ChatGPT

Blockchain-Based Decentralized Identity Platform Architecture

Overview and Requirements

Decentralized Identifiers (DIDs) provide a self-sovereign identity mechanism where entities (people, organizations, devices) have identifiers not reliant on any central authority [1]. The goal is to enable users to control their own digital identities, with DIDs linking to DID Documents (containing public keys, endpoints, etc.) that others can trust and verify [2]. To meet the needs of a large-scale identity system, we propose a **blockchain-based distributed database** architecture designed specifically for DIDs and related identity data. Key requirements include:

- **DID Method Agnosticism:** Support all W3C-conformant DID methods (e.g. `did:ethr`, `did:key`, `did:web`, etc.) without being tied to one specific ledger or protocol. The system should implement generic DID operations (create, update, deactivate) and resolution interfaces that work across methods [3]. This ensures interoperability and future extensibility as new DID methods emerge.
- **Scalability:** The design must scale to millions of DIDs and transactions. It should handle high DID registration and update throughput, and efficiently store or index large volumes of identity data. Techniques to achieve this include batching, off-chain storage, and high-performance consensus, as discussed later.
- **Data Types Support:** Store and manage all types of identity-related data: DID Documents, Verifiable Credentials (VCs) and their status/revocations, public keys and revocation registries, and even IoT device identity data. The storage model should accommodate both small metadata (e.g. DID Documents) and larger objects (e.g. credential or IoT data) in a secure and cost-effective way.
- **Consistency First:** Prioritize strong consistency of data across nodes – changes to identity records should be immediately reflected network-wide (analogous to a CP system in CAP theorem). Partition tolerance is desirable, but not at the expense of a single source of truth for identity state. In practice, this means the network may sacrifice some availability during network splits to avoid forked or divergent identity states [4].
- **Blockchain-Based Trust:** Use a blockchain or distributed ledger as the backbone for immutable, auditable logging of identity events (creations, updates, revocations). The choice of blockchain technology should be flexible – the architecture could be implemented on a public chain, a permissioned consortium ledger, or a hybrid model – but must provide decentralization and tamper-resistance.
- **Security and Privacy:** Protect sensitive identity data by design. Only store public or hashed data on-chain, keep private data off-chain, enforce access controls, and support key recovery and rotation. The system must prevent unauthorized DID modifications (only allowing updates signed by the DID controller) and provide audit trails for all operations. Privacy-enhancing techniques (selective disclosure, zero-knowledge proofs, etc.) should be supported for credential data.
- **IoT Integration:** Anticipate integration of IoT devices and sensors as identity subjects. The architecture should accommodate potentially billions of IoT DIDs, lightweight crypto for constrained devices, and secure onboarding processes where devices can be issued DIDs and owners can manage device credentials. Mechanisms for device authentication, credential

issuance, and data sharing via DIDs/VCs are required to bring IoT into the decentralized identity ecosystem [5].

With these requirements in mind, we outline a detailed architecture covering the choice of ledger platform, data storage structure, consensus mechanism, DID operations, scalability strategies, security/privacy features, and IoT integration.

Platform and Ledger Selection

Choice of Blockchain/DLT: We recommend using a high-performance **permissioned blockchain network** as the core ledger, employing a Byzantine Fault Tolerant consensus for strong consistency. For example, a Tendermint-based Proof-of-Stake network (like the Cosmos SDK) would be suitable – Tendermint ensures that *"every non-faulty machine sees the same transaction log and computes the same state"*, providing immediate finality and consistency across nodes [6]. This BFT consensus (tolerating up to 1/3 faulty nodes) means all identity transactions are agreed in order, preventing conflicting DID states. Tendermint's consensus has fast block times (~1 second) and can handle thousands of transactions per second with instant finality (no forking) [7], aligning with our scalability needs.

Alternatively, other BFT consensus platforms (like Hyperledger Fabric's Kafka/Raft ordering or Algorand's Pure PoS) could be used, but the core requirement is **CP-style behavior** – the system prefers consistency and will halt updates during network partitions rather than risk divergent states. In a CP approach, if connectivity is lost between nodes, the network might pause DID operations until quorum is restored [8]. This ensures that at any given time there is a single authoritative view of each DID Document (no split-brain scenarios). Partition tolerance is addressed by quickly reconciling state once the network heals, at which point all nodes catch up to the one true state.

Permissioned vs. Public: A **consortium (permissioned) blockchain** is recommended for this identity ledger to achieve predictable performance, low latency, and governance control (important for trust frameworks). Consortium validators (e.g. trusted organizations, government or industry stakeholders) can run the nodes, using a BFT algorithm (such as Tendermint Core or Hyperledger Indy's variant of PBFT) to validate identity transactions. This yields a controlled environment with strong consistency guarantees. However, the design remains *blockchain-agnostic* – it can also be deployed on a public chain if desired (e.g. anchoring identity transactions on Ethereum or Bitcoin). In fact, the architecture draws from the **Sidetree protocol**, which is a blockchain-agnostic DID layer that *"can run atop any existing decentralized anchoring system (e.g. Bitcoin, Ethereum, distributed ledgers)"* [9]. Just as Sidetree overlay networks can be implemented on different base chains, our platform could use different ledger backbones depending on context (public, private, or hybrid).

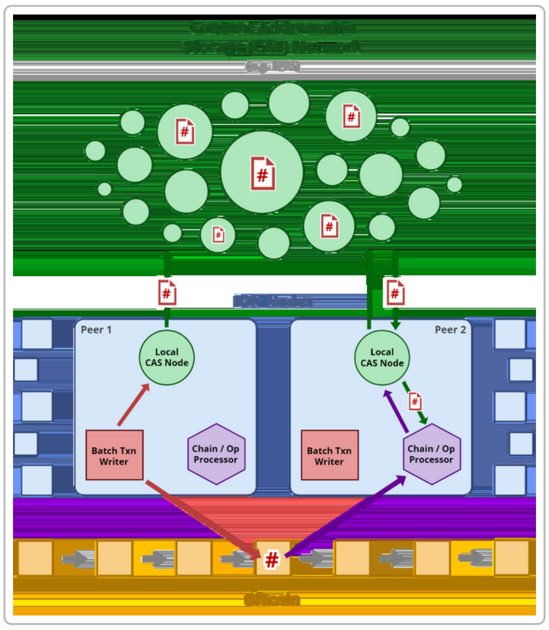
Hybrid Anchoring: For additional robustness, a hybrid model can be employed: The primary DID ledger runs on a fast permissioned network for handling millions of users, but periodically it anchors cryptographic checkpoints to a public blockchain (e.g. writing a hash of the latest state to Ethereum or Bitcoin). This provides an extra layer of auditability and partition tolerance – even if the private network is briefly partitioned or its operators act maliciously, the public anchor (which is highly partition-tolerant by design) serves as an immutable reference point. This approach leverages the strength of public blockchains (censorship-resistance and global availability) without incurring their performance and cost limitations for every transaction. It is similar to how some Layer-2 networks commit checkpoints to Layer-1 for security.

