IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 31, 2020, accepted January 18, 2021, date of publication January 27, 2021, date of current version February 9, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3054887

A Security Analysis of Blockchain-Based Did Services

BONG GON KIM[1], (Graduate Student Member, IEEE), YOUNG-SEOB CHO[2], SEOK-HYUN KIM[2], HYOUNGSHICK KIM[3], AND SIMON S. WOO[4], (Member, IEEE)

[1]Department of Computer Science, Stony Brook University, Stony Brook, NY 11794, USA
[2]Electronics and Telecommunications Research Institute, Daejeon 34129, South Korea
[3]Department of Software, Sungkyunkwan University, Suwon 440746, South Korea
[4]Department of Applied Data Science, Sungkyunkwan University, Suwon 440746, South Korea

Corresponding authors: Hyoungshick Kim (hyoung@skku.edu) and Simon S. Woo (swoo@g.skku.edu)

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Ministry of Science and ICT (MSIT), Korean Government through the Developing Blockchain Identity Management System with Implicit Augmented Authentication and Privacy Protection for O2O Services under Grant 2018-0-01369, in part by the SUNY Korea's ICT Consilience Creative Program through the Institute for Information and Communication Technology Planning and Evaluation (IITP) under Grant IITP-2020-2011-1-00783, in part by the Institute for Information and Communication Technology Planning and Evaluation (IITP) Grant funded by the Ministry of Science and ICT (MSIT), Korean Government through the Regional Strategic Industry Convergence Security Core Talent Training Business under Grant 2019-0-01343, in part by the Basic Science Research Program through the NRF of Korea funded by the Ministry of Science and ICT (MSIT) under Grant 2020R1C1C1006004, and in part by the High-Potential Individuals Global Training Program through the Institute for Information and Communications Technology Planning and Evaluation (IITP) funded by the Ministry of Science and ICT (MSIT) under Grant 2019-0-01579.

ABSTRACT Decentralized identifiers (DID) has shown great potential for sharing user identities across different domains and services without compromising user privacy. DID is designed to enable the minimum disclosure of the proof from a user's credentials on a need-to-know basis with a contextualized delegation. At first glance, DID appears to be well-suited for this purpose. However, the overall security of DID has not been thoroughly examined. In this paper, we systemically explore key components of DID systems and analyze their possible vulnerabilities when deployed. First, we analyze the data flow between DID system components and analyze possible security threats. Next, we carefully identify potential security threats over seven different DID functional domains, ranging from user wallet to universal resolver. Lastly, we discuss the possible countermeasures against the security threats we identified.

INDEX TERMS DID, decentralized key management system (DKMS), universal resolver, blockchain, data exfiltration, blockchain redaction, attack surface.

I. INTRODUCTION

DID is a new paradigm, where users can securely control their own identity (ID) and sovereignty without relying on any single central authority or third-party entities for managing users' credentials [1]. The decentralized autonomy and capability to control his/her own identity enable the DID ID management system to be a fully working Self-Sovereign Identity (SSI) model [2], [3] compared to other identity management schemes. For the principles of SSI, users can control and manage their ID by themselves with minimum disclosure of their personal information and claims, providing consistent usability across different contexts.

To realize SSI, the DID system has been proposed, leveraging state-of-the-art technologies such as DKMS, verifiable credentials (VC), universal resolver, microservice architecture (MSA), and blockchain. More specifically, DID is a pairwise relationship-oriented identifier, which is contextualized over different relationships. DID uses the delegated entities called edge agents or cloud agents by authorizing them to access the user's credentials, where the access level is controlled by the authorization policy in the public ledger per each agent. Each agent, belonging to the same identity subject but used in different devices or domains, uses a special type of key (a link secret). Thereby, each agent can act as if it is all from one logically unified identity. Detailed information about the DID, such as public key verification information, service endpoints, and specific authentication methods, is encapsulated in the DID document. If a user is requested by a certain DID to assert specific claims (e.g., the capability of operating a vehicle), the agent of the user's DID uses the VC, which contains the

The associate editor coordinating the review of this manuscript and approving it for publication was Kuo-Hui Yeh.

required official attestation information (e.g., driver license) from an issuer. Therefore, it is securely verified through the presentation of VCs with DID to prove the requested claims.

On the other hand, the detailed DID resolution to the corresponding DID document varies across different DID methods such as Sovrin [4], UPort [5], Jolocom [6], etc. Therefore, a universal resolver is needed to provide the unified DID resolution between different DID methods. Both the universal resolver and drivers for individual DID methods are orchestrated by the microservice-deploying platforms such as Kubernetes from Google[1] or Docker-Compose.[2]

However, due to the paramount complexity of leveraging many different security services and new system building blocks, we hypothesize that DID can introduce additional security risks and vulnerabilities during deployment. Prior research [7], [8] has examined the specific attacks on service endpoints in a DID document and DID authentication in IoT context. However, the research on the security analysis of the entire DID system has not been systematically studied. In this paper, we first analyze the data flow between DID components using Microsoft's threat modeling tool [9] to characterize the interactions and information flow in an entire DID system. For analysis, we use W3C's DID core standard [10] and VC data model [11], the open-source wallet platform developed by Hyperfabric Aries [12], DKMS [13], DID communication protocol [14] and the universal resolver [15]. In particular, we categorize the entire DID system into the following DID main component entities based on their functionalities: 1) DID Subject (User), 2) DID, 3) DID Document (DDO), 4) Identity Wallet Software, 5) Universal Resolver (UR), 6) DID method, and 7) Verifiable Data Registry (VDR).

Next, we investigate the underlying security threats for the above seven DID system component categories and analyze detailed attack surfaces. In particular, we identify the following seven attack domains as shown in Figure 2: 1) Wallet System Attack, 2) Phishing/Impersonation Attack, 3) Cache Poisoning/Pollution Attack, 4) Microservice Attack, 5) DDO forgery Attack, 6) VDR Partitioning/ Information-Centric Networking (ICN) Attack, and 7) Social Recovery Attack. Lastly, we discuss the possible countermeasures for those attacks. In summary, we make the following contributions in this paper:

- We propose a holistic view of the DID system to provide a comprehensive understanding of different building blocks and their interactions with the state-of-the-art DID system components.
- We identify the underlying seven major security attack domains by exploring vulnerabilities in the entire DID system on the path, starting from a user's query to the actual acquisition of the target DID document and its use.
- We also present the possible countermeasures and defenses to each of the security attack vectors in the DID system, providing directions for future research.

The organization of this paper is as follows. In Section II, we show the related work on the background of SSI with DID system and their security issues in using DID. In Section III, we present an overview of the DID system. In Section IV, we present a holistic view of the entire DID system and describe the interactions among individual DID components. In Section V, we examine the DID system's attack surface and identify the DID system's possible security threats. In Section VI, we walk through different countermeasures per each security threat identified and discuss the future work of our research. Finally, we offer our conclusion in Section VII. For the terms and abbreviations in this paper, the reader is referred to the nomenclature in Appendix.

II. RELATED WORK

In this section, we present the prior research related to the background of the emergence of SSI, DID systems to enable SSI, and their security issues.

A. DECENTRALIZED ID MANAGEMENT

Over the past few decades, *centralized identity* management (CIM) systems have been popularly used for web and network applications. However, the centralized approach would have a single point of failure and generally does not scale well. Therefore, federated identity management (FIM) scheme has emerged to address the above issues, which allows the federation across multiple organizations by bridging different identity providers and thereby achieving cross-organizational access to each service provider.

To further help users control their own identity information and enable the minimized exposure of their personal data, user-centric identity management (UIM) schemes are needed to verify and manage user's ID in a more user-controllable and privacy-respecting manner. The concept of the SSI [2], [3] management scheme has been proposed to further achieve these goals. SSI allows users to generate and manage their own identity data in a decentralized way. SSI is expected to leverage the state-of-the-art decentralized identity technologies, providing the DID as the fundamental layer for machine-verifiable unique identifiers. And VC provides secure, privacy-ensuring credentials along with DKMS for handling decentralized key management and UR for ensuring the successful retrieval of DDO across different DID methods. In particular, Kim *et al.* [1] provide the overview of the DID-based distributed ID management system over the blockchain. In Section III-A, we further delineate different ID management schemes.

B. VC OVER BLOCKCHAIN

Storing and sharing VC via a blockchain system has been studied by Takemiya and Vanieiev [16], where VC's integrity and non-repudiation can be ensured. The authors leveraged

[1]https://github.com/kubernetes/kubernetes
[2]https://docs.docker.com/compose/

the use of 128-bit cryptographic salt on top of the key stretching method of a password-based key derivation function 2 (PBKDF2) [17]. The key pair to access blockchain is generated from the PBKDF2 method. The VC is then separated into public and private parts; the former is added with another salt and hashed using SHA3-256 [18]. The salted hash of the VC's public part is published in a transaction to the blockchain, where the key pair is used for creating and propagating the transaction in the blockchain. This design effectively increases pseudonymity and VC protection in a public blockchain setting. Besides, the notarization method for VC can be utilized by stacking each notary's non-repudiation signature to the VC and publishing it in the blockchain.

C. SECURITY CHALLENGES IN USING DID

For DID authentication, Pennino *et al.* [7] identified several issues when binding a user's real identity to the service endpoint in a claimed DDO. For example, the user's *proof-of-possession* (PoP) of the service endpoint was used with a digital signature scheme to establish and publish the proof to a blockchain system. They proposed a challenge-response cycle through which a user, i.e. the DID subject, demonstrates their DID ownership, using the PoP of the claimed service endpoint to enable a secure DID authentication. However, the proof establishing process requires a committee consensus, where a set of predefined authorized service endpoint owners are *randomly* elected as committee members. To publish the PoP, the committee members all need to be on-line in the previous proof establishing process. They identified that this process could incur a considerable processing delay if some partial committee members are absent, and thereby the committee consensus is not processed in time. This procedure can introduce a new attack. Besides, it will increase the on-chain data traffic and system complexity for the committee's governance framework. Another research on DID authentication was conducted by Omar [8]. The author suggested using HMAC key derivation function (HKDF) [19]-based DID mutual authentication method to counter impersonation attack, replay attack, and reflection attack in IoT device authentication.

Regarding security and privacy issues in DID, Halpin [20] identified the threats on the use of DID and VC on COVID-19 immunity passport [21]. However, storing hashed VC containing personal medical information in a public blockchain can introduce a significant privacy issue. In particular, the hash of the personal data without appropriate salting can violate the privacy law such as General Data Protection Regulation (GDPR) [22]. For anonymous authentication, Biswas *et al.* [23] proposed a ring signature-based Multi-DID design to encrypt the transactions between an Electric Vehicle (EV) user and charging station, while using a mixed-use of Sovrin [4] and, for ephemeral DIDs, the blockchain-independent *Peer DID method* [24] to prevent the Man-In-The-Middle attack. However, the limitation of this work is the excessive resource consumption on the processing of the ring signature and storing of the transaction data required for the design.

Consequently, none of the existing studies explore the systematic attack surfaces according to the entire DID system components.

III. OVERVIEW OF DID SYSTEMS

First, we overview different identity management schemes, and describe the major components in the DID-based SSI systems.

A. DIFFERENT ID MANAGEMENT SCHEMES

First, the CIM system was introduced, which brought the inchoate concept of an identity provider (IDP) and single sign-on (SSO) [25] to meet the burgeoning needs of managing multiple identity credentials across different internet service providers. Examples of CIM system subsume digital certificates, OpenAM (successor to OpenSSO), PGP [26], Kerberos [27], and Radius [28]. However, users had no other recourse but to fully trust the centralized IDP, despite the limited control over their identities, that it will not misuse their identity. Also, the CIM system lacks cross-organizational accessibility, expedited to transition to the federated identity management system.

To circumvent the limitation in the CIM system, the FIM system has emerged, where the federation means a circle of trust that was made available by bridging IDPs and sharing identities across different service providers. The era of the federated identity coined SAML [29] of XML-based data format standard for exchanging authentication and authorization data among parties such as IDPs (also known as asserting parties (APs)) and service providers (SPs) (also known as relying parties (RPs)), and OIDC, the identity authentication layer on top of Open Authorization (OAuth) 2.0. Nevertheless, the FIM system still failed to let users fully control their own identity data, and there are still too many intermediate authorities. The drawbacks of the superfluous user data exposure among federated IDPs and less controllability of the user on their own identity data led to the new identity scheme, which is the UIM system.

The UIM system has emerged by allowing users to complete control over their digital identities, mainly focused on user's consent and interoperability. OpenID and Facebook Connect [30] are the examples of this user-centric identity scheme enabling users to broadly use their IDs of one public website in other third-party SP's websites. However, the same problem still exists, where users have to depend entirely upon the trusted identity providers for their personal data.

To further overcome such shortcomings of central authority and minimize user information exposure, the SSI has proposed providing full user identity autonomy and control over their own identity. According to each contextualized relationship and user-determined delegation level, SSI enables users to harness security and flexibility about their own identity credentials (or attributes).

Moreover, SSI leverages blockchain technology and several new emerging standards, at the epicenter of which comes the DID [10] as the main building block. Those crucial new emerging standards for SSI are as follows: (1) VC [11] for the globally unique tamper-evident credentials of claims by issuers, (2) DKMS [13], which is a decentralized version of centralized cryptographic key management systems (CKMS) for ensuring safe key custodianship while enforcing the anti-correlation capability lacking in CKMS. In this work, we specifically focus on the security issues of the DID-based SSI, along with their implications when deployed in real-world scenarios.

