CLedger: A Secure Distributed Certificate Ledger via Named Data

Tianyuan Yu*, Hongcheng Xie†, Siqi Liu*, Xinyu Ma*, Varun Patil*, Xiaohua Jia†, and Lixia Zhang*
*Computer Science Department, University of California, Los Angeles, CA, USA
†Department of Computer Science, City University of Hong Kong, Hong Kong, China

***Abstract*—Named-Data Networking (NDN) is a novel network that secures network communication by fetching semantically named and secured data. All data packets in NDN are signed by producers and verified by data consumers. Therefore, it is vital to have producers' certificates available all the time. In this paper, we describe the design of CLedger, a secure distributed certificate ledger, to ensure certificate availability in NDN. CLedger logs certificate records in an immutable Directed Acyclic Graph (DAG) structure and replicates the DAG among a set of distributed loggers. We implemented CLedger using NDN's pub/sub API, and evaluated our design through an emulated deployment setting. Our initial evaluation results show that CLedger is effective, efficient, and resilient to failures.**

I. Introduction

Named-Data Networking (NDN) secures network communication by fetching semantically named and secured data [1], [2]. NDN Data packets are cryptographically signed by their producers' keys, this enables data consumers to use the producers' certificates to authenticate all received data packets. Thus certificate availability is a prerequisite to enabling secure communications.

In today's IP-based Internet, almost all web communications run over secure HTTP, which also needs to authenticate web servers. The availability of web servers' certificates is assured by IP's *always-on* model: all web servers are online and reachable all the time, so that browsers can directly obtain the certificates from web servers. In contrast, NDN's data-centric design enables consumers to fetch data even when their producers are offline, or unreachable due to temporary failures. These features enable NDN to support disruption-tolerant networking natively, but also raise the need for making certificates available all the time, even when certificate owners are offline or unreachable.

We propose CLedger, a distributed, secure, and resilient certificate ledger, to meet the above critical need. CLedger consists of a set of distributed loggers that store certificates from authorized certificate owners. CLedger makes use of NDN's State Vector Sync (SVS), a distributed dataset synchronization protocol to replicate existing certificates among all loggers; and SVSPS, a pub/sub API built on top of SVS for secured multiparty communications. CLedger utilizes an append-only data structure, HashDAG, which is based on Directed Acyclic Graph (DAG). HashDAG assures that, once a certificate is stored in CLedger, it cannot be removed.

Our contributions can be summarized as follows.

- To address certificate availability problem, we developed a secure distributed certificate ledger, CLedger, to run in NDN networks. CLedger uses an append-only data structure shared among all loggers to store certificates.
- We evaluate the performance of CLedger. The evaluation results show that our proposed ledger network is efficient and practical to be deployed.

The rest of this paper is organized as follows. §II discusses basic building blocks we use in our proposed system. §III formulates the problem and states CLedger design goals. We present the details of our proposed system in §IV. §V gives the performance evaluation that verifies CLedger design. We summarize related works in §VI, discuss other potential usage of CLedger and some of our design decisions in §VII. Finally, we conclude our work and discuss future work in §VIII.

II. Background

In this section, we provide an overview of the Named-Data Networking and the two major building blocks used in the CLedger design, Pub/Sub support over State Vector Sync and Directed Acyclic Graph.

A. Named Data Networking

Instead of using end-host addresses, Named-Data Networking (NDN) [1] uses application data names for communications. Consumers fetch data by names carried in *Interest* packets, and the network returns the named and secured *Data* packets.

To effectively verify the trustworthiness of received data, data consumers use a set of rules, called *trust schemas*, to define trust relations among NDN entities[1]. These rules defines which key (identified by the key's semantically meaningful name) should be used to sign which piece of data. All the entities managed by the same administrator form a *trust domain* [3], and each entity is bootstrapped with the administrator's self-signed certificate as its *trust anchor*, together with its trust schema which defines its trust relations with other entities in the same domain. All certificate signing chains in this trust domain derive from the trust anchor.

B. State Vector Sync (SVS) and SVSPS

State Vector Sync (SVS) [4] is a distributed dataset synchronization protocol running over NDN networks. Utilizing SVS,

[1]An NDN entity is an application process or network communication participant in an NDN network.

