# A Robust Trust Model for Named-Data Networks

**Vahab Pournaghshband and Karthikeyan Natarajan**
Computer Science Department
University of California, Los Angeles

**Abstract**—*Any future Internet architecture must offer improved protection and resilience over today's network, which is subject to pervasive and persistent attacks. A recently emerging architecture, Named-Data Network (NDN), treats content as the primitive entity. This leads to decoupling location from identity, security and access, and retrieving content by name. NDN security is based on the establishment of a trustworthy routing mesh, relying on singed routing messages and an appropriate trust model. Signature verification of NDN content merely indicates that it was signed with a particular key. Making this information useful to applications requires managing trust, allowing content consumers to determine acceptable signature keys in a given context.*

*In this paper, we propose a robust trust model for NDN to securely learn public keys of content publishers so that applications can determine what keys are trustworthy. In doing so, the user asks for publisher key recommendations from all entities in its community of trust, which consist of people the user personally knows, as in real world interactions. A local policy is then used to decide consistency of responses, and hence trustworthiness of the publisher's key. Also, we present a suitable key revocation approach for this model. We then provide a discussion on robustness of this model against various attacks.*

**Keywords:** decentralized trust model, named-data network, web of trust

## 1. Introduction

In the current Internet, packets are routed based on IP addresses irrespective of the content the user is looking for. NDN advocates for a routing infrastructure where forwarding decisions are made on the content requested. NDN also caches data at the routers along the path through which the response (data) travels. An interest packet travels until it encounters a valid source of data, which could either be a router's cache, or the publisher of the data.

From a security standpoint, there are two concerns that are to be addressed in such architecture:

1. Did the user receive the message as sent by the publisher?
2. Is the publisher really the person whom he claims to be?

Here we are trying to establish the integrity of the data and the authenticity of the data source. A user depends on the trust model provided by the Internet infrastructure to determine these two. We will present a trust model that is decentralized, providing the user with necessary information to make a decision to establish trust on an entity.

### 1.1 NDN Basics

In NDN, when an interest is sent out for a particular name, we receive the data back from the publisher or from a node that had cached the data. The response has the following components, which are of specific interest to the discussion here: Signed info and Signature (Figure 1).

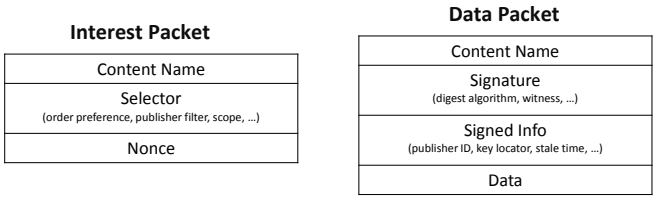


Figure 1: NDN Packet Types

The content publisher signs the hash of the content and the name of the content. Hence, given the publisher's key, the integrity of content received is verified by comparing the given signature.

A more challenging problem, however, is to verify the authenticity of the publisher's public key. Every trust model would provide means of retrieving a published public key. Here, the "Signed info" component of the data consists of the keylocator and the publisher's public-key name, to facilitate key verification problem.

By querying the publisher using the information provided in the signed info part of the message one can fetch the publisher's public key. However, this is susceptible to man-in-the-middle attacks sitting between the user and the publisher, supplying fake keys to the user. Hence, there is a need for secure retrieval of public keys. The conventional PKI scheme trusts a third party to provide the user with the correct public key of the publisher. In most of the cases, the end user has no idea of who this trusted entity is. We address this problem by distributing trust among a set of people whom the user would trust.

### 1.2 Review of Current Approaches

This section provides a brief overview of the current approaches of establishing trust in the Internet. We will briefly

cover leap-of-faith, PGP, Certificate Authorities, DNSSEC, and SPKI/SDSI. We will also discuss the shortcomings of each of these approaches.