B. DID SYSTEM BUILDING BLOCKS

1) DID

A DID is the decentralized identifiers, which is a self-registered globally unique identifier pointing to a DDO that contains the information of the DID subject. DID is comprised of three syntactic components (e.g., *did:example:ABC*), which are URI scheme identifier (*did*), DID method name (e.g., *example*), and DID method-specific identifier (e.g., *ABC*).

The DID method specifies how the DID is being created, resolved, and deactivated, and how the DDO is written and updated. DID is designed to separate the proof of the user's identity from credentials. Thus, DID allows the identity owner to prove credentials in a selective disclosure through cryptographic methods such as Zero-Knowledge Proof (ZKP) and digital signature schemes.

Also, the DID is recommended to be unique per each situational context and relationship to provide privacy. Therefore, DID can be adopted for the standard identity scheme for various cross-domain environments such as IoT, Smart Factory, Smart City, Cooperative-Intelligent Transportation System (C-ITS), Smart Government, E-Health, and 3rd Party Data Consumer and more.

2) DDO AND DID SUBJECT

A DDO is the DID document [31] composed of a data set that describes the DID subject, which is retrieved by a DID resolver. The DDO contains the core properties of DID subject, DID controller, verification method, verification relationship, and service endpoint. Also, the DDO follows the representation form as specified by the core data model of DID [10]. The representation forms are JSON, JSON-LD, and Concise Binary Object Representation (CBOR).

A DID subject is the primary entity that holds one unique DID. Also, the DID subject is identified by a DID and described by the DDO. A DID subject can be not only a person but also a group, organization, or even physical object that requires a unique identifier without depending on a central authority. The DID subject can authorize a DID controller to make changes on behalf of the DID subject. The DID subject and DID controller relationship is also applicable to the data subject and data controller in GDPR [22].

3) VC

VC is the verifiable credentials that contain a set of claims, credential metadata to cryptographically prove the issuer, and proofs of the issuer's digital signature. In VC, the credential subject is identified by DID and likewise other entities composing the claims are using their DIDs for identification.

The assertion of the claim is expressed using *subject-property-value* relationships. For instance, *someone* (subject) is *alumni of* (property) *XYZ university* (value). The VC should be securely stored within an identity wallet. To abide by privacy, the credential subject can expose only some part of the VC in verifiable presentation form. Besides, it can combine multiple VCs to generate a single verifiable presentation, which can be passed to verifiers to prove the authenticity of VC's claims. To further enhance security, the credential subject can also add its own digital signature to the proof of the original VC's issuer to protect against a replay attack.

4) DKMS

DKMS is a new cryptographic key management approach intended to be used with blockchain and Distributed Ledger Technologies (DLTs). Unlike the conventional CKMS, DKMS does not rely on X.509 certificate and central authority. In fact, there is no central entity that protects and distributes the keys on behalf of the identity owner. Instead, each identity owner is responsible for the key management via an identity wallet and delegation to each edge agent or cloud agent authorized by the identity owner.

To present the ownership of credentials, one special key called *link secret* is used in every credential of an identity owner in a blinded form. Thereby, the link secret proves that all those credentials were issued to one logical identity owner even though the identity owner uses different wallet SW on different devices. Also, DKMS handles the key revocation along with the agent revocation as well as the key rotation methods and recovery schemes. DKMS is currently being developed and hosted by Hyperledger Aries project [13].

5) AGENT

An agent is the delegated entity by a DID subject, which is responsible for agent-to-agent DID communication, operation of cryptographic functions of DID identity wallet, and use of credentials with authorized identity sovereignty according to each relationship. There are two types of agents: edge agent and cloud agent. Edge agent is within the local user's wallet software, while cloud agent is in the cloud to provide extended features such as key management, identity wallet backup / recovery, data storage, and the 24/7 DID communication even when the peer edge agent is not reachable or offline.

The agents can access the partial or whole credentials of the identity owner based upon the authorization policy, which is stored in the public ledger. When checking the authorization of each agent, there is no indication on which identity owners

the agent is belonged to, because the agent is using the blinded commitment of secret value to prove the authorization. Thereby, the ZKP is ensured without revealing the secret value itself. However, we show that various impersonation attacks and phishing attacks are feasible against various edge agents in Section V-B.

6) IDENTITY WALLET

The identity wallet software (SW) stores the credentials including VCs, DID signing and verification keys, link secret, each edge agent's policy keys, and secret value commitments. The wallet provisions each edge agent to handle each different relationship. Besides, the wallet has an interface for tamper-proof secure element or TPM, which each agent will interact with, using cryptographic secret key management.

Also, it provides each edge agent with storage for each microledger to record DID events over each agent-to-agent communication. In the case of Hyperledger Indy [32], the wallet data is stored with a tagging mechanism, where a tag is defined by key-value pair. Then, the wallet data is queried and filtered using a JSON-based query language, i.e., Wallet Query Language (WQL) [33], similar to MongoDB's syntax. Also, WQL can be mapped to SQL, or other types of DB languages depending on the specific pluggable storage used by the wallet SW.

In particular, *Connect.Me* [34] by Evernym is the first Sovrin-based digital wallet. *Connect.Me* provides aforementioned features, which are also partly adopted to the open-source wallet projects such as Hyperledger Aries [35] (for Agent, Key Management, DID-to-DID communication Protocol), and URSA [36] (for the shared cryptographic library including Camenisch-Lysyanskaya signature, zero-knowledge proofs used by Indy in exchanging credentials).

7) MICROLEDGER

A microledger is the implementation of the Relationship State Machine (RSM), which is a very small local Merkle tree-based ledger. As shown in Figure 2, microledger resides in each agent secured by DID wallet, and each agent is recommended, for privacy, to have a unique DID for every relationship. Since a relationship requires two parties, each relationship shall have two microledgers.

Thus, each party shall record the DID events such as *New DID Created, Agent Key Updated, Authorization Set / Updated / Revoked, Recovery Policy Set*, and *Added Recovery Key* on their microledger. Then, the DID events are organized in a Merkle tree. Thus, each party maintains their half of the relationship and replicates their microledger to the other party. Therefore, if there are n relationships, it will have a total of $2 \times n$ microledgers.

8) UR

A UR is a universal resolver [15], which is a DID resolver that provides the unified lookup and resolution of DIDs across different DID methods. The UR has a collection of DID method drivers that interface with various identifier system providers. A DID or DID URL are parsed across different DID methods and the corresponding DDO or DID resolution result are returned as an output. The UR has been developed and hosted by Decentralized Identity Foundation (DIF), where currently DIF has deployed two types of URs as a community service based on Amazon Web Services (AWS[3]) and IBM Cloud.[4]

In essence, the DID resolution mechanism is functionally similar to DNS resolution in that the local stub resolver invokes the remote recursive resolver. Hence, DID UR can have a layer of caches to provide the fast resolving output for a given DID input. Also, it contains a collection of drivers to ensure interoperability among different DID methods such as Sovrin [4], Uport [5], Jolocom [6], etc.

In particular, the drivers are comprised of each DID method-specific wrapping functions. Thus, regardless of the input DID using different DID methods, it bootstraps to return the DID resolution output accordingly. Since UR is analogous to the DNS resolver in resolving mechanism, UR is vulnerable to the similar cache poisoning and pollution attack performed on the DNS resolver.

9) VERIFIABLE DATA REGISTRIES (VDR)

A VDR [37] is the verifiable data registries, which is a storage system facilitating the creation, update, verification, and revocation of DIDs, DDOs, and VCs. Besides, VDR provides the storage of the credential schema of a VC, for the purpose of the verification of the VC. VDR can be designed as a distributed ledger, peer-to-peer file system, decentralized file system, or any database. The specific implementation of VDR may vary according to each different DID method and its mechanism. In this work, we assume blockchain or ICN as an underlying data registry for VDR for convenience to illustrate the VDR interactions within SSI.

IV. HOLISTIC VIEW OF THE ENTIRE DID SYSTEM

In this section, we present the holistic view of the DID system in detail using the Data Flow Diagram (DFD) [38] in Figure 1. We use DFD to describe the data flow and interactions among different entities in the DID system and analyze the comprehensive attack surfaces on those later.

We assume the most common use case scenario, where the local user A's edge agent (EA) aims to communicate with the remote party B's EA through DID communication and perform the wallet backup and recovery, as shown in Figure 1. First, the A's EA queries the DID of B's EA to the Community Resolver (CR) within UR. Then, the DID method returns the DDO of B's EA to the UR through the interaction with VDR. Thereafter, the A's EA retrieves the DDO from the UR and establishes the DID communication by referencing the data in the DDO. Besides, the A's EA encrypts the DID wallet for backup and stores the backup data in the cloud agent (CA). Lastly, the A's EA restores the DID wallet through the interaction with the CA via a social recovery process.

[3]https://dev.uniresolver.io/
[4]https://resolver.identity.foundation/

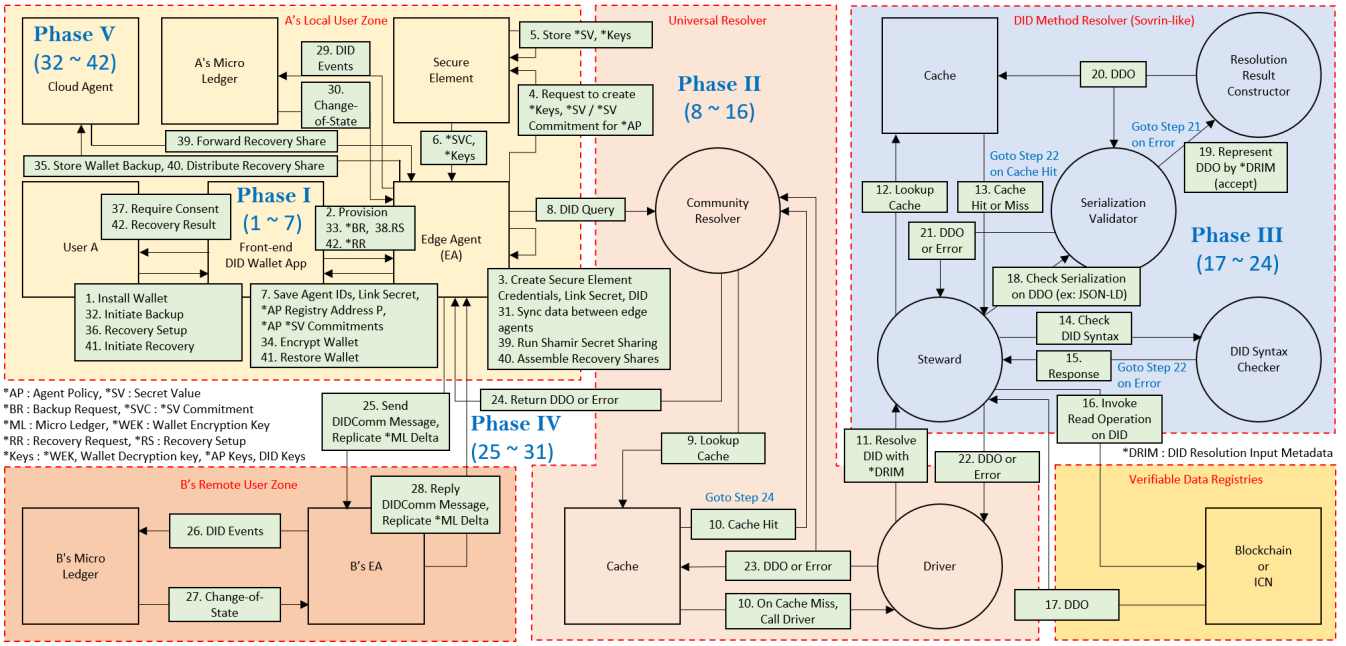

FIGURE 1. **The holistic DID system view using DFD, where the EA of user A attempts to communicate with the EA of user B through DID communication via five operational phases (Phase I - Phase V) across Local User Zone, Remote User Zone, Universal Resolver, DID Method, and Verifiable Data Registries, and the green boxes indicate the specific sequential parameters and interactions among different functional blocks.**

In particular, we divide the entire DID system into five blocks as follows: 1) *A*'s Local User Zone, 2) *B*'s Remote User Zone, 3) UR, 4) DID Method Resolver, and 5) VDR, as shown in Figure 1. In particular, the Local User Zone comprises six components: user A, front-end DID wallet app, *A*'s EA, microledger, secure element, and CA. Also, UR has the following three main components, as shown in Figure 1: a community resolver, a cache, and a driver for the DID method. For the DID method, we choose Sovrin [4] as a representative example to demonstrate the interactions of DID resolution, where a Sovrin-like DID resolver is comprised of the following five main components: a steward, cache, DID syntax checker, serialization validator, and resolution result constructor. For specific details of DID resolution, we also utilize the reference DID resolution algorithm from the W3C standard [39]. For VDR, we assume the network type is either blockchain or ICN. And the Remote User Zone is comprised of B's EA and microledger.

To describe detailed interactions among these components, we divide the operation phase into five phases (Phase I to V), as shown in Figure 1. The data flow starts from the installation of a DKMS-compatible identity wallet (Phase I), showing the process of DID query via UR and DID method (Phase II). Also, the retrieval of DDO (Phase III) and agent-to-agent DID communication [14] are shown in the following Phase (Phase IV). Lastly, wallet backup and recovery interactions are presented in Phase V.

A. PHASE I. INITIALIZATION OF IDENTITY WALLET AND AGENT

In Phase I, the user *A* first installs the wallet SW and initiates the process to provision the first EA, as shown in Step 1 and 2 in Local User Zone. Then, the EA creates a credential for Secure Element (SE), link secret, and DID for agent-to-agent communication in Step 3, where the link secret is used to establish each DID relationship through a blinded commitment, which can be used across different identity wallets that belong to the same identity user. Thus, the link secret can prove that the claimed credentials, or each identity wallet, are all belonged to the same logical identity.