SVS Publish/Subscribe API (SVSPS) provides secured multi-party communications. SVSPS enables application developers to write distributed applications by focusing on high-level application logic, naming conventions and security policies, without dealing with low-level protocol details. Applications can join the same Pub/Sub group to start secured data communications. When an application process wishes to publish a data object to the group, SVSPS takes the object, segments as needed, and signs into Data packets, then informs others of the new data through SVS protocol. All packet exchanges within SVSPS are signed so that SVSPS is able to determine whether a Pub/Sub participant is authorized based on the trust schema, and discards packets from unauthorized participants.

C. Directed Acyclic Graph

Directed Acyclic Graph (DAG) is a directed graph without cycles. In the field of storage, DAG is a graph-based data structure where nodes are data records and directed edges are reference relations among data records.

DAG has been used in several distributed storage designs [5]–[7]. Unlike blockchain-based designs, a storage node can insert a received record to its local DAG copy immediately, and the DAG in each storage node can be different at a specific time point. These DAGs can merge efficiently and reach eventual consistency, i.e., every record is stored by all DAGs eventually. Moreover, if references are built by hash functions, a record will contain a cryptographic hash of the previous record. Modifications to that record are detectable unless its referrers are also compromised.

III. Problem Statements

A. Problem Formulation

We consider a scenario as follows. There is a distributed ledger CLedger, multiple CertOwners who want their certificates to be stored in the ledger, and multiple CertFetchers who want to fetch certificates. As shown in Figure 1, their roles are summarized as follows.

- *Logger:* CLedger consists of Loggers who accept legitimate certificate submissions from CertOwners and publish *records* within CLedger. Each record is a semantically named and secured data object that includes a certificate from CertOwner.
- *CertOwner:* CertOwners are legitimate certificate owners who are authorized by the trust schema of CLedger to submit certificates to Loggers.
- *CertFetcher:* CertFetchers are application data consumers who use CLedger to fetch the data producers' certificates.

A viable design that consists of the above components should have no single point of failure and ensure record immutability, so that a recorded certificate cannot be removed from the system.

B. Design Goals

We design CLedger for systems which have less than K Loggers making operational errors or performing malicious attacks,

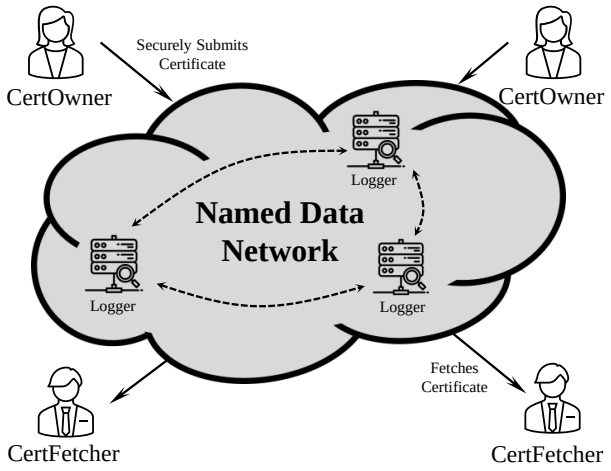


Fig. 1: A scenario where CLedger has Loggers deployed in the network, and CertOwners would like to use CLedger to provide their certificate availability, while CertFetchers want to obtain certificates from CLedger in order to validate application data.

by either refusing CertFetcher certificate fetching request or deleting the record from their local storage. This assumption is a generalization of the assumption in Practical Byzantine Fault Tolerance [8]. With the above assumption, our design achieves the following three goals:

- Certificates are available if any Logger is available.
- Only authorized CertOwners are allowed to submit certificates to Loggers.
- If a Logger accepts an authorized submission from a CertOwner, this record is stored in at least K loggers.

IV. CLedger

In this section, we first identify questions that must be addressed in the CLedger design and describe our solutions to each question. We then show the overall system workflows when CLedger is in operation.

Design Question 1: How do CertOwners submit certificates to Loggers?

We utilize the submission protocol developed by our previous work [9] to enable secured certificate submission.

