CertRevoke: A Certificate Revocation Framework for Named Data Networking

Tianyuan Yu
UCLA
Los Angeles, USA
tianyuan@cs.ucla.edu

Hongcheng Xie
City University of Hong Kong
Hong Kong, China
hongcheng.xie@my.cityu.edu.hk

Siqi Liu
UCLA
Los Angeles, USA
siqi.liu@ucla.edu

Xinyu Ma
UCLA
Los Angeles, USA
xinyu.ma@cs.ucla.edu

Xiaohua Jia
City University of Hong Kong
Hong Kong, China
csjia@cityu.edu.hk

Lixia Zhang
UCLA
Los Angeles, USA
lixia@cs.ucla.edu

ABSTRACT

Named Data Networking (NDN) secures network communications by requiring all data packets to be signed upon production. This requirement makes usable and efficient NDN certificate issuance and revocation essential for NDN operations. In this paper, we first investigate and clarify core concepts related to NDN certificate revocation, then proceed with the design of CertRevoke, an NDN certificate revocation framework. CertRevoke utilizes naming conventions and trust schema to ensure certificate owners and issuers legitimately produce in-network cacheable records for revoked certificates. We evaluate the security properties and performance of CertRevoke through case studies. Our results show that deploying CertRevoke in an operational NDN network is feasible.

CCS CONCEPTS

• **Networks** → **Security protocols**; • **Security and privacy** → **Authentication**.

KEYWORDS

Named data networking, Information-centric networking, Trust management, Certificate revocations

ACM Reference Format:
Tianyuan Yu, Hongcheng Xie, Siqi Liu, Xinyu Ma, Xiaohua Jia, and Lixia Zhang. 2022. CertRevoke: A Certificate Revocation Framework for Named Data Networking. In *9th ACM Conference on Information-Centric Networking (ICN '22), September 19–21, 2022, Osaka, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3517212.3558079

1 INTRODUCTION

Securing data communications requires usable solutions for data integrity, authenticity, and confidentiality. Named Data Networking (NDN) [45] offers essential building blocks for data security by signing data during production and encrypting data whenever needed [49]. These security supports require effective and efficient mechanisms to handle certificate issuance and revocation. A number of existing works [20, 26, 46, 48] have explored the design space of NDN certificate issuance. However, a systematic understanding of certificate revocation design approaches is still missing. Although NDNCERT [46] proposed a simple procedure to let the certificate owner notify its issuer that the certificate should be revoked, this procedure does not provide a general solution for certificate revocation.

A certificate revocation designs need to answer the following three research questions:
***RQ1*: Which entity can legitimately revoke a certificate?** Each certificate has its issuer and verification chain, which terminates at the trust anchor. If the certificate issuer (*e.g.,* a site CA on the NDN Testbed [39]) is not the trust anchor, one also needs to consider the role that the trust anchor may play in the revocation process.
***RQ2*: What procedures are needed to revoke a certificate?** Certificate revocation needs protocol designs to ensure the legitimacy of each step in the process. For example, one needs to define data structures to record certificate revocation events regarding *who revokes which certificate at when*, and the corresponding procedures to legitimately produce and secure the recordings.
***RQ3*: How do data consumers learn certificates' revocation status?** To enable data consumers to learn the validity status of the revoked certificate, the revocation design must consider data authenticity and timeliness without incurring high communication costs in the data validation process.

In this work, we propose a framework for NDN certificate revocation and make the following contributions:

- We clarify the concepts between certificate revocation and key revocation.
- We investigate the design space of certificate revocation by articulating the system model and design requirements.
- We design and implement CertRevoke as a certificate revocation framework for NDN.

In the rest of this paper, §2 provides an overview of NDN security design. §3 describes the necessity of certificate revocation and clarifies the concept of certificate versus key revocation. §4 presents our certificate revocation framework CertRevoke, and §5 discusses our prototype implementation and evaluations. We describe related works in §6 and discuss some of our design decisions and lessons learned in §7. Finally, we conclude our work in §8.

2 BACKGROUND

NDN views a network as a collection of named entities with trust relations with each other. NDN entities are applications or any communicating parties in the network, each belonging to a *trust zone* [24]. In order to join a trust zone, an entity needs to go through the *security bootstrapping* step to obtain three basic pieces of NDN security information from the *trust zone controller* [43]: trust anchor, certificate, and trust schema.

- *Trust anchor*: The trust anchor is a self-signed certificate for the trust zone, which the trust zone controller owns. It is the termination point of cryptographical authentication inside the trust zone, thereby establishing an entity's initial trust relation to its trust zone controller.
- *Certificate*: The trust zone controller endorses an entity by signing the binding between a name and public key; the result of this endorsement is an NDN certificate. A certificate is a piece of named data, and its name follows the naming convention "`/<prefix>/KEY/<keyid>/<issuer>/<version>`". A certificate named "`/ndn/siteA/depart/cs/KEY/223/alice/001`" conveys that the entity "`/ndn/siteA/depart/cs`" is the certificate owner, and its key ID (e.g. app-1) uniquely identifies a public-private key pair belonging to this entity. Furthermore, the certificate name also indicates it is issued by "`alice`" with version number "`001`".
- *Trust Schema*: NDN defines trust schema [44] to restrict the signing power of keys. In authenticating received data packets, applications use trust schema to verify whether the data is produced by the right party by checking whether its name and signer's key follow defined name patterns.

After security bootstrapping, an NDN entity can produce and consume data authentically.

3 WHEN WE NEED CERTIFICATE REVOCATION

3.1 Invalidating Bad Certificates

A certificate represents the trust zone controller's endorsement of a name-key binding within a validity period. However, there can be times when this endorsement may need to be terminated before the validity period expires. For example, when the entity renews its certificate before the expiration time, when the entity's private key gets compromised, or when the trust zone controller finds that it issued a certificate to a wrong entity by mistake. All the above cases require the issued certificates to be invalidated.

There exist several possible approaches to invalidate an NDN certificate.

Trust Schema Renewal: A revised trust schema can inform data consumers to reject the certificates that have been invalidated. This approach requires the trust zone controller to promptly distribute the renewed trust schema to all affected entities within its trust zone. Data consumers that do not receive the trust schema renewal in time will still consider the invalidated certificates legitimate. Therefore, although revoking certificates by renewing the trust schema can be a viable solution for small trust zones with intra-zone data communications, it is not an effective solution in general.

Short-lived Certificates: If a certificate is issued with a short validity period, it automatically becomes invalidated upon expiration, reducing the chance of getting compromised or causing big damages even in case it gets compromised. A certificate's validity period may be a few days [41] or even a few hours [12]. The drawback of this approach is the increased workload of certificate issuers, which goes up reversely proportional to the certificates' validity period lengths. However, the recent success of Let's Encrypt [1] in the existing Certificate Authorities (CA) market suggests that one can effectively manage certificate renewal tasks through automation, such as by using the Automatic Certificate Management Environment (ACME) protocol [4] or CertBot [7]. NDN-based security solutions can further facilitate automated key management by leveraging naming conventions.

