future internet

MDPI

Article

A PEKS-Based NDN Strategy for Name Privacy

Kyi Thar Ko [1,*], Htet Htet Hlaing [1] and Masahiro Mambo [2]

[1] Graduate School of Natural Science and Technology, Kanazawa University, Ishikawa 920-1192, Japan; htethtethlaing@stu.kanazawa-u.ac.jp

[2] Institute of Science and Engineering, Kanazawa University, Ishikawa 920-1192, Japan; mambo@ec.t.kanazawa-u.ac.jp

* Correspondence: kyitharko@stu.kanazawa-u.ac.jp; Tel.: +81-742170056

Received: 8 June 2020; Accepted: 28 July 2020; Published: 31 July 2020

Abstract: Named Data Networking (NDN), where addressable content name is used, is considered as a candidate of next-generation Internet architectures. NDN routers use In-Network cache to replicate and store passing packets to make faster content delivery. Because NDN uses a human-readable name, it is easy for an adversary to guess what kind of content is requested. To solve this issue, we develop a PEKS-based strategy for forwarding packets, where PEKS stands for public key encryption with keyword search. We implement the PEKS-based strategy based on the best route strategy and multicast strategy of NDN and show the performance of the PEKS-based NDN strategy. We also discuss the issues of the PEKS-based NDN strategy.

Keywords: Named Data Networking (NDN); name privacy; PEKS; next generation internet architecture

1. Introduction

Today's Internet is growing incredibly year-by-year with big content providers. At the same time, people are trying to retrieve enormous online content from those providers. However, the current TCP/IP internet architecture based on end-to-end communication between two hosts becomes gradually insufficient [1]. In TCP/IP, Internet Protocol (IP) is the only protocol taking care of Network Layer, and it is known that it is not easy to add new functional components to the IP. Name Data Networking (NDN) [2], which is a branch of Information Centric Networking (ICN), has been proposed as a new Internet architecture for next-generation networks to address the existing problems of TCP/IP. Instead of using source and destination addresses, NDN identifies data content with a unique name that does not depend on the location. Security and strategy are added as two new features at the network layer in NDN architecture. These new features bring several benefits to NDN such as built-in data security and multipath forwarding. Furthermore, the in-network cache is used to store data packets for future retrieva [3,4].

NDN names have a hierarchical human-readable structure and consist of several name components which are usually delimited by ('/') or ('.') [5]. They are semantically related to content and can accordingly expose information about the content to an attacker who can then trace users' interests by examining names in their requests [6,7]. Therefore, names should be protected to avoid these threats and to enhance users' privacy by ensuring that names in a packet reveal no information about the data inside the content object [8].

There are some proposed approaches for name privacy including an overlay-based network called Anonymous Named Data Networking Application (ANDāNA) [9], and privacy-preserving content retrieval in information-centric networking (PrivICN) [7]. However, these approaches still have drawbacks.

Due to data encryption, the caching mechanism cannot be fully used in ANDāNA. Proxy re-encryption is used in PrivICN to protect name privacy but this scheme is only designed for Intra-domain usage. Moreover, a pseudorandom function, which is common in PrivICN, is used by different content requesters to encrypt names. Consequently, if each of those content requesters encrypts the same plaintext name, generated encrypted names become the same. This fact allows attackers to identify multiple packets requesting the same content so that unlinkability between encrypted names is not guaranteed.

The goal of our work is to protect names in NDN by applying "public key encryption with keyword search (PEKS)". Contributions of this paper are as follows:

- It is ensured in our design that there exists unlinkability between encrypted names even though plaintext names are identical because random values chosen by content requesters are used in every encrypted names.
- There is no master secret key in our work and key generation of private key does not depend on any master secret key.
- Network nodes are able to check whether the data content is stored in the cache even though names are encrypted because the PEKS scheme provides searching encrypted name without revealing the plaintext information. Therefore, our work maintains cache ability on all routers in NDN network.

We implement a new strategy, called PEKS-based NDN Strategy, based on Best Route Strategy and Multicast Strategy which are described in [10]. However, applying the PEKS scheme to NDN names occurs other issues. For example, string matching algorithms cannot be applied to lookup in tables, which cause a negative impact on computational efficiency. Other than this, there exist other efficiency problems such as packet size overhead and large table size because the encrypted names have a large and fixed length according to parameters of the PEKS scheme. Number of PEKS operations increases as the number of name components in a name, which results in a significant execution time overhead. Therefore, we examine to reduce the number of PEKS operations by inputting several components at a time to the PEKS scheme without affecting the operations of NDN.

The rest of this paper is organized as follows. In Section 2, we explain about the background and related work in NDN. In Section 3, we introduce the fundamentals of PEKS-based NDN. In Section 4, we present about implementation details. Section 5 shows experiments and their results. We make a discussion in Section 6. Finally, we conclude the paper in Section 7.

2. Background and Related Work

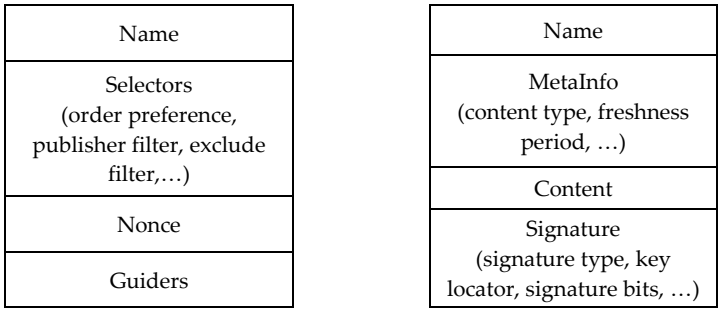
2.1. NDN Architecture

2.1.1. Name

NDN uses hierarchically structured names, for example, a document produced by Kanazawa University may have the name "/japan/kanazawa/ac/gnst/info.pdf". A content consumer uses this name to retrieve data content "info.pdf", whose location is Kanazawa University. This states that there is a significant relationship between names and data content. Names must be globally unique to retrieve global data. They are application-specific and not transparent to the network, which means naming schemes do not depend on the network [1,2,11]. Therefore, one prefix has many aggregated names. This fact provides not only a faster lookup mechanism but also better routing scalability. The users can easily memorize content names because of readability [12,13].

2.1.2. Packets in NDN

In NDN, data consumers start communication using a pull request. NDN uses two types of packets: An interest packet and a data packet. A name is set to both types of packets to uniquely identify a piece of data [2,14]. An interest packet is created by a consumer and sent to the network to retrieve the desired data content. Forwarding the interest packet to its corresponding producer is done by routers using Forward Information Base (FIB) table contained in each router. When the Interest packet arrives at the data producer, the producer node will reply a data packet that is composed of both the name of the data and as well as the content. This data packet is returned to the consumer by following the incoming Interest path [2]. Figure 1 describes formats of both Interest and Data packets. A Nonce field is contained in the Interest packet. The purpose of the Nonce field is to prevent interest packets from looping in the network. Data packet is composed of Content requested by a Consumer, name of the interest packet and the Signature.