When submitting an certificate, which is a Data packet, the CertOwner encapsulates it into another Data packet, and sends an notification Interest to inform CLedger that there is a certificate ready to be submitted. CLedger Loggers register a shared routing prefix (*e.g.,*"`/ndnfit/LEDGER`"). Therefore, when CertOwners express an Interest with CLedger's prefix, the network will anycast the packet to its nearest Logger.

After receiving the notification, a Logger sends another Interest back to retrieve the submission. After validating both the CertOwner's submission Data and the submitted certificate in this Data, this Logger replies to the CertOwner's notification Interest with a Data packet as an acknowledgment. Thus,

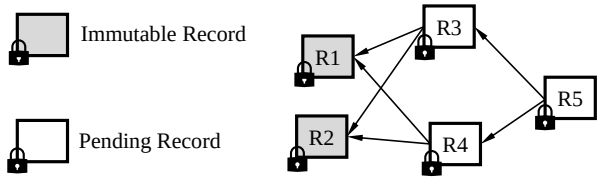


Fig. 2: A DAG example where $K = 3$ and each record is published by an unique Logger.

the CertOwner is informed the submission process has been completed.

When submitting certificates, a CertOwner signs the submission with its certificate private key, so Loggers are able to authenticate the CertOwner and validate whether the CertOwner is authorized by the trust schema to use CLedger. Upon receiving a validated submission, a Logger produces a record to log the submission.

Loggers securely synchronize records via SVSPS (§II-B), where Loggers join the same Publish/Subscribe group and subscribe to new records produced by other Loggers.

Design Question 2: How does CLedger ensure a record is immutable?

Since CLedger defines record immutability as at having replications in least K Loggers, a natural answer to this question is having K Loggers' signatures on the record, claiming they have replicated it.

CLedger realizes this idea by introducing a new data structure HashDAG inspired by IOTA Tangle [5]. In HashDAG, each vertex is a record, and each directed edge refers to a previous record, which is implemented by storing the hash of the previous record in the current record. HashDAG edges represent replication relations. For instance, $R5$ in Figure 2 refers to $R3$ and $R4$ directly, and refers to $R1$ and $R2$ indirectly. This reference relation indicates the Logger who produces $R5$ claims it has seen and replicated $R3$ and $R4$ and all their dependencies (*i.e.,* $R1$ and $R2$) and replicated them into its local copy of HashDAG. Therefore, if a record has been referred by more than K unique Loggers, that record becomes immutable; otherwise, we refer to its status as pending. For example, $R1$ and $R2$ in Figure 2 are immutable since their parents are produced by three unique Loggers, while $R3$, $R4$, and $R5$ are still pending.

Obviously, the latency for a pending record to become immutable is up to how often Loggers are publishing. However, Logger publication rate further depends on how many nearby CertOwners are submitting certificates. If only a few Loggers are publishing records, the immutability latency increases, or even becomes infinite when there is no new Logger contribute references by producing record and refer to the existing tail.

In order to address this problem, we design the *dummy record* mechanism for CLedger. Each Logger maintains a timer and refreshes it whenever the Logger receives a new record from SVSPS. If no records are received in a given amount of time, the Logger publishes a dummy record that does not contain a certificate but only refers to all tail records.

We made two decisions to prevent CLedger keeping publishing dummy records when there is no new certificate submission. First, a Logger will not refresh its timer when receiving a dummy record. Second, a dummy record can only refer to the records that contain certificates. Therefore when HashDAG growth stalls, more Loggers can participate to help pending records to be immutable.

CLedger In Operation Now we use an example shown in Figure 3 to further demonstrate CLedger operations.

We consider a case where Alice is a CertOwner and Bob is a CertFetcher. At the beginning, Alice submits her certificate to CLedger using the submission protocol. Alice's nearest Logger receives the submission, validates it with trust schema, and publishes a record $R5$ that includes Alice's certificate and references to all HashDAG tail records $R3$ and $R4$.

As all Loggers are in the Publish/Subscribe group, other Loggers receive this $R5$ from record subscription, and append it into their HashDAG. Because $R5$ has become the tail record, when other Loggers publish new records, their records will refer to $R5$. When there are at least K Loggers have directly or indirectly referred to $R5$ in their record publications, $R5$ becomes immutable.

We assume that Bob starts fetching Alice's certificate after Alice's certificate submission finishes. Bob expresses an Interest with Alice's certificate name[2]. Bob's nearest Logger receives the Interest and looks up its local copy of HashDAG for the record that contains Alice's certificate.