In Step 4, the EA requests the SE to create a secret value (SV) and a SV commitment (SVC) for Agent Policy (AP). In addition, the EA requests for creating different types of keys such as wallet encryption and decryption keys, AP keys, and DID keys for signing and verification. In Step 5, the SE stores 1) the SV and 2) private (signing) keys of DIDs, a wallet decryption key, and agent policy keys. Then, the SE returns the wallet encryption key for wallet backup, SVC for AP, and DID verification keys in Step 6.

In Step 7, the EA requests the front-end DID wallet app to store agent IDs, a link secret, and an AP registry address $\mathcal{P}$, where the address $\mathcal{P}$ is pointing to the stored AP location in the public ledger. Then, the AP determines the authorization level permitted to each agent per each different credential. Each time when a new agent is added, the new agent stores its SVC in the *PROVE* section of the AP at the address $\mathcal{P}$.

As a result, the DID system achieves SSI, preserving the privacy of the identity system via *Confidentiality and Control* [40]. The confidentiality is satisfied such that the user's credential is secured within the DKMS-compatible identity wallet. Also, the control is achieved via minimum *controllable* disclosure of the proof.

Also, using the SVC in Step 6, the EA can prove the authorization from the user, without disclosing the real secret value. For herd privacy, the SVC with the AP address is also converted into another commitment, called AP

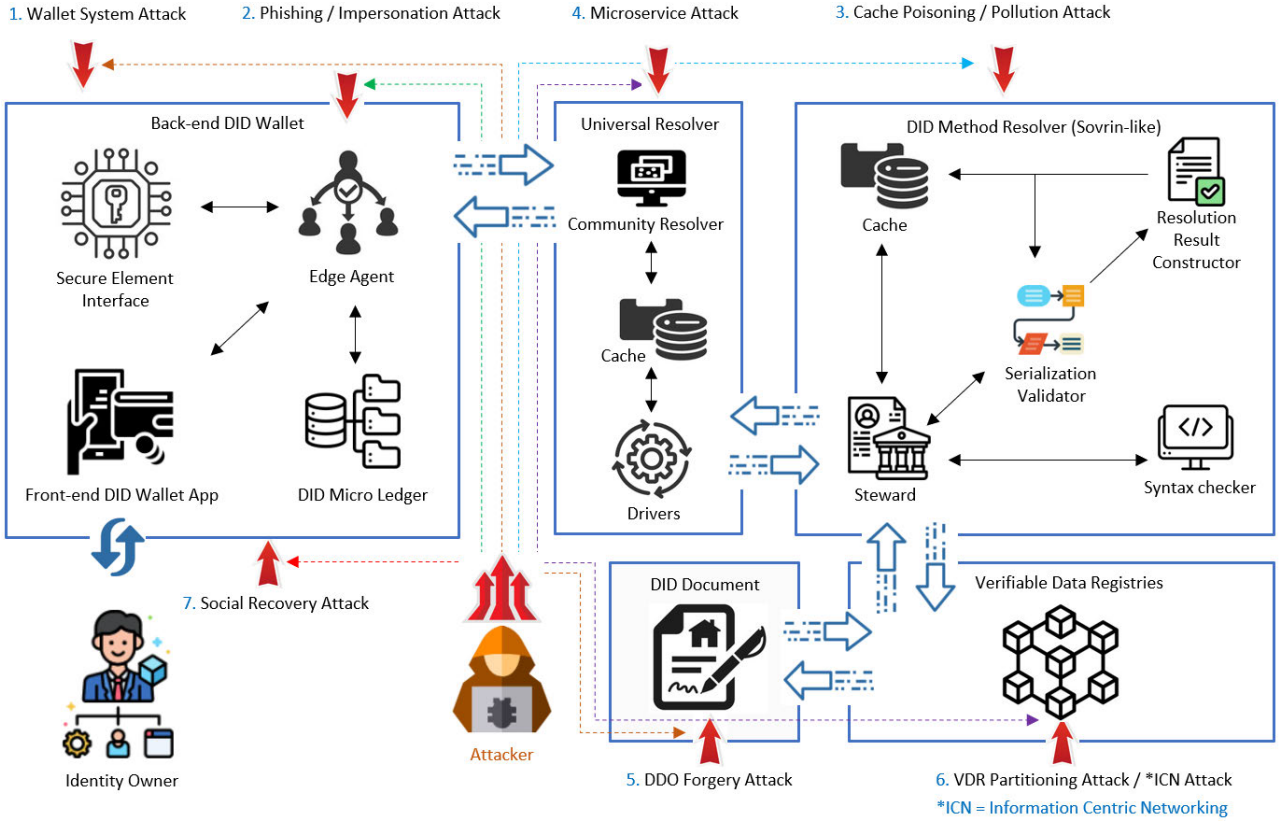

FIGURE 2. The entire DID system's overview with each possible attack surface of vulnerabilities, where those attack vectors are divided into the seven areas in the proposed DID system: 1. Wallet System Attack, 2. Phishing / Impersonation Attack, 3. Cache Poisoning / Pollution Attack, 4. Microservice Attack, 5. DDO Forgery Attack, 6. VDR Partitioning Attack / ICN Attack, and 7. Social Recovery Attack.

Address Commitment (APAC). Then, the APAC is stored in the global prover registry of the public ledger shared by all identity owners.

In particular, the DID keys, in Steps 4 and 6, are composed of a verification key and a signing key, based on ED25519 [41], where ED25519 is the signature scheme of Edwards-curve Digital Signature Algorithm (EdDSA) using SHA-512 (SHA-2) and Curve25519. Next, the DID and verification key are passed to the other party each time when a new relationship is established. Lastly, the EA creates a CA to pair with, where CA stores the encrypted backup of the DID wallet as shown in Step 35 in Phase V. From this, the CA creates the recovery endpoint only known to the CA itself. Also, the CA forwards messages between the EA of the local user and the remote party's CA or EA.

As shown in Figure 1, the DID wallet encompasses essential components such as agents, secure element interactions, and microledgers. Therefore, the wallet can be possibly exploited by various attacks. We describe the details of the DID wallet attack surface in Figure 2 and possible attacks in Section V-A.

B. PHASE II. DID QUERY TO UR AND DID METHOD INTERACTION

In Phase II, the EA of the local user *A* receives a DID connection invitation from a remote party *B*. Then, *A*'s EA sends the DID query to the CR of UR, as shown in Step 8. Upon receiving the DID query, in Step 9, the CR first checks the cache. In Step 10, if the cache hits the DID query, then CR immediately returns the stored DDO, as shown in Step 24, where the DDO can be returned with other metadata (e.g., DDO metadata and DID resolution metadata based on the input metadata of resolution) in Step 11 in Figure 1. Otherwise, on the cache miss in Step 10, the CR invokes the resolution process by first choosing the driver that matches the DID method given in the input DID. In the proposed DID system, Sovrin is selected for the DID method.

In Step 11 at the DID Method block shown in the top right corner in Figure 1, the driver requests the resolution of the DID to the DID method, passing the DID and DID Resolution Input Metadata (DRIM). Upon successful reception of those, the *accept* [42] property of the DRIM can be passed to determine the representation form [10] of the returned DDO in Step 22. In Step 12, the steward in Sovrin checks the cache if the DDO for the input DID is present. Upon cache hits in Step 13, the steward immediately returns DDO. Otherwise, the steward prepares to resolve the DID query. In Step 14, the steward checks if the input DID is conformed to the standard DID format. If the syntax error occurs, then the steward returns the corresponding error in Step 22. Otherwise, in Step 16, the steward invokes a read operation to VDR with the input DID. If the input DID was a public

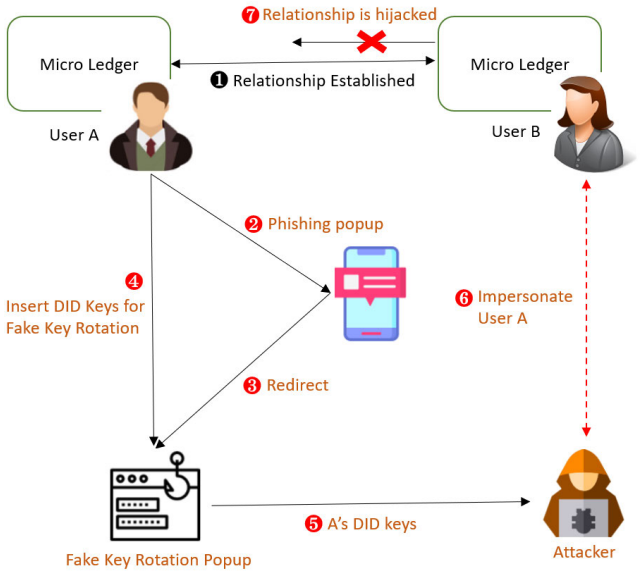

FIGURE 3. Phishing and impersonation attacks ($\mathcal{A}_8$) being executed, while exploiting the DID signing key via UI-redressing and data exfiltration.

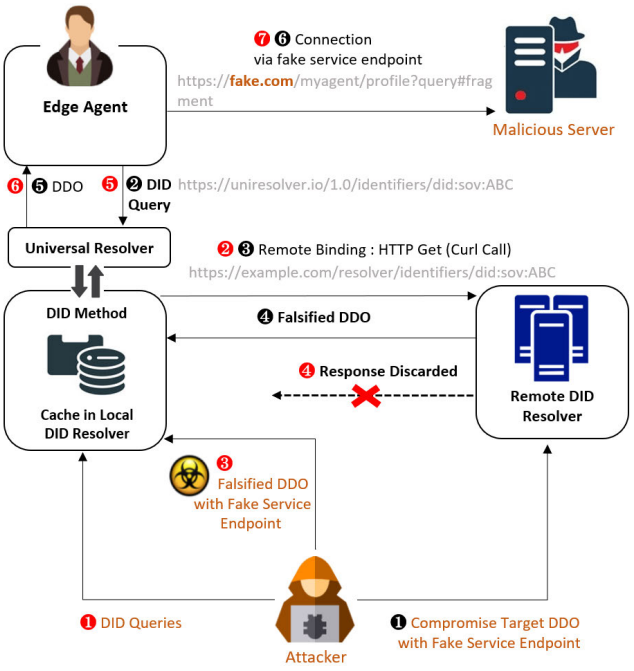

FIGURE 4. Cache poisoning attack ($\mathcal{A}_9$) on the DID method server via proxied DID resolution with each sequential step, for Scenario 1) showing in red circles, and Scenario 2) showing in black circles, to illustrate two different cache poisoning attack scenarios.

DID URL [43], then the DID URL needs to be dereferenced [44] by the DID method, where DID URL includes more information on specific resources to be referenced in DDO. The URI components of the DID URL such as *path*, *query*, and *fragment* are used to identify the resources in DDO.

For practical implementation, the UR and the drivers are containerized applications orchestrated by microservice-deploying platforms such as Kubernetes or Docker-Compose, where the microservices are easy and flexible to be deployed across different systems. However, the microservices are vulnerable to system attacks, exploiting homogeneous deployment from the base image, misconfiguration, failed audit, etc. Besides, using a cache layer in UR and DID method can also introduce the exploitability of cache poisoning or cache pollution attacks. We discuss the detailed attacks on the caches in UR and microservices in Section V-C and V-D, respectively. And the corresponding attack surfaces are also illustrated in Figures 4 and 5.

C. PHASE III. DDO RETRIEVAL INTERACTION

As discussed, VDR can be any kind of database, peer-to-peer networks, or other forms of trusted data storage. In this work, we assume that the VDR uses the blockchain or ICN for analysis. The first VDR operation is shown in Step 17, where the VDR processes the read operation and return the DDO to the steward. Then, in Step 18, the serialization validator in the DID method validates that the DDO is conformed to the serialization formats, where the formats can be JSON, JSON-LD, and CBOR, as specified in the DID core data model standard [10].

If the serialization validator in Step 21 returns an error, the steward will forward the error to the driver in UR in Step 22. Otherwise, in Step 19, the DDO is passed directly to

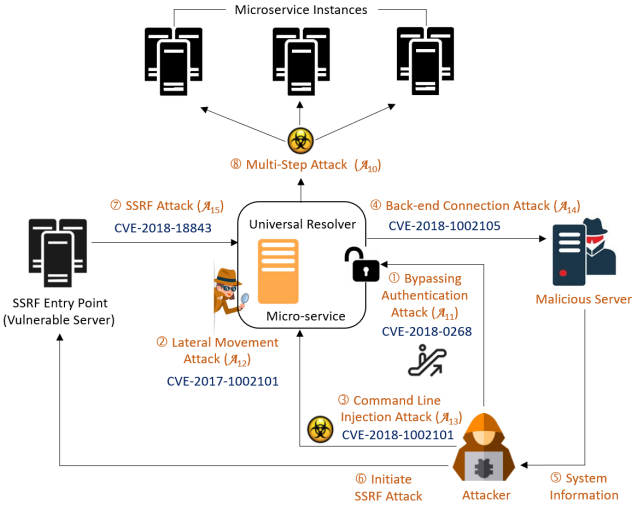

FIGURE 5. Microservice Attack on UR and DID method driver. Those attack vectors are including Multi-step attack ($\mathcal{A}_{10}$), Bypassing authentication attack ($\mathcal{A}_{11}$), Lateral movement attack ($\mathcal{A}_{12}$), Command line injection attack ($\mathcal{A}_{13}$), Back-end connection attack ($\mathcal{A}_{14}$), and SSRF attack ($\mathcal{A}_{15}$).

the steward. However, if the DRIM in Step 11 has an *accept* [42] property and the value is *application/ld+json;profile ="https://w3id.org/did-resolution"*, then the DDO returns the DID resolution result in Step 22. The returned DID resolution result contains DID resolution metadata, DDO metadata in addition to the DDO. If the *accept* property has the value, then the DDO is passed, as shown in Step 19, to the resolution

result constructor to further make the representation form of DID resolution result.