(a) Interest Packet (b) Data Packet

Figure 1. Named Data Networking (NDN) Packets (reproduced according to "Name Data Networking") [2].

The data producer signs the data packet and adds a signature in the data packet. Key locator in the Signature field contains the name of public key that is used to verify data packets and the public key can be retrieved using the name. The owner of the public key can be considered as a legitimate signer [15]. In NDN, a key is also a named data content and it can also be signed by administrators' keys and/or a pre-authenticated root key [16]. A signed data packet, containing the public key as data content, becomes a certificate.

Interest packet authentication can be also done by signing an interest packet. A generated signature is appended to the last component of the interest name. The signature contains InterestSignatureInfo and InterestSignatureValue as shown in Figure 2. InterestSignatureInfo contains the elements such as a type of signature algorithm and key locator. To ensure that a signed interest packet is unique and to prevent possible replay attacks, InterestSignatureInfo must contain at least one of the elements:

- SignatureNonce: This number is added to assure the uniqueness of the signature,
- SignatureTime: This number is the timestamp of the signature represented in milliseconds since 1970-01-01T00:00:00Z (Unix epoch),
- SignatureSeqNum: This element is also used to provide the assurance of the signatures' uniqueness. The number is used to protect against replay attacks [17].

InterestSignatureValue contains the generated signature value. When a producer, a consumer or a network node receives an interest or data packet with a signature, it can authenticate the sender of the packet by using the information contained in InterestSignatureInfo and InterestSignatureValue [11]. An entity, which receives a signed interest packet or data packet, can recursively follow the key locator in

each packet to retrieve the public-key certificate of the signer who has created the packet. These procedures are necessary if the entity needs to authenticate the packet.

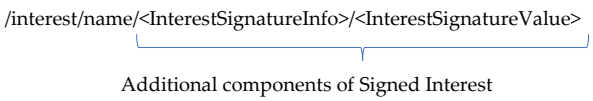


Figure 2. Signed interest name format [17].

2.1.3. Routing and Forwarding

Routing and forwarding in NDN are based on names, which eliminates the problems caused by IP addressing scheme such as managing the addresses, Network Address Translation, and IP address exhaustion. An NDN Forwarder called Named Forwarding Daemon (NFD) is responsible for forwarding packets. Every NDN router contains three types of data structures which are shown in Figure 3 [2]:

- Pending Interest Table (PIT): This table is used to store incoming interest packets which arrive at a router. The router reads the interest packet's parameters such as incoming interface, interest lifetime, and Nonce number. Then it stores them in the PIT. It also checks the name and the Nonce field of the incoming packets with PIT. If there is the same Nonce Number, the router assumes that this is an interest loop and simply discards it [2,4,18]. The router will then discuss the interest name with FIB and forward the packet to the nexthop.
- Content Store (CS): NDN router can cache the data packets passing through them and store those packets in CS. This property called in-network caching allows NDN hosts to share contents with low usage of bandwidth and latency [2,4,18].
- Forwarding Information Base (FIB): As traditional routers do, NDN router needs to have the information that shows where the incoming interest packets should be forwarded. Therefore, the role of FIB is to record the information for the name to reach the destination. The FIB can be updated by routing protocol such as OSPF based Routing Protocol for NDN (OSPFN) [2,4,18].

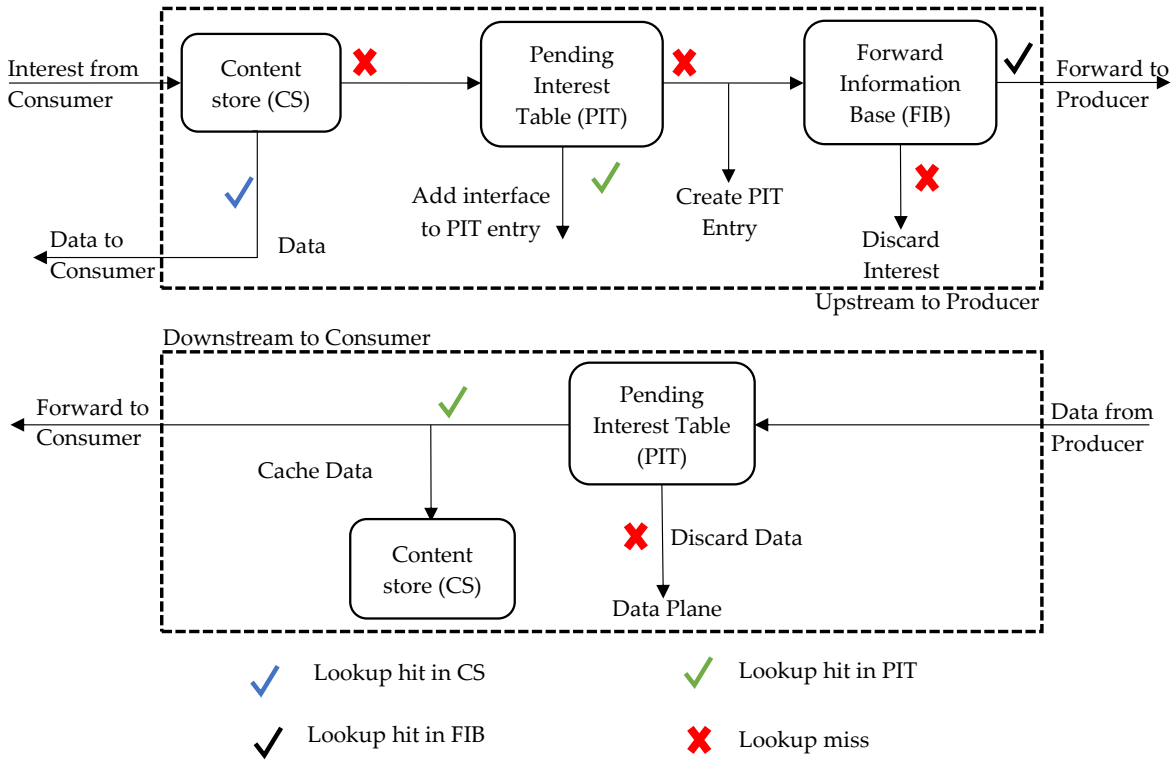


Figure 3. NDN packet forwarding process (reproduced according to "Name Data Networking") [2].