If such a record exists in HashDAG, the Logger extracts Alice's certificate from it and replies to the Interest with the certificate found. Otherwise, the Logger replies to the Interest with a Data packet that shares the same name prefix with Bob's Interest but has an empty content. Logger signs the reply to inform Bob that CLedger has no record for *which* certificate[3].

In the case shown in Figure 3, the Logger nearest to Bob has learned $R5$ from SVSPS, hence successfully looks up $R5$ from its local copy of HashDAG and replies to Bob's Interest with Alice's certificate.

V. Experimental Evaluations

In this section, we evaluate the performance of CLedger with various parameter settings, including the submission latency, the fetching latency, the immutability latency, and the Certificate/Total record ratio.

A. Evaluation Setup

To evaluate CLedger's performance, we implemented a prototype[4] based on ndn-cxx. Our prototype uses LevelDB as loggers' local database.

[2]Interest packets can carry forwarding hint that hints the routers where to forward the packet. In our case, Bob's certificate fetching Interest carries the forwarding hint to the CLedger shared routing prefix.

[3]The Logger can use the Data FreshnessPeriod to control the timeliness of this information, so that Bob knows when to resend the request.

[4]https://github.com/UCLA-IRL/cert-ledger

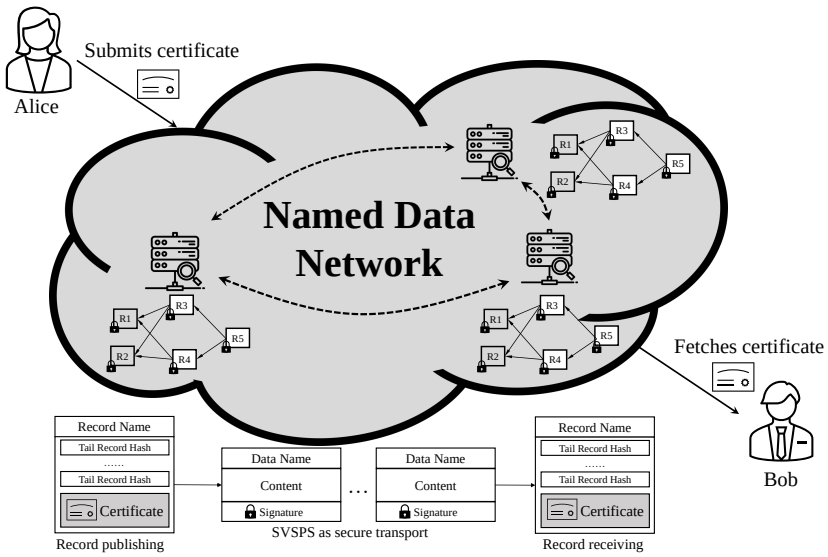
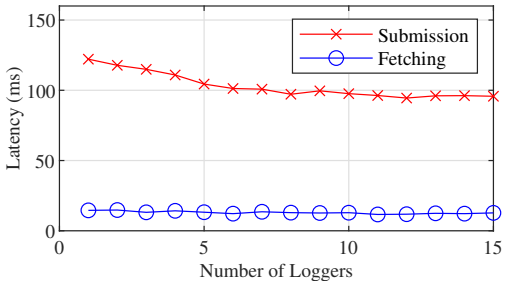
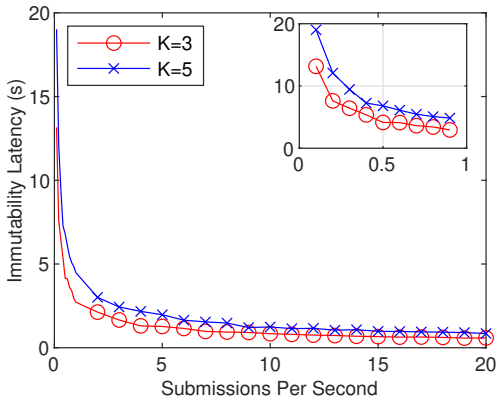


Fig. 3: An CLedger example where Alice is the CertOwner while Bob is the CertFetcher.