DID Method Agnosticism: The platform's software will implement a **universal DID resolution and management** layer. Rather than only supporting a native DID method, it adheres to the *W3C DID Resolution specification* (for a standard resolve() interface) and the *Decentralized Identity Foundation's DID*

Registration specs for create/update operations [3]. In practice, this means the system can manage DIDs from any method: e.g., it could store or cache a `did:ethr` DID Document (after verifying it on Ethereum), resolve a `did:web` by fetching the web-hosted document and optionally recording it on-chain, or generate new DIDs in its own ledger namespace. All DID methods share certain universal operations (create, read, update, deactivate) [10], so our ledger's transaction model is built around those. A future iteration of the system could even act as a **universal DID directory**, where users register pointers to their DIDs of various methods. For now, the design focuses on supporting a native DID method (for DIDs created on this ledger) *and* providing extensibility to reference external DIDs. For example, the **cheqd network** (a Cosmos-based SSI ledger) illustrates this approach: it was built for the `did:cheqd` method but is designed with robust DID management features and APIs that could be extended to other DID methods [11]. Likewise, our platform's architecture does not hard-code any single method's logic, making it *method-agnostic* and future-proof.

Data Storage Model and Structure

Storing identity data on-chain verbatim can be expensive and raise privacy concerns. We propose a **hybrid on-chain/off-chain storage model** that balances consistency, cost, and privacy [12]. In summary, the blockchain will store **small critical pieces of data** (DID identifiers, pointers/hashes, and revocation/status flags) while **large or sensitive data** (like full DID documents, credential contents, or IoT data streams) are kept off-chain in a distributed storage network. This yields an efficient, scalable system where on-chain records act as an index or registry, and off-chain stores hold the bulk data.



Architecture illustrating the hybrid storage approach (inspired by ION/Sidetree): DID operations from many users are aggregated off-chain in a Content-Addressable Storage (CAS) network (green cloud) like IPFS. Periodically, hashes (`#` representing content IDs) of batches of DID operations are anchored to the blockchain (bottom). Peer nodes (blue) each run a local CAS node to fetch/store identity data and a batch writer process. This design allows the blockchain to only store immutable references to identity data, while the detailed DID documents and transaction batches reside in off-chain storage. [13] [14]

On-Chain Data: The blockchain ledger stores an *immutable registry* of DIDs and metadata. For each DID, the ledger holds a record containing: a unique DID identifier (or its cryptographic digest), a pointer to the latest DID Document data (e.g. an IPFS content hash or URL), and possibly a small set of attributes like a timestamp or version number. Only **hashes or references** of the actual DID Document are on-chain [12] [15]. This ensures the ledger remains lightweight and efficient. Storing only the hash (or CID – Content Identifier) guarantees data integrity: anyone resolving the DID can retrieve the off-chain document and verify that its hash matches the on-chain value, proving it hasn't been tampered with [12]. The ledger entry may also include a *checksum of the DID Document* and the public key of the controller (for additional trust at a glance).

In addition, the blockchain stores **credential status registers** and revocation registries. Importantly, we do *not* store entire Verifiable Credentials on-chain (to avoid bloating the ledger and leaking personal data). Instead, for each issued credential, the issuer can write a **credential status entry** to the ledger indicating whether that credential is valid, revoked, suspended, etc., along with a cryptographic identifier for the credential (such as a hash or a registry index) [16]. This allows verifiers to check the status of a credential by querying the ledger without revealing the credential itself. For example, an issuer might register a credential's ID and set its status to "Live" (good) or "Revoked" on the ledger; no private information is stored, but anyone with the credential can look up its revocation status via the ID [16]. The on-chain revocation registry can take the form of a simple smart contract or state tree mapping credential IDs to status values (and perhaps a revocation reason code). This design meets the requirement of supporting credential revocations in a scalable way.

Off-Chain Data (Distributed Storage): The full DID Documents and other large data (credential contents, IoT data, etc.) reside in an off-chain storage layer. This could be a decentralized storage network like **IPFS or Filecoin**, a content-addressable peer-to-peer file system that many identity systems use for scalability [17] [18]. Off-chain storage ensures that large JSON documents or binary data are not put on the blockchain (saving space and cost), while still retaining decentralization. We store each DID Document as a file in IPFS, which gives us a content hash (CID). That CID is what gets anchored on-chain as described. Whenever a DID Document is updated, a new version is stored in IPFS (generating a new CID) and the ledger record is updated with the new hash reference. Because IPFS is content-addressed and tamper-evident, *"nodes don't need to worry about the data being manipulated… the cryptographic hash-based addressing means verification of the data is built into the request itself"* [14]. In other words, if someone tries to maliciously alter a DID Document off-chain, its CID would change and no longer match the ledger's stored hash, so the inconsistency would be detected immediately.

Off-chain storage can also be backed by cloud or database infrastructure if needed (for performance or backup), but any data stored that way must be content-addressable or cryptographically verifiable. For example, an organization could use a distributed database for quick access to DID docs, but still publish the hash of each doc to IPFS and the chain to maintain global verifiability. Additionally, off-chain storage solutions can implement **access control and encryption** for sensitive data [19] [20]. The DID Document itself is generally not encrypted (it's meant to be public keys and endpoints), but any personal attributes or verifiable credential data stored off-chain could be encrypted such that only authorized parties (with the proper keys or via a secure DID-controlled endpoint) can decrypt it [19]. This approach supports privacy: the ledger might list an IPFS hash for a data record, but only someone with the subject's permission can actually fetch and decrypt the content behind that hash.

Batching and Anchoring (Layer 2): To further improve throughput, the system can batch multiple DID operations off-chain and anchor them to the blockchain in a single transaction. This concept is taken from Sidetree-based networks like ION. For example, the software running on each node can collect a batch of DID create/update operations from users, assemble them into a Merkle tree or file stored in IPFS, then write one transaction to the blockchain with the root hash (CID) of that batch [13]. One on-chain transaction could thereby commit hundreds of DID updates. All nodes would see the hash on-chain, retrieve the batch from IPFS, and apply each operation in order. This drastically reduces on-chain transaction load while still maintaining an immutable, timestamped anchor for every DID operation. **Microsoft's ION network** on Bitcoin uses this model: it *"rolls up batches of identity transactions, publishes* *(them) through IPFS, and then writes the address (CID) of that batch to the Bitcoin blockchain"* [13] . Our architecture can do similarly on the chosen ledger. The result is a **log of batched operations** that all nodes can replay deterministically, achieving eventual consensus on DID states without needing a blockchain transaction per DID change. This technique lets the system scale to very high DID volumes (ION, for instance, targets supporting millions or billions of DIDs by batching) while the underlying blockchain only handles a manageable number of anchored records.