#### 1.2.1 Leap-of-faith

This is the approach widely adopted by the secure shell application. When a client connect to an SSH server, the server sends its certificate to the client. The client, in order to communicate securely afterward, accepts this certificate to be the trusted key of the server. This certificate is cached for future access to the server. The application trust the certificate obtained, assuming that no Man-in-the-Middle attack was performed during the initial key request. Such a trust model, while simple, is clearly vulnerable to attacks especially in the case of key changes and hence is not being used for sensitive applications like online banking or e-commerce.

#### 1.2.2 PGP

Pretty Good Privacy (PGP) is a computer program that provides cryptographic privacy and authentication. PGP is often used for signing, encrypting and decrypting e-mails to increase the security of e-mail communications. It uses the concept of a web of trust. To trust a certificate (i.e., the public key of an entity), the user requires someone he trusts to endorse the untrusted entity. The trusted person who endorses the certificate is likely to have used out-of-band means to get the certificate in question.

#### 1.2.3 Certifying Authority

Certifying Authority (CA) is a trusted third party that issues certificates for publishers' keys. The certificate contains the public key and the identity of the publisher. The public key of the CA is publicly known and is normally hard-coded within all the nodes connected to the Internet. The CA computes the hash of the key as well as the publisher identity and encrypts it with its private key, which serves as the certificate of the publisher. When a client wants to communicate to a server, it can verify the authenticity of the server by comparing the hash of (key,name) pair obtained from the publisher to the hash signed by the CA. This is essentially trusting whatever the CA has signed. A typical certificate issued by a CA for some publisher would contain the publisher's public key, publisher's name, the key's expiration date, and the signature of this information signed by the CA's private key. The expiration date is used to identify the certificate's lifespan.

#### 1.2.4 DNSSEC

This model is used to secure the DNS service running on top of the IP Network. It provides authenticated response to the lookup queries. All DNS answers are signed by the authority server that is responsible for a particular domain. Consider `A.B.com` : `A`'s key is signed by `B`. `B`'s key is signed by `com` and `com`'s key is signed by the root. The root's key is hard-coded in all browsers. They follow this chain of keys to establish trust in a DNS response. All domains are signed by their respective parent domains. The key of the root is installed in your system. Key revocation and rollover has been also a problem for this model.

#### 1.2.5 SPKI/SDSI

Simple Public Key Infrastructure (SPKI) is designed to be an alternative to the X.509 standard for digital certificates. SPKI views authority as being associated with principals, which are identified by public keys. It allows binding authorizations to those public keys and delegation of authorization from one key to another.

# 2. Design

## 2.1 Principles and Assumptions

There are numerous characteristics of a robust trust model that any design at the Internet level should have such as scalability, easy enrollment, decentralized authority, and resiliency against attacks such as Denial-of-Service (DoS) and Man-in-the-Middle (MitM) attacks. In designing our trust model for NDN, besides taking these characteristics into consideration, we primarily concentrated on preserving the liberty of choosing who to trust by avoiding complete trust on third parties which are trusted not at the user's will. In addition, flexibility in local policies is also desirable, in the sense that each person defines security individually which would give them locality of control. This is important since not all users care at the same level and not all contents need the same level of attention. And lastly, which is provided by the nature of NDN, the notion of trust should be contextual, i.e. narrowly determined in the context of particular content and the purpose for which it will be used. In our design, we assume that attacks are either localized to a particular network scope or of limited duration, since a larger attack is more easily detected and further remedied. This is true mainly since most network level attacks on integrity and secrecy of information need to remain undetected to be successful.

## 2.2 Overview

We define the notion of community of trust to be a combination of our friends (whom we know) and a selective subset of publicly available network notaries. Notaries (or pseudo-friends), as defined in Wendlandt et al. [4], are servers which are solely designed to monitor and record history of public keys used by a network service. A notary frequently asks for the keys from particular servers and updates its database upon a key change. As its service, each notary responds to queries from clients who ask notaries about content publishers' public keys.

Upon request for a particular data, the user receive the data signed by its claimed publisher. If the user has the publisher key associated to the data cached, then the authenticity of data can be verified immediately using the cached key. If the cached publisher's key is suspected as invalid or not cached, the client issues a public key request for the publisher to all its friends and notaries in its community of trust. Upon receiving responses from them, as well as an offered key from the publisher itself, the user applies its local policy to judge the consistency of, possibly weighted, responses to make its trust decision on whether to accept the key or not. In the following sections we discuss this approach in details.