(a) Availability Latency



(b) Immutability Latency

Fig. 4: Latency Performance

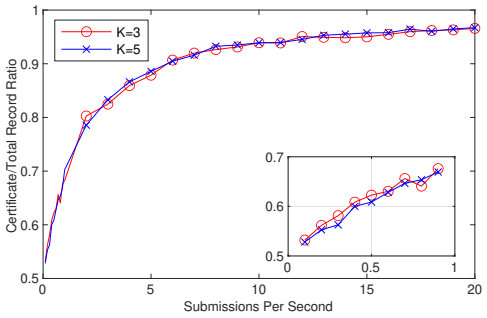


Fig. 5: Certificate/Total Record Ratio

We use Mini-NDN [10] to emulate CLedger in a real-world ISP network topology [11] that includes 115 nodes. They are connected to each other via 10 ms per-link delay. Our evaluations were hosted on a server equipped with Ubuntu 20.04, AMD EPYC 7702P with 64 cores and 256 GB RAM. Each evaluation result is the mean of 10 trials.

B. Submission Latency and Fetching Latency

We measure the certificate submission latency and the fetching latency in this section, to evaluate the performance of certificate availability provided by CLedger. According to §IV, we define that submission latency is the time elapsed from CertOwner sending the notification interest to receiving the acknowledgement. Fetching latency is the time elapsed from CertFetch sending a certificate Interest to receiving the requested certificate. In order to serve more CertOwners, Loggers in actual deployments are more likely on the nodes which have more neighbours. Therefore to emulate the logger selection process, we sort all the nodes in descending order of edge counts, and select the top N nodes as Loggers in each trial[5], with N ranging from 1 to 15.

For submission latency evaluation, we assume that submission events have a Poisson distribution and set λ as 10, i.e., the expected number of submissions per second is 10. For each submission, we randomly choose a node as a CertOwner

[5]We exclude the top 2% nodes before the selection, because these nodes are probably routers in the real-world scenario.

and submit its certificate. The submission evaluation lasts for 120 seconds, and we evaluate the mean submission latency of all submitted certificates. As shown in Fig. 4a, the submission latency decreases as the number of Loggers increases, because the larger number of Loggers, the higher probability that a CertOwner can find a nearby Logger with small hops.

After all certificates are submitted into the system, we evaluate the certificate fetching latency. We also assume that fetching events have the same distribution as submission events. For each fetching request, we randomly choose a node as a CertFetcher and fetch a random certificate already in the system. We assume that the fetched certificates have the Zipf distribution. It also lasts for 120 seconds, and we evaluate the mean latency of all fetching requests. As shown in Fig. 4a, the fetching latency remains stable as the number of Loggers increases, because popular certificates are fetched several times under Zipf distribution. Due to the cache mechanism in NDN, they are cached by the intermediate nodes so that a CertFetcher can fetch them from the intermediate routers.

C. Immutability Latency

In this section, we measure the immutability latency to evaluate the immutability performance provided by CLedger. In this experiment, we picked 15 Loggers with the same node selection process. The dummy record timer for each Logger is set randomly from 0.1 s to 20 s. As illustrated in Fig. 4b, we evaluate the immutability latency with different submission speed rates and K settings, where the figure on the corner shows the immutability latency with low submission speed. The immutability latency in both cases $K = 3$ and $K = 5$ decrease as the submission rate increases. The increasing number of submitted records per second makes a newly submitted record referenced quickly. The latency in the case $K = 5$ is higher than that in the case $K = 3$, because a submitted record needs more descendent records for larger K. Our evaluation results show that our record can be immutable after a short period of time. When the submission speed is 20 records per second and $K = 5$, a record only needs 862.89 ms to be immutable.

D. Certificate/Total Record Ratio

As CLedger uses dummy records to accelerate the immutability, we evaluate the proportion of certificate records in all records. In Fig. 5, we evaluate the proportion of certificates from CertOwners with different submission speed rates and K settings, where the figure on the corner shows the immutability latency with low submission speed. The number of Loggers is set as 15, and we select them based on the same policy we used in §V-C. We can find that the proportions in both cases $K = 3$ and $K = 5$ increase when the submission speed rate increases. When the submission speed rates are over 15 records per second, the proportions of certificate records are over 95%.