Data Model Structure: On-chain, the identity data could be structured as smart contract state or in the ledger's key-value store (depending on the platform). A possible structure is: a mapping from **DID -> (DocumentHash, DocumentMetadata, CredentialStatusList, IoTMetadata)**. The *DocumentHash* is the CID or hash pointing to the latest DID Document in IPFS. *DocumentMetadata* might include the creation and last-updated timestamps (for audit) and perhaps a version number or previous hash (to link history). The *CredentialStatusList* could be a list of IDs of credentials issued by that DID (if the DID is an issuer) along with their status (active/revoked), or this could be an index in a separate global revocation registry keyed by credential ID. *IoTMetadata* might include pointers to IoT-specific info (like a URL of a device info file, or an indicator of the device's public key if the device itself doesn't hold the whole DID Document due to memory constraints). The exact schema can be adjusted, but the guiding principle is minimal on-chain, rich off-chain.

Consensus Mechanism and Consistency

To satisfy the requirement for strong consistency and scalability, the platform uses a **Byzantine Fault Tolerant (BFT) consensus mechanism** on the blockchain layer. This ensures that all validating nodes agree on the order of DID transactions and hence on the state of the DID registry. Every write (creation or update of a DID document, revocation, etc.) goes through consensus so that it is irreversibly committed in a block with finality.

We recommend a **Proof-of-Stake BFT consensus** protocol such as *Tendermint Core*. In Tendermint's algorithm, a rotating leader validator proposes a block of transactions, and validators vote in rounds to commit the block; if >2/3 of validators agree, the block is finalized immediately. This provides immediate consistency: once a DID update transaction is in a block, all nodes have the same view of it and it cannot be reversed (barring >1/3 Byzantine failure). As noted, *"Tendermint Core ensures that all nodes in a network agree on the same state of the blockchain… using a BFT algorithm that achieves fast and final block confirmation"* [21] . Blocks can be produced in ~1 second intervals, giving near real-time confirmation of identity operations to users.

A BFT consensus inherently leans towards **CP (Consistency/Partition tolerance)** in CAP terms. In the event of a network partition, Tendermint validators will not be able to reach the required supermajority consensus, so they will stop committing new blocks until connectivity is restored – thus consistency is maintained (no split-brain ledger), at the expense of availability during the outage. This behavior aligns with the requirement that consistency is paramount. The system "shows an error and blocks users from accessing (during partition) until the issue is fixed," which is characteristic of CP systems [8] . Once the network is whole, the blockchain resumes and all nodes process any backlogged transactions in the agreed order. This design means users might have to wait if the network is severely partitioned, but they will never get two conflicting versions of a DID document.

Consensus Throughput and Scalability: Modern BFT blockchains are quite scalable. Tendermint, for instance, has been benchmarked to **thousands of TPS and ~1 second latency** with dozens of validators [7] . That means even without off-chain batching, the ledger could directly handle millions of DID operations over time (e.g. 1000 tx/sec ~ 86.4 million/day). If more throughput is needed, the network can scale by increasing block size or using parallel chains (sharding). However, our architecture's use of batched anchoring further **amplifies throughput**: hundreds of DID operations can ride in one transaction [13], multiplying the effective TPS. This ensures the consensus layer will not be the bottleneck.

Flexibility in Mechanism: While Tendermint PoS is a strong candidate, the design could adopt other consensus algorithms that prioritize consistency and finality. For example, **Hyperledger Fabric**'s approach (endorsement + ordering service with Raft) could be used in a private deployment – Fabric achieves consistency via an ordering service that totally orders transactions, and supports features like multi-channel partitioning for scaling identity namespaces. Another option is **PBFT variants** (Practical Byzantine Fault Tolerance) used in networks like Hyperledger Indy (which was specifically built for DIDs). The choice may depend on governance: if run by a consortium, a permissioned BFT with known validators is ideal. If a more open network is desired, a permissionless PoS chain with fast finality (e.g. Polkadot's GRANDPA finality or Ethereum 2's Casper FFG with shorter finality windows) could be considered. The overriding requirement is that the consensus offers *finality (no probabilistic forks)* and *high consistency.*

In our design, we assume a **set of validators** that are either pre-selected or elected based on a staking mechanism to participate in consensus. These validators validate each identity transaction's integrity (checking signatures, schemas) before voting to commit it. All transactions are cryptographically linked in blocks, and each block has a hash chain to its predecessor, forming the tamper-proof ledger. Clients (users or agents) can wait for one block confirmation (in BFT that's enough, as finality is immediate) before considering their DID operation complete.

Ordering and Time: Consensus also provides a global ordering of operations, which is crucial for correct DID state. For example, if two update transactions for the same DID are submitted around the same time (perhaps from different devices of the user), the consensus order decides which one is applied first. Because all nodes see the same order, the potential conflicts can be resolved deterministically (e.g., the later operation could be rejected or applied on top of the earlier one, depending on the design). The blockchain timestamp of each block serves as a trusted **time source** for identity events (useful for auditing changes, token expiration, etc.). The **chronological ordering** from the ledger acts as a "global clock" that all parties trust – similar to Sidetree's use of the underlying chain as a *"linear chronological sequencing oracle"* for DID operations [22].

In summary, the consensus mechanism ensures the identity database has a single, agreed-upon state at all times (aside from transient network halts). This strong consistency foundation is critical for reliable DID resolution – users can query any node and get the same DID Document for a given DID, without concern that another part of the network might have a divergent answer.

DID Resolution, Updates, and Verification Mechanisms

A core function of the system is performing **DID operations** – creation, resolution (read), update, and deactivation – in a method-agnostic yet consistent manner. We detail how each of these works in the proposed architecture:

- **DID Creation (Registering a DID):** To create a new DID, an identity owner (or device) generates a cryptographic key pair and chooses a DID method (if the DID is on this native ledger, it might be something like `did:xyz:`). The owner submits a *create DID transaction* to the blockchain, which includes the initial DID Document (or a reference to it) and is signed by the new DID's public key (to prove control). The transaction is endorsed and ordered via consensus, and once on-chain, the DID becomes globally resolvable. The ledger assigns it a unique identifier (e.g. a sequence number or the transaction hash could serve as the DID suffix). The result is an initial DID Document stored off-chain (in IPFS) with an on-chain hash pointer. The **DID method specification** for our ledger will describe how exactly the DID string is formed and how to parse it – but importantly, the create operation conforms to the standard of writing an entry in a decentralized registry. *All nodes* will validate the signature on the create operation (ensuring the public key in the DID Document matches the signature) and then record the new DID in state. This ensures the DID is **self-certified** (the DID contains or can derive a public key that was used to sign its own registration) which prevents malicious registrations. If supporting external DID methods, a similar create flow happens but with method-specific checks – e.g. for an imported did:ethr, the system could require proof of ownership of the corresponding Ethereum address.