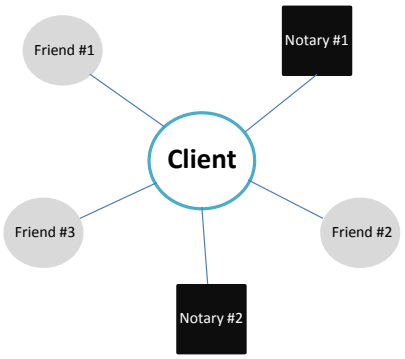


Figure 2: Notion of Community of Trust

## 2.3 Trust Bootstrapping

A provably secure, and yet its practicality being still in question, approach to bootstrap trust is the use of out-of-band mechanisms to learn the public keys of the user's friends in his community of trust. This could be less of the problem since the user is expected to have personal relationship with his friends and hence utilizing a large variety of such out-of-band means. To obtain, the notaries' public key, however, the user can request it from his friends, the same way to get any other publisher's key. Some other mechanisms such as security through publicity [5] can be applied for some well-known notaries.

## 2.4 Notion of Master Key

Every publisher maintains a master key which is used to sign only other keys under the publisher's domain. In fact master key is the only offered publisher key that invoke key recommendations from friends. Once that key is validated and cached, the publisher would play as the Certifying Authority (CA) for all its sub-domains by generating certificates and revocation notifications.

The master key needs to be highly secure and requires a longer life span. That is why it is only used to sign top level keys in the publisher's domain and not content, mainly to protect it against known-plaintext attacks by having very few samples publicly available. Besides, due to higher measures of security considered in choosing master keys, the signature verification process is expected to be relatively expensive, hence not suitable for signing contents.

Note that even though the notion of master key shares similarities with the current CA mechanism, they are fundamentally different. One reason is that there is no third party involvement in the process, since the publisher manages certificates for its own domain. Also, it incurs no cost to generate or revoke certificates, unlike current approaches that involves third party companies that charge for such services.

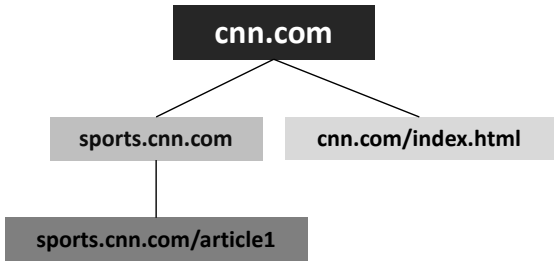


Figure 3: Master Key Signing its Sub-domains

## 2.5 Key Revocation

An important part of every public-key trust model is key revocation. The public-key will be well known to a large but unknown set of users. The revocation system must have the ability to replace the private-public key pair, distribute the new key, and notification to the users of the revocation should happen in a timely manner. Key revocations have two fundamental approaches, implicit and explicit. An implicit revocation is asserted by the ability to retrieve the certificate. Any certificate retrievable is guaranteed to be valid near the time of retrieval. Also common is expiration or Time-To-Live (TTL), which is the maximum time the certificate is valid before requiring a new certificate from the issuer. This is more commonly known as certification expiration. However, from the time the certificate is retrieved to the expiration date is a window of vulnerability if a key has been explicitly revoked. Explicit revocation is when the issuer states that certificates are no longer valid, or have been compromised (also indirectly which certificates have not been revoked). Current explicit revocation approaches come in two forms, online and offline. Some popular offline models include Certificate Revocation Lists (CRL) [13], $\Delta$-CRL [13], and Certificate Revocation Trees [14]. These models explicitly state that particular certificates have been revoked, and they can be obtained through the issuer or a third-party. On the other hand, some online models include Online Certificate Status Protocol (OCSP) [11], and Semi-Trusted Mediator (SEM) [15], which provide an online service used to obtain the revocation status. Similar to offline models, which provides the URI to request the location of the issuers CRL, online models also include the location of

this service within the issuers certificate. The main drawback of offline models is that they require frequent downloads to keep the revocation list current. Online models overcome this limitation by checking the certificate status in real time.