Next, the resolution result constructor returns the DDO to the serialization validator and updates caches with the DDO in Step 20. Then, the serialization validator returns the DDO to the steward in Steps 21, and the steward returns the DDO to the driver of UR in Step 22. In Step 23, the driver passes the DDO to CR and also updates the cache with the DDO. Eventually, in Step 24, the CR returns the DDO to the EA of the user.

However, during the DDO retrieval process in VDR, the partitioning attack and cache pollution attack can occur in blockchain and ICN network, respectively. The partitioning attack in blockchain results from the skewed distribution of the full nodes participating in the network maintenance. Also, the cache pollution attack in ICN can occur due to the frequent random DID queries or repeated queries of unpopular DID set to disrupt the cache operation for DID service. We investigate more in-depth attacks in Section V-F.

D. PHASE IV. AGENT-TO-AGENT COMMUNICATION INTERACTION

In Step 25 in Remote User Zone, *A*'s EA of Local User Zone communicates with *B*'s EA by sending a DIDComm [14] messages and the delta of microledger. The DID communication uses a message-based protocol between EAs. In other words, each agent exchanges a series of messages. Consequently, such an agent-to-agent communication way can lead to the replication delay when exchanging the delta of microledger to share each EA's DID events over the asynchronous communication where the delta of microledger contains the DID events recorded from each EA. Here, the delta can be represented as the updates of microledger organized in a Merkle tree and exchanged via *authenticated encryption* [13] using the verification key of each EA. In Step 26, *B*'s EA stores the DID events in the relationship between *A* and *B*. The microledger delta of DID events received from *A* is also stored in *B*'s microledger to make sure each party of *A* and *B* shares the same copy of DID events of the other.

In Step 27, *B*'s microledger sends a change of state as a response to the recording operations in Step 26. In Step 28, *B*'s EA sends a reply to the DIDComm message and replicates *B*'s microledger delta to *A*'s EA. Likewise, in Step 29, *A*'s EA will store the DID events in *A*'s microledger, and the microledger delta received from *B*. As shown in Step 30, *A*'s microledger sends a state change as a response to *A*'s EA. Then, *A*'s EA checks that the DID events are in sync with *B*'s EA in Step 31.

In this phase, we assume that the DIDComm is transport-agnostic, and the protocol can operate on top of various transport layers such as HTTP(s) 1.x and 2.0, Bluetooth, NFC, Advanced Message Queuing Protocol (AMQP), WebSockets, and more. Also, as the EA is directly involved into the agent-to-agent communication, the region around EA becomes the target of the attacks such as impersonation or phishing attacks. In particular, we examine the attacks that exploit the vulnerabilities of the latency inherent to the DID communication protocol and the attack surface for the agent-to-agent communication in Figure 3.

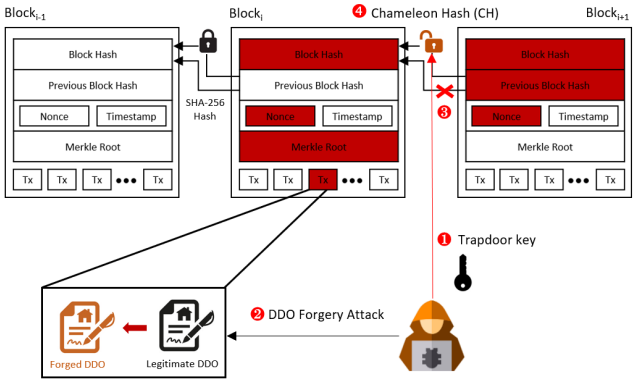

FIGURE 6. DDO Forgery Attack ($\mathcal{A}_{16}$) on Redactable Blockchain with sequential steps of the attack scenario. The attacker exploits the CH using the trapdoor key to launch the forgery attack on DDO.

E. PHASE V. WALLET BACKUP AND RECOVERY INTERACTION

At Local User Zone, we assume the user initiates the backup process for the identity wallet via the front-end DID wallet app (Step 32). Then, the backup request is forwarded to the EA in Step 33. The EA encrypts the wallet using the wallet encryption key in Step 34, where the key was already received from the secure element in Step 6 in Phase I. Then, the EA stores the encrypted backup of the wallet in the cloud via the CA in Step 35. Thereafter, in Step 36, we assume that the user initiates the recovery for the encrypted wallet backup due to unexpected compromise of device or theft and the user selects the social recovery when asked by the front-end DID wallet app in Step 37.

We assume that the *A*'s EA has already set up the social recovery before Step 36. For the recovery setup, the decryption key for the encrypted wallet backup along with the link secret of the user and special recovery endpoint are packed into a recovery data file. The recovery endpoint is set up by the CA when the EA requests CA to create the recovery buddy invitation, where the buddy means a social trustee of the user.

In the social recovery setup, the recovery endpoint is used to communicate with the recovery buddies. The recovery endpoint is only known to the CA itself. Then, the recovery data file is sharded into recovery data shares based on Shamir Secret Sharing Scheme (SSSS) [45] using Eq.-(1). Thereafter, the EA requests the CA to forward the recovery invitation to the recovery trustees. The recovery invitation contains the recovery data share to be distributed to the recovery trustees.

In Step 38, the EA receives the recovery request. The EA will request CA to start the social recovery process and collect the recovery data shares from the trustees. In Step 39, the CA forwards the collected recovery data shares to the EA. Then, the EA assembles the recovery shares in Step 40. Using

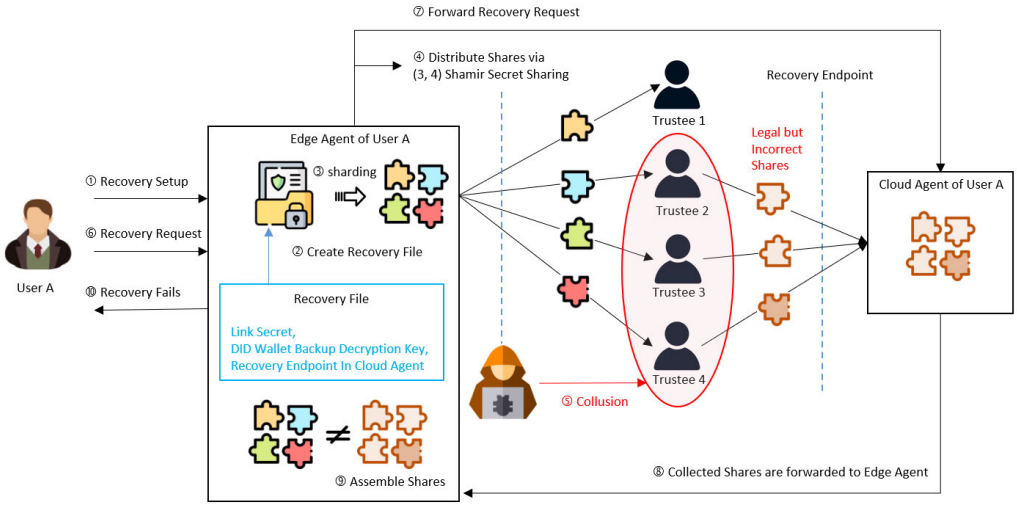

FIGURE 7. Social Recovery Attack ($\mathcal{A}_{18}$) with sequential steps of attack scenario. The attacker exploits the weakness of Shamir Secret Sharing Scheme and colludes with other recovery shareholders in Step 5. Thus, the reconstruction of the recovery file becomes legal but incorrect. Consequently, the social recovery procedure is failed.

the reconstructed decryption key, the EA can finally restore the wallet in Step 41. Consequently, the recovery result is forwarded to the user in Step 42. In this social recovery step, however, several attacks are feasible and we describe the details of the social recovery attack in Figure 7.

V. SECURITY THREATS AND ATTACK SURFACES

In this section, we investigate the security threats comprising the attack surfaces of the DID system, where Figure 2 illustrates the components of the entire DID system and the attacks that can be mounted on each component. As shown in Table 1, we identify eighteen different attack vectors ($\mathcal{A}_1$~$\mathcal{A}_{18}$) and categorize them into the following seven categories based on the functional blocks attacks are performed on:

1) Wallet System Attack
2) Phishing / Impersonation Attack
3) Cache Poisoning / Pollution Attack
4) Microservice Attack
5) DDO Forgery Attack
6) VDR Partitioning Attack / ICN Attack
7) Social Recovery Attack

A. WALLET SYSTEM ATTACK ($\mathcal{A}_1 \sim \mathcal{A}_7$)

First, as discussed, the DID identity wallet plays a pivotal role in the SSI system to provide security and privacy in the local user zone. The wallet system comprises EA(s), SE interface, microledgers, and also built-in user-interactive applications, as shown in Figure 2. Therefore, attackers can precisely target the wallet system and it can become a honeypot for many potential attacks via various types of spoofing, data tampering, repudiation, malware intrusion, information disclosure, and data exfiltration measures, etc. Hence, we start the analysis of the wallet system attack by examining a key exposure attack.

1) KEY EXPOSURE ATTACK ($\mathcal{A}_1$)

An attacker can trigger a key exposure attack via various data exfiltration measures to target the sensitive data such as DID keys (e.g., agent-to-agent keys of signing key, verification key, etc.), DID wallet keys, DID agent keys, a link secret, private keys, etc. In particular, Davenport and Shetty [46] used a Markov chain model to show how an external attacker can infiltrate into an air-gapped target device, to establish a covert channel and exfiltrate data, when a blockchain-based wallet is running. Hence, they show that it is feasible for attackers to leverage the lack of perimeter security of the wallet system in the device. For the DID wallet system, similar attacks by Davenport and Shetty [46] can be performed to exfiltrate private keys through an established communication channel.

Also, depending on data transfer bandwidth in the wallet system, the attacker can secretly infect the wallet system with the exfiltration malware, which can make themselves covert via code obfuscation, debugger detection, execution environment-aware binary packing, virtual machine detection, etc.

2) MAN-IN-THE-MIDDLE (MITM) ATTACK OVER DID COMMUNICATION ($\mathcal{A}_2$)

The wallet system is also vulnerable to MITM, where the typical MITM attack involves two endpoints and the third-party attacker during DID communications. In DID agent-to-agent communication, the initial setup messages for invitation between the two endpoints are in plain texts as described in Step 25 in Figure 1. This is because both entities have no trusted communication channel yet. Only when a connection request based on the invitation is completed and each DID is exchanged [47], the public key-based encryption can be used.

Hence, problems can occur for *trust on first use* [48] or *blind trust before verification* [49] cases, which implicate that the user has to blindly trust the other party

TABLE 1. The categorization of DID system components and their potential security threats. We specifically identified eighteen attack vectors across the seven attack domains.

Main Category	DID System Components	Wallet System Attack	Phishing / Impersonation Attack	Cache Poisoning /Pollution Attack	Micro service Attack	DDO Forgery Attack	VDR Partitioning / ICN Attack	Social Recovery Attack
User	DID Subject	$\mathcal{A}_1$~$\mathcal{A}_7$	$\mathcal{A}_8$	$\mathcal{A}_9$	$\mathcal{A}_{10}$~$\mathcal{A}_{15}$	$\mathcal{A}_{16}$	$\mathcal{A}_{17}$	$\mathcal{A}_{18}$
DID	Decentralized Identifiers		$\mathcal{A}_8$	$\mathcal{A}_9$		$\mathcal{A}_{16}$		
DDO	DID Document			$\mathcal{A}_9$		$\mathcal{A}_{16}$	$\mathcal{A}_{17}$	
Identity Wallet Software	Front-end Wallet App	$\mathcal{A}_1$~$\mathcal{A}_7$	$\mathcal{A}_8$					$\mathcal{A}_{18}$
	Microledger	$\mathcal{A}_1$~$\mathcal{A}_7$	$\mathcal{A}_8$			$\mathcal{A}_{16}$		$\mathcal{A}_{18}$
	Edge Agent	$\mathcal{A}_1$~$\mathcal{A}_7$	$\mathcal{A}_8$			$\mathcal{A}_{16}$		$\mathcal{A}_{18}$
	Secure Element Interface							
Universal Resolver	Community Resolver				$\mathcal{A}_{10}$~$\mathcal{A}_{15}$			
	Cache			$\mathcal{A}_9$				
	Drivers for DID Methods				$\mathcal{A}_{10}$~$\mathcal{A}_{15}$			
DID Method Resolver (Sovrin-like)	Syntax Checker							
	Cache			$\mathcal{A}_9$				
	Steward						$\mathcal{A}_{17}$	
	Serialization Validator							
	Resolution Result Constructor							
VDR	Blockchain / ICN			$\mathcal{A}_9$		$\mathcal{A}_{16}$	$\mathcal{A}_{17}$	

before DID connection [47] is made. In such an environment, the attacker can insert himself in the middle and trigger the MITM attack at the initial connection phase in Step 25 in Figure 1 during Phase IV. In addition, the *mediator* and *relay* [50], supporting routes of multiple transports for cross domain inter-operability [51] in agent-to-agent communication, also open another security loophole for the MITM attacker to exploit.