2.2. Public Key Encryption with Keyword Search

An encryption scheme supporting a search mechanism is called searchable encryption. Symmetric Searchable Encryption (SSE) and Public Key with Keyword Search (PEKS) are the most popular Searchable Encryption (SE) techniques. A Searchable Encryption Scheme (SSE) [19] proposed in 2000, solves the problem of searching encrypted data on the storage servers. The SSE allows the user to upload data to the cloud with provable secrecy by issuing an isolated and hidden query. The hidden and isolated query allows the server to learn nothing about the plaintext except for the ciphertext. The query is running as an encrypted query which is called trapdoor. Trapdoors are always generated using a secret key. It has the following limitations: While the construction is proven to be a secure encryption scheme, it is not proven to be a secure searchable encryption scheme; the distribution of the underlying plaintexts is vulnerable to statistical attacks; searching is linear in the length of the document collection [20]. Furthermore, the scheme suffers from a complicated secret key distribution [21]. On the other hand, the PEKS scheme introduced by Boneh et al. in 2004 allows one to search encrypted documents on the untrusted server without revealing any information as shown in Figure 4. This scheme overcomes the secret key distribution issue in the SSE. In order to enhance name privacy in NDN, we adopt the PEKS scheme.

The PEKS scheme [22] by Boneh et al. incorporates keyword search functionality into public key encryption and it is secure based on the Bilinear Diffie–Hellman assumption in the random oracle model. In the PEKS scheme, three parties, Alice, Bob, and Server, are involved. Bob wants to share sensitive data with Alice. He uses Alice's public key A_{pub} to encrypt a message msg with keywords W_1, ..., W_k. The following ciphertext (1), which is generated by Bob, is sent to the server.

$$Ciphertext = [E_{A_{pub}}(msg), PEKS(A_{pub}, W_1), ..., PEKS(A_{pub}, W_k)] \quad (1)$$

where $E_{A_{pub}}(msg)$ is the encrypted message and PEKS(A_{pub}, W_k) is a ciphertext obtained by encrypting W_k under A_{pub} using PEKS algorithm. When Alice wants to download the ciphertext containing the keyword she specifies, she generates trapdoors T_W by using her secret key and keyword. She then sends it to the server. The server gets no information about the message inside the ciphertext but is able to search a specific keyword by using the trapdoor T_W. When matching ciphertexts are found, the server sends those ciphertexts to Alice. She can decrypt the ciphertext using her private key A_{priv}.

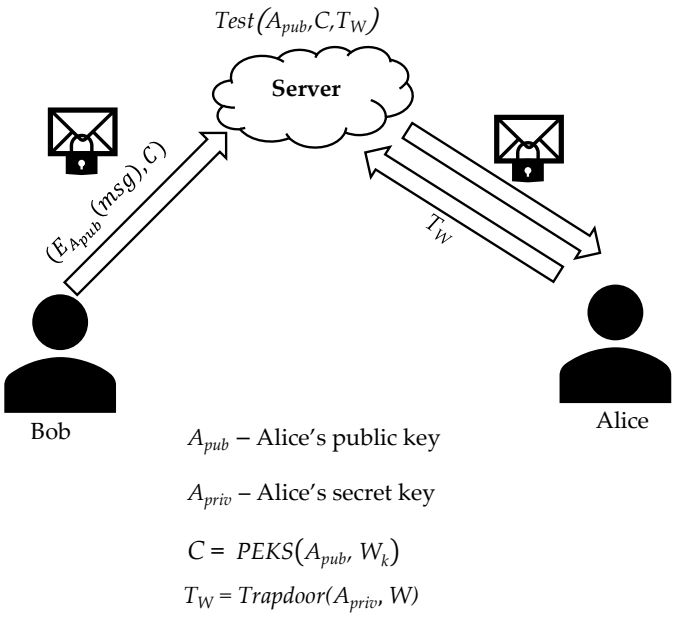


Figure 4. Public key encryption with keyword search (PEKS) system framework.

The details of the PEKS scheme are described as follows. The scheme uses two groups G_1, G_2 with a prime order p and a bilinear map e: $G_1 \times G_1 \rightarrow G_2$ between them. The map satisfies the following properties:

- Computable: Given $g, h \in G_1$ there is a polynomial time algorithm to compute $e(g, h) \in G_2$.
- Bilinear: For any integers $x, y \in [1, p]$ we have $e(g^x, g^y) = e(g, g)^{xy}$.
- Non-degenerate: If g is a generator of G_1 then $e(g, g)$ is a generator of G_2.

Two hash functions are given as $H_1 : \{0,1\}^* \rightarrow G_1$ and H_2: $G_2 \rightarrow \{0,1\}^{logp}$. The PEKS scheme uses the following four polynomial-time algorithms:

- $KeyGen(1^k)$: The input security parameter determines the size of p and the groups G_1, G_2 . The algorithm picks a random $\alpha \in Z_p^*$ and a generator g of G_1. It outputs A_{pub} = [g, $h = g^{\alpha}$] and $A_{priv} = \alpha$.
- PEKS(A_{pub}, W): First compute t = e(H_1(W), h^r) for $r \in Z_p^*$. Output PEKS (A_{pub}, W) = [g^r, H_2(t)]
- Trapdoor (A_{priv}, W): Output $T_W = H_1(W)^{\alpha} \in G_1$
- Test (A_{pub}, C, T_W) : Let [g^r, H_2(t)]. Test if H_2(e(T_W, g^r)) = H_2(t). If so, output "yes". Otherwise, output "no".

When the search finishes, the server returns the search result to Alice. During the search, the server does not know the content of the documents, nor the keyword [22,23].

2.3. Related Solutions on Name Privacy

Name Privacy issues have been considered in several related work. Proxy re-encryption is used in [7] to protect the name privacy of the packets and they apply the ElGamal encryption scheme. In their work,

there are ICN hosts (content consumer and producer) and ICN nodes (routers). The edge node nearest to a consumer or a producer behaves as a proxy. A Key Management Server (KMS), which holds a master key $X_{MK} = X_{client} + X_{proxy}$, where X_{client} is a client key and X_{proxy} is a proxy key. The consumer encrypts interest names using its own client key and sends interest packets to the proxy. The proxy re-encrypts the incoming interest names with the associated proxy key to the consumer, and then it forwards to the network. It seems like the ciphertexts are encrypted using the master key after the proxy re-encryption. In their work, each name component is input to a pseudorandom function, which is common to all consumers and producers in the network. The output value of the pseudorandom function is used in place of random value which means the encryption has the same output for same plaintexts. Furthermore, their scheme relies on a single Key Management Server for the whole domain. Privacy-aware transmission scheme based on homomorphic proxy re-encryption for NDN (PATS_NDN) systems in [24] uses proxy re-encryption and a blind algorithm for privacy-aware content retrieval. In their work, Content Provider publishes generated content to the network. Then consumer uses a shared master secret key to encrypt the content name and requests alias name to the provider. The provider then generates an alias of content name by using a quadratic function and also generates a re-encryption key with respect to the consumer's public key. It then uses a blind algorithm to cover the alias. At last, the producer sends the information to the consumer. The latter then creates an interest packet and sends it to the network. Any router that receives the interest packet will check if there exists the matched data content. If it finds out a match, it will reply data. Otherwise, it will forward to the network. In their work, many encryption and decryption are involved. This creates some latency during transmission.