Our revocation approach is inspired by OCSP [11]. Once a node requests a key through our trust model, and it has been accepted by our policy, it will cache the key/certificate until it implicitly expires or is explicitly revoked. When the node sends a key request interest message, the key response message contains an ordered list of Revocation Authorities (RA), providing an online service that states the status of the issuers certificate (similar to the OCSP URI in the certificate [11]). The next time a node request data from the same issuer with a cached key, it would only need to request the status of the certificate from the RAs. RAs would reply back with a signed *Valid*, *Invalid*, or *Unknown* response. A nonce would also be included in this reply to prevent replay attacks when a malicious user was to capture a valid response and replay it during another status request. Note that the RA themselves would also have a trusted certificate which will be obtained using our trust model. Any âĂIJinvalidâĂİ responses to a certificate status results in the node clearing its certificate cache and falling back to the trust model to obtain a trusted key.

However, this model is still vulnerable to attack. If a DoS attack on the RA would prevent a response with the current status of the certificate, this model considers this timeout to be an *invalid* response. Further, if a malicious user does compromise an RA there are a few possible outcomes. Either the key is still valid but the compromised RA would respond with *invalid*, which would result in the node clearing its cache and returning to the start of the trust model. The other possibility and more threatening to the user is the key has been revoked, but the compromised RA response back with a *valid* response. The node maybe be fooled into accepting non-valid content from the issuer of the certificate. However, this alone would not be a meaningful attack. For a more successful attack, a node would have to first trust a key through our trust model, say for example `cnn.com`. If the publisher then revokes its compromised key, the RA's would be notified, and would respond with *invalid*. The malicious user must then compromise both the RA's for `cnn.com` and `cnn.com` itself. Then wait to intercept a request for both `cnn.com` data and replace it with its malicious content signed with the compromised key, then also intercept the certificate status request to any of the RA's and respond with a *valid* message. This makes an attack against our revocation model costly, specially if the RA list contains more than one RA in which all RA's responses must be forged.

## 2.6 Key-trust Policies

Once the user has key response messages from its trust community, it uses a key-trust policy to accept or reject an offered key based on the collected information. Basically the user must make a decision which leads to a security vs. availability trade-off. The user can take the risk of accepting the offered key based on the responses from its friends or reject the key and discard the content received. Every user has a liberty of choosing a local policy most suitable for them. Here, we discuss Perspectives's policy model [4] as an example of a simple and yet effective local policy implementation. The notion of quorum [4] as a key-trust primitive is defined as followed:

**Definition:** For a set of $n$ entities in a community of trust, a service $S$, and a threshold $q$ ($0 \leq q \leq n$) we say that a key $K$ has a quorum at time $t$ iff at least $q$ of the $n$ entities report that $K$ is the key for $S$ at time $t$.

As an example, consider our trust community depicted in Figure 2. Let's assume that the user has received the following responses from the community as illustrated in Table 1.

Table 1: An example of key recommendations from the community of trust

| Notary#1 | Friend#1 | Notary#2 | Friend#2 | Friend#3 |
|---|---|---|---|---|
| $K_A$ | $K_A$ | $K_C$ | $K_B$ | $K_A$ |

If the user's quorum is $0.6n$, then it accepts the offered key from `cnn.com`. Any higher quorum, however, would reject it. Notice that the security vs. availability trade-off is apparent in this case when a user can simply set the quorum to total number of friends, $n$, where this provides the strongest protection against accepting a false key, but it means that a single unavailable or misinformed friend could cause the user to reject a valid key.