- **DID Resolution (Read):** Resolving a DID means retrieving the DID Document (the latest version) for that DID. In this architecture, resolution is straightforward: a resolver (which could be a function on each node or a client library) looks up the DID on the blockchain state. Because each DID is a key in the ledger's state, the node can fetch the record in O(1) time from its state database. The record tells it the content hash (and possibly an IPFS address or other URI) for the DID Document. The resolver then fetches the document from the off-chain storage using that hash (e.g. retrieve from IPFS or a cache). The integrity is verified by recomputing the hash of the fetched document and comparing it to the on-chain hash – if they match, the document is valid and returned to the requester. This process is local for any full node of the network. Clients can either query a node via an API (e.g. a REST or DID Resolver API endpoint that the node provides, like `GET /did/{did}` returning the document JSON) or run a light client that does the lookup and verification. Because of the ledger's consistency, any honest node will give the same result. This design aligns with the W3C DID Resolution spec which defines a `resolve(did)` function that could be backed by our ledger's query interface [3]. For DID methods not natively on this ledger (say `did:web`), the resolver might fetch from the external source – but potentially, the system could cache that externally fetched document on-chain as well for performance and immutability (with appropriate validation and trust logic).

- **DID Update:** DID controllers must be able to rotate keys, add/remove authentication methods, or update service endpoints in their DID Document. An update is performed by submitting a *DID update transaction* to the blockchain. This transaction will reference the DID to update and include the new DID Document or an explicit patch. It must be **authorized** by the current controller of the DID: the transaction is signed with a key that matches one of the verification methods in the existing DID Document (on record). All validating nodes will check the signature against the latest known public keys for that DID. If valid, the transaction proceeds and the DID's on-chain record is updated to point to the new document hash. The new DID Document is stored in IPFS (the client likely does this and includes the CID in the transaction, or the network might do it if given the raw document). This yields an immutable history: the old document remains available via IPFS (since content-addressed storage can preserve it), and the new one is now current. We might also store a previous-document hash or version number in the state to maintain a link to history, or rely on event logs for reconstructing history. In any case, the **DID Document versioning** can be achieved by blockchain logs: every update transaction is recorded, and by replaying them one can build the timeline of changes [23] [24]. Each update, like creation, is a deterministic state transition that all nodes apply. If an update transaction is not properly signed or is malformed, it is rejected by the validators. The effect is that only the rightful DID owner (or someone they delegated via keys) can change their DID information, fulfilling the self-sovereign control aspect.

- **DID Deactivation (Revocation of DID):** A DID might need to be deactivated (e.g. identity no longer valid, key compromised beyond recovery, or an IoT device is decommissioned). The system will support a *deactivate* operation, which is a special kind of update that marks the DID as deactivated. For example, a boolean flag could be set on-chain indicating deactivated status, and the DID Document might be replaced with a minimal document indicating deactivation. Like updates, this must be signed by the controller (unless in emergency the network policy allows an admin or guardian to do it for safety, but typically it's the controller). Once deactivated, resolution of the DID will indicate its revoked status (and ideally no longer return the old public keys as valid). Deactivation operations are recorded immutably too, so anyone auditing can see that at block X the DID was active and at block Y it was deactivated. This ties into **revocation registries**: deactivating a DID could automatically revoke all credentials issued by it (if the business logic dictates), and verifiers checking credentials can be aware if the issuer's DID itself is no longer trustworthy.

- **Verification of Identity Data:** There are two levels of verification: **document authenticity** and **credential verification**. For DID Documents, the blockchain ensures authenticity. When a DID Document is fetched via resolution, trust comes from the blockchain's consensus (the document pointer was written by the DID owner and agreed by the network) and the content hash check. Additionally, the DID Document contains public keys; a relying party can verify that those keys correspond to the ones that were used in the on-chain update signatures (since each update is signed). In fact, because all updates require signatures from authorized keys, the blockchain acts as a **PKI** where the binding of DID to public key is maintained by collective agreement [25]. If needed, a relying party can query the ledger for a proof (like a cryptographic proof or receipt of the transaction) to verify a DID Document's provenance. We can expose a function to get a verifiable credential (in the trust framework sense) about a DID Document's latest state, signed by the ledger or by multiple nodes as witnesses.

For **Verifiable Credentials**, verification involves checking the credential's digital signature (using the issuer DID's public key from its DID Document) and checking the credential's status on the ledger. The verifier would: resolve the issuer's DID via our blockchain to get their public key; verify the VC's signature; then lookup the credential's ID in the on-chain revocation registry to ensure it is not revoked [16]. Our system thus provides the needed *verifiable data registry* functions defined by the W3C: a trusted source for keys and status [26]. If the credential is shown as revoked (or the issuer DID is deactivated), the verifier should reject it. The ledger might support querying credential status by credential hash or an index – a standardized API (perhaps per W3C Credential Status List 2021 or DIF Revocation List spec) can be provided. By keeping these status lists on-chain, we ensure consistency: no matter which node or participant is asked, the answer about revocation will be the same.

- **DID Method Flexibility in Resolution:** Because our architecture is DID method agnostic, the resolution process can be extended to DID methods that aren't "managed" by this ledger. For example, if asked to resolve a `did:web:example.com:alice`, a node's resolver component could fetch `https://example.com/.well-known/did.json` as per the did:web method, then optionally store that document's hash on the ledger for caching. Similarly, for `did:ethr:0x1234...`, the resolver could query the Ethereum network (or a cached state in our ledger if we maintain a synchronization) to get the document from the Ethr DID Registry smart contract. However, to maintain the **agnostic approach**, the platform might integrate with a **universal DID resolver** library that knows how to handle many methods. The architecture allows plugging in such resolution adapters. This way, even if a user's DID lives on another system, our platform can include it in its scope (with appropriate trust – perhaps requiring a proof from that other network anchored into our chain). Over time, if a future identity metasystem emerges (often dubbed **DID:X**), our ledger could transition to being part of that larger network rather than a standalone method [27] [28].

In essence, the design treats the DID operations much like a distributed database CRUD operations, but with cryptographic enforcement of authorization and immutability. The **DID controllers are the only ones who can create or update their records**, and the **ledger ensures all readers get a verified, up-to-date view**. This meets the core DID principle of user control and the requirement of supporting all DID methods (by not hardcoding any single method logic, using a generic interface for resolution and registration) [10].