Paper [9] designs a tool called AND$\bar{a}$NA which contains a number of features from Tor. It is an onion routing overlay network and provides privacy and anonymity to consumers. A pair of distinct anonymizing routers (ARs) are used to send encrypted interest packets. A consumer generates two symmetric keys that will be used to encrypt the retrieved content packet and shares to respective ARs. When the AR, which is closed to the producer, receives a data packet, it encrypts the whole packet with the key provided by the consumer. Then it creates a data packet containing the ciphertext as content and sends it to the next AR. The latter retrieves the ciphertext from the received data packet and encrypts it by using the consumer provided key. The second AR forwards the result of the encryption to the consumer. The consumer then decrypts the payload. However, due to data encryption, caching effectiveness is reduced. Name and naming convention explicitly convey desired information and can facilitate trust and key management in NDN. An identity-based architecture is introduced in [25] to prevent the operation traceability. Domain Trusted Entity (DTE) is a special entity to manage and protect the communication aspects of its domain identities. The usage of Bloom filters is suggested in [26] as a possible solution for name privacy. By using Bloom filter, the architecture would contain a hierarchical bloom filter used as a routing table, a counting bloom filter for each interface used as a PIT table, and a hierarchical bloom filter used as the router storage. However, using Bloom filters gives false positives and it is necessary to reset periodically.

3. Fundamentals of PEKS-Based NDN

In PEKS-based NDN, Alice in Figure 4 is a data producer. She owns a pair of private and public keys. She uses her own private key to create a trapdoor for each component N_k of a name. The outputs are then concatenated by using separator "/" and a trapdoor name is produced as $/T_{N_1}/T_{N_2}/.../T_{N_k}$. Hereafter, the notation T_W in Figure 4 is changed to T_{N_k} to represent a trapdoor of each name component N_k. When a consumer, Bob, wants to send an interest packet to Alice, he uses Alice's public key to transform each name component N_k to PEKS ciphertext C_k and obtains an encrypted name $/C_1/C_2/.../C_{N_k}$. Routers in the network take the role of the server of Figure 4 and use the test algorithm of the PEKS scheme to decide

where to forward packets or to search in CS. In these operations, the PEKS scheme is applied to the name of NDN packets. Name components becomes keywords in PEKS-based NDN. Only name is encrypted in this work.

3.1. Threat Model

In NDN, there are several routers in the network. These routers are responsible to check interest packet's name whether to forward or to retrieve the data content from CS. So, we need to use these routers as semi-trusted parties in our design. These nodes have an ability to check the incoming interest name with the trapdoors which are stored in the local storage of each router. For this reason, each node may operate honest-but-curiously to do packet forwarding and reply. However, in this paper, we mainly focus on the attackers outside of this scenario. The one who is listening to the consumer's interest may get no information about what the consumer is requesting to the producer because the name in every packet is encrypted using PEKS as shown in Figure 5. These nodes will have ability to check the incoming interest name with the trapdoors which are stored in the local storage of each router. For this reason, each node may operate honest-but-curiously to do packet forwarding and reply.

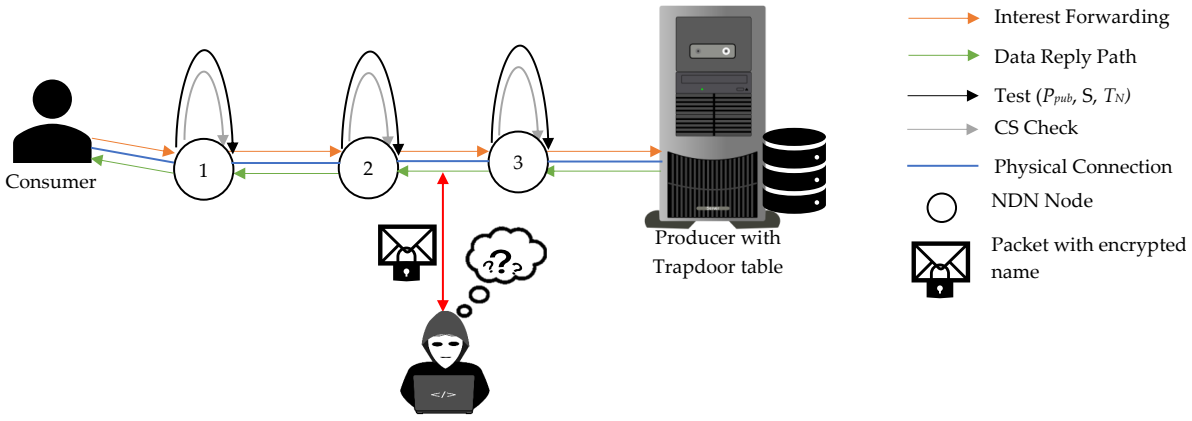


Figure 5. Thread model to PEKS-based NDN.

3.2. Framework of PEKS-Based NDN

The purpose of using PEKS in our work is only to check names with tables inside NFD. So, the encryption part $E_{A_{pub}}[msg]$ which is shown in Equation (1), is not used. For this reason, we are able to reduce PEKS ciphertext creation time. In NDN, names are composed of several components N_j, i.e., name has a form of $/N_1/N_2/$ ... $/N_k$. The framework of PEKS-based NDN is shown in Figure 6.

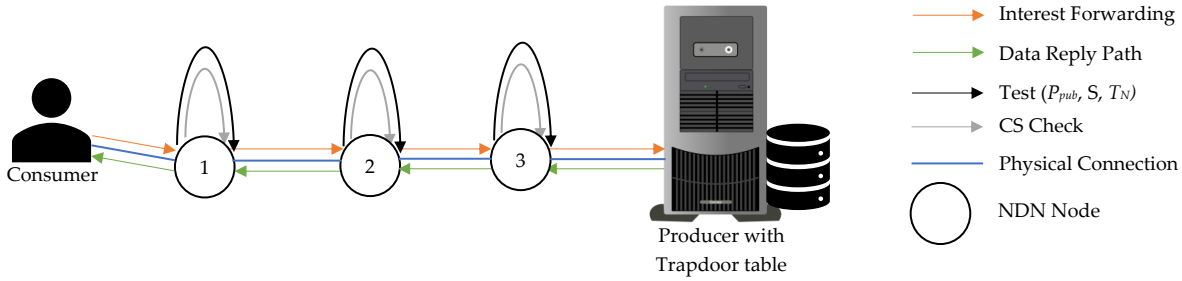


Figure 6. PEKS-based NDN network.