VI. Related Works

A. Certificate Transparency

Today's Internet has a decade of experience in logging certificates in append-only data structures. Certificate Transparency (CT) [12] is a solution that logs issued certificates in Merkle Tree, an append-only data structure. The Merkle Tree with logged certificates is stored in multiple independent CT log servers. CT helps mitigate the misissuance problem in Certificate Authorities (CA) through browser-end forcing. Browsers, such as Chrome, require that a certificate must be logged in multiple CT servers. Otherwise, it will be rejected by the browser. Meanwhile, monitors periodically check the latest update of CT log servers. Thus they can detect misissued certificates in Web through analyzing CT logs.

We acknowledge that CT has made significant contributions to today's Internet security. But we also note that, system consistency is a non-goal when CT was first proposed. A key difference in CT design is that, each Logger independently maintains its own certificate storage. The stored certificates may differ among different Loggers, which leads to system inconsistency. Therefore, browser vendors control the trust policy that decides to accept which Logger. On the contrary, CLedger synchronizes records among Loggers via SVSPS, and secures the system by executing trust schema that is bootstrapped to all domain entities.

Moreover, as we mentioned in §I, today's Internet follows an always-on model. CT aims at addressing the transparency issue. It supposes that the browser should have the certificate from the server, which cannot be used to address the certificate availability problem.

B. Blockchain-based Certificate Ledger

Many Proposals [13]–[15] are built on using the blockchain to achieve X.509 certificate transparency. Cecoin [13] uses Merkle Patricia Tree to represent its domains and the certificates records and accepts modification to this tree using blockchain and its cryptographic verification methods. The proposal by Ze et al. [14] builds the function of CT with blockchain and builds domain ownership proof with a group of verifying parties. The proposal by Kubilay et al [15] provides a model of building a decentralized Byzantine fault tolerant trust on a blockchain, with inherent support of public append-only log property provided by the clock-chain.

However, CLedger differs from the blockchain-based designs in the following two ways. First, based on whether require authorization to publish and access records or not, blockchain-based ledger can be further categorized into permissioned and permissionless ledger. Different from IP-based permissioned ledgers, CLedger realizes record authorization with a data-centric design that leverages semantically meaningful names in the network layer, which can authorize record publications according to trust schema, without adding a map from ledger identifier to its address.

Second, the blockchain technique requires the system achieving global consensus on record orders, which leads to extra communication overhead. However, as certificates are stateless, the consistency of record order is not important. DAG in CLedger does not require a total order among all records but still ensures the immutability for each record. It is a lightweight design that is more suitable for certificate availability.

VII. Discussion

Revocation Availability Although CLedger aims to address the certificate availability problem, the same solution can also be applied to the availability of certificate revocation records. The revocation framework CertRevoke [9] requires a ledger that immutably stores the revocation records and guarantees their availability. CLedger is a natural fit for that need.

Supporting Inter-domain Availability Although the current CLedger design started from providing the certificate availability within one trust domain, CLedger can also achieves the same goals for multiple domains. In a multi-domain deployment, each domain contributes Loggers with trust schema that specifies rules on how to authenticate and authorize packets from other trust domains [16]. Therefore, federated Loggers can provide immutable certificate storage for CertOwners and CertFetchers from multiple domains.

Dummy Record Timer CLedger relies on a dummy record timer to enable records to be immutable timely. However, system operators should carefully consider the timer interval.

First, the interval of the timer should present system operators' consideration of how urgently records need to be immutable. As shown in Figure 5, when CLedger only has few submissions per second, the ratio between CertOwner-submitted records and dummy records drops significantly. This indicates CLedger immutably stores certificates with higher communication overhead.

Second, in order to prevent multiple Loggers from publishing dummy records simultaneously, each Logger should use a variable length timer. In our evaluation, each Logger uses a randomized timer whose interval is evenly distributed but with the same mean value. This might not be the optimal choice, since the urgency of record immutability is increasing as time increases when no one is publishing new records. A better randomization strategy is to have the probability distribution function prefers shorter timers.

VIII. Conclusion and Future Works

In this paper, we proposed a secure distributed certificate ledger, CLedger, via named data. CLedger provides certificate and revocation availability based on immutably replicated records. Our evaluation results showed that our proposed CLedger is feasible and efficient.