Using short-lived certificates can make an effective engineering alternative to certificate revocations for leaf or edge (*e.g.,* user) certificates. However, renewing higher-level certificates may still incur substantial cost, as a change to a high-level certificate leads to re-issuing the certificates for all the entities beneath it. Therefore, upstream certificates in a trust chain of non-trivial length may need a relatively long validity period (*e.g.,* at the scale of multiple months). The flip side of a lengthened validity period is an increased chance of needing to revoke a certificate before it expires.

Certificate Revocation: Certificate revocation becomes necessary in cases where both of the above two approaches are deemed infeasible. This infeasibility can be illustrated by using the NDN Testbed as an example. The Testbed trust anchor certifies each participating organization, which may further delegate its sub-namespaces to individual departments and issue corresponding certificates. Because certificates for user applications may be further downstream along this multihop trust chain, short-lived organization-level certificates burden user applications with reissuing their downstream certificates frequently[1]. The above operational feasibility concerns lead to the need for issuing organizational certificates with a relatively long lifetime. This lengthened lifetime must be combined with a certificate revocation tool to handle the cases where a long-lived certificate must be revoked before its expiration. The revocation tool needs to record revocation events and provide means to inform data consumers of the revocation status of the certificates.

3.2 An Example Scenario

We introduce an example scenario below, which will be used throughout the paper to facilitate the explanation and discussion.

As shown in Figure 1, an organization site owner delegates her signing power to a few site administrators to manage each department, and each department owner delegates the signing power to a few department administrators to manage individual users. The routers of each department register department-level prefixes to the site router, while the site router aggregates and registers site prefix with the external network. Figure 1 shows two routers from "`ee`" and "`cs`" department register name prefixes "`/ndn/siteA/depart/cs`" and "`/ndn/siteA/depart/ee`" on the site A router, respectively. The two routers use their monthly renewed certificate to sign prefix announcements [40] and propagate them to the site

[1]In practice, we observe that the current NDN Testbed issues year-long certificates to participating organizations.

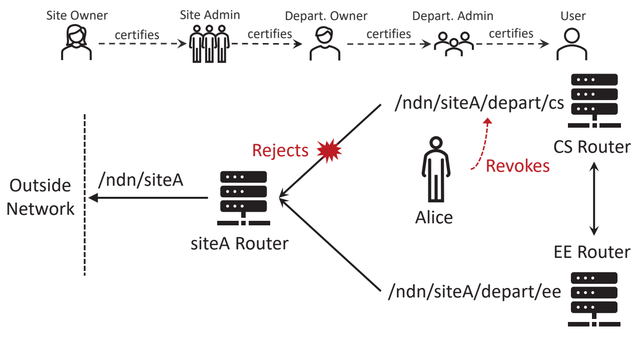


Figure 1: A Prefix Registration Scenario

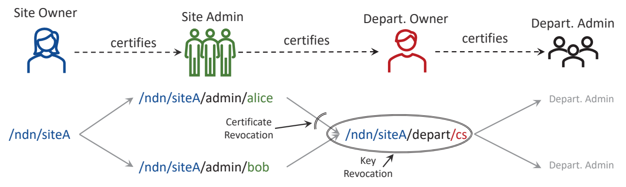


Figure 2: An Example of Multi-Path Authentication

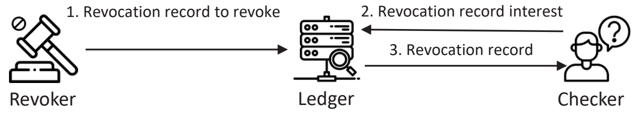


Figure 3: CertRevoke System Model

router. The site router validates the certificate and registers routes for the two departments, then further uses site certificate to sign prefix announcement for "`/ndn/siteA`" and propagates them to the outside network.

Consider a site administrator Alice discovers that the certificate of "`/ndn/siteA/depart/cs`" was mis-issued and revokes it. Then, the site A router should know about this revocation event and reject the next prefix announcement signed by this certificate.

3.3 Revoking Name-Key Endorsements

Revoking a certificate indicates the *removal of a name-key endorsement* from a specific certificate issuer, which *breaks the corresponding edge* in the authentication graph. Note that revoking one's certificate does not necessarily revoke one's authenticity completely, when multi-path authentications exist [44]. For the example shown in Figure 2, the trust schema allows the site owner "`/ndn/siteA`" to certify administrators Alice and Bob for the name prefix "`/ndn/siteA/admin`", and both administrators can further certify department owners for the name prefix "`/ndn/siteA/depart`". Thus the entity "`/ndn/siteA/depart/cs`" has two authentication chains, and entities who trust "`/ndn/siteA`" can validate "`/ndn/siteA/depart/cs`" through either Alice or Bob. In this case, when Alice revokes her issued certificate for "`/ndn/siteA/depart/cs`", it does not disable the downstream authenticity but only removes one of the authentication chains. If the CS router receives certificates from both Alice and Bob, it can use the other certificate from Bob to sign prefix announcements. The site A router will consider the routing announcement as legitimately produced data.

Revoking a certificate means removing an edge that connects two entities in the authentication graph. In this paper, we allow either end node of an edge to remove the edge. In order words, the name-key binding endorser (*i.e.,* the certificate issuer) and the endorsee (*i.e.,* certificate owner) can legitimately invalidate the certificate. Although the certificate owner is not the "producer" of the certificate, it can use its private key to sign a revocation record.

4 THE CERTREVOKE FRAMEWORK

In this section, we show how the CertRevoke design answers the three research questions and fulfills the design goals.

4.1 Problem Statements

In this section, we analyze the certificate revocation problem in NDN and propose a model to investigate the design spaces.

System Model: We consider a certificate revocation framework, as shown in Fig. 3. There are three categories of entities in this protocol, *i.e.,* the *ledger*, *revokers*, and *checkers*. Their roles are discussed as follows:

- *Ledger*: The ledger is a set of entities that as a whole provides immutable data storage. It receives the revocation records from revokers and responds to checkers' Interests to retrieve revocation records. In a practical deployment, the ledger's immutable data storage may be implemented by a distributed-style scheme, such as Merkle Tree [17, 18] or DAG replicas [21, 47]. Thus, the single-point failure can be avoided.
- *Revoker*: Revokers are the entities that revoke certificates. They send a revocation record to the ledger when they revoke a specific certificate.
- *Checker*: Checkers are the entities that want to know whether a given certificate is revoked. They send an Interest to retrieve the revocation record from Ledger when they check the revocation status of a certificate.

Assumptions: Since the checker is the party interested in the actual revocation status, it has no incentive to be malicious. We assume that some malicious revokers may try to revoke the certificates illegitimately, *i.e.,* to revoke the certificates they are not permitted to revoke. Ledger is honest but not trustworthy. Similar to Certificate Transparency [19], the ledger's honesty needs to be checked by external auditors, and we leave the specific design of CertRevoke auditing for future research. Ledger's data immutability can be realized in several ways, e.g., immutable database and distributed ledger technologies (DLT). For example, following the designs of DLedger [47] or Mnemosyne [21], revocation records can be linked together with their names and hash value; when multiple parties are involved, records generated from different parties can interlock each other for better security. We also assume that the ledger, revokers and checkers are in same trust zone. The trust zone controller bootstraps the aforementioned entities with the zone's trust anchor, issues them certificates and defines their trust schema. Thus their operations will follow the security policies defined by the controller. Moreover, the ledger has a well-known prefix inside the trust zone (*e.g.*,"`/ndn/siteA/LEDGER`").