A producer owns a table in which each trapdoor is mapped to a plaintext name of the data content name. At first, a producer creates a trapdoor for its own prefix and disseminates it to the network. Routers in the network store the producer's prefix trapdoor and also register this trapdoor together with the ingress interface to FIB. A consumer then generates PEKS for each name component for the interest packet and producer generates trapdoors. When a router receives an interest packet, it uses Test (P_{pub}, C_j, T_{N_j}) to match the PEKS values and trapdoors of each name component N_j, where, P_{pub} is producer's public key, C_j is PEKS(P_{pub},N_j) and T_{N_j} is the trapdoor of N_j. The black arrow in the figure shows the test process inside a node. The gray arrow in the figure shows the process of CS lookup if the result of the test is an exact match. The router replies the data packet to the consumer if the data exists in CS. If the result of the test becomes longest match, the router queries to nexthop information by using the matched trapdoor in the FIB. Using the information received from FIB, the router forwards to nexthop. When the packet arrives at the producer, it also uses Test (P_{pub}, C, T_{N_j}). When the exact matching is found, the producer creates a data packet and replies to the consumer.

In this work, we assume that trapdoor is securely disseminated to NFDs and securely stored at routers. We also assume that producers, consumers, and routers are connected through normal NDN network. The format of Interest packets' name generated by a consumer will become as shown in (2) and The format of interest packets' name generated by a producer to disseminate the trapdoors is shown in (3).

$$/peks\_strategy/C_1/...../C_k \tag{2}$$

where, each C_k is PEKS ciphertext transformed from each name component N_k by using PEKS(P_{pub}, N_k) algorithm, and P_{pub} is the public key of the producer.

$$/peks\_strategy/Tw/T_{N_1}/...../T_{N_k} \tag{3}$$

where, "*Tw*" is used to inform NFDs that the interest packet contains a name in the form of trapdoors, T_{N_k} is the trapdoor of each name component N_k generated by the producer using Trapdoor (P_{priv}, N_k), and P_{priv} is the private key of the producer. The purpose of using "/peks_strategy" in both (2) and (3) is that our newly developed strategy named "PEKS_Strategy" should be triggered by NDFs to forward the packets.

3.3. Public Key Dissemination

In PEKS-based NDN, a producer needs to disseminate its trapdoor and public key to the network to check PEKS ciphertexts C_k. There are two types of public key dissemination process: Dissemination to the network and Dissemination to consumers. Dissemination to the network is done by sending an interest packet which is created by the producer with use of public key and parameters as a name. When a network node receives the interest packet, it retrieves public key from the name of the interest packet. In Dissemination to consumers, each consumer sends an interest packet to request public key of the producer before using PEKS-based NDN. The producer replies public key and parameters by using a data packet. To make sure the authenticity of public key, both interest and data packets need to be signed by the producer. In this subsection, the detailed procedures of public key dissemination are explained.

- Dissemination to the network: After the producer generates a pair of public and private keys for the PEKS scheme, it starts a dissemination process of public key to the network. It encodes public key and parameters to the string format. The result strings are concatenated by using a separator "/". An interest packet is then created by the producer with use of the concatenated string, which serves as a name, such as /peks_strategy/KEY/public-key-string/PARAM/parameter-string, where "KEY" indicates that public key string is appended after it, and "PARAM" indicates

that parameter string is appended. The producer also owns an RSA key pair, which is signed by administrators' keys or a root key, for signing both interest and data packets. It then signs the interest packet as shown in Figure 2. Therefore, the name in the interest packet becomes /peks_strategy/KEY/public-key-string/PARAM/parameter-string/<InterestSignatureInfo>/<InterestSignatureValue>. SignatureSha256WithRsa algorithm, which is provided by ndn-cxx library [27], is used to sign the packets. Key locator, which contains the name of RSA public key, is set to InterestSignatureInfo. To make sure the uniqueness of the signature, SignatureTime and SignatureSeqNum are also used. The type of signature algorithm, SignatureTime and SignatureSeqNum are added to InterestSignatureInfo. The generated signature is set to InterestSignatureValue. The producer then sends the interest packet to the network. Routers that receive the interest packet verify the sender of the interest packet by using the information which are contained in InterestSignatureInfo and InterestSignatureValue. The dissemination processes of the trapdoor of the producer prefix's name, which is known as producer's prefix, and public key can be combined by using a single interest packet.

- Dissemination to consumers: When a consumer wants to use PEKS_based NDN, it sends an interest packet to request public key of the producer. The producer creates a data packet consisting of public key and parameters as data content. The producer also signs the data packet according to NDN protocol. The signature algorithm that is used to sign interest packets is also used in signing the data packet. When the consumer receives the data packet, it verifies the data packet with the signature contained in the data packet. In this way, public key for the PEKS scheme can be verified.

In both cases, an entity which receives a signed packet, needs to follow recursively key locator to authenticate RSA public keys and signers as explained in Section 2.1.2. Note that the name in these processes is not encrypted.

4. Details of Implementation

The implementation is written in C++. For NDN operations, we use ndn-cxx [27] library and NFD software [28]. For all cryptographic operations, we use PBC library [29]. In our work, the producer is responsible to generate public/private key pair and public parameters. The consumers who have the producer's public key are able to retrieve the content by using PEKS-based NDN. The NDN nodes need to check incoming interest packets' names by using the PEKS scheme.

4.1. Choosing Parameters for PEKS

Public key encryption with keyword search is based on pairing and bilinear mapping. So, we need to choose parameters that meet certain security levels. Table 1 shows that there are multiple options to achieve a certain security level. The bit sizes in the first column are those matching the security levels of the SKIPJACK, Triple-DES, AES-Small, AES-Medium, and AES-Large symmetric key encryption schemes. We consider choosing 80 bits security level to achieve better performance. To achieve that security level, it is necessary to choose r, which is a subgroup size, 160 bits. Choosing the size of q^k, which is a prime number, to the subgroup r is dependent not only on the extension field size k but also on an embedding degree ρ according to Equation (4) [30].

Table 1. Bit sizes of curve parameters and corresponding em-bedding degrees to obtain commonly desired levels of security [30].

Security Level (in bits)	Subgroup Size r (in bits)	Extension Field Size q^k (in bits)	Extension Field Size k $\rho \approx 1$	$\rho \approx 2$
80	160	960–1280	6–8	2*, 3–4
112	224	2200–3600	10–16	5–8
128	224	3000–5000	12–20	6–10
192	384	8000–10000	20–26	10–13
256	512	14000–18000	28–36	14–18

$$\frac{\log q^k}{\log r} = \rho \cdot k \tag{4}$$

In our case, we choose a curve with embedding degree 4 and $\rho = 2$ such that the size of q^kshould lie between 960 bits and 1280 bits. With the value 512 bits for q size, q^k size becomes 1024. This value exists between the range as shown in Table 1. After choosing parameters, a pair of public and private keys are generated by a producer. The producer keeps private key for itself, and announces public key together with parameters.

4.2. Producer Operations

In key generation, $KeyGen(1^k)$ algorithm is used and announces the public parameters G_1, G_2, H_1, H_2 as well as Producer's public key P_{pub} = [g, h = g^{α}]. Figure 7 shows that the producer also stores trapdoor T_{Name_j} = /Trapdoor(P_{priv}, N_1)/.../Trapdoor(P_{priv}, N_1) and content name $Name_k = /N_1/.../N_k$ in the Producer's trapdoor table, where N_j is a name component.