Besides, if the key exfiltration attack in $\mathcal{A}_1$ is successfully carried out, then the attacker can further establish a backdoor channel to monitor the agent-to-agent message traffic. Thus, the miscreant can trigger MITM attacks by eavesdropping and manipulating the data in the agent-to-agent communication. Moreover, since the DID agent-to-agent communication is transport-agnostic [14], the MITM attack can be executed in various network types. Conti *et al.* [52] show the wide range of MITM over underlying network communication protocols.

Thus, based on these MITM tactics on the various communication channels, the attacker can intercept the message-based DID communication and manipulate the messages at their disposal in the DID system. Furthermore, it is possible to carry out the MITM attack in other types of DID communication systems involving UR, SSL/TLS, and Board Gateway Protocol (BGP), as follows:

- DID Spoofing-based MITM attack via UR.
- DID SSL/TLS MITM attack via two separate SSL connections maintained by the attacker.
- DID BGP MITM attack via traffic tunneling through attacker's Autonomous Station (AS).

3) DID WALLET REVERSE ENGINEERING ATTACK ($\mathcal{A}_3$)

Also, it is possible for an attacker to use various tools to reverse engineer SW binaries of the DID wallet. For instance, in android-based DID wallet applications, the attacker can use a *bakSMALI* [53] to disassemble the *class.dex* file of a target DID application into *SMALI* code [53], where the attacker can examine and modify the code and reassemble the code into class.dex in SMALI assembly code. Hence, reverse engineering the DID wallet application can be clearly feasible using the existing reverse engineering methods [54]. Lastly, the attacker can use *Keytool* and *Jarsigner* tools in JDK to go through the signing process of the DID wallet applications for further compromise.

In particular, reverse engineering becomes easier when there are no preventive measures such as obfuscation on the generated binaries of the DID wallet applications. Thus, the attacker can perform reverse engineering and tamper the DID wallet application binary on the wallet to achieve their purpose.

4) WQL INJECTION ATTACK ($\mathcal{A}_4$)

WQL [33] injection attack can occur, which is leveraging code injection techniques, similar to SQL injection. In particular, an attacker can inject malicious codes into WQL strings for key exposure attack in $\mathcal{A}_1$ and other malware to ease data exfiltration out of DID wallet. Then, those injected codes are later passed to the DID wallet's SQL Server for parsing and execution.

The typical form of WQL injection is the direct insertion of the malicious codes into user-input variables that are concatenated with WQL commands, which can be executed when the WQL commands are parsed and executed later. A less direct attack injects the malicious code into strings that are stored in the table or metadata of DID wallet database. Therefore, when the stored strings in the DID wallet are subsequently concatenated into a dynamic WQL command, the malicious code can be executed. A loose login auditing, privilege account abuse,

or insecure input validation in the wallet database can be the typical entry points for WQL injection attack. Using the WQL injection attack, the attacker can execute arbitrary commands in the DID wallet database.

5) DID WALLET DB INFORMATION DISCLOSURE ATTACK ($\mathcal{A}_5$)

In addition, an attacker can exploit the lack of encryption on either *data at rest* or *data in use* of DID wallet database and gain access to sensitive data such as personally identifiable information (PII) or high business impact (HBI) data. For example, in Hyperledger Indy wallet, some information is stored in the plain text, usually for richer and faster searchability in the DID wallet database via WQL. A tagging mechanism is used for searchability, where a tag is a *key-value* pair. In general, the *tag names*, *record ids*, and *record values* are always encrypted.

However, *tag values* are not encrypted when a special prefix *~(tilde)* is attached to the name of a tag value. This is usually the case when a user wants to perform some complex searches. In particular, the un-encrypted tag value enables the use of extended query operators for comparison or predicates such as *$gt (greater than), $lt (less than), $like*. In addition, if the malware attack in $\mathcal{A}_1$ is successfully carried out and DID wallet stores the sensitive PII or HBI data on untrusted SD card or local storage, then the data may get covertly extracted and exfiltrated out of DID wallet.

6) ELEVATION OF PRIVILEGES ATTACK ($\mathcal{A}_6$)

If the DID wallet reverse engineering attack in $\mathcal{A}_3$ is successful, then an attacker can gain the knowledge of target DID identity wallet applications. For instance, in the Indy wallet case, the knowledge on the following important system parameters[5] can be obtained: *wallet id*, *storage type*, and *storage configuration* including the path of wallet files, *key derivation method*, etc. Based on the combined knowledge and information, the attacker can further infiltrate into a target device via the data exfiltration techniques in $\mathcal{A}_1$ to escalate the privileges.

When the attacks in $\mathcal{A}_1$ and $\mathcal{A}_3$ are successful, the attacker can easily exploit unauthorized access to the database within the identity wallet. The attacker can also exploit the loose authorization rules such as user access permission to data or type of allowed actions to the data per user. Moreover, the attacker can exploit the privileged account abuse such as root account, domain admin account, and other accounts that can access security elements and more in the DID wallet system.

7) JAILBREAK/ROOTING ATTACK AGAINST DID IDENTITY WALLET ($\mathcal{A}_7$)

The techniques in reverse engineering ($\mathcal{A}_3$) and the elevation of privilege ($\mathcal{A}_6$) can be combined together to carry out a rooting attack against the DID identity wallet. The attacker can exploit unauthorized access to the secret area of the wallet application through elevated privileges. Consequently, the attacker can run the data exfiltration process, as shown in $\mathcal{A}_1$, to obtain wallet keys and user credentials.

[5]https://github.com/hyperledger/indy-sdk/blob/master/wrappers/python/indy/wallet.py

B. PHISHING / IMPERSONATION ATTACK ($\mathcal{A}_8$)

When DID EAs communicate with each other on an already established relationship, as shown in Step 28 in Figure 1, an attacker can trigger a phishing attack on a target user to gain access to the user agent's keys or credentials. In general, the phishing attack is a type of *social engineering* attack, tricking victims into disclosing their secret information. Thus, the attack can be launched through various impersonation forms such as legitimately-looking websites, emails, a third-party app UI, and more. Moreover, the attacker can spoof the victim user using advanced tools such as a Phishing kit [55] or Click-jacking [56]. For example, the attacker can use fake website templates and execute a server-side data collection using the Phishing Kit. Also, the Click-jacking enables the UI-redressing of a victim app through a fake overlay on top of the screen.

Specifically, the attacker can impersonate the EA of the user by masquerading the UI of DID wallet using the tools above. Then, the attacker shows a fake pop-up message, asking if the user wants to rotate the DID keys for security. If the user accepts and inputs the DID keys in the fake UI, then the DID keys are transferred to the attacker's remote server.

Figure 3 illustrates this specific scenario, where phishing and impersonation ($\mathcal{A}_8$) attacks are executed taking advantage of the phishing kit and UI redressing of DID wallet mentioned above.

Detailed Phishing / Impersonation Attack Scenario: In Step 1, the user *A* establishes DID connection with the user *B*. And, in Step 2 in Figure 3, a legitimate-looking pop-up message UI is rendered to the user *A* to impersonate the EA of the user *A*. The pop-up message asks the user *A* to rotate the DID signing key, saying the rotation should be performed periodically to maintain high security.

If the user selects the message in Step 3, then the phishing pop-up message redirects to the fake key rotation pop-up message. In Step 4, the user *A* provides the old DID signing key and a new DID signing key for rotation. In Step 5, the new DID signing key plus the old DID signing key are transferred to the remote side attacker. Using the old DID signing key in Step 5, the attacker replaces the key with a new key and propagates the DID event of *Key replaced for an agent* to the other party's microledger

However, the DID event is not immediately transferred to the user *A* due to the microledger replication delay. Taking advantage of the latency, in Step 6, the attacker impersonates the user *A* and communicates with the user *B* via the new DID signing key. Consequently, in Step 7, since the old key is changed, the message from user *A* becomes invalid to the user *B* and the attacker can successfully hijack the relationship.

C. CACHE POISONING / POLLUTION ATTACK ($\mathcal{A}_9$)

In the DID system, as shown in Figure 4, a DID agent sends a DID query to the UR and the query is checked if the local cache of UR has a matching DDO. If the cache misses the DDO, the query is forwarded to the DID method via driver layer in UR, as shown in Step 11 during Phase II in Figure 1. Then, each DID method runs its own process to retrieve the DDO. To improve the performance, most DID methods may use a cache of DDOs. In addition, each local DID resolver in the DID method can use a proxied resolution [57], called a *remote binding* mechanism, by which a resolver can invoke another DID resolver that runs on a different network host.

Because of the strong similarity between DID resolver and DNS resolver, these DID resolver processes operationally inherit similar vulnerabilities from DNS spoofing or cache poisoning. In DNS cache poisoning [58], the response verification is relatively simple based on those three metrics: IP destination/source addresses, UDP destination/source ports, and the original transaction ID. In the DID system, the attacker can exploit the analogous security loopholes in the verification process of the response from the DID remote resolver.

In Figure 4, we illustrate how such cache poisoning attacks against DID resolver can be executed in two different ways,[6] showing each sequential step in red circles for Scenario 1) and black circles for Scenario 2) below:

1) Detailed DID Resolver Cache Poisoning Attack Scenario. First, an attacker sends a DID query to a target DID method server, as shown in red circle. Next, the DID method server forwards the DID query to the remote DID resolver via the URL of "https://example.com/resolver/identifiers/did:sov:ABC" in Step 2. In Step 3, the attacker, masquerading the remote DID resolver, sends a response with the falsified DDO before the response from the legitimate remote DID resolver arrives. Then, the DID method updates the cache with the falsified DDO and the attacker succeeds in the cache poisoning attack ($\mathcal{A}_9$).

Since the query transaction via remote binding is completed in Step 3, the response from the legitimate remote DID resolver is discarded in Step 4. Then, the EA sends a DID query to UR via the URL of "https://uniresolver.io/1.0/identifiers/did:sov:ABC" in Step 5. Then, the cache in the DID method server returns the falsified DDO to the UR as a response. Thereafter, the UR forwards the falsified DDO as shown in Step 6. Finally, the EA connects to the fake service endpoint to the attacker's malicious server via the URL of "https://fake.com/myagent/profile?query#fragment" in Step 7, and the attack is successful at last.

[6]The main difference between Scenario 1) and 2) is the legitimacy of the response. Scenarios 1) assumes the valid response from the legitimate remote DID resolver.

2) Detailed DID Resolver Cache Poisoning Attack Scenario.

In addition, another cache poisoning attack (show in black circles) can be executed if the remote DID resolver gets compromised by the attacker. First, an attacker compromises a target remote DID resolver via the cache poisoning attack in the above Scenario 1). Next, the EA sends a DID query to UR via the URL of "https://uniresolver.io/1.0/identifiers/did:sov:ABC" in Step 2.

Then, the UR forwards the query to the DID method server. Thereafter, the DID method server invokes the compromised remote DID resolver for the query, as shown in Step 3 via the remote binding URL of "https://example.com/resolver/identifiers/did:sov:ABC". In Step 4, the compromised remote DID resolver returns the falsified DDO. In Step 5, the UR forwards the falsified DDO. Finally, the EA receives the falsified DDO and makes a connection to the fake service endpoint via the URL of "https://fake.com/myagent/profile?query#fragment" in Step 6.

Cache Pollution Attack. Lastly, the cache pollution attack can be performed, where the attacker can skew the content popularity of DID query by issuing multiple valid DID queries frequently that are less popular. The cache pollution attack could be in two forms depending on the intent of the attack, which are *locality disruption* and *false locality* [59]. The locality disruption is the case when the attacker repeatedly issues *new* DID queries to disrupt the locality of the cache in the DID method server. The false locality occurs when the attacker repeatedly requests those DID queries which are in a set of unpopular DIDs.

D. MICROSERVICE ATTACK ($\mathcal{A}_{10} \sim \mathcal{A}_{15}$)

As briefly mentioned in Section IV, the UR and DID method drivers are implemented in containerized applications and hence managed by the microservice orchestration platforms such as Kubernetes or Docker-Compose. Microservice is an independently deployable and autonomous component for a bounded scope that can support interoperability through message-based communication [60]. Thus, microservice architecture (MSA) can be highly automated, flexible, and scalable, comprised of microservice instances. And MSA is built from the identical base image, which is popularly used in the UR by DIF.

However, there are several known vulnerabilities with MSA. Especially, in Figure 5, we present the attack surface of microservices in the UR and DID method drivers, considering existing well known Kubernetes vulnerabilities in National Vulnerability Database (NVD)[7] of National Institute of Standards and Technology (NIST).

Detailed DID Microservice Attack Scenario: Each vulnerability in NVD is cited with Common Vulnerabilities and Exposures (CVE) ID, where Common Vulnerability Scoring System (CVSS) score is above 9.0 of critical severity.

[7]https://nvd.nist.gov/vuln/search

We identified the following six attack vectors with five specific CVEs that are directly applicable to existing UR and DID method drivers:

1) MULTI-STEP ATTACK ($\mathcal{A}_{10}$)

The derivation from base images introduces a *homogeneity* [61] in MSA, providing simplicity in managing multiple technologies. However, the simplicity also becomes another security loophole. For instance, the base image's vulnerabilities can be directly propagated, as shown in Step 8, to all the derivative microservice instances of the UR and DID method drivers. The multi-step attack exploits the re-occurrence of such base image's vulnerabilities in multiple microservices.