In this preliminary work, we consider providing revocation support for intra-zone data communications only and KeyLocators are certificate names. Generally speaking, practical deployments will utilize inter-zone communication, thus certificate checkers may query the revocation records from ledgers operated by entities external to the local trust zones. Securing *inter-zone* data communications is a research topic which is being explored at the moment [42]. Once the inter-zone trust design matures, we plan to add inter-zone revocation check to CertRevoke as part of our future work.

Design Goals: Based on the above system model and assumptions, the design goals of our proposed framework are defined as follows:

- *Validating Revocation Legitimacy*: The framework can determine the legitimacy of a revoker in a certificate revocation attempt.
- *Maintaining Revocation Recording*: Revocation attempts are recorded by the ledger with the revoker information to prove its legitimacy.
- *Providing Record Accessibility*: Checkers can access legitimate revocation records to determine the revocation status of a certificate.

4.2 Revocation Records

Each revocation record is a semantically named, signed Application Data Unit (ADU) [3]. A revoker generates a revocation record to revoke a certificate. Let R^C denote the revocation record of a certificate C. R^c is defined as Eq. 1.

$$R^c = \{name, \{timestamp, rCode, cKeyH\}, sig\} \tag{1}$$

where *name* is the name of R^C, *timestamp* indicates the time of data signing, *rCode* is the code that indicates the reason why C is revoked, *cKeyH* is the hash value of the public key in C, and *sig* is the corresponding data signature produced by revoker.

R^C binds itself to the corresponding certificate C by naming conventions. R^C name is defined as "`/<prefix>/REVOKE/<keyid>/<issuer>/<version>/<revoker>`", where the name components "`<prefix>`", "`<keyid>`", "`<issuer>`" and "`<version>`" are the same as the corresponding parts of the name of C, while "`<revoker>`" carries the revoker information. The naming convention enables checkers to automate the revocation status checking by converting the C's name into the corresponding R^C name and expressing the Interest to retrieve it (see §4.4) .

Revocation Legitimacy: Since revokers sign revocation records with their certificates and the R^C names reveal the corresponding Cs, the trust zone controller manages the revoker's legitimacy by controlling the signing restrictions in the trust schema. *CertRevoke considers the certificate issuer and certificate owner are legitimate revokers (see §3.3).*

We illustrate how we leverage the trust schema to ensure revocation legitimacy in Figure 4, where Alice as the certificate issuer revokes the certificate of "`/ndn/siteA/depart/cs`" by producing the revocation record "`/ndn/siteA/depart/cs/REVOKE/223/alice/001/alice`". Trust schema enforces the revoker component in R^C name must be equal to the issuer component; (ii) the records under the name prefix "`/ndn/siteA/depart`" must be signed by revokers under the name prefix "`/ndn/siteA/admin`"; (iii) "`/ndn/siteA/admin/alice`" uses the same signing certificate to produce the corresponding revocation record.

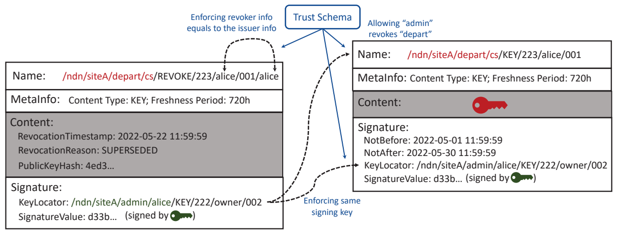

Enforcing revoker info equals to the issuer info
Trust Schema
Allowing "admin" revokes "depart"
Name: /ndn/siteA/depart/cs/REVOKE/223/alice/001/alice
MetaInfo: Content Type: KEY; Freshness Period: 720h
Content:
RevocationTimestamp: 2022-05-22 11:59:59
RevocationReason: SUPERSEDED
PublicKeyHash: 4ed3...
Signature:
KeyLocator: /ndn/siteA/admin/alice/KEY/222/owner/002
SignatureValue: d33b... (signed by)
Name: /ndn/siteA/depart/cs/KEY/223/alice/001
MetaInfo: Content Type: KEY; Freshness Period: 720h
Content:
Signature:
NotBefore: 2022-05-01 11:59:59
NotAfter: 2022-05-30 11:59:59
KeyLocator: /ndn/siteA/admin/alice/KEY/222/owner/002
SignatureValue: d33b... (signed by)
Enforcing same signing key

Figure 4: Site administrator Alice revokes the certificate of CS department

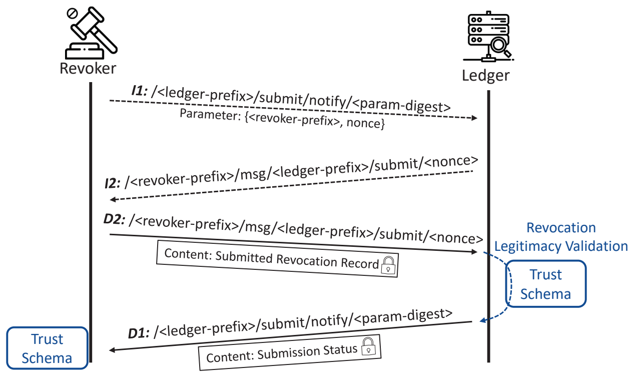

Revoker
Ledger
I1: /<ledger-prefix>/submit/notify/<param-digest>
Parameter: {<revoker-prefix>, nonce}
I2: /<revoker-prefix>/msg/<ledger-prefix>/submit/<nonce>
D2: /<revoker-prefix>/msg/<ledger-prefix>/submit/<nonce>
Content: Submitted Revocation Record
Revocation Legitimacy Validation
Trust Schema
D1: /<ledger-prefix>/submit/notify/<param-digest>
Content: Submission Status
Trust Schema

Figure 5: Interest-Data exchanges between Revoker and Ledger

If the certificate owner revokes the certificate, CertRevoke utilizes trust schema to enforce (i) the revoker component must be "`self`"; (ii) the signing key must be the same as the key name prefix in this certificate.

4.3 Revocation Submission Protocol

In this section, we show the protocol that revokers submit the revocation records to the ledger, and the ledger utilizes the trust schema and key matching policy to ensure the revocation legitimacy.

As shown in Figure 5, a revoker attempts to revoke a certificate C by generating a revocation record R^C and submitting R^C to Ledger. It initiates the submission process by expressing Interest "`/<ledger-prefix>/submit/notify/<param-digest>`" to notify the ledger on the submission event. The notification Interest *I1* carries the revoker's prefix and a nonce to reach this revoker[2], while "`<param-digest>`" is the digest of three parameters. *I1* indicates the submission is under name "`/<revoker-prefix>/msg/<ledger-prefix>/submit/<nonce>`" and retrievable.

[2] *I1* can also carry an optional forwarding hint if this revoker is not directly reachable

After receiving *I1*, the ledger further expresses the Interest *I2:* "`/<revoker-prefix>/msg/<ledger-prefix>/submit/<nonce>`" to retrieve the submission Data *D2*. Then the revoker encapsulates the revocation record into *D2* with the same name, signs it, and replies.