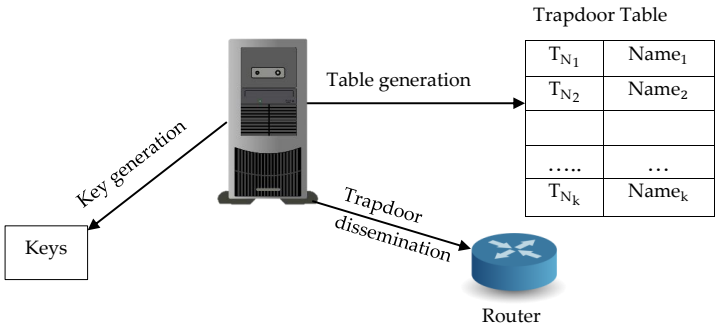


Figure 7. Producer initial setup.

When an interest packet arrives at the producer, it uses Test (P_{pub}, C, T_N) to check incoming the interest name component by component with the use of the trapdoor table. If the data name is found, it then creates a data packet and replies to the consumer. It then starts to disseminate the trapdoor, which corresponds to the incoming interest name, to the network.

4.3. Router's State

In NDN routers, there are three data structures: Pending Interest Tables (PIT), Content Store (CS), and Forwarding Information Base (FIB) as we mentioned in Section 2. Those tables are accessed by NFD during the forwarding process. So we will use NFD when we refer to the processes inside routers. Our work needs to use trapdoors to check incoming interest names. So we added a new table called

Trapdoor as shown in Figure 8. In the figure, NFD stores a trapdoor T_{N_1}, which is generated by a producer, in the Trapdoor data structure $[T_{W_1}]$. When an interest with a component "/*Tw*" as shown in (3), NFD needs to update the data structure, it needs to check them in its data structure such as $[T_{N_1}, T_{N_2},...,T_{N_k}]$, where k is a positive integer. NFD also registers the incoming trapdoor at FIB with the incoming Face ID when the producer disseminates trapdoors. Then NFD uses multicast the incoming interest packet to all adjacent routers. Every NFD in each router does the same procedure to disseminate the trapdoors.

PIT

Name	None
Nonce	None
Face	None

CS

Name	None
Content	None

FIB

Name	$/T_{N_1}$
Next Hop	Face i

Trapdoor

$[T_{N_1}]$

Figure 8. Router's state after producer's first dissemination.

4.4. Interest Packet Creation

During interest packet creation, consumer uses producer's public parameters and uses $PEKS(P_{pub}, N_j)$ and produces $[g^r, H_2(t_j)]$, where t_j is $e(H_1(N_j), h^r)$ for each $r \in Z_p^*$. The consumer then creates interest packets with the PEKS ciphertexts as shown in Figure 9. It also sets other necessary parameters such as Nonce, interest lifetime [31] and, etc. Then the consumer sends the interest packet to the network.

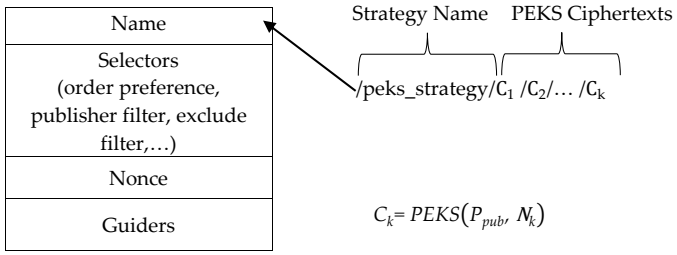


Figure 9. Interest packet with PEKS ciphertexts.

4.5. NFD Operations

When NFD sees "/peks_strategy" in the name of an interest packet, it then triggers PEKS_Strategy to match with the Trapdoor table. This Trapdoor is a nested list. It contains several inner lists that hold each name component as an element. Each inner list is meant for a name. We use exhaustive search to match every PEKS ciphertexts with the Trapdoor table as shown in Algorithm 1.

In the Algorithm 1, the created PEKSList[] is obtained by extracting name components of an interest name. The output becomes PEKSList$[C_1, C_2, ..., C_k]$. Then NFD uses Test (P_{pub}, C_k, T_{N_k}) algorithm to check the first element of PEKSList and the first element of the first element of Trapdoor. If it is not a match, NFD moves to the next element in the Trapdoor. If it is a match and the length of PEKSList is equal to the length of the entry list inside Trapdoor, every C_j is checked with T_{N_l} in inner list. Every result with a match is marked as true in an array matchCount[]. NFD continues this iteration until the end of the PEKSList[].

Algorithm 1: Test Operation

Input: $InterestName$, $Trapdoor$, P_{pub}
Require: $i = 0$, $oldMatchCount[] = 0$, $PEKSList[]$
Output: ExactMatch or LongestMatch

```
1  while not at end of InterestName do
2      component = getComponent(InterestName)
3      PEKSList[i] = component
4      i = i +1
5  iterator = index of PEKSList[]
6  for row = 0 to end of Trapdoor[noOfRow] STEP 1 do
7      if Test(P_pub, PEKSList[0], Trapdoor[row][0]) then
8                                                  ▷ First component equal
9          matchCount[iterator] = true
10         iterator = iterator + 1
11         for iterator to end of PEKSList[], col = 1 to end of Trapdoor[row][noOfColumn] col ← col + 1,
            iterator ← iterator + 1 do
12             if Test(P_pub, PEKSList[0], Trapdoor[row][col]) && len of PEKSList[] == len of
                Trapdoor[row] then
13                                                 ▷ Find the exact match
14                 matchCount[iterator] = true
15             else
16                 Break
17         if col == iterator then
18             exactMatch == false
19             if all of matchCount == true then
20                                                 ▷ Found an exact match
21                 exactMatch = true
22                 return Trapdoor[row]
23         else
24             if No. true in oldMatchCount[] < No. true in matchCount[]) then
25                                                 ▷ Longest match found
26                 tmp = Trapdoor[row]
27                 exactMatch = false
28                 update oldMatchCount[]
29 return tmp
```

Then NFD checks whether all of the elements in matchCount[] are true at a condition, where the length of PEKSList[] is equal to the length of inner_list. If it is true, there is an "Exact match". Then NFD uses the exact match trapdoor to check CS. If there is a match, the data exists in CS and NFD will reply Data(InterestName + Content) to the consumer. If the result of Test (P_{pub}, C_k, T_{N_k}) algorithm is false for C_k and T_{N_k}, NFD breaks the operation then store the number of match count in oldMatchCount[] list. The process continues until the end of the Trapdoor table. If there is no update in the oldMatchCount[], then it becomes "Longest match" trapdoor. With this trapdoor, NFD gets nexthop from FIB. Then the packet is

forwarded to the nexthop by choosing the best route. If there is no single matched trapdoor, NFD will multicast the interest packet. NFD also uses a hash function to generate a hash value from the interest name and makes tuple(hash, interest name). It then stores in a temporary location. NFD uses this tuple to update the Trapdoor table. The matching process by using hash value is O(1) time. This is much faster than using polynomial-time algorithm Test (P_{pub}, C, T_{N_l}).