A lesson we learned from designing CLedger is that a trust schema enables CLedger security design through semantically matching data name with its signer name. Therefore, CLedger is able to control not only who can submit certificates, but also *who* can submit *which* certificate, without adding another layer of indirection. Hence, CLedger does not need a separate security design to authorize CertOwners, but relies on the trust schema to validate their submission data and ensure every submission is authenticated and legitimate.

We also learned SVSPS is a useful tool to build secure NDN applications. By using SVSPS, CLedger securely synchronizes records without dealing with low-level protocol details, but only handles high-level data naming and security policies.

In the future, we will investigate an adaptive timer for the dummy record design to further reduce the records' immutability latency while remaining the HashDAG efficiency. We also plan to deploy CLedger on NDN Testbed [17] as a solution to Testbed certificate availability problem and support certificate availability across multiple trust domains.

Acknowledgement

We would like to thank the anonymous reviewers for their valuable comments. This work was supported in part by US National Science Foundation under awards 2019085 and 2126148, and Research Grants Council of Hong Kong under CityU 11213920 and R1012-21.

References

[1] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," pp. 66–73, 2014.

[2] A. Afanasyev, T. Refaei, L. Wang, and L. Zhang, "A brief introduction to Named Data Networking," in *Proc. of MILCOM*, Oct. 2018.

[3] K. Nichols, V. Jacobson, and R. King, "Defined-Trust Transport (DeftT) Protocol for Limited Domains," Internet Engineering Task Force, Internet-Draft draft-nichols-tsv-defined-trust-transport-00, 07 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-nichols-tsv-defined-trust-transport

[4] P. Moll, V. Patil, N. Sabharwal, and L. Zhang, "A brief introduction to state vector sync," *NDN, NDN Memo, Technical Report NDN-0073, Revision 2*, 2021.

[5] Y. Li, B. Cao, M. Peng, L. Zhang, L. Zhang, D. Feng, and J. Yu, "Direct acyclic graph-based ledger for internet of things: Performance and security analysis," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1643–1656, 2020.

[6] Z. Zhang, V. Vasavada, R. King, and L. Zhang, "Proof of authentication for private distributed ledger," in *Proceedings of the NDSS Workshop on Decentralised IoT Systems and Security (DISS)*, 2019.

[7] S. Liu, P. Moll, and L. Zhang, "Mnemosyne: An immutable distributed logging framework over named data networking," in *Proceedings of the 8th ACM Conference on Information-Centric Networking*, ser. ICN '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 130–132. [Online]. Available: https://doi.org/10.1145/3460417.3483375

[8] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. USA: USENIX Association, 1999, p. 173–186.

[9] T. Yu, H. Xie, S. Liu, X. Ma, X. Jia, and L. Zhang, "Certrevoke: A certificate revocation framework for named data networking," in *Proceedings of the 9th ACM Conference on Information-Centric Networking*, 2022.

[10] The NDN Team, "Mini-NDN: A Mininet based NDN emulator," 2022, accessed: 2022-11-01. [Online]. Available: https://github.com/named-data/mini-ndn

[11] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[12] B. Laurie, "Certificate transparency: Public, verifiable, append-only logs," *Queue*, vol. 12, no. 8, pp. 10–19, 2014.

[13] B. Qin, J. Huang, Q. Wang, X. Luo, B. Liang, and W. Shi, "Cecoin: A decentralized pki mitigating mitm attacks," *Future Generation Computer Systems*, vol. 107, pp. 805–815, 2020.

[14] Z. Wang, J. Lin, Q. Cai, Q. Wang, D. Zha, and J. Jing, "Blockchain-based certificate transparency and revocation transparency," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 681–697, 2022.

[15] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar, "Certledger: A new pki model with certificate transparency based on blockchain," *Computers & Security*, vol. 85, pp. 333–352, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818313014

[16] T. Yu, X. Ma, H. Xie, Y. Kocaoğullar, and L. Zhang, "Intertrust: establishing inter-zone trust relationships," in *Proceedings of the 9th ACM Conference on Information-Centric Networking*, 2022, pp. 180–182.

[17] The NDN Team, "Ndn testbed," Online at https://named-data.net/ndn-testbed/, 2022.