Now that we have learned the details of the design, to better understand the model, let us turn to the example depicted in Figure 3. The scenario is that the user downloads an article from `cnn.com`. We assume that the user's cache does not contain `cnn.com`'s master key. Hence, upon receipt of the article, the user issues a series of key interest requests to his trust community and asks for `cnn.com`'s master key. The user asks `cnn.com` directly for its master key as well (offered key). If the friend (or notary) has the key in interest cached, which in that case he would prepare the signed key response message that includes the key. But if the friend does not have the key, he issues his own key interest request messages to his community of trust, accepts or rejects the key, and forwards his decision to the user. Once the key recommendation responses are received, the user's local key-trust policy decides whether to accept the key or not. If he accepts, the user caches the `cnn.com`'s master key, and will be able to validate any certificate generated by `cnn.com` or any of its sub-domains. Using this information,

the user can verify the authenticity of any article published by `cnn.com`.

To summarize, the content name in the key request interest message from user $A$ to a friend $B$ in $A$'s community of trust would be:

$$B : A, \sigma_{k_A}(P, \tilde{n})$$

where $k_A$ is $A$'s private key, $P$ is the publisher whose key is in question, and $\tilde{n}$ is the nonce used for this interest message. And the data in the key response message from $B$ would be:

$$B, \sigma_{k_B}((k_P, t, RA), \tilde{n})$$

where $k_B$ is $B$'s private key, and $(k_P, t, RA)$ is the publisher's key information (key, expiration date, and the RA ordered list).

# 3. Security Analysis

To better understand the robustness of our approach against attacks, we examine the following attack scenarios and analyze their effectiveness in details.

## 3.1 Man-in-the-Middle Attack

In this scenario, the attacker compromises the link between the user and one of his trust community entities, in an attempt to make the user to believe a false key. However, to successfully launch this attack, the attacker needs to forge the signatures of a sufficient number of the user's friends in the key response messages. Note that the number of messages need to be forged depends on the user's local policy. Forging signatures from multiple friends is a costly attack on just a single user, specially given that the friends' keys are assumed to be obtained by the user through out-of-band means.

In a different scenario the attacker who has compromised the link from the user to his friends could drop the legitimate key response messages, in an attempt to make the user to accept the false key by majority rule. The way to prevent this is by checking the quorum by the percentage of queries sent and not but by the percentage of responses received. Considerable amount of lost responses is suspicious and relevant actions must be taken.

## 3.2 Denial-Of-Service Attack

Attackers can perform a Distributed Denial-of-Service (DDoS) attack on a particular Revocation Authority, overwhelming it by generating many queries to it. This will lead to the RA not being responsive to legitimate validation queries by users. In this case, the user will not know whether the key of interest is still valid or not. This form of attack is not potentially helpful if not coupled with key compromise of a well-known publisher associated to the RA. But still, this could be mitigated by having ordered list of RA's to ask for instead of only one for particular content-sensitive publishers.

## 3.3 Replay Attack

In this case, the attacker stores a user's friend's response for a particular key for later replay. This is particularly effective when an emergency rollover happens, making the key no longer valid. By replaying the same invalidated key, the user will receive false key information. To remedy this, a nonce is used in key request interest messages which needs to be included in response messages. Same technique can be used to protect the user against replay attacks against RA responses.

## 3.4 Compromised Notary

There could be a scenario that notary gets compromised and turns malicious. In that case it could potentially send incorrect information to the user. However, not following the complete trust principle immunes the user from harms of a single compromised notary.

## 3.5 Compromised Friend

A compromised friend, similar to a compromised notary, could send false key information. As discussed in the previous section this is generally not an effective attack.

A compromised friend can also perform a DoS attack on the user by abusing the user's resources. To achieve this, the compromised friend sends overwhelming number of key request interest messages to the user. However, this can be mitigated by liming the rate of queries sent by a particular friend.

## 3.6 Compromised Revocation Authority

A compromised revocation authority lies about the validity of a particular key. We examine both of possible scenarios. In the first scenario, the key is still valid and RA advertises that it's invalid. In this case, the user removes the key from the cache and generate key interest requests to his friends for the key. The user is expected to still believe in the correct key after consensus from his friends. In the other scenario, RA can be potentially harmful by advertising a prematurely revoked key as valid. The attacker, in this case, will only benefit from such false information, if the key for the publisher is also compromised by the same attacker. This is believed to be a hard task since the attacker must learn the keys for both the publisher and the associated RA at the same time and remain undetected throughout the attack.

