al., "On de-synchronization of user pseudonyms in mobile networks," in *Proc. Int. Conf. Inf. Syst. Secur.*, 2017, pp. 347–366.

[3] M. Ylianttila et al., "6G white paper: Research challenges for trust, security and privacy," 2020, *arXiv:2004.11665*.

[4] L. F. Zhang, Z. Yan, and R. Kantola, "Privacy-preserving trust management for unwanted traffic control," *Future Gener. Comput. Syst.*, vol. 72, pp. 305–318, 2017.

[5] Z. Yan, R. Kantola, and Y. Shen, "A generic solution for unwanted traffic control through trust management," *New Rev. Hypermedia Multimedia*, vol. 20, no. 1, pp. 25–51, 2014.

[6] Z. Yan and C. Prehofer, "Autonomic trust management for a component-based software system," *IEEE Trans. Dependable Secure Comput.*, vol. 8, no. 6, pp. 810–823, Nov./Dec. 2011.

[7] Z. Yan, X. Li, M. Wang, and A. V. Vasilakos, "Flexible data access control based on trust and reputation in cloud computing," *IEEE Trans. Cloud Comput.*, vol. 5, no. 3, pp. 485–498, Third Quarter 2017.

[8] T. Minkus and K. W. Ross, "I know what you're buying: Privacy breaches on eBay," in *Proc. Int. Symp. Privacy Enhancing Technol.*, 2014, pp. 164–183.

[9] W. Feng, Z. Yan, L. T. Yang, and Q. Zheng, "Anonymous authentication on trust in blockchain-based mobile crowdsourcing," *IEEE Internet of Things J.*, vol. 9, no. 16, pp. 14185–14202, Aug. 2022.

[10] E. Androulaki et al., "Reputation systems for anonymous networks," in *Proc. Int. Symp. Privacy Enhancing Technol.*, 2008, pp. 202–218.

[11] D. Christin et al., "IncogniSense: An anonymity-preserving reputation framework for participatory sensing applications," *Pervasive Mobile Comput.*, vol. 9, no. 3, pp. 353–371, 2013.

[12] X. O. Wang, W. Cheng, P. Mohapatra, and T. Abdelzaher, "ARTSense: Anonymous reputation and trust in participatory sensing," in *Proc. IEEE INFOCOM*, 2013, pp. 2517–2525.

[13] E. Zhai et al., "AnonRep: Towards tracking-resistant anonymous reputation," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2016, pp. 583–596.

[14] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Proc. IEEE Symp. Secur. Privacy*, 2014, pp. 475–490.

[15] Z. Yan et al., "Social-chain: Decentralized trust evaluation based on blockchain in pervasive social networking," *ACM Trans. Internet Technol.*, vol. 21, no. 1, pp. 1–28, 2021.

[16] W. Feng and Z. Yan, "MCS-Chain: Decentralized and trustworthy mobile crowdsourcing based on blockchain," *Future Gener. Comput. Syst.*, vol. 95, pp. 649–666, 2019.

[17] G. Liu et al., "SeDID: An SGX-enabled decentralized intrusion detection framework for network trust evaluation," *Inf. Fusion*, vol. 70, pp. 100–114, 2021.

[18] Decentralized identity foundation, 2020. [Online]. Available: https://identity.foundation/

[19] W3C, "Decentralized identifiers (DIDs) V0.11: Data model and syntaxes for decentralized identifiers," 2018. [Online]. Available: https://w3c-ccg.github.io/did-spec/

[20] D. Maram et al., "CanDID: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 1348–1366.

[21] J. Wang, Z. Yan, H. Wang, T. Li, and W. Pedrycz, "A survey on trust models in heterogeneous networks," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 4, pp. 2127–2162, Fourth Quarter 2022.

[22] Y. Sun et al., "A trust-augmented voting scheme for collaborative privacy management," *J. Comput. Secur.*, vol. 20, no. 4, pp. 437–459, 2012.

[23] M. A. Al-Garadi, A. Mohamed, A. K. Al-Ali, X. Du, I. Ali, and M. Guizani, "A survey of machine and deep learning methods for Internet of Things (IoT) security," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1646–1685, Third Quarter 2020.

[24] G. Liu, H. Dong, Z. Yan, X. Zhou, and S. Shimizu, "B4SDC: A blockchain system for security data collection in MANETs," *IEEE Trans. Big Data*, vol. 8, no. 3, pp. 739–752, Jun. 2022.

[25] S. Fei et al., “Security vulnerabilities of SGX and countermeasures: A survey,” *ACM Comput. Surv.*, vol. 54, no. 6, pp. 1–36, 2021.