Because signed *D2* contains the R^C, the ledger executes trust schema and key matching policy on *D2* to validate (i) whether *D2* itself is legitimately produced; (ii) whether R^C is legitimately produced; (iii) whether R^C's signer and the C's signer satisfy certain name constraints. Upon successful validation, the ledger inserts the record into its storage backend. Note that CertRevoke decouples the submitter legitimacy from the revoker legitimacy via validating *D2* and its content separately, which enables other parties voluntarily submit collected revocation records to the ledger.

After the submission validation, the ledger replies *I1* with signed *D1*, which has the same name and encapsulates the submission status. Like the ledger, the revoker also executes the trust schema to validate *D1*'s legitimacy and learns the submission status from the validated data content.

Batch Submission: The record submission protocol allows the submitter to batch revocation records in *D2*. Accordingly, the ledger encodes the submission status for each record in *D1*. When the batch size is large, the submitter needs to segment *D1* into "`/<D1-name>/<segment-number>`", and use the *FinalBlockId* field of the Data packet to specify the highest segment number.

4.4 Certificate Revocation Checking

CertRevoke designs each R^C as a piece of semantically named and secured data. Thus checking a certificate's revocation status is a data accessibility problem: *How can the checker access the revocation record in the ledger?* CertRevoke addresses this problem by using *Application Layer NACK* and dynamically controlling *Freshness Period*.

Since we consider the certificate issuer and owner are legitimate revokers, checkers can start from C's name to automatically construct the R^C's name, if it exists, by following the defined naming convention below. In order to access the records, it expresses two Interests "`/<prefix>/REVOKE/<keyid>/<issuer>/<version>/<issuer>`" and "`/<prefix>/REVOKE/<keyid>/<issuer>/<version>/self`" together with the forwarding hint to the ledger. If the ledger finds R^C corresponding to the data name carried in the Interest in its backend storage, it replies to the Interest with the R^C. Receiving either record indicates the checked certificate was revoked. If the corresponding R^C does not exist, the ledger replies with a Data packet that carries an application layer NACK. This Data packet has the name "`/<record-prefix>/nack/<timestamp>`" with empty content and is signed by the ledger. After checkers validate the received NACK with trust schema, it learns that the certificate has not been revoked at this time.

In-network Caching: Because an R^C NACK is a named Data packet, the ledger can dynamically adjust its *Freshness Period* to balance the NACK timeliness and traffic load. A long freshness period indicates the chance that future checkers' Interest packets may hit the R^C NACK from the in-network cache, and reduces the workload at the ledger. On the flip side, it increases the chance that the certificate in question may be revoked, making the R^C NACK in the cache no longer valid. In short, the Freshness Period of each R^C NACK must be carefully selected to minimize the chance of obsolete data in the cache while keeping ledger and network traffic at a reasonable level.

4.5 Putting Together a Case Study

The previous sections explained the individual procedures of CertRevoke. This section shows how everything works together to build a certificate revocation framework that answers the three design questions (see Section 1).

When starting the site network, Site A specifies the trust schema only administrators can legitimately revoke a department's certificate. The revoker component in R^C must be the same as the issuer component in C's name. Site A's owner also runs a ledger instance "`/ndn/siteA/LEDGER`" on Site A's router. During security bootstrapping, Site A's owner installs the trust schema into all site routers. Alice, as the site administrator, revokes its previously issued certificate "`/ndn/siteA/depart/cs/KEY/223/alice/001`" by producing a revocation record (see Figure 4), then submits R^C to "`/ndn/siteA/LEDGER`" following the protocol we mentioned in Section 4.3. An hour later, the CS router signs the prefix announcement and sends it to the site router. The routing application on Site A's router checks the revocation status of the signing certificate using the mechanism in §4.4 and learns the signing certificate was revoked for the reason "`SUPERSEDED`" thereby rejecting the announcement.

Meanwhile, the routing application on the site router also receives the prefix announcement from the EE department router. It checks the revocation status and receives an R^C NACK from the ledger. Upon successful R^C NACK validation, the routing application proceeds to validate the prefix announcement and renews the route of "`/ndn/siteA/depart/ee`".

5 IMPLEMENTATION AND EVALUATION

5.1 Implementation

We have implemented CertRevoke in C++ and provide the library for developers to integrate the CertRevoke protocols into their specific applications[3]. Our implementation supports the ledger for serving revocation records using memory and persistent storage (*e.g.,* SQL database). It can also seamlessly work with other ledger implementations (*i.e.,* DLedger [47]) to provide distributed and immutable data storage.

5.2 Evaluation Scenario

Figure 6 depicts the evaluation scenario. We assume an environment with three forwarders. The hosts of each forwarder are connected through IP overlay links. Checkers and the revoker are virtually connected to the forwarder running on the same host through a Unix socket. The ledger can be connected to Forwarder 1 via IP overlay or virtual links. We assume there is no packet loss on each link, but the latency of each IP overlay link between the forwarders varies from 0.5 ms to 6 ms. Forwarders know the route to each entity. Thereby no route configuration is needed. Additionally, all caches on the forwarder are empty when the evaluation started.

We emulated the hosts by running three containers on a Ubuntu 20.04 server. This server features an AMD EPYC 7702P processor

[3]Code has been published at https://github.com/UCLA-IRL/ndnrevoke

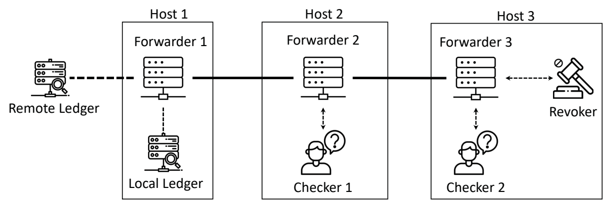


Figure 6: Evaluation scenario where checker and ledger, revoker and ledger are multiple hops away.

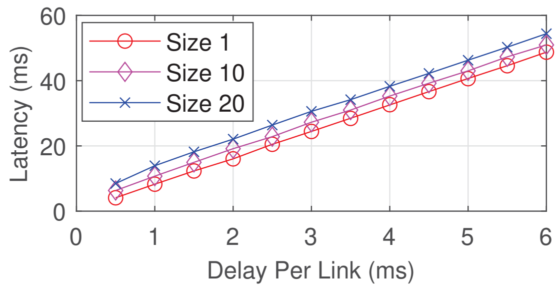


Figure 7: The latency of Revoker submitting revocation records to the Local Ledger

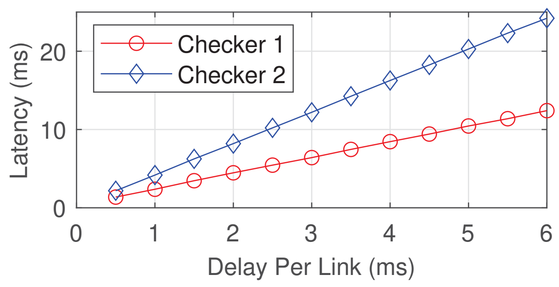


Figure 8: The latency of checking revocation records from the Local Ledger

with 64 physical cores (128 threads) and 256 GB of RAM. We first evaluate the record submission and check performance with a local ledger connected to Forwarder 1 (see §5.3 and §5.4). We also deployed the ledger at a remote host that is connected to Host 1 with 100 ms propagation delay and investigated the effect of cache against data freshness period and certificate popularity in §5.5.