2) BYPASSING AUTHENTICATION ATTACK ($\mathcal{A}_{11}$)

Using the vulnerability of CVE-2018-0268 (CVSS:10), the attacker, who can access the service port of Kubernetes, can bypass authentication and obtain the elevated privileges due to the misconfiguration of Kubernetes container management subsystem for UR and DID method drivers, as shown in Step 1 in Figure 5.

3) LATERAL MOVEMENT ATTACK ($\mathcal{A}_{12}$)

In Step 2, the attacker can exploit lateral movements via containers of UR and DID method drivers using a subpath[8] volume mount. The attacker can perform the attack outside of the container's volume and the host's file system, specifically using the vulnerability of CVE-2017-1002101 (CVSS:9.6).

4) COMMAND LINE INJECTION ATTACK ($\mathcal{A}_{13}$)

In Step 3, the attacker can exploit a command-line argument injection, leveraging the vulnerability of CVE-2018-1002101 (CVSS:9.8) insecure command line input validation when mounting a volume[9] in UR and DID method driver microservices via command-line interfaces. The attacker can execute malicious commands which can be used for other attacks such as $\mathcal{A}_{14}$ and $\mathcal{A}_{15}$.

5) BACK-END CONNECTION ATTACK ($\mathcal{A}_{14}$)

In Step 4, the attacker can establish a back-end connection that can be used as an egress point of data exfiltration (CVE-2018-1002105, CVSS:9.8) in the UR and DID method driver microservices. The target data for the exfiltration can be system information of the UR and DID method microservices such as port numbers, details of currently running workloads, and privileged account information (e.g., /etc/sudoers). Then, the system information is forwarded to the attacker in Step 5.

6) SERVER SIDE REQUEST FORGERY (SSRF) ATTACK ($\mathcal{A}_{15}$)

In principle, a Server Side Request Forgery (SSRF) attack [62] allows an attacker to induce a server-side web application in vulnerable servers, such as SSRF entry point as shown in Step 6 to make HTTP requests. Next, the HTTP requests can be executed in the attacker's target domain. In Step 7, with the vulnerability of CVE-2018-18843 (CVSS:10), the attacker can successfully trigger the SSRF attack in the UR and DID method driver microservices.

E. DDO FORGERY ATTACK ($\mathcal{A}_{16}$) ON REDACTABLE BLOCKCHAIN

In general, a DDO is stored at public ledgers of VDR [37]. Each EA's authorization policy or secret value commitment is also recorded at the public ledgers, where the root of trust in the public ledgers is built on top of the blockchain's immutability.

In recent years, however, there has been paramount interest in making blockchain redactable, allowing deleting and modifying a transaction in the middle of the chain. For example, Chameleon Hash (CH) enables a redactable blockchain with a key exposure-free CH function when a trap door key is given [63]. The CH, based on the minimum disclosure proofs of knowledge [64] and Chameleon Signature (CS) [65], efficiently finds the collision of a hash when the trap door key is known.

Derler *et al.* [66] extended a CH-based blockchain redaction via a policy-based access control mechanism using Attribute-Based Encryption (ABE). However, when the VDR of DID method is implemented using the redactable blockchain schemes, the immutable blockchain transaction data is severely jeopardized by tampering attacks. For example, if the attacker can acquire the trapdoor key of the CH function, then the attacker can exploit the CH-based block redaction. Hence, the attacker can trigger the DDO forgery attack. Also, in DDO, the attacker may tamper the data such as service endpoint URL, properties of verification relationships, public key information, etc.

In Figure 6, we introduce how the DDO forgery attack can be invoked in the redactable blockchain scheme [63] by Ateniese *et al.*, which is already available for commercial solution.[10] Each sequential step is marked with the corresponding number in Figure 6. In particular, we assume the blockchain scheme has the CH along with the legacy SHA-256 hash.

Detailed DDO Forgery Attack Scenario: When the blocks are altered, their SHA-256 hash values are also changed, and the CH is connected instead. In Step 1, the attacker acquires the trapdoor key via the key exposure techniques, introduced in $\mathcal{A}_1$. Then, the attacker accesses the transaction Tx in $Block_i$ that stores the target DDO and forges the DDO, as shown in Step 2 in Figure 6.

Since the change of Tx data on $Block_i$ affects the Merkle-tree root, Nonce, and the hash of the $Block_i$, the legacy SHA-256 hash of the *Previous Block Hash* in $Block_{i+1}$ is not consistent as shown in Step 3. In Step 4, the attacker

[8]https://kubernetes.io/docs/concepts/storage/volumes/#using-subpath
[9]https://kubernetes.io/docs/concepts/storage/volumes/
[10]https://www.accenture.com/us-en/insight-editing-uneditable-blockchain

uses the trap door key to unlock the CH and find the hash collision. Therefore, the next $Block_{i+1}$ is not affected by the inconsistent SHA-256 hash, and the blockchain becomes persistent due to the CH. Consequently, the execution of the DDO forgery attack becomes successful.

F. VDR PARTITIONING / ICN ATTACK ($\mathcal{A}_{17}$)

The VDR partitioning attack can occur when the hashing power on blockchain is unequally distributed. In particular, this attack can be caused by the excessive aggregation of maintenance nodes such as full nodes participating in the mining process of Bitcoin network.

According to Saad *et al.* [67], the distribution of the full node is usually concentrated around a few major mining pools that have a high hashing rate. The geospatially centralized distribution makes the local autonomous systems (ASes) of the internet service provider, responsible for hosting the mining pools, to be the main target of attack such as BGP hijacking [68].

Detailed Partitioning / ICN Attack Scenario: Typically, the mining pool uses the stratum overlay protocol server that has a public IP address, which is vulnerable to routing attack and flood attacks due to the open public IP address. According to research by Apostolaki *et al.* [69] and Saad *et al.* [70], the impact of these BGP hijacking and routing attacks is severe such that the network hash rate can be reduced up to 50% due to the hijacking. Besides, the block propagation time of Bitcoin network can increase up to 20 mins by routing attacks.

Therefore, such network partition increases the likelihood of the stale blocks or orphan blocks being created and incurs significant damages such as blockchain fork, network congestion causing consensus latency, and more. As a result, all these network partitioning attacks jeopardize the DID services provided from the blockchain-based VDR. The operations of VDR, such as *Create, Read, Update, Deactivate* (CRUD) methods on DDO and DID, can be severely disturbed. Consequently, the partitioning attack makes VDR exposed to Denial of Service or system resource hijacking.

Moreover, the in-network caching system in ICN can be further exploited by the attacker through the cache pollution attack. The cache pollution attack targets the cache performance degradation in ICN-based VDR, whereas the partitioning attack targets the significant damage of hashing power in the blockchain-based VDR network.

The in-network caching is used to ensure low-latency content delivery in ICN. Each router node in ICN maintains a cache along with content storage. If the attacker repeatedly invokes random DID queries through UR or DID method, then the node in ICN suffers the churning of locality in the cache [59]. Besides, the attacker can invoke a massive number of unpopular DID queries to spoil the caching system in VDR.

G. SOCIAL RECOVERY ATTACK ($\mathcal{A}_{18}$)

For an encrypted DID wallet backup recovery, we observe the reconstruction of the decryption key can be threatened through a social recovery process. Specifically, we identify the vulnerabilities of the secret sharing scheme and corresponding attack scenario in this section.

1) SECRET SHARING AND TOMPA-WOLL ATTACK

For example, when a user invokes the DID wallet backup process, the EA generates an encrypted backup of the DID wallet. And the backup file is passed to the CA of the user to be restored later and the decryption key is then sharded using Eq.-(1) of threshold-based secret sharing scheme and distributed to the trustees of the user, as described in Step 4 in Figure 7.

In particular, we consider the Shamir Secret Sharing Scheme (SSSS) [45], where the threshold is the number of secret shares necessary to recover the original secret. The collection of the shares for the recovery is called the *access structure* for the secret sharing scheme. If the number of shares is less than the threshold k out of a total number of shares n, the secret cannot be determined. Thus, with knowledge of $k-1$ or fewer pieces, no information about the secret can be obtained, which is equivalent to the case that no share was given. Given the k shares, the original secret D can be uniquely determined through *Lagrangian interpolation* as below, where the k shares become the k pairs of $(x_i, q(x_i))$. The $q(x_i)$ is a random $k-1$ degree of a polynomial with $q(0) = D$:

$$q(x) = a_0 + \sum_{i=1}^{k-1} a_i x^i = D + \sum_{i=1}^{k-1} a_i x^i, \tag{1}$$

However, another secret $D^{'}$ out of $k-1$ pairs of $(x_i^{'}, q(x_i^{'}))$ plus $(x_k, q(x_k))$ can also determine a *legal* secret, where $D \neq D^{'}$. For a thorough meaning of the legality, we refer to Tompa and Woll [71]. Therefore, an attacker can collude with the shareholders of $k-1$ pairs and deceive the shareholder of $(x_k, q(x_k))$, claiming the $D^{'}$ to be the legal secret D. This is the core mechanism of Tompa-Woll attack [71], which is used by the social recovery attack. In Figure 7, we describe the attack scenario of social recovery for DID wallet via the DKMS method of Hyperledger Aries [13].

2) DETAILED SOCIAL RECOVERY ATTACK SCENARIO

First, the user A initiates a recovery setup in Step 1 in Figure 7. Next, the EA creates a recovery file containing the link secret of the user, DID wallet backup decryption key, and recovery endpoint of the CA. Then, the file D is divided into shares via Eq.-(1). We assume that the shares are distributed to four trustees using (3,4) access structure of SSSS, where 3 is the threshold and 4 is the total number of shares in Step 4.

In Step 5, the attacker colludes with the three of trustees to trigger Tompa-Woll attack. Thereafter, when the DID wallet is compromised, the user A invokes the social recovery request in Step 6. In Step 7, the EA forwards the request and the CA collects the shares from the trustees. Then, the

collected shares from the group in collusion by the attacker are forwarded to the EA in Step 8.

In Step 9, the EA assembles the shares and generates the legal secret $D^{'}$. Since the $D^{'}$ is legal but not D, the recovery of the DID wallet backup comes to fail in Step 10 in Figure 7. Therefore, the execution of the social recovery attack becomes successful.

VI. DISCUSSIONS ON POSSIBLE COUNTERMEASURES

In this work, we introduce seven new attack domains against the entire DID system when it is actually deployed with the latest technologies using Kubernetes, ICN, Chameleon-hashes, two-level secret sharing scheme, etc. Some attacks are direct extensions of the existing vulnerabilities from the underlying system components, while other attacks can be specifically carried out due to complex interactions within the entire DID system. In this section, we describe the possible mitigation approaches to the attack vectors from the previous section.

A. DEFENSE FOR WALLET SYSTEM ATTACK

Generally, Wallet System Attack is rooted from the key exposure attack ($\mathcal{A}_1$) vulnerability. Therefore, preventing the key exposure in the first place is of the utmost importance. Therefore, an approach such as a Software Data Diode [72] can be used. The mechanisms to provide the path integrity, packet integrity, and packet confidentiality using commodity hardware (HW) Trusted Execution Environments [73] can also be employed to mitigate the key exposure exfiltration.

To prevent MITM Attack over the agent-to-agent DID communication ($\mathcal{A}_2$), a certificate pinning can be used, where pinning is the process of associating a host with their expected X.509 certificate or public key. Once a certificate or public key is known or seen for each EA in DID wallet SW, the certificate or public key is associated or 'pinned' to the EA. Thus, when an adversary attempts to perform a TLS MITM attack, during TLS handshaking, the key from the attacker's server will be different from the pinned certificate's key of the EA, and the request will be discarded. Also, to defend against reverse engineering attack ($\mathcal{A}_3$) of DID wallet app, existing defense mechanisms such as Crypto Obfuscator [74] can be actively deployed to prevent reverse engineering threats.

For WQL Injection Attack ($\mathcal{A}_4$), a login auditing to the DID wallet database must be enabled to detect password guessing attacks. It is important to capture failed login attempts, which would help to identify the attacker's footprints. To defend against WQL injection attacks, existing defenses for SQL injection can be jointly applied here. In particular, the least-privileged accounts should be granted to connect to the database within the DID wallet. And DID service application login should be restricted in the DID wallet database and should only execute the allowed procedures.

Besides, a DID service application's login should not have direct table access to the DID wallet database. All WQL-mapped SQL statements (including the SQL statements in stored procedures) must be parameterized. This is because the parameterized SQL statements only accept characters that have predetermined meaning to SQL.

Moreover, sensitive data in the DID wallet database columns should be encrypted. For example, data can be encrypted using column-level encryption or by an application using encryption functions. In particular, database-level encryption of Transparent Data Encryption (TDE) can be enabled, where TDE in SQL server helps in encrypting sensitive data in the database of DID wallet and protecting the keys that are used to encrypt the data with a certificate.

B. DEFENSE FOR IMPERSONATION FROM STOLEN KEY

First, in case the attacker attempts to replace the DID key of the victim with the stolen key, the other party's EA should be able to notify the event to the CA of the victim. Thereby, the EA of the victim can report the event to the victim and revoke the compromised DID key.

Also, the DID system can consider applying a group consensus agreement protocol, where more than two group members are selected from the victim's EAs to allow the update of the target agent's DID key. At that time, the proof of the agreement should be presented on the DID event of *Key replaced for an agent*. The proof of agreement can be the ring signature [75] of the selected EAs of the victim.

If the DID event is propagated to the other party's EA without the proof of agreement, then the EA can be aware of the malicious attempt at the moment. Thereafter, the other party's EA can notify the victim's EA of the malicious attempt. Then, the victim's EA can either trigger the revocation process for the compromised DID agent's key so that the attacker cannot easily impersonate the victim using the compromised key and inflict another damage to the hijacked relationship.