Scalability Strategies for Millions of DIDs

Scaling to millions of users (and potentially billions of identity documents) requires careful architectural choices. Our solution incorporates several layers of scalability:

1. High-Throughput Ledger Engine: As discussed, using a fast BFT consensus like Tendermint gives a strong baseline: thousands of transactions per second throughput and sub-second confirmation times [7]. This ensures the ledger itself can handle a large volume of operations. The validator set can be sized appropriately; more validators can increase security (at slight cost to performance), but tests have shown networks like Cosmos handle 100+ validators without major throughput loss. The block size can be tuned to accommodate many transactions per block (e.g. if each DID update tx is small, a block could contain thousands). The identity transactions themselves are lightweight – since the heavy data is off-chain, on-chain we mostly deal with hashes and signatures, which are a few hundred bytes each. This means we can pack many operations into each block.

2. Operation Batching (Layer 2 Sidetree Approach): We leverage the Sidetree protocol ideas to offload transaction load. By batching multiple DID updates into one anchored batch, we can scale linearly in the number of operations per transaction. For example, if one batch file contains 1000 batched operations (which is plausible, as each might be a small JSON patch and the whole batch maybe a few hundred KB, storeable on IPFS), and we anchor such a batch every block, then even 1 tx per block yields 1000 updates per block effectively. The ION implementation on Bitcoin demonstrated this kind of scale: it allowed *up to 600,000 DID operations anchored in a single Bitcoin block* by batching (which corresponded to ~60 batched transactions on Bitcoin) – showing how bulk throughput can be achieved [29] [30]. Our network, being purpose-built, could allow similarly large batch sizes if needed. Sidetree's design goal was to *"scale to billions of DIDs while maintaining security and decentralization"* [31] by using batches and content-addressable storage. We adopt that goal here. In practice, we might not need to push to billions immediately, but this strategy provides headroom well beyond millions.

All participating nodes run the logic to **process batches of operations**: they retrieve each batch from IPFS (or whichever CAS) in order, then apply each DID operation in it to their local state. This is effectively a **CRDT (Conflict-Free Replicated Data Type)** approach to DID state, as Sidetree describes – by applying all operations in the same sequence, all nodes converge on the same state [32] [33]. It's eventually consistent (with strong eventual consistency once the anchor is confirmed) without needing per-DID consensus beyond ordering. This drastically cuts down the consensus overhead per DID.

3. Horizontal Scaling and Read Replica Nodes: The network can scale out by adding more nodes to distribute query load. For reading (resolving DIDs), any full node or even read-replica node can serve queries. As millions of devices and users start resolving DIDs, they can query a local node or a nearby gateway node rather than hitting a single centralized server. This is the beauty of decentralized infrastructure – load is naturally distributed. We can also introduce **caching layers**: e.g. an organization might run a caching resolver that stores recently requested DID Documents (since they are immutable by hash, caching is safe) to serve high-volume verification requests quickly. Because data is immutable (addressed by content hash) until updated, caches can be valid for a long time and can be invalidated by the next update event (which changes the hash).

4. Namespacing and Sharding (Future Consideration): If the volume truly grows to many millions of DIDs, we could shard the identity ledger by *DID namespaces*. For instance, DIDs could be partitioned by an identifier prefix or hash range to different blockchain instances that run in parallel, all anchored periodically to a main chain for synchronization. This approach might resemble the design of systems like **Veres One or DIDcomm routing**, but it's optional. In our current design, we assume one global ledger for simplicity (since a single modern blockchain can handle millions of users). However, we could deploy multiple instances of the ledger (say, one per geographic region or one per IoT manufacturer) that cross-register DIDs. They would either use a common global DID method with unique prefixes or interoperate via a federation. This is analogous to scaling databases by splitting by key range. The downside is added complexity for cross-shard resolution. So we lean on the single-ledger with high throughput for now, which should suffice up to a very large scale.

5. Efficient Data Structures and Indexing: The implementation will use optimized data stores for state. For example, if built on Cosmos SDK, it uses a Merkle key-value store (IAVL tree) under the hood, which can handle a huge number of keys (DIDs) while still allowing proofs. We ensure that DID lookups (by exact DID) are indexed – essentially the DID string or its hash is the key in a map, yielding O(log n) or better access. For revocation registries, storing a set or map of revoked credential IDs on-chain could be done with a bitvector or accumulator for efficiency (e.g. using a cryptographic accumulator to represent many revoked credentials in one small digest). Indeed, *AnonCreds* (used in Sovrin/Indy) uses accumulators to allow thousands of revocations with a single on-chain accumulator state. Our design can incorporate a similar approach if needed for credentials – allowing a verifier to get a succinct proof that a credential is not revoked, rather than scanning a list. This is an implementation detail that can greatly reduce on-chain footprint for large revocation lists.

6. Off-chain Computation and Client-Side Filtering: Another way to reduce load on the core system is to push certain queries to the client side. For example, if someone wants to find all DIDs with a certain attribute, rather than making the blockchain handle complex queries, one could use off-chain indexers or search services. The blockchain primarily functions as the source of truth for direct lookups and verifications; more complex directory-like functions (say, searching DIDs by name or credential type) can be handled by separate services that subscribe to the ledger (via events or block feed) and build queryable indices. This separation of concerns keeps the blockchain lean and fast.

Real-World Example: A concrete demonstration of scalability is the **DIAM-IoT framework** which envisions *"a large-scale IoT ecosystem with billions of users and devices"* using DIDs [34]. In that model, blockchain smart contracts manage device DIDs at scale, and our architecture is aligned with it. By combining a capable blockchain core and off-chain distribution of data, the system will be equipped to handle not just millions of current users, but growth into the future.

In summary, by combining a high-performance consensus engine, off-chain data anchoring, batching of operations, and careful indexing, the architecture is designed for **scalability** from day one. We also ensure that adding more nodes improves read throughput (linearly scaling the number of resolution queries that can be served). The design avoids any single bottleneck or centralized component that would cap the scale.

Security and Privacy Considerations

Security is paramount in an identity system. Our design incorporates multiple layers of security:

- **Immutability and Auditability:** The blockchain ledger provides an immutable log of all identity transactions. Once a DID operation (create/update/deactivate) is confirmed, it's tamper-proof and transparently recorded. This enables auditing of identity changes – one can trace the entire history of a DID Document through the transaction log [23] [24]. The immutability protects against malicious actors trying to alter identity data undetected. Even administrators cannot surreptitiously change a DID Document; any change would produce a new transaction on record. This audit trail is critical for trust in verifiable credentials and regulatory compliance (e.g. demonstrating who had access when).