5.3 Revocation Submission

In this section, we evaluate the record submission performance with different link delays and different batch sizes, as illustrated in Fig. 7. We let the revoker revoke 100 certificates and submit the revocation records to the ledger. The results show that the time cost grows linearly with the increasing link delay but is stable at 2RTT. With the increasing batch size, the latency also grows slightly. Specifically, it just takes 54.4 ms when the RTT is set as 24 ms, and the batch size is 20. The results show that our submission protocol is usable in a practical network.

5.4 Revocation Record Checking

In this section, we assume Checker 1 and 2 learn a certificate out-of-band and check its revocation status by expressing Interests for corresponding revocation records. We assumed that both Checker 1 and Checker 2 were interested in 100 potential certificate revocation records, in which two percent of them existed. We disabled the cache so that every Interest packet reaches the ledger. As we can see from Fig. 8, the time costs of both Checker 1 and Checker 2 increase linearly along with the per-link delay. With the same per-link delay, the latency of Checker 1 is smaller because it is closer to the ledger. For instance, when the per-link delay is 6 ms, the latency of Checker 2 is 24.21 ms, while the latency of Checker 1 is 12.41 ms.

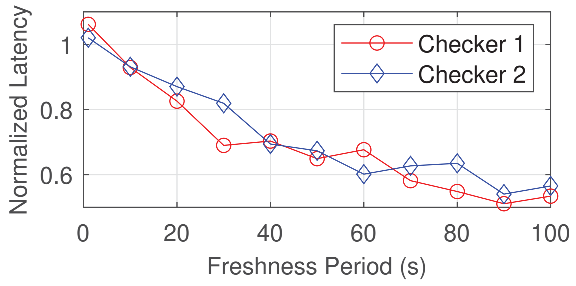
5.5 Benefit from In-network Caching

CertRevoke relies on in-network caching for efficiently distributing revocation records and record NACK packets. If the local network does not have enough resources to run a ledger, the trust zone administrator may bootstrap an entity remotely as ledger deployment. The performance improvement from cache depends on the network topology, data freshness period, and average RTT of record checking. In this experiment, we particularly focused on the record NACK distribution due to two reasons. First, revocation records are supposed to be long-lived data. Therefore the cache benefit largely depends on the network topology and cache capacity. Second, revoked certificates only account for a small portion of all issued certificates. In most cases, when an application checks a revocation record, it will receive a record NACK.

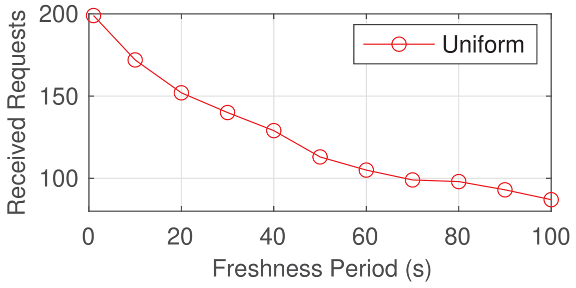
In this experiment, we deployed the ledger on the NDN Testbed [39] and investigated the record-checking latency. We assumed Checker 1 and 2 were interested in the same 100 potential revocation records in §5.4, and randomly requested one of them every second. Each examination event was scheduled with a 1 to 250 ms random backoff. In order to have a better comparison, we also normalized the latency with the average RTT latency from the checker to the ledger.

As we can see from Figure 9a, the normalized latency follows a similar downward trend as the data freshness period increases. That is because the caches in intermediate forwarders can respond some requests. The longer the freshness period, the greater proportion of requests that hit the cache. When the caching benefit converged, the record checking latency was improved by 45%. We also investigate the number of requests that Remote Ledger exactly received in Figure. 9b. For the same reason, the number of received requests also decreases along with the data freshness period. For example, among the first 200 requests that the two Checkers sent, the ledger only received 87 requests when the freshness period was 100 s. In other words, in-network caching decreases the ledger's load by 56.5%.

However, if the average RTT takes 200 ms (approximately from Southern Europe to North America on NDN Testbed), 45% improvement still requires the checker application to wait >100 ms while checking a record. Fortunately, in real applications, not every certificate has the same possibility of being checked. We know that content popularity on today's Internet closely follows a Zipf distribution [5, 8, 37]. We can assume NDN certificates, as data content signers, follow the same popularity distribution. Higher ranked

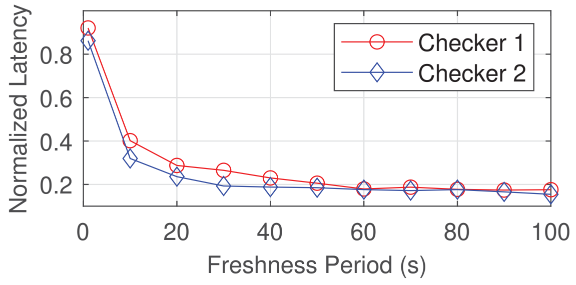


(a) Normalized latency of revocation record checking

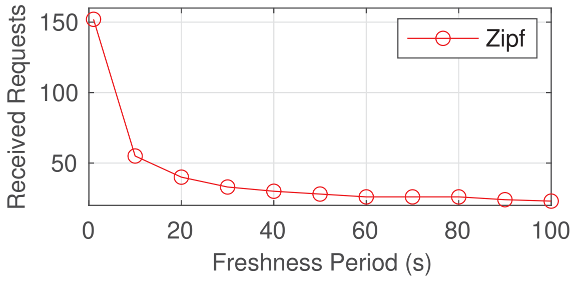


(b) Requests received by Local Ledger

Figure 9: Evaluation when checked records follow Uniform distribution



(a) Normalized latency of revocation record checking



(b) Requests received by Remote Ledger

Figure 10: Evaluation when checked records follow Zipf distribution

certificates are checked more often and thus benefit more from in-network caching.

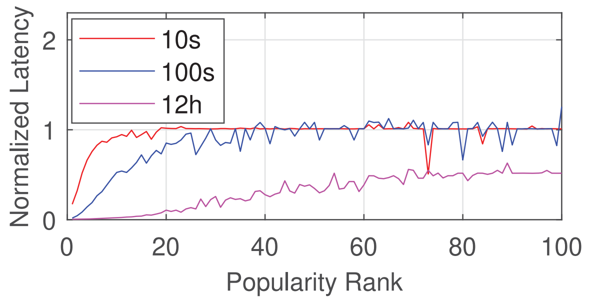


Figure 11: Normalized latency with popularity rank

Therefore in the third setting, to illustrate the real performance, we use the same ledger deployment on the NDN Testbed but assume the certificate popularity follows Zipf distribution with $s = 2$. As illustrated in Fig. 10a and Fig. 10b, both average normalized latency and the received requests decrease exponentially. Moreover, they are also significantly lower than those when certificates share the same popularity. For example, the latency of Checker 2 in Zipf distribution is only 15.5% of original RTT when the freshness period is 100 s, which is significantly smaller than the converged performance 45% in Figure 9a. The number of received requests among the first 200 requests was only 23 when the freshness period was 100 s.