C. DEFENSE FOR CACHE POISONING / POLLUTION ATTACK

To prevent cache poisoning attack, as shown in Scenario 1) of Figure 4 against DID resolver, the core ideas from DNS Security (DNSSEC) [76] can be adopted to check the validity of the response given a DID query. Thus, the local DID resolver should be able to check if the response has been really received from the authentic DID resolver invoked through the remote binding. In our case, the DID proxied resolution process can strengthen the integrity of the received resolution response via the verification of the digital signature from the remote resolver.

Also, recent work such as Jin *et al.* [77] using a machine learning-based DNS traffic model can be used to characterize the cache poisoning attack for Scenario 2) of Figure 4. For example, the local DID resolver can employ two-level phases of *Global Training* (GT) phase and *Selective Training* (ST) phase for analyzing DID query-response pairs.

In the GT phase, all DID query-response pairs are analyzed based on the features extracted from the general DID

resolution protocol [39]. The main purpose of the GT phase is similar to anomaly detection by classifying abnormal DID query-response pairs from normal ones. In the ST phase, we can apply the selective features of DID query-response pairs based on the heuristic analysis on the anomaly detection in the GT phase.

The selective features can be the following: 1) the geographic information of the IP address used in the DID query, 2) the distance between a current DID query and the previous DID query, and 3) the abnormal DID query and the response of cached DID resolution. Consequently, the DID resolver can utilize the above analyzer of the query-response pairs between the local DID resolver and the remote DID resolver to detect the cache poisoning attacks.

D. DEFENSE FOR MICROSERVICE ATTACK

To prevent MSA attacks, prior work on moving target defense approach by Torkura *et al.* [61] can be considered. Their defense introduces the diversification index that can be used to determine the degree of diversification in the container image and the code. The main purpose of this approach is to increase the uncertainty for attackers, while the attack surface is diversified.

Hence, the inherent homogeneity of the microservice is improved against the multi-step attack $\mathcal{A}_{10}$, as shown in Figure 5, on the UR and the driver of DID methods. Also, the verification of the base image's origin and integrity check can be incorporated to prevent the multi-step attack in the UR and drivers of DID methods.

Except for $\mathcal{A}_{10}$, other microservice attacks from $\mathcal{A}_{11}$ to $\mathcal{A}_{15}$ have been partially patched with each of the mitigations in the NVD of NIST that can be further checked with each CWE ID in Section V-D. However, the attacker may expand the domain of those attacks, in a more sophisticated way, by sidestepping the known mitigation and trigger a new attack on top of the old ones. Therefore, further research on the security of microservice needs to be performed to prevent the attacks.

E. DEFENSE FOR PARTITIONING / ICN ATTACK

To prevent partitioning attacks in VDR, the geospatial concentration of hashing power should be distributed to the broader geographical regions with network diversity as pointed out by Saad *et al.* [67]. This can be achieved through the decentralized hosting of the mining pools and full nodes over the internet. In parallel, the secure and scalable relay network of Bitcoin [78] can be considered, using an inter-domain routing policy to provide a secure and fast routing path among relay nodes.

In the case of an ICN attack, the router node in the ICN should be able to cache the DID with the matching DDO according to the popularity of the contents. Also, the popularity of the cached DID and DDO should be periodically evaluated to detect the cache pollution attack. For detecting the attack in ICN-based VDR, the research by Xie *et al.* [79] on the robust cache shielding scheme can be considered to prevent cache pollution attacks. Their approach leverages the difference between a normal request having Zipf-like [80] distributions and a malicious request, which can be generated from a totally different distribution.

Another work by Karami and Guerrero-Zapata [81] proposed a fuzzy logic-based cache content evaluation, based on the membership function to return a *goodness* value ranging from 0 to 1, where the boundary value of 1 means normal content, 0.5 cache disruption attack, and 0 the false locality attack, respectively. However, both approaches of [79], [81] incur a high computation overhead due to the probabilistic cache replacement and fuzzy logic-based goodness derivation, which should be considered for further research in deployment.

F. DEFENSE FOR SOCIAL RECOVERY ATTACK

The core threat against the social recovery attack is the weak integrity on the distributed shares of DID wallet's recovery file. To prevent such threats, we need to confirm the correctness as well as the legality of the reconstructed secret and avoid the collusion attack from dishonest participants (cheaters).

To confirm the correctness of the reconstructed secret, the integrity on the share of DID wallet recovery file, as shown in Step 2 of Figure 7, needs to be protected. A recent approach by Sprenkels [82] can be employed by applying a hybrid mode of encryption and Shamir's scheme. First, the secret (DID wallet recovery file) is encrypted using a random key.

Then, the random key is sharded and distributed to the participants. When reconstructing the secret, it will first obtain the reconstructed random key, and use it to decrypt the secret so that the secret will be recovered successfully, only when the reconstructed random key is valid through Shamir's process.

Also, to avoid the collusion attack, two-level scheme for the secret sharing by TREZOR team [83] can be used such that the secret, i.e. the DID wallet recovery file, is divided into Group share and Member share. It will first apply the threshold-based secret sharing scheme in a large group (e.g., 3/5). Then, the scheme divides each of the groups into members so that the possibility of collusion within each individual group becomes impotent to recover the entire key, unless the collusion of other remaining groups is made. This scheme adds more safeguards to the original Shamir's approach in the social recovery of the DID wallet.

Nevertheless, the problem of cheater identification is not addressed in the above defenses. Also, the verification of an individual share still remains as an open problem to be addressed. Thus, further research is needed in this area for the social recovery. Moreover, system performance metrics should also be considered such as the overall recovery speed of DID wallet, the availability for non-interactive verification of secret share, the size of each secret share, the expandability for multi-secret sharing solution, etc.

VII. CONCLUSION

In this work, we systemically explored the blockchain-based DID system's attack surfaces. First, we considered the cutting-edge underlying technologies such as DID, VC, blockchain to enable SSI and characterized the detailed information flows and interactions among different entities. Next, we carefully analyzed the possible attacks that are native to each of the technologies and new attacks that can occur due to the intertwined complexity of the combined DID and blockchain technologies. In particular, we presented the full-scale attack surfaces across the different DID functional block systems. Some of these attacks should be urgently considered to build a more secure DID-based SSI system for a wide range of practical application deployment.

APPENDIX

NOMENCLATURE

ABBREVIATIONS

AP	Agent Policy
APAC	Agent Policy Address Commitment
BGP	Board Gateway Protocol
CA	Cloud Agent
CBOR	Concise Binary Object Representation
CH	Chameleon Hash
CIM	Centralized Identity Management
CKMS	Centralized cryptographic Key Management Systems
CR	Community Resolver
CS	Chameleon Signature
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DDO	DID Document
DFD	Data Flow Diagram
DID	Decentralized Identifiers
DIF	Decentralized Identity Foundation
DKMS	Decentralized Key Management System
DNSSEC	DNS Security
DRIM	DID Resolution Input Metadata
EA	Edge Agent
FIM	Federated Identity Management
GDPR	General Data Protection Regulation
GT	Global Training
HBI	High Business Impact
HKDF	HMAC Key Derivation Function
ICN	Information-Centric Networking
IDP	Identity Provider
MITM	Man-In-The-Middle
MSA	Microservice Architecture
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
PBKDF2	Password-Based Key Derivation Function 2
PII	Personally Identifiable Information
PoP	Proof-Of-Possession
SSI	Self-Sovereign Identity
SSO	Single Sign-On
SSRF	Server Side Request Forgery
SSSS	Shamir Secret Sharing Scheme
ST	Selective Training
SV	Secret Value
SVC	Secret Value Commitment
TDE	Transparent Data Encryption
UIM	User-centric Identity Management
UR	Universal Resolver
VC	Verifiable Credentials
VDR	Verifiable Data Registry
WQL	Wallet Query Language
ZKP	Zero-Knowledge Proof

REFERENCES

[1] E. Kim, Y.-S. Cho, B. Kim, W. Ji, S.-H. Kim, S. S. Woo, and H. Kim, "Can we create a cross-domain federated identity for the industrial Internet of Things without Google?" *IEEE Internet Things Mag.*, vol. 3, no. 4, pp. 82–87, Dec. 2020.

[2] C. Allen. *The Path to Self-Sovereign Identity*. Accessed: Aug. 17, 2020. [Online]. Available: http://www.lifewithalacrity.com/2016/04/the-path-to-self-sovereign-identity.html

[3] K. Cameron, "The laws of identity," *Microsoft Corp*, vol. 5, pp. 8–11, May 2005.

[4] *A Protocol and Token for Selfsovereign Identity and Decentralized Trust*. Sovrin. Accessed: Aug. 12, 2020. [Online]. Available: https://sovrin.org/wp-content/uploads/Sovrin-Protocol-and-Token-White-Paper.pdf

[5] *Uport Specs*. uPort. Accessed: Aug. 7, 2020. [Online]. Available: https://github.com/uport-project/specs

[6] *A Decentralized, Open Source Solution for Digital Identity and Access Management*. Jolocom. Accessed: Sep. 10, 2020. [Online]. Available: https://jolocom.io/wp-content/uploads/2019/12/Jolocom-Whitepaper-v2.1-A-Decentralized-Open-Source-Solution-for-Digital-Identity-and-Access-Management.pdf

[7] D. Pennino, M. Pizzonia, A. Vitaletti, and M. Zecchini, "Binding of endpoints to identifiers by on-chain proofs," 2020, *arXiv:2005.00794*. [Online]. Available: http://arxiv.org/abs/2005.00794

[8] A. S. Omar, "Decentralized identity and access management framework for Internet of Things devices," Ph.D. dissertation, Univ. Waterloo, Waterloo, ON, Canada, 2020.

[9] I. Williams and X. Yuan, "Evaluating the effectiveness of microsoft threat modeling tool," in *Proc. Inf. Secur. Curriculum Develop. Conf.*, Oct. 2015, pp. 1–6.

[10] D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, and M. Sabadello. *Decentralized Identifiers (DIDS) V1.0. Core Architecture, Data Model, and Representations*. Accessed: Jun. 2, 2020. [Online]. Available: https://www.w3.org/TR/did-core/

[11] *Verifiable Credentials Data Model 1.0 Expressing Verifiable Information on the Web*. Accessed: Jun. 2, 2020. [Online]. Available: https://www.w3.org/TR/vc-data-model/W3C

[12] *Aries RFC 0004: Agents*. Hyperledger Aries. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/hyperledger/aries-rfcs/tree/master/concepts/0004-agents

[13] D. Reed, J. Law, D. Hardman, and M. Lodder. *DKMS (Decentralized Key Management System) Design and Architecture V4*. Hyperledger Aries. Accessed: Jul. 12, 2020. [Online]. Available: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0051-dkms/dkms-v4.md

[14] D. Hardman. *Aries RFC 0005: Did Communication*. Hyperledger Aries. Accessed: Jul. 3, 2020. [Online]. Available: https://github.com/hyperledger/aries-rfcs/tree/master/concepts/0005-didcomm

[15] *Universal Resolver*. Decentralized Identity Foundation. Accessed: Jul. 15, 2020. [Online]. Available: https://github.com/decentralized-identity/universal-resolver

[16] M. Takemiya and B. Vanieiev, "Sora identity: Secure, digital identity on the blockchain," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 2, Jul. 2018, pp. 582–587.

[17] L. Ertaul, M. Kaur, and V. A. K. R. Gudise, "Implementation and performance analysis of PBKDF2, Bcrypt, Scrypt algorithms," in *Proc. Int. Conf. Wireless Netw. (ICWN)*, 2016, p. 66.