# 4. Evaluation

While there are no standard means for evaluating various trust model implementations [10], there is a set of desirable characteristics that any Internet-scale system should have to succeed. These characteristics are distributed authority, independent policy, scalability, and easy enrollment. Our proposed trust model has these characteristics since there is no single or multiple globally central authorities in this trust model which also leads to scalability. Also in our

model, every user has the liberty of who to trust and what local policy to implement. This model also promotes easy enrollment since any user at any time can join and gradually expands its community of trust.

# 5. Conclusion and Future Work

In this paper, we introduced and presented the details of a trust model for NDN that gives more freedom to the user to make his own trust decisions. We then analyzed various attack scenarios and discussed how this design would thwart each attack.

This system could be further improved by a well defined reputation system that assists the user in deciding the degree of trust on a friend or a notary. It would then required well-defined metrics, such as correct responses ratio and responsiveness. This reputation system should be able to detect friend's misbehavior over time, and remove him from the user's friend list. Also, privacy issues involved in this design should be further investigated. As an example, one concern involved in this trust model, potential privacy issues by sharing what content the user is interested in with his friends.

# References


[1] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L. 2009. "Networking named content". *In Proceedings of the 5th international Conference on Emerging Networking Experiments and Technologies* Rome, Italy, December 01 - 04, 2009.

[2] D. K. Smetters and V. Jacobson. "Securing network content", October 2009. PARC Technical Report.

[3] Osterweil, E., Massey, D., and Zhang, L. 2009. "Managing Trusted Keys in Internet-Scale Systems". *In Proceedings of the 2009 Ninth Annual international Symposium on Applications and the internet (July 20 - 24, 2009)*. SAINT. IEEE Computer Society, Washington, DC, 153-156.

[4] Wendlandt, D., Andersen, D. G., and Perrig, A. 2008. "Perspectives: improving SSH-style host authentication with multi-path probing". *In USENIX 2008 Annual Technical Conference on Annual Technical Conference (Boston, Massachusetts, June 22 - 27, 2008)*. USENIX Association, Berkeley, CA, 321-334.

[5] Osterweil, E., Massey, D., Tsendjav, B., Zhang, B., and Zhang, L. 2006. "Security through publicity". *In Proceedings of the 1st USENIX Workshop on Hot Topics in Security (Vancouver, B.C., Canada)*. USENIX Association, Berkeley, CA, 3-3.

[6] Blaze, M., Feigenbaum, J., and Lacy, J. 1996. "Decentralized Trust Management". *In Proceedings of the 1996 IEEE Symposium on Security and Privacy (May 06 - 08, 1996)*. SP. IEEE Computer Society, Washington, DC, 164.

[7] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. "SPKI Certificate Theory", September 1999. RFC2693.

[8] R. L. Rivest and B. Lampson. "SDSI - A Simple Distributed Security Infrastructure". Technical report, MIT, 1996.

[9] A. Lenstra and E. Verheul. "Selecting cryptographic key sizes". Journal of Cryptology, 14(4):255-293, 2001.

[10] Wojcik M, Venter HS, Eloff JHP: 2006. "Trust Model Evaluation Criteria: A Detailed Analysis of Trust Evaluation", *In Proceedings of the ISSA 2006 from Insight to Foresight Conference, Information Security South Africa*, pp 1-9.

[11] Myers, M., R. Ankney, A. Malpani, S. Galperin and C. Adams, "Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

[12] `http://www.ccnx.org/`

[13] Housley, R., Polk, W., Ford, W. and D. Solo, "Internet X.509 Public Key Infrastructure: Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[14] Paul C. Kocher. "On certificate revocation and validation". *In Financial Cryptography*, pages 172-177, 1998.

[15] Dan Boneh, Xuhua Ding, and Gene Tsudik. 2004. "Fine-grained control of security capabilities". *ACM Trans. Internet Technol. 4, 1* (February 2004), 60-82.