After NFD has received a reply data packet, it also stores data packet at a temporary location. When the producer sends trapdoor update information, it also append the hash value at the end. NFD checks the hash value in the tuple(hash, interest name) to get the interest name. Using this name, NFD retrieves data packet. It then updates CS by using incoming trapdoor and the retrieved data packet.It also adds the new trapdoor to the Trapdoor data structure. In our case, we created CS relating to PEKS_Strategy. However, this is also an unsorted data structure in our scheme.

4.6. Strategy

The forwarding plane in NDN is called the strategy module [32] and the key to the resilience and efficiency of the NDN [2]. Strategies in NDN provide to make decision forwarding interest packets. Even though there are multiple strategies in NFD, only a single strategy handles to forward an interest packet. There are triggers to decide where the forwarding strategy needs to send the packet. Those triggers are:

- After Receive Interest Trigger: When an interest packet arrives at a router, NFD checks whether the packet violates "/localhost" scope, whether it is looped, etc,. After all of those necessary checks are done, the incoming interest packet is needed to be forwarded. Then this trigger is invoked by the Incoming Interest pipeline, which is a module of NFD. If the interest packet is using PEKS_Stratey, testing with Trapdoor data structure done at this moment and forwards to PEKS_Strategy module to match InterestName with Trapdoor as described above subsection.
- After Receive Data Trigger: This trigger is invoked by the Incoming Data pipeline. In PEKS_Strategy, storing data packet to CS is done after this trigger is invoked.

In abstract, when an interest packet is forwarded to nexthop, PEKS_Strategy adapts the best route and when NFD receives trapdoor dissemination, it uses multicast. If there is no single match, it also multicast the interest packet to the network. If it is not a PEKS packet, NFD will use one of the other strategies to forward interest packet. PEKS_Strategy should be applied to all NFD in the network in order to use PEKS-based NDN.

5. Experiments and Results

In this section, we explain experiments and results. The first experiment shows that if a consumer wants to adopt a straight forward construction of the PEKS-based strategy, it needs to encrypt each name component by using a producer's public key. After PEKS ciphertext is computed, output elements, g^r and H_2(t) are concatenated by a 1 byte character to create a single string. Then the string becomes a ciphertext component C_j of an encrypted name $/C_1/C_2/.../C_k$. Once parameters of PEKS are determined, the output size of the ciphertext is fixed. This means that the size of encrypted ciphertext generated by a consumer is the multiplication of the name size and the number of components. Furthermore, using PEKS scheme's algorithm multiple times is a painful process for a router in a network. Unlike NDN, PEKS-based NDN includes a hash function in the encryption process. It is possible to input multiple name components of plaintext names at a time. Such a construction reduces the size of the encrypted name and has less number of executions. Therefore, we can design a more efficient construction in which multiple name components rather than each name component are hashed in the encryption process. Hashing all name components

except the producer's prefix component seems to yield the most efficient scheme. We discuss in Section 5.2 that such a whole-but-prefix name components encryption construction does not always provide the best efficiency. The purpose of the second experiment to show the caching ability of PEKS-based NDN and evaluation of download with respect to the cache hit ratio.

5.1. Experimental Setup

We set up an experiment as shown in Figure 10. In the figure, the router R1 is running on a machine with Intel(R) Xeon(R) CPU E5-1680 v4 @ 3.40 GHz and 64 GB of Memory, while other are running on VMs with 1CPU and 1024 MB of Memory. The producer is located at the same node as router R1. The consumers, Con1, Con2, Con3, and Con4, run also at the individual node with a different router. Small and fixed size of content is retrieved in these experiments and the producer is using a name list $\mathbb{N}$ which contains 100 names.

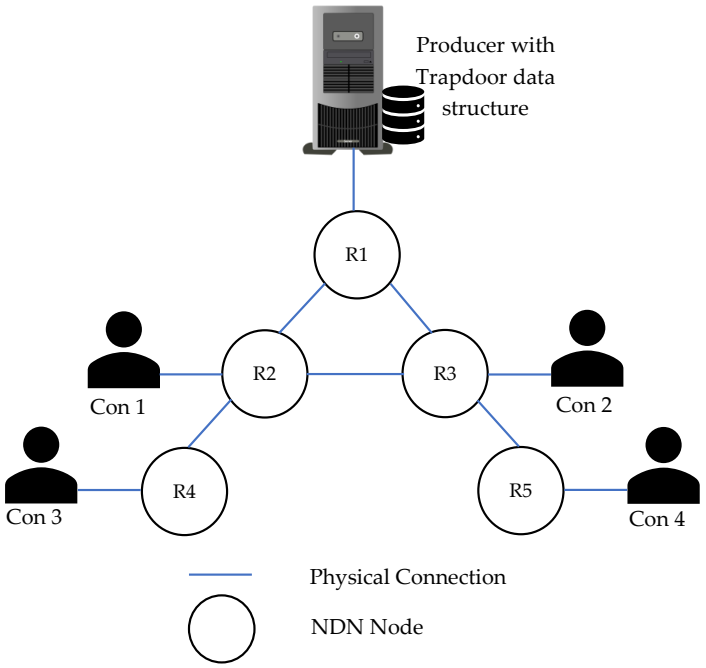


Figure 10. Experimental setup.

5.2. Finding the Length of Input for the PEKS Scheme

In the straight forward construction, the size of encrypted names becomes larger if there are many name components in a name. Encrypting each name component result in a large amount of size in a packet and a remaining space for content becomes smaller. The number of name components also affects the execution time because ciphertext generation and test algorithms are applied to each of the name components. As explained in this section's introduction, the multiple name components encryption construction is expected to provide faster processing and shorter ciphertext size. Therefore, in this section, we examine to find how many name components should be used as an input to the encryption algorithms. To this end, we assume the following setup. A list of names shared by the producer and the consumers is available for determining a suitable number of name components fed into the hash function.

Compute $\lceil \#C_j/\mathrm{d} \rceil$ to get the number of component inputs to the PEKS scheme, where divisor d is a positive integer and $\#N_j$ is the number of name components of a name entry in the name list. The result

of this division determines the number of PEKS ciphertexts generation operations for the name entry. One can count the number of name entries having the same value for $\lceil \#C_j/\mathrm{d} \rceil$, which is related to the number of PEKS ciphertext generation operations. As shown in Equations (2) and (3), the number of PEKS ciphertext components and that of corresponding trapdoor components are the same. Therefore, one can predict how many times one should execute PEKS ciphertext generation operations. In the same way, one can predict how many times the Test algorithm is run in the network if one can roughly figure out how many operations executed in the intermediate nodes. In our experiment, the producer's prefix is input to the PEKS scheme and we process the following steps:

1. Each component of a content name without including the producer's prefix is used to generate PEKS ciphertexts and trapdoors.
2. The outputs, PEKS(P_{pub}, N_j) or Trapdoor(P_{pub}, N_j), of the PEKS scheme are concatenated to create interest packet.
3. Consumers send interest packets to retrieve data from the producer.
4. The producer disseminates trapdoors.