- **Cryptographic Authentication for Updates:** Every update to a DID's data must be signed by the proper key. The ledger's validation logic enforces that the signature on a transaction matches a currently authorized key in the DID's document. If an attacker somehow tries to submit an update for someone else's DID, it will fail signature validation. This is the fundamental self-sovereign security model: *only the DID controller (or an authorized delegate key) can make changes*. We also support **key rotation and recovery** – if a key is compromised or lost, the user can use a recovery method (perhaps a secondary key or social recovery mechanism defined in the DID Document) to rotate to a new key via a special recovery transaction (also secured by either the old key if still in control, or by a pre-authorized recovery process). The Sidetree protocol outlines *"Recovery and Update keys"* for DIDs [35] – we can incorporate a similar concept where a DID Document might list a recovery key or secret, allowing one to recover control if the primary keys are lost.

- **Consistency and Anti-Forking:** By using a single source of truth ledger, we avoid scenarios where two conflicting DID Documents exist for the same DID. This consistency is a security feature: it prevents impersonation or confusion. In some blockchain DID methods (like purely on Ethereum), if a chain fork occurs you could briefly have two histories; our BFT approach with finality avoids that. Additionally, to prevent malicious forks, all validating nodes are incentivized to maintain the single chain (in a permissioned network via governance, in a PoS network via stake slashing for misbehavior). The consensus's fault tolerance also means it can withstand some node compromises without affecting correct operation (Tendermint tolerates up to 1/3 corrupted nodes and still keeps consistency) [6].

- **Privacy of Data:** We design such that **Personally Identifiable Information (PII)** is never required on-chain. DIDs and public keys are generally not PII by themselves (they're pseudonymous identifiers). Verifiable Credentials, which might contain PII (name, age, etc.), are kept off-chain and shared peer-to-peer only with parties that need to see them. The blockchain may store hashes or status of those credentials, which reveals nothing about the content. This addresses privacy regulations: e.g., under GDPR, putting personal data on an immutable ledger is problematic – our approach avoids that by only putting non-personal or hashed data on the immutable ledger. Moreover, by using **hashes and CIDs**, we ensure that even if someone scans the blockchain, they cannot derive the underlying data without access to the actual off-chain content.

- **Selective Disclosure and ZK Integration:** Though not required by the question, our architecture can support advanced privacy techniques like **zero-knowledge proofs (ZKP)** for credentials. For instance, a user could prove to a verifier that their credential is valid (not revoked) by providing a proof that references the ledger's accumulator of revocations, without revealing which credential they hold. The ledger can facilitate this by storing ZK-friendly revocation data (like accumulators or Merkle trees). This means verifiers might check a proof against a ledger-stored Merkle root rather than querying the ledger directly for the credential ID, preserving user privacy. Our design is compatible with such enhancements.

- **Encryption and Access Control:** Any sensitive data that must be stored off-chain (perhaps IoT data or large documents) can be encrypted such that only authorized parties (with the proper decentralized identity or cryptographic key) can decrypt. For example, an IoT sensor's data stream might be encrypted with the public key of authorized subscribers; the DID Document could list a service endpoint for data access which requires authentication. The ledger could store an encrypted symmetric key for data that only the subject or specific verifiers can retrieve using their DIDs (this overlaps with Decentralized Web Nodes / personal data stores concepts). Essentially, **off-chain data management includes access control** – storing a hash on-chain proves data integrity, but the actual data can be gated. This mitigates risks of exposing private info if the storage is public (like IPFS can be public, but data can be ciphered).

- **Protection Against Spam and Misuse:** With a public-facing system, one must consider denial-of-service or spam attacks (e.g., someone registering millions of fake DIDs to bloat the system). We incorporate measures such as **transaction fees or rate limits**. If using a PoS chain, transaction fees (even minimal) provide economic disincentive for spamming. In a permissioned network, we can enforce quotas per participant or require authentication for DID creation (for instance, only authorized organizations can create IoT DIDs in bulk, or users must prove humanity via CAPTCHA/PoW for mass registrations). The design could also include an *allowance for anonymous creation* but throttle it to protect the network's capacity for legitimate users.

- **Smart Contract Security:** If our ledger uses smart contracts (e.g., if built on Ethereum or Fabric chaincode for storing the records), we will adhere to security best practices in those contracts. However, for simplicity, the core DID registry logic might be implemented at the chain application level (like a native module in Cosmos SDK), reducing attack surface compared to arbitrary contracts. The logic for DID operations is relatively straightforward, which helps in auditing and formally verifying it. If any governance or special roles exist (for example, a governance committee can resolve disputes or emergency-revoke a DID in case of misuse), those would be clearly coded and limited in scope to avoid abuse.

- **Trust Model and Governance:** Since identities can be sensitive, governance of the network is a consideration. In a permissioned setup, the consortium members act as trustees of the network's integrity. Regular audits of node software, consensus logs, and maybe even external audits could be employed to ensure no collusion or malicious activity. In a public network, the open community and economic incentives (stake) provide security, as any attempt to attack (fork or alter records) would be evident and economically costly.

- **Complete User Control:** The architecture supports **user-controlled identifiers** in the full sense: users can have **local DID generation** (like `did:key` which is just generated keys) for cases where even the ledger is not needed. Our system can accommodate such DIDs by allowing users to register them if desired or just use them off-ledger when appropriate. The presence of `did:key` or peer DIDs in the ecosystem means sometimes interactions don't hit the ledger at all (which is fine). The ledger primarily is needed for DIDs that require discovery or third-party trust verification. So users can choose which identities to anchor on-chain and which to keep purely peer-to-peer. This gives individuals and organizations flexibility to balance privacy and trust needs.

- **Data Retention and Privacy Compliance:** If a user wants to "remove" their identity (for example, under a right-to-be-forgotten request), how do we handle that given an immutable ledger? This is tricky: by design, we don't put personal data on-chain, so the ledger doesn't actually contain PII that would need removal. A DID could be deactivated, and the DID Document (which might have contained public keys or endpoints) can be wiped from off-chain storage after deactivation if desired (IPFS can "forget" if nodes garbage-collect and no one pins it, though one cannot guarantee all copies deleted). However, since DID Documents by nature have public keys, it's not highly sensitive data. Still, our approach ensures minimal personal data exposure. For credentials, since they're off-chain, the user can simply delete them or not share them further; revoking on-chain just marks them invalid, which is privacy-safe.

- **IoT Security:** Many IoT devices are resource-constrained and often lack secure enclaves. We recommend using **hardware security modules (secure elements)** in devices to store private keys where possible. The architecture supports registering devices with keys that are generated on-device (so the private key never leaves the device). If a device is too constrained, it could be represented by a **delegate or proxy identity** (e.g., a mobile phone or edge gateway holds the credentials for the sensor and interacts with the ledger on its behalf). The DID Document for a device can list a controller DID (perhaps the owner or manufacturer) so that those controllers can rotate keys or revoke the device if it's compromised [5]. This ties into IoT lifecycle management which we address next.