In order to have a closer look at how certificate popularity affects the record checking latency, we also measured the average record checking performance for each certificate. Similar to previous experiments, we normalized the latency with the average RTT from the checker to the ledger. When data expires more often (*e.g.*, 10 s), checking a popular certificate benefits less from in-network caching. As we can see from Fig. 11, the normalized latency quickly converges to 1 as the popularity rank goes down. Only the top 20% certificates take a shorter time to examine. This number becomes the top 40% when the data freshness period is set to 100 s, and the top 10% takes 0.5 RTT for Checkers to examine their revocation statuses. According to the above evaluations, we show that even when the ledger is deployed remotely, the record checking latency is acceptable with the appropriate configuration of caches.

Cache Utilization in Practical Deployments: There are two major factors that affect the cache utilization in practical CertRevoke deployments: network topology and revocation record NACK freshness period.

We are aware that the network topology used in our experiments enables every record Interest hitting the cache. In order to improve the cache utilization in a practical deployment, one may bootstrap distributed ledger [21, 47] instances in different locations in the network.

A practical revocation record NACK's freshness period can be multiple hours or even a few days. Our evaluation shows the performance of record checking promptly converges as the NACK freshness period increases, even the freshness period in this experiment is significantly smaller than the practical network uses (*e.g.*, one day or more).

6 RELATED WORKS

Today's Internet has a history of addressing the need for certificate revocations. In fact, about 2% of all TLS certificates in today's Internet are revoked, according to a study in 2021 [15]. The Certificate Revocation List (CRL) [34] and the Online Certificate Status Checking (OCSP) [35] have become the two pillars of today's revocation system.

CRL: CRLs are files containing lists of unexpired revoked certificates. They are signed by the corresponding CA or a party it delegates to. A certificate can only be revoked by the CA, and the CA revokes certificates by appending the identification of the certificate and the revocation reason to the CRL. CRLs are usually hosted at stable URLs, and the CA informs the consumer about the URL in the certificate. Data consumers check the certificate status by fetching the CRL and confirming that the certificate in question is not on the list.

However, CRL suffers from performance issues. The consumer needs to make another request for the CRL, which creates another RTT. In particular, as the service of the CA expands, the size of the CRL may grow. Although a number of solutions [9, 14, 16, 25, 27, 38] are proposed to enable fast CRL checkings, the fundamental issue of CRL is that the revocation list is merely a file or data structure rather than a piece of named data at the network layer. As a result, one has to find the server address and retrieve the file at the application layer. A lesson we learned from CRL is that revocation records should be named data and only contain the revocation information of one certificate, so the revocation status of each certificate can be fetched separately.

OCSP: OCSP is another approach for providing revocation updates. Like CRL, CA configures OCSP responders at stable URLs, and informs the responder URL in the issued certificate. Data consumers follow the OCSP protocol to query the certificate status from the URL, and the OCSP responders provide signed revocation status for each query. OCSP avoids the CRL problem on the size of the data responses, but it still contains the same problem as the CRL on the additional RTT. It also requires the responder to be always online for OCSP queries, which creates a large burden for OCSP responders for responding all queries. OCSP Stapling [6] and the later OCSP Must-Staple [10] solve the problems by allowing the server to include the cached OCSP responses from the issuer during a TLS handshake with the data consumer.

Compared to CRL, OCSP makes revocation records as named data at the application layer. From OCSP, we learned that record accessibility is an important factor that affects adoption. Therefore, CertRevoke takes one step further by directly making revocation record as named data at the network layer, thereby enabling in-network caching and providing better data accessibility.

Ledger-based Revocation System: There are also works to design the revocation system by leveraging ledger technologies such as Certificate Transparency [17]. Enhanced Certificate Transparency [33] proposes the use of a Merkle tree to store revoked certificates in the Certificate Transparency log. Revocation Transparency [18] by Google uses append-only log to store and verify certificate revocations. There are also proposals to use blockchain to build revocation logs [2, 28]. CertRevoke shares a similar approach of utilizing immutable logs, but also enjoys its unique advantage of efficient distribution of all revocation records via NDN's built-in multicast delivery and in-network caching.

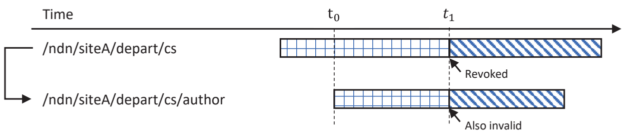


Figure 12: Example of Revocation Impact

DNSSEC and DANE: In today's Internet infrastructure, DNSSEC [30–32] is another solution to endorse name-key bindings between domain names and cryptographic keys. Leveraging DNSSEC, DANE [11] can further specify a TLS certificate for a domain name in TLSA Resouce Records (RR) published under the certificate owner's domain name. In a recent presentation [13], Huston pointed out that, as DNS uses relatively short TTLs to control the TLSA RR liveness, after cached results time out quickly, future requests for TLSA records will be forwarded to the DNS server to handle, eliminating the needs of certificate revocation. Huston argued taht using DANE to manage TLS certificates can achieve the same, or better and simpler, effect in TLS key management, as compared to today's practices of either deploying revocation solutions, or otherwise letting CAs issue short-lived X.509 certificates.

7 DISCUSSION

After presenting the lessons learned from related works in existing X.509 certificate revocation solutions, this section discusses potential optimizations and remaining questions.

7.1 Revocation Impact

When an upstream, long-lived certificate is revoked, it invalidates all downstream Data in the authentication chain, including the potentially innocent packets. For example, as shown in Figure 12, Alice discovers the certificate of "`/ndn/siteA/depart/cs`" is misused at *T1* and decides to revoke it. The downstream certificate "`/ndn/siteA/depart/cs/author`" issued at *T0*, which is earlier than *T1*, might be innocent. In order to reduce the revocation impact, one can add an optional field "`notBefore`" in the revocation record to indicate the start timestamp of invalidation. When the checker receives a revocation record containing "`notBefore`", it compares the certificate validity period with "`notBefore`" to determine whether it can be exempted. Note that "`notBefore`" is based on the revoker's best estimate. Thereby the usage of "`notBefore`" objectively increases the security risk and is not recommended.

7.2 Record Checking Latency

Checking revocation status for each certificate along the certificate chain requires data consumers to take at least one RTT per certificate to retrieve revocation records. Although in-network caching mitigates this issue, one can take the following approaches to further reduce the record checking latency: (i) Sending the record fetching Interests together with the Interest to fetch the upstream certificate. (ii) Similar to OCSP Stapling, providing fresh Record NACK packets in the certificate bundle [22]. (iii) Only performing revocation record checking for long-lived certificates (e.g., validity period longer than a threshold).

Also note that, unlike OCSP, a CertRevoke revocation record is *not* a certificate status, and the record name can be directly converted from the corresponding certificate name. Therefore a CertRevoke ledger does not need to manage the revocation status for each certificate, but rather a key-value storage. This advantage enables fast record look up and at the ledger side.