[26] S. Brenner et al., “Secure cloud micro services using Intel SGX,” in *Proc. IFIP Int. Conf. Distrib. Appl. Interoperable Syst.*, 2017, pp. 177–191.

[27] N. Dokmai et al., “Privacy-preserving genotype imputation in a trusted execution environment,” *Cell Syst.*, vol. 12, no. 10, pp. 983–993, 2021.

[28] M. Lipp et al., “PLATYPUS: Software-based power side-channel attacks on x86,” in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 355–371.

[29] C. A. Neff, “A verifiable secret shuffle and its application to e-voting,” in *Proc. ACM Conf. Comput. Commun. Secur.*, 2001, pp. 116–125.

[30] L. Nguyen, R. Safavi-Naini, and K. Kurosawa, “Verifiable shuffles: A formal model and a Paillier-based three-round construction with provable security,” *Int. J. Inf. Secur.*, vol. 5, no. 4, pp. 241–255, 2006.

[31] H. Zhou et al., “SDN-RDCD: A real-time and reliable method for detecting compromised SDN devices,” *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2048–2061, Oct. 2018.

[32] Y. Li et al., “A survey on truth discovery,” *ACM SIGKDD Explorations Newslett.*, vol. 17, no. 2, pp. 1–16, 2016.

[33] Y. Li, S. Liu, Z. Yan, and R. H. Deng, “Secure 5G positioning with truth discovery, attack detection and tracing,” *IEEE Internet of Things J.*, vol. 9, no. 22, pp. 22220–22229, Nov. 2022.

[34] S. Li, K. Xue, D. S. L. Wei, H. Yue, N. Yu, and P. Hong, “SecGrid: A secure and efficient SGX-enabled smart grid system with rich functionalities,” *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 1318–1330, 2020.

[35] B. Waters, “Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization,” in *Proc. Int. Workshop Public Key Cryptogr.*, 2011, pp. 53–70.

[36] D. L. Chaum, “Untraceable electronic mail, return addresses, and digital pseudonyms,” *Commun. ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[37] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization,” in *Proc. Int. Conf. Inf. Syst. Secur. Privacy*, 2018, pp. 108–116.

Gao Liu received the PhD degree in information security from the School of Cyber Engineering, Xidian University, in 2022. He is currently an assistant professor with the College of Computer Science, Chongqing University. His research interests include network security measurement, data collection, blockchain, federated learning, and data aggregation in smart grid.

Zheng Yan (Senior Member, IEEE) received the BEng degree in electrical engineering and the MEng degree in computer science and engineering from the Xi’an Jiaotong University, Xi’an, China, in 1994 and 1997, respectively, the MEng degree in information security from the National University of Singapore, Singapore, in 2000, and the licentiate of science and the doctor of science in technology in electrical engineering from the Helsinki University of Technology, Helsinki, Finland. She is currently a professor with the Xidian University, Xi’an and a visiting professor with the Aalto University, Espoo, Finland. Her research interests include trust, security, privacy, and security-related data analytics. She serves as a general or program chair for more than 30 international conferences and workshops. She is a steering committee co-chair of IEEE Blockchain international conference. She is also an associate editor of many reputable journals, e.g., the *IEEE Internet of Things Journal*, *Information Sciences*, *Information Fusion*, *Journal of Network and Computer Applications*, *IEEE Access*, *Security and Communication Networks*, etc.

Dongliang Wang is currently working toward the undergraduate degree majoring in cyberspace security with the School of Cyber Engineering, Xidian University. His research interests include privacy preservation, trusted computing, cryptography, and data mining.

Haiguang Wang (Senior Member, IEEE) received the bachelor’s degree from Peking University, in 1996, and the PhD degree in computer engineering from the National University of Singapore, in 2009. He is an expert on identity management and network security. He joined Huawei at year 2013 and currently he is a senior researcher with Huawei. He was a research engineer/scientist with I2R Singapore since 2001.

Tieyan Li received the PhD degree in computer science from the National University of Singapore. He is an expert on security and applied cryptography, and a technology generalist on applications, systems and networks. He is currently leading research on Digital Trust-building the trust infrastructure for future digital world, and previously on mobile security, IoT security, and AI security at Shield Lab., Singapore Research Center, Huawei Technologies. He was a security scientist with Institute for Infocomm Research, I2R Singapore. He has more than 20 years of experiences and is proficient in security design, architect, innovation, and practical development. He was also active in academic security fields with tens of publications and patents. He has served as the PC members for many security conferences, and is an influential speaker in industrial security forums. His current research topics include: Trustworthy AI, trustworthy computing, trustworthy identity, and network infrastructure.