Overall, the security design is multi-faceted: **blockchain security, cryptographic security, operational security, and privacy by design** are all employed to protect the integrity of the identity data and the privacy of the users.

Integrating IoT Devices and Future Support

The architecture is built to onboard not just people but also **IoT devices** as first-class identities. This opens up use cases like device authentication, data integrity, and autonomous device-to-device interactions. Here's how we accommodate IoT:

DID Onboarding for Devices: When a device is manufactured, it can be given a DID (e.g., in a QR code or embedded in firmware) and an initial DID Document. Manufacturers can register these device DIDs on the blockchain, acting as the initial controllers. The **DIAM-IoT model** suggests that during device provisioning, the owner (user) and manufacturer interact: *"users can choose to share device data, and if so, the user receives Verifiable Credentials during the device binding process and registers device DIDs using a manufacturer-managed smart contract"* [36]. In our system, that means a manufacturer could deploy a contract or service to automate registering its devices' DIDs to the ledger at production time. Each device might have an asymmetric keypair injected; the public key goes into its DID Document and the private key stays on the device (ideally in secure storage). The manufacturer's DID could be set as a **controller** on the device's DID Document initially, allowing them to update certain fields or deactivate if needed (for example, if a device is recalled).

Ownership Transfer and Binding: When a consumer buys an IoT device, the device's DID control may need to be transferred to the new owner. The architecture supports **transfer of DID control**: the manufacturer (current controller) issues an update transaction that adds the user's DID as a new controller or transfers control entirely. This is recorded on-chain, establishing the user's authority over that device DID [37] [38]. The user might prove ownership via a credential from the manufacturer (like a VC that says "User X owns Device Y"). Our ledger can store such a credential's status and thus verify ownership. The **device binding** process might involve the user scanning a code and the manufacturer's backend invoking the DID update on the ledger to tie the device to the user. After this, the device can authenticate to services using its DID (the user's keys or delegated keys might sign assertions on its behalf if the device itself is constrained).

IoT Device Credentials: Devices can have Verifiable Credentials too – e.g., a maintenance record, or a certificate of calibration. These credentials can be issued by relevant authorities and recorded (as hashes) on-chain for verification by service technicians or other devices. Our identity storage can easily store IoT-related credential statuses similarly to user credentials. For example, a sensor might present a VC that it's certified by UL; the verifier checks the UL issuer DID in the ledger and the VC status.

Lightweight Crypto for Constrained Devices: Recognizing IoT limitations, we favor **efficient cryptographic algorithms** for keys and credentials. Elliptic curve cryptography (like ED25519 or ECDSA secp256r1) provides strong security with small key sizes and faster computations. *"ECDSA achieves significant security strength, even with shorter key lengths, making it suitable for resource-constrained environments such as IoT devices"* [39]. So, IoT DIDs will typically use elliptic curve keys. Our DID Document schema will accommodate multiple key types, but we'll ensure at least one IoT-friendly algorithm is supported (which is the case for all modern DID methods). Additionally, signatures and proofs used in IoT interactions (like authenticating data messages) can be kept minimal (JSON Web Signatures or COSE signatures in binary form for efficiency).

Edge Proxy and Offline Scenarios: Many IoT devices might not connect directly to the blockchain (due to limited network capability or power). In such cases, an **edge gateway or cloud service** can act on behalf of the device's DID. The gateway can hold a delegate key (or simply forward signed messages from the device). For example, a smart home hub might aggregate the DIDs of all sensors and handle blockchain interactions (resolving other devices, posting events). Our architecture supports this because identity is decentralized – as long as the device's keys sign data, any other party can submit that to the ledger or verify it using the ledger's data. This means a device could be entirely offline and still have a DID on our ledger; others can trust data from it by verifying the signature with the DID Document's key.

Secure Data Exchange and Transparency: IoT generates data that often needs sharing (smart city sensors, etc.). By using DIDs and VCs, we enable a **user-centric access control** for IoT data [40] [41]. For example, a device owner can issue a VC to a service granting it access to the device's data feed. The ledger can host a **Trust Registry** of which parties (DIDs) are trusted manufacturers or service providers [42] [43]. Using smart contracts as **service endpoints** listed in the DID Document, devices could even automate data selling or sharing with consent recorded on-chain. The blockchain's transparency means that any data sharing agreements (encoded as VC or contract) are auditable – aligning with the notion that *"smart contracts guarantee transparency and fairness while sharing IoT data"* [44].

Device Lifecycle Management: Throughout a device's life – manufacture, ownership transfer, maintenance, decommission – the DID method supports transitions. For instance, when a device is decommissioned, its DID can be deactivated on-chain, and perhaps a note added that it's retired (so no one should accept data from it post a certain date). If a device gets compromised (e.g., someone hacks a security camera), the owner or manufacturer can rotate its keys or revoke its DID quickly on the ledger, alerting all relying services. Because of the strong consistency of the ledger, such security actions propagate instantly to all verifiers.

Performance and Network Traffic: IoT integration means potentially a huge number of devices doing identity operations. But note, devices typically will not be constantly updating their DID Document. Most will register once, maybe rotate keys rarely. The more common operations will be those devices producing signed telemetry that verifiers check. Our ledger is optimized for the latter scenario by making resolution fast and cheap. Also, if billions of devices are resolving DIDs simultaneously, the decentralized nature means queries can be served by nearest nodes or caches rather than all hitting one server. If necessary, special **lightweight resolver nodes** can be deployed in IoT networks (like a local blockchain node in a factory serving all machines' queries).

Example Use Case: Consider a smart vehicle. It has a DID from the manufacturer. When sold, the DID's controller updates to the new owner. The vehicle periodically gets software updates – the update server verifies the vehicle's DID (to ensure it's not cloned) and the vehicle verifies the update package's signer DID (to ensure it's an authorized manufacturer). Both use the ledger to resolve each other's DIDs and trust the keys. The vehicle also might share driving data with an insurance company via a VC; the insurer verifies that data's origin by checking the vehicle's DID signature. If the vehicle is taken out of service, the manufacturer deactivates its DID. This entire lifecycle is managed by the combination of on-chain identity state and off-chain credentials.

In conclusion, the architecture readily extends to IoT ecosystems by giving every device a secure, verifiable identity. DIDs bridge the **"identity of things"** gap, providing interoperability across manufacturers and platforms [45] [46]. Our design ensures even constrained devices can participate using efficient crypto and delegate architectures, and it places owners in control of their device identities and data, aligning with user-centric IoT principles. As IoT networks scale, the combination of blockchain and DIDs will help *"break data silos"* and enable devices to interact in a **universal identity framework** [5] securely and autonomously.