7.3 Certificate Revocation versus Key Revocation

In Section 3.3, we mentioned that certificate revocation is to remove the relation between two entities. It is worth mentioning that certificate revocation does not revoke the corresponding key pair.

Key revocation is a separate concept that directly invalidates the key, *i.e.,* removing the corresponding node in the authentication graph. We show the difference between the certificate revocation and key revocation in Figure 2. If each entity node in the authentication graph has only one key, revoking the key of entity "`/ndn/siteA /depart/cs`" breaks all authentication chains from "`/ndn/siteA`" to this entity by distrusting the revoked key itself. Of course an entity may possess multiple keys to mitigate the impact of losing one key. Nevertheless, we believe that key revocation is a separable research problem in the trust management, and leave it for our future work.

7.4 Privacy Implications

The privacy of a certificate checker is an important concern in certificate revocation designs. For example, The OCSP design lets each client check with an OCSP server about whether a given certificate has been revoked, and this OCSP request allows the OCSP server to learn that a specific client is visiting the domain being checked [16]. In contrast, because NDN Interest packets do not carry identity information of consumers, the Interests for checking revocation records do not disclose any checker information. If a checking request does not hit a cache along the way before reaching a ledger, the ledger can only learn the domain being checked, but not the consumer who sent the record checking Interest.

7.5 Certificate Revocation versus Short Certificate Lifetime

Under the assumption that there is a general need for NDN certificate revocation, this paper presented the design of CertRevoke and some preliminary evaluation of its effectiveness and efficiency. The need for certificate revocations becomes evident when the implementation of an NDN-based application directly puts the certificate name into the `KeyLocator` field in its Data packets. Assuming that the authentication chains of Data packets are defined by the application's trust scheme, using `KeyLocator` to carry the certificate name is a simple, straightforward way to enforcing the authentication chain starting from the trust anchor down to the Data packet's producer. At the same time, this implementation approach also ties together the entire authentication chain, resulting in any change to the trust anchor or other keys at higher level triggering a rippling changes to all the certificates downstream from it. Reducing the frequency of such undesired ripple effects requires using longer certificates lifetime (see Section 3), which in turn increases the chance that a certificate may need to be revoked before it expires.

As we described in Section 6, Huston suggested to do away with certificate revocation entirely by assigning *all* certificates with relatively short lifetime [13]. To avoid the ripple effect due to higher level key changes, one can *decouple* each step in the authentication chain by putting the signing key name, instead of the certificate name, in Data's `KeyLocator` field. To retrieve the corresponding certificate, the consumer can construct a certificate name of the signing key from the information provided by trust schema, but without knowing the certificate's version number. Thus the consumer needs a means to retrieve the certificate of the latest version quickly.

We note that the strict authentication chain should be *explicitly* defined by the trust schema. and that carrying certificate name in `KeyLocator` and the proposed solution in [13] represent different design tradeoffs in authentication chain enforcement. The former is a simple way to strictly enforce the chain, by paying the cost of deploying certificate revocation solutions (such as the one designed in this paper). The latter does away from certificate revocation but requires an effective means to quickly retrieve the latest certificate without knowing its version number.

7.6 Remaining Work

We have identified two pieces of remaining work to be done in order to operate CertRevoke effectively and securely.

Realization of a Distributed Immutable Ledger. CertRevoke requires a ledger design to provide immutable data storage. The design of this ledger can be tailored specifically to meet the needs for certificate management requirements. The questions of how to design a simple certificate management ledger and bootstrap it into a given trust zone remain to be answered. We consider this as one of our future works.

Effective Cache Poisoning Mitigation. Malicious attackers can poison in-network caches by requesting a prepared fake revocation record or revocation NACK from a colluding server. Such fake data packets do not conform to the trust schema, therefore checkers can easily detect and discard them. However the cache poisoning essentially becomes a Denial-of-Service (DOS) attack, we need effective means to either remove fake data packets from in-network caches, and/or route checkers requests around poisoned caches. Since cache poisoning is a well-recognized problem in NDN, many efforts have been devoted to identifying effective solutions [23, 29, 36]. We hope to identify, and implement, an effective solution from the existing literature.

8 CONCLUSION AND FUTURE WORK

NDN architecture secures communication by semantically named and signed data packets, which requires an easy-to-use mechanism to issue and revoke certificates. Certificate revocation is an essential part of NDN certificate management, which has not attracted adequate attention. This work articulated the necessity of certificate revocation, and presented a usable certificate revocation framework CertRevoke. CertRevoke leverages NDN's naming convention, trust schema, and in-network caching to systematically validate revocations and enable efficient distribution of certificate revocation records. Our evaluation shows CertRevoke significantly benefits from in-network caching and provides acceptable latency for revocation checking. We also identified the remaining work to be done, and plan to deploy a fully operational certificate revocation system in the NDN Testbed in a near future.

ACKNOWLEDGEMENT

We want to thank all the anonymous reviewers and the shepherd Nikos Fotiou for their valuable comments. This work was supported in part by National Science Foundation under awards 2019085 and 2126148, and Research Grants Council of Hong Kong under CityU 11202419.

REFERENCES

[1] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, et al. 2019. Let's Encrypt: an automated certificate authority to encrypt the entire web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* 2473–2487.

[2] Yves Christian Elloh Adja, Badis Hammi, Ahmed Serhrouchni, and Sherali Zeadally. 2021. A blockchain-based certificate revocation management and status verification system. *Computers & Security* 104 (2021), 102209.

[3] Alex Afanasyev, Jeff Burke, Tamer Refaei, Lan Wang, Beichuan Zhang, and Lixia Zhang. 2018. A brief introduction to Named Data Networking. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM).* IEEE, 1–6.

[4] Richard Barnes, Jacob Hoffman-Andrews, Daniel McCarney, and James Kasten. 2019. *Automatic certificate management environment (acme).* Technical Report.

[5] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. 1999. Web caching and Zipf-like distributions: Evidence and implications. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, Vol. 1. IEEE, 126–134.

[6] D. Eastlake. 2011. *Transport Layer Security (TLS) Extensions: Extension Definitions.* Technical Report. IETF Network Working Group, Fremont, CA, USA, 2021.

[7] EFF. 2022. Certbot. Online at https://certbot.eff.org/.

[8] Seyed Kaveh Fayazbakhsh, Yin Lin, Amin Tootoonchian, Ali Ghodsi, Teemu Koponen, Bruce Maggs, KC Ng, Vyas Sekar, and Scott Shenker. 2013. Less pain, most of the gain: Incrementally deployable icn. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 147–158.

[9] Mark Goodwin. 2015. Revoking Intermediate Certificates: Introducing OneCRL. Online at https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/.

[10] P. Hallam-Baker. 2015. *X.509v3 Transport Layer Security (TLS) Feature Extension.* Technical Report. IETF Network Working Group, Fremont, CA, USA, 2021.

[11] Paul E. Hoffman and Jakob Schlyter. 2012. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698. https://doi.org/10.17487/RFC6698

[12] Yung-Kao Hsu and Stephen Seymour. 1997. Intranet security framework based on short-lived certificates. In *Proceedings of IEEE 6th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises.* IEEE, 228–234.