[18] M. J. Dworkin, "SHA-3 standard: Permutation-based hash and extendable-output functions," NIST, Gaithersburg, MD, USA, Tech. Rep. Federal Inf. Process. Stds. (NIST FIPS) - 202, 2015.
[19] H. Krawczyk, "Cryptographic extraction and key derivation: The HKDF scheme," in *Proc. Annu. Cryptol. Conf.* Berlin, Germany: Springer, 2010, pp. 631–648.
[20] H. Halpin, "Vision: A critique of immunity passports and W3C decentralized identifiers," in *Proc. Int. Conf. Res. Secur. Standardisation*. Cham, Switzerland: Springer, 2020, pp. 148–168.
[21] M. Eisenstadt, M. Ramachandran, N. Chowdhury, A. Third, and J. Domingue, "COVID-19 antibody test/vaccination certification: There's an app for that," 2020, *arXiv:2004.07376*. [Online]. Available: http://arxiv.org/abs/2004.07376
[22] *Art. 4 GDPR Definitions*. EU. Accessed: Jul. 23, 2020. [Online]. Available: https://gdpr-info.eu/art-4-gdpr/
[23] S. Biswas, "Enhancing the privacy of decentralized identifiers with ring signatures," M.S. thesis, Aalto Univ., Espoo, Finland, 2020.
[24] *Peer Did Method Specification*. W3C. Accessed: Sep. 11, 2020. [Online]. Available: https://identity.foundation/peer-did-method-spec/index.html
[25] J. De Clercq, "Single sign-on architectures," in *Proc. Int. Conf. Infrastruct. Secur.* Berlin, Germany: Springer, 2002, pp. 40–58.
[26] P. Zimmermann, "Why i wrote PGP," *Part Original*, pp. 1–3, Jun. 1991. [Online]. Available: http://palmer.wellesley.edu/~ivolic/pdf/Classes/Handouts/NumberTheoryHandouts/WhyIWrotePGP-Zimmermann.pdf
[27] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, "Kerberos authentication and authorization system," in *Proc. Project Athena Tech. Plan*. Cambridge, MA, USA: Citeseer, 1988.
[28] J. Hassell, *RADIUS: Securing Public Access to Private Resources*. Newton, MA, USA: O'Reilly Media, 2002.
[29] H. Simon, "SAML: The secret to centralized identity management," InformationWeek, San Francisco, CA, USA, Tech. Rep. 1028656, 2004.
[30] M. Miculan and C. Urban, "Formal analysis of Facebook connect single sign-on authentication protocol," in *SOFSEM*, vol. 11. Nový Smokovec, Slovakia: Citeseer, 2011, pp. 22–28.
[31] *Did Documents*. W3C. Accessed: Aug. 13, 2020. [Online]. Available: https://w3c.github.io/did-core/#did-documents
[32] *Indy Walkthrough*. Hyperledger Indy. Accessed: Aug. 6, 2020. [Online]. Available: https://github.com/hyperledger/indy-sdk/blob/master/docs/getting-started/indy-walkthrough.md
[33] *Wallet Query Language*. Hyperledger Indy. Accessed: Aug. 8, 2020. [Online]. Available: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0050-wallets/README.md#wallet-query-language
[34] *Meet Connect.Me, the First Sovrin-Based Digital Wallet*. Evernym. Accessed: Aug. 17, 2020. [Online]. Available: https://www.evernym.com/blog/connect-me-sovrin-digital-wallet/
[35] *Hyperledger Aries Explainer*. Hyperledger Aries. Accessed: Jul. 3, 2020. [Online]. Available: https://github.com/hyperledger/aries
[36] *Hyperledger Ursa Explainer*. Hyperledger Ursa. Accessed: Jul. 11, 2020. [Online]. Available: https://github.com/hyperledger/ursa
[37] *Verifiable Data Registry*. W3C. Accessed: Sep. 3, 2020. [Online]. Available: https://w3c.github.io/did-core/#dfn-verifiable-data-registry
[38] Q. Li and Y.-L. Chen, "Data flow diagram," in *Modeling and Analysis of Enterprise and Information Systems*. Berlin, Germany: Springer, 2009, pp. 85–97.
[39] *Decentralized Identifier Resolution (Did Resolution) V0.2*. W3C. Accessed: Jul. 11, 2020. [Online]. Available: https://w3c-ccg.github.io/did-resolution/#resolving-algorithm
[40] G. Danezis and S. Gürses, "A critical review of 10 years of privacy technology," *Proc. Surveill. Cultures A, Global Surveill. Soc.*, pp. 1–16, Apr. 2010. [Online]. Available: https://fahrplan.events.ccc.de/congress/2010/Fahrplan/attachments/1694_DanezisGuersesSurveillancePets2010.pdf
[41] J. Brendel, C. Cremers, D. Jackson, and M. Zhao, "The provable security of ed25519: Theory and practice," in *Proc. IEEE Secur. Privacy*, Oct. 2020, pp. 1–18.
[42] *Input Metadata Properties*. W3C. Accessed: Aug. 3, 2020. [Online]. Available: https://w3c-ccg.github.io/did-resolution/#resolving-input-accept
[43] *Did Url Syntax*. W3C. Accessed: Aug. 7, 2020. [Online]. Available: https://www.w3.org/TR/did-core/#did-url-syntax
[44] *Did Url Dereferencing*. W3C. Accessed: Aug. 11, 2020. [Online]. Available: https://w3c.github.io/did-core/#dfn-did-url-dereferencing
[45] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[46] A. Davenport and S. Shetty, "Air gapped wallet schemes and private key leakage in permissioned blockchain platforms," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Jul. 2019, pp. 541–545.
[47] *Did Exchange Protocol 1.0*. Hyperledger Aries. Accessed: Aug. 16, 2020. [Online]. Available: https://github.com/hyperledger/aries-rfcs/blob/master/features/0023-did-exchange/README.md
[48] *Trust on First Use*. Wikipedia. Accessed: Aug. 21, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Trust_on_first_use
[49] D. Gultsch. *Blind Trust Before Verification*. Accessed: Aug. 22, 2020. [Online]. Available: https://gultsch.de/trust.html
[50] *Mediators and Relays*. Hyperledger Aries. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0046-mediators-and-relays/README.md
[51] *Cross-Domain Messaging*. Hyperledger Aries. Accessed: Aug. 7, 2020. [Online]. Available: https://github.com/hyperledger/aries-rfcs/blob/master/concepts/0094-cross-domain-messaging/README.md
[52] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2027–2051, 3rd Quart., 2016.
[53] B. Gruver. *Smali/Baksmali Explainer*. Accessed: Aug. 17, 2020. [Online]. Available: https://github.com/JesusFreke/smali
[54] Q. Do, B. Martini, and K.-K.-R. Choo, "Exfiltrating data from Android devices," *Comput. Secur.*, vol. 48, pp. 74–91, Feb. 2015.
[55] A. Oest, Y. Safei, A. Doupe, G.-J. Ahn, B. Wardman, and G. Warner, "Inside a phisher's mind: Understanding the anti-phishing ecosystem through phishing kit analysis," in *Proc. APWG Symp. Electron. Crime Res. (eCrime)*, May 2018, pp. 1–12.
[56] A. Possemato, A. Lanzi, S. P. H. Chung, W. Lee, and Y. Fratantonio, "ClickShield: Are you hiding something? Towards eradicating clickjacking on Android," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 1120–1136.
[57] *Decentralized Identifier Resolution (Did Resolution) V0.2*. W3C. Accessed: Aug. 12, 2020. [Online]. Available: https://w3c-ccg.github.io/did-resolution/#resolver-architectures-proxied
[58] C. Wright, "Understanding kaminsky's dns bug," *Cory Wright's Blog*, Jul. 2008. [Online]. Available: https://www.linuxjournal.com/content/understanding-kaminskys-dns-bug
[59] R. Tourani, S. Misra, T. Mick, and G. Panwar, "Security, privacy, and access control in information-centric networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 566–600, 1st Quart., 2018.
[60] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice Architecture: Aligning Principles, Practices, and Culture*. Newton, MA, USA: O'Reilly Media, 2016.
[61] K. A. Torkura, M. I. H. Sukmana, A. V. D. M. Kayem, F. Cheng, and C. Meinel, "A cyber risk based moving target defense mechanism for microservice architectures," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl., Ubiquitous Comput. Commun., Big Data Cloud Comput., Social Comput. Netw., Sustain. Comput. Commun. (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, Dec. 2018, pp. 932–939.
[62] *Server-Side Request Forgery (SSRF)*. Web Security Academy of PortSwigger. Accessed: Aug. 17, 2020. [Online]. Available: https://portswigger.net/web-security/ssrf
[63] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable blockchain–or–rewriting history in bitcoin and friends," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS P)*, Apr. 2017, pp. 111–126.
[64] G. Brassard, D. Chaum, and C. Crépeau, "Minimum disclosure proofs of knowledge," *J. Comput. Syst. Sci.*, vol. 37, no. 2, pp. 156–189, Oct. 1988.
[65] H. Krawczyk and T. Rabin, "Chameleon hashing and signatures," IACR, Bellevue, WA, USA, Cryptol. ePrint Arch. Rep. 1998/010, 1998.
[66] D. Derler, K. Samelin, D. Slamanig, and C. Striecks, "Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 406, Feb. 2019.
[67] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and A. Mohaisen, "Exploring the attack surface of blockchain: A systematic overview," 2019, *arXiv:1904.03487*. [Online]. Available: http://arxiv.org/abs/1904.03487
[68] M. Saad, A. Anwar, A. Ahmad, H. Alasmary, M. Yuksel, and A. Mohaisen, "RouteChain: Towards blockchain-based secure and efficient BGP routing," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, May 2019, pp. 210–218.
[69] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 375–392.
[70] M. Saad, V. Cook, L. Nguyen, M. T. Thai, and A. Mohaisen, "Partitioning attacks on bitcoin: Colliding space, time, and logic," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 1175–1187.

[71] M. Tompa and H. Woll, "How to share a secret with cheaters," *J. Cryptol.*, vol. 1, no. 3, pp. 133–138, Oct. 1989.
[72] B. G. Schlicher, L. P. MacIntyre, and R. K. Abercrombie, "Towards reducing the data exfiltration surface for the insider threat," in *Proc. 49th Hawaii Int. Conf. Syst. Sci. (HICSS)*, Jan. 2016, pp. 2749–2758.
[73] D. E. Asoni, T. Sasaki, and A. Perrig, "Alcatraz: Data exfiltration-resilient corporate network architecture," in *Proc. IEEE 4th Int. Conf. Collaboration Internet Comput. (CIC)*, Oct. 2018, pp. 176–187.
[74] *Crypto Obfuscator For.Net (V2020)*. LogicNP Software. Accessed: Aug. 11, 2020. [Online]. Available: https://www.ssware.com/cryptoobfuscator/obfuscator-net.htm
[75] H. Wang, F. Zhang, and Y. Sun, "Cryptanalysis of a generalized ring signature scheme," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 2, pp. 149–151, Apr. 2009.
[76] S. Weiler and D. Blacka. *Clarifications and Implementation Notes for DNS Security (DNSSEC)*. Accessed: 2013. [Online]. Available: http://www.ietf.org/html/rfc6840
[77] Y. Jin, M. Tomoishi, and S. Matsuura, "A detection method against DNS cache poisoning attacks using machine learning techniques: Work in progress," in *Proc. IEEE 18th Int. Symp. Netw. Comput. Appl. (NCA)*, Sep. 2019, pp. 1–3.
[78] M. Apostolaki, G. Marti, J. Müller, and L. Vanbever, "SABRE: Protecting bitcoin against routing attacks," 2018, *arXiv:1808.06254*. [Online]. Available: http://arxiv.org/abs/1808.06254
[79] M. Xie, I. Widjaja, and H. Wang, "Enhancing cache robustness for content-centric networking," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 2426–2434.
[80] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. IEEE Conf. Comput. Commun. 18th Annu. Joint Conf. IEEE Comput. Commun. Societies Future Now (INFOCOM)*, vol. 1, Mar. 1999, pp. 126–134.
[81] A. Karami and M. Guerrero-Zapata, "An ANFIS-based cache replacement method for mitigating cache pollution attacks in named data networking," *Comput. Netw.*, vol. 80, pp. 51–65, Apr. 2015.
[82] D. Sprenkels. *Implementing Threshold Schemes*. Accessed: Sep. 26, 2020. [Online]. Available: https://github.com/WebOfTrustInfo/rwot8-barcelona/blob/master/topics-and-advance-readings/implementing-threshold-schemes.md
[83] TREZOR. *Shamir's Secret-Sharing for Mnemonic Codes*. Accessed: Sep. 21, 2020. [Online]. Available: https://github.com/satoshilabs/slips/blob/master/slip-0039.md

BONG GON KIM (Graduate Student Member, IEEE) received the B.S. degree in electrical and electronics and the M.S. degree in computer science from Yonsei University, Seoul, South Korea. He is currently pursuing the Ph.D. degree with the Department of Computer Science, University of Stony Brook. From 2003 to 2018, he worked as a Senior Engineer with the SW platform Lab, Samsung Electronics, South Korea. He has been involved in numerous projects of mobile communication, Android applications, and Linux systems during the time in Samsung. He is also interested in the redactable blockchain system using chameleon hashes. His current research interests include mainly the security and privacy management in blockchain systems.

YOUNG-SEOB CHO received the Ph.D. degree in computer science from Inha University, South Korea. From 1998, he worked on research projects with the Electronics and Telecommunications Research Institute (ETRI) and continuously conducted numerous projects on national and international level. He is currently a Principal Researcher with the Information Security Research Division, ETRI. His current research interests include blockchain identity management, AI security, authentication, and authorization.

SEOK-HYUN KIM received the M.S. degree in information security from Chonnam National University, South Korea. From 2010, he worked on research projects with the Electronics and Telecommunications Research Institute (ETRI) and continues to carry out various national projects. He is currently a Senior Researcher with the Information Security Research Division, ETRI. His current research interests include blockchain identity management, FIDO, authentication, and privacy.

HYOUNGSHICK KIM received the B.S. degree from the Department of Information Engineering, Sungkyunkwan University, the M.S. degree from the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), and the Ph.D. degree from the Computer Laboratory, University of Cambridge, in 1999, 2001, and 2012, respectively. From 2004 to 2008, he worked with Samsung Electronics, as a Senior Engineer. He worked with the Department of Electrical and Computer Engineering, The University of British Columbia, as a Postdoctoral Fellow. He is currently with the Department of Software, Sungkyunkwan University, as an Associate Professor. His current research interests include usable security and security engineering.

SIMON S. WOO (Member, IEEE) received the B.S. degree in electrical engineering from the University of Washington (UW), Seattle, the dual M.S. degree in computer science from the University of Southern California (USC), Los Angeles, and in electrical and computer engineering from the University of California at San Diego (UCSD), San Diego, and the Ph.D. degree from the University of Southern California (USC). He was a member of Technical Staff (Technologist) for nine years with the NASA's Jet Propulsion Laboratory (JPL), Pasadena, CA, USA, conducting research in satellite communications, networking, and cybersecurity areas. He also worked with Intel Corporation and the Verisign Research Lab. Since 2017, he has been a Tenure-Track Assistant Professor with SUNY Korea, South Korea, and a Research Assistant Professor with Stony Brook University. He is currently a Tenure-Track Assistant Professor with the SKKU Institute for Convergence and Department of Applied Data Science and Software, Sungkyunkwan University, Suwon, South Korea.