Conclusion

We have outlined a comprehensive architecture for a blockchain-based distributed identity database that meets the requirements of a large-scale decentralized identity (DID) system. The proposal is **DID method-agnostic**, leveraging standardized DID resolution and registration interfaces to support any current or future DID methods [3]. A robust **blockchain/DLT platform** forms the core, with a recommended BFT consensus mechanism (e.g. Tendermint PoS) providing strong consistency (CP-style) and immediate finality for identity transactions [6]. This ensures all nodes share a single, reliable view of identity data, critical for trust in verifiable credentials.

We employ a **hybrid storage model** where only hashes and indexes are kept on-chain, and actual DID documents and credentials reside in distributed off-chain storage (like IPFS) [13]. This hybrid approach balances performance, cost, and privacy – the blockchain remains lightweight and fast, while content-addressable storage guarantees data integrity and availability off-chain [14]. Batching of DID operations and anchoring via content hashes allows the system to scale to millions of users (and beyond) by amortizing on-chain transactions across many updates [13]. The design borrows from proven approaches like Sidetree/ION to achieve *eventual strong consistency* of DID state without sacrificing decentralization [47].

We detailed the **mechanics of DID operations**: how DIDs are created with cryptographic proof of control, how resolution yields authentic documents via on-chain pointers, how updates are authorized by keys and immutably recorded, and how credentials are verified including checking revocation registries [16]. The system provides a uniform DID layer for people, organizations, and things, making it flexible enough for diverse identity use cases. Each DID's lifecycle (including key rotations and deactivation) is securely handled through ledger transactions and reflected globally to all participants.

In terms of **scalability**, the architecture uses high-throughput consensus and data partitioning strategies to support a very large number of DIDs and transactions. We discussed horizontal scaling, caching, and even optional sharding for future growth. Coupled with consensus optimizations (BFT finality, no mining delays), this ensures we can onboard **millions of identities with low latency** and handle enterprise or nationwide identity systems on this platform.

Security and privacy considerations are embedded throughout the design. The ledger acts as a secure trust anchor (immutable and auditable), while sensitive personal data is kept off the ledger to preserve privacy [16]. Only hashed or public key information is on-chain, and even that is pseudonymous by design. Each identity change is cryptographically authenticated, thwarting unauthorized tampering. The use of modern crypto (elliptic curve keys, etc.) offers strong security with performance suitable for even constrained IoT devices [39]. Additionally, the architecture is compatible with advanced privacy techniques (like zero-knowledge proofs for credentials) to minimize data exposure when proving attributes.

Finally, we extended the design to **integrate IoT device identities**. By giving devices DIDs and embedding them in the same framework, we enable unified identity management across users and devices. IoT devices gain unique identifiers that can be verified and managed just like human identities, solving interoperability and security issues in current IoT systems [48] [49]. The architecture supports secure device onboarding, ownership transfer via DIDs, and fine-grained access control using verifiable credentials [41]. This positions the platform well for future expansions into autonomous device networks and smart infrastructure where trusted identity for every node (human or machine) is crucial.

In summary, the proposed architecture is a **technical blueprint for a scalable, secure, and flexible decentralized identity platform**. It marries the strengths of blockchains (decentralization, immutability, trust) with the needs of identity management (privacy, control, interoperability). By carefully choosing the ledger technology, using a hybrid storage strategy, and adhering to DID standards, the design achieves a balance of consistency (CP), partition tolerance, and performance suited for real-world deployment. This architecture can serve as the foundation for next-generation identity systems – from nationwide self-sovereign identity schemes to IoT authentication networks – empowering millions of entities with control over their digital identities in a decentralized yet reliable manner.

Sources: The solution integrates insights from W3C and DIF specifications, such as the DID Core spec and Sidetree protocol [9] [47], which guided the method-agnostic and layer-2 aspects. We also referenced real-world implementations and research like Microsoft's ION on Bitcoin (for IPFS batch anchoring) [13], the Cosmos-based cheqd network (for DID operations and PoS BFT usage) [11], and academic proposals for IoT identity frameworks [5], to ensure the design is grounded in state-of-the-art approaches. This architecture, therefore, stands on a solid foundation of existing decentralized identity knowledge while presenting a cohesive plan tailored to the specified requirements.

---

[1] [2] [39] BDIDA-IoT: A Blockchain-Based Decentralized Identity Architecture Enhances the Efficiency of IoT Data Flow
https://www.mdpi.com/2076-3417/14/5/1807

[3] [10] [11] [23] [24] [37] [38] DIDChain: Advancing Supply Chain Data Management with Decentralized Identifiers and Blockchain
https://arxiv.org/html/2406.11356v2

[4] [8] Inery DLT - Blog | Overview of the CAP Theorem and the Blockchain
https://inery.io/blog/article/overview-of-the-cap-theorem-and-the-blockchain/

[5] [34] [36] [40] [41] [44] [45] [46] [48] [49] Decentralized Identity Access Management for IoT Devices | HSC
https://www.hsc.com/resources/blog/decentralized-identity-access-management-for-iot-devices/

6 What is Tendermint | Tendermint Core
https://docs.tendermint.com/v0.34/introduction/what-is-tendermint.html

7 21 Tendermint: The Consensus Engine Powering Next-Gen Blockchains - HeLa
https://helalabs.com/blog/tendermint-the-consensus-engine-powering-next-gen-blockchains/

9 22 25 32 33 35 47 DIF Sidetree Protocol
https://identity.foundation/sidetree/spec/

12 15 19 20 On-chain vs Off-chain - peaq
https://docs.peaq.xyz/build/basic-operations/on-chain-vs-off-chain

13 14 17 18 Decentralized Identity, IPFS and ION | IPFS Blog & News
https://blog.ipfs.tech/2020-06-11-identity-ipfs-ion/

16 Credential Revocation Registry | Hypersign.Id
https://docs.hypersign.id/core-concepts/verifiable-credential-vc/credential-revocation-registry

26 The Role of Verifiable Data Registries in the Verifiable Credential ...
https://technology.a-sit.at/en/the-role-of-verifiable-data-registries-in-the-verifiable-credential-ecosystem/

27 28 31 Toward DID:X
https://decentralgabe.xyz/toward-did-x/

29 ion/docs/design.md at master · decentralized-identity/ion - GitHub
https://github.com/decentralized-identity/ion/blob/master/docs/design.md

30 Aries and ION: Two Different Perspectives of Decentralized Identity ...
https://www.linkedin.com/pulse/aries-ion-two-different-perspectives-decentralized-jesus-rodriguez

42 43 cheqd and walt.id — walt.id
https://walt.id/ecosystem/cheqd