[13] Geoff Huston. 2022. Revocation. Presentation at RIPE 84. https://www.potaroo.net/presentations/2022-05-20-revocation-ripe84.pdf

[14] Paul C Kocher. 1998. On certificate revocation and validation. In *International conference on financial cryptography.* Springer, 172–177.

[15] Nikita Korzhitskii and Niklas Carlsson. 2021. Revocation Statuses on the Internet. In *International Conference on Passive and Active Network Measurement.* Springer, 175–191.

[16] James Larisch, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. 2017. CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers. In *2017 IEEE Symposium on Security and Privacy (SP).* 539–556. https://doi.org/10.1109/SP.2017.17

[17] Ben Laurie. 2014. Certificate transparency: Public, verifiable, append-only logs. *Queue* 12, 8 (2014), 10–19.

[18] Ben Laurie and Emilia Kasper. 2012. Revocation transparency. *Google Research, September* 33 (2012).

[19] B. Laurie, E. Messeri, and R. Stradling. 2021. *RFC 9162-Certificate Transparency Version 2.0.* Technical Report. IETF Network Working Group, Fremont, CA, USA, 2021.

[20] Yanbiao Li, Zhiyi Zhang, Xin Wang, Edward Lu, Dafang Zhang, and Lixia Zhang. 2019. A secure sign-on protocol for smart homes over named data networking. *IEEE Communications Magazine* 57, 7 (2019), 62–68.

[21] Siqi Liu, Philipp Moll, and Lixia Zhang. 2021. Mnemosyne: an immutable distributed logging framework over named data networking. In *Proceedings of the 8th ACM Conference on Information-Centric Networking.* 130–132.

[22] Manika Mittal, Alex Afanasyev, and Lixia Zhang. 2017. NDN Certificate Bundle (Version 0.1). *University of California, Los Angeles, Tech. Rep. NDN-0054* (2017).

[23] Tan Nguyen, Xavier Marchal, Guillaume Doyen, Thibault Cholez, and Rémi Cogranne. 2017. Content poisoning in named data networking: Comprehensive characterization of real deployment. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM).* IEEE, 72–80.

[24] Kathleen Nichols. 2021. Trust Schemas and ICN: Key to Secure Home IoT *(ICN '21).* Association for Computing Machinery, New York, NY, USA, 12 pages.

[25] Kobbi Nissim and Moni Naor. 1998. Certificate Revocation and Certificate Update.. In *USENIX Security Symposium.* Citeseer.

[26] Davide Pesavento, Junxiao Shi, Kerry McKay, and Lotfi Benmohamed. 2022. PION: Password-based IoT Onboarding Over Named Data Networking. In *2022 IEEE International Conference on Communications.* IEEE.

[27] The Chromium Projects. 2022. CRLSets. Online at https://www.chromium.org/Home/chromium-security/crlsets/.

[28] Bo Qin, Jikun Huang, Qin Wang, Xizhao Luo, Bin Liang, and Wenchang Shi. 2020. Cecoin: A decentralized PKI mitigating MitM attacks. *Future Generation Computer Systems* 107 (2020), 805–815.

[29] Zeinab Rezaeifar, Jian Wang, and Heekuck Oh. 2018. A trust-based method for mitigating cache poisoning in name data networking. *Journal of Network and Computer Applications* 104 (2018), 117–132.

[30] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends. 2005. DNS Security Introduction and Requirements. RFC 4033. https://doi.org/10.17487/RFC4033

[31] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends. 2005. Protocol Modifications for the DNS Security Extensions. RFC 4035. https://doi.org/10.17487/RFC4035

[32] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends. 2005. Resource Records for the DNS Security Extensions. RFC 4034. https://doi.org/10.17487/RFC4034

[33] Mark D. Ryan. 2014. Enhanced Certificate Transparency and End-to-End Encrypted Mail. In *NDSS Symposium 2014.*

[34] S Santesson, S Farrell, S Boeyen, R Housley, W Polk, and D Cooper. 2008. *RFC 5280-Internet X. 509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* Technical Report. IETF Network Working Group, Fremont, CA, USA, 2008.

[35] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. 2013. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP.* Technical Report. IETF Network Working Group, Fremont, CA, USA, 2021.

[36] Divya Saxena, Vaskar Raychoudhury, Neeraj Suri, Christian Becker, and Jiannong Cao. 2016. Named data networking: a survey. *Computer Science Review* 19 (2016), 15–55.

[37] Thomas C Schmidt, Sebastian Wölke, Nora Berg, and Matthias Wählisch. 2016. Let's collect names: How PANINI limits FIB tables in name based routing. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops.* IEEE, 458–466.

[38] Trevor Smith, Luke Dickinson, and Kent Seamons. 2020. Let's revoke: Scalable global certificate revocation. In *Network and Distributed Systems Security (NDSS) Symposium 2020.*

[39] The NDN Team. 2022. NDN Testbed. Online at https://named-data.net/ndn-testbed/.

[40] The NDN Team. 2022. Prefix Announcement Protocol. Online at https://redmine.named-data.net/projects/nfd/wiki/PrefixAnnouncement.

[41] Emin Topalovic, Brennan Saeta, Lin-Shung Huang, Collin Jackson, and Dan Boneh. 2012. Towards short-lived certificates. Web.

[42] Tianyuan Yu, Xinyu Ma, Hongcheng Xie, Yekta Kocaoğullar, and Lixia Zhang. [n.d.]. Intertrust: Establishing Inter-Zone Trust Relationships. In *9th ACM Conference on Information-Centric Networking (ICN 2022).* ACM.

[43] Tianyuan Yu, Philipp Moll, Zhiyi Zhang, Alexander Afanasyev, and Lixia Zhang. [n.d.]. Enabling Plug-n-Play in Named Data Networking. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM).* IEEE, 562–569.

[44] Yingdi Yu, Alexander Afanasyev, David Clark, KC Claffy, Van Jacobson, and Lixia Zhang. 2015. Schematizing trust in named data networking. In *proceedings of the 2nd ACM Conference on Information-Centric Networking.* 177–186.

[45] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, KC Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named data networking. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 66–73.

[46] Zhiyi Zhang, Alexander Afanasyev, and Lixia Zhang. 2017. Ndncert: universal usable trust management for ndn. In *Proceedings of the 4th ACM Conference on Information-Centric Networking.* 178–179.

[47] Zhiyi Zhang, Vishrant Vasavada, Xinyu Ma, and Lixia Zhang. 2019. Dledger: An iot-friendly private distributed ledger system based on dag. *arXiv preprint*

arXiv:1902.09031 (2019).

[48] Zhiyi Zhang, Su Yong Wong, Junxiao Shi, Davide Pesavento, Alexander Afanasyev, and Lixia Zhang. 2020. On Certificate Management in Named Data Networking. *arXiv preprint arXiv:2009.09339* (2020).

[49] Zhiyi Zhang, Yingdi Yu, Alexander Afanasyev, Jeff Burke, and Lixia Zhang. 2017. NAC: Name-based access control in named data networking. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*. 186–187.