Then d is increased by one for each experiment and do the same steps described above. These processes continue until there exits only one component in the ciphertext of content name. In these processes, interest packets are sent only by the consumer Con1 in Figure 10. Then we measure download time starting from PEKS ciphertexts creation until the data packet retrieval. In this experiment, consumer Con1 requests all content name listed in $\mathbb{N}$.

In general, if the number of PEKS ciphertexts decreases, the number of tests should also decrease. However, in PEKS_Strategy, the test process depends not only on the number of components but also on whether the number of trapdoor components in the trapdoors and PEKS ciphertexts in the interest packet is equal.

As shown in Figure 11, when we increase the number of input components to the PEKS scheme, the test process executions significantly drop at 3. However, those executions increase again at 5 where several ciphertexts with the same length are required by several interest packets and trapdoor table is to lookup several times for the exact match according to Algorithm 1. It is significant when the whole of the content name is used as input to the PEKS scheme producing only one ciphertext. Figure 11 also shows the average download time for each number d of input components to the PEKS scheme. The average number of tests and the average download time behaves in a similar way and the correlation ratio between them is 0.895189. However, these results actually depend on the name list which is shared between the consumers and the producer and also network operations.

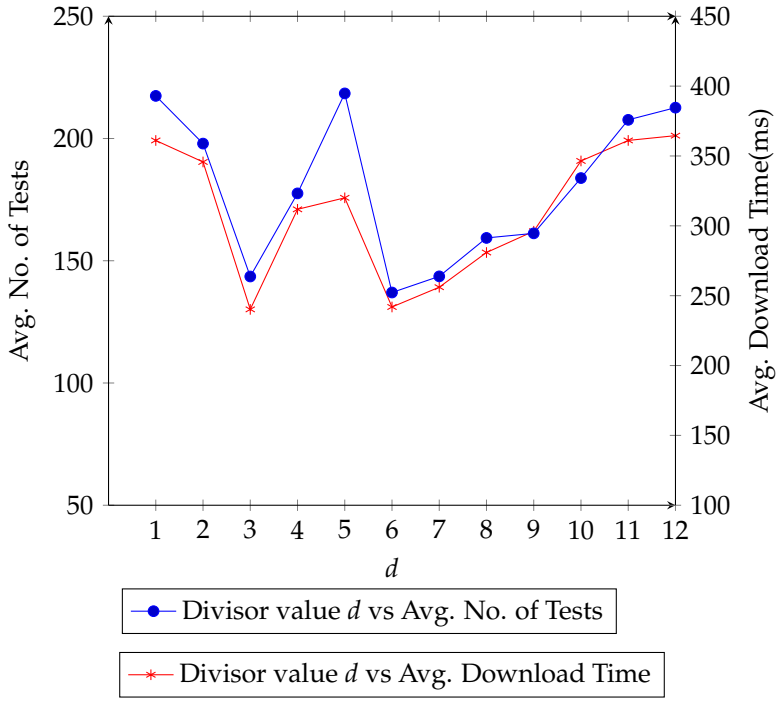


Figure 11. Comparison for number of tests counts and download time. d is the divisor to $\#N_j$ in the plaintext name to get the number of input to the PEKS scheme.

5.3. Measuring Cache Hit Rate and Download Time

We examine the relationship between the cache-hit ratio versus download time to check whether the in-network caching mechanism is preserved in the proposed scheme. For that purpose, we use the experimental setup shown in Figure 10.

In the experiment, all consumers, Con1 to Con4, randomly request using 25 content out of $\mathbb{N}$, which contains 100 names. Each consumer selects a random name and retrieves 100 data content from the producer by using PEKS_Strategy. The cache hit rate for each router is measured and also download time for each consumer is recorded. Then the results are then observed. Figure 12 shows the download time with respect to the cache hit ratio at router R4 which is the nearest to consumer Con1. In the figure, when the percentage of hit ratio increases at Router R4, the average download time drops. This clearly shows that even though the PEKS-based NDN strategy is applied to NDN, in-network caching, which is one of the NDN properties as described in Section 2.1.3, the mechanism is maintained.

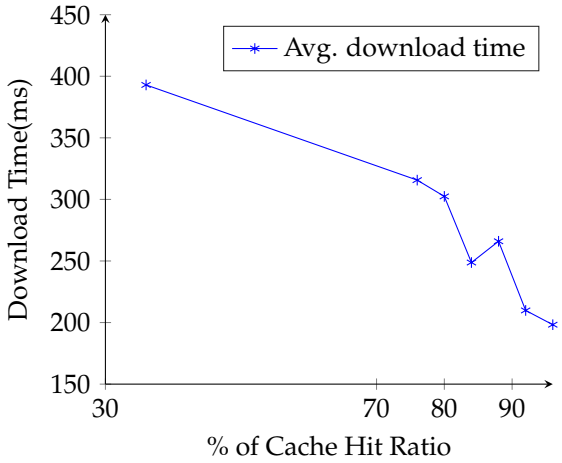


Figure 12. Average download time of Con1 vs cache hit ratio at router R4.

6. Discussion

As we introduced above, our main purpose is to protect the name privacy from outside attackers. We discuss and compare PEKS-based NDN with other two schemes: PrivICN [7] and ANDāNA [9] as shown in Table 2.

Table 2. Comparison between PEKS-based NDN and other works.

Scheme	Un-L	MK	DS	SD	TP	NL	C
ANDāNA	–	✓	–	Setup	Exit-routers (fully)	fast	✗
PrivICN	✗	✗	✗	Period	KMS(fully), Proxy(semi)	fast	–
PEKS-based	✓	✓	✓	On-demand	Routers(semi)	slow	✓

- Un-L - Unlinkability between ciphertexts
- MK - Non-Usage of Master Secret Key
- DS - Derivation of Secret Keys
- SD - Assumption of Secure Secret-value Delivery and Management
- TP - Trusted Party
- NL - Name Lookup
- C - Fully supported cache ability on all routers in NDN network
- ✓- True
- ✗- False
- "-" - Unknown or unable to determine

Privacy: As shown in Table 2, Unlinkability between ciphertexts (Un-L), Non-Usage of Master Secret Key (MK), Derivation of Secret Keys (DS), and Assumption of Secure Secret-value Delivery and Management (SD) are the issues relating to the privacy issues. We discuss these issues as follow:

(*a*) Unlinkability between ciphertexts (Un-L): PrivICN uses proxy re-encryption which is done only at edge nodes and does not need to be done multiple times. The generation process of ciphertexts for the same name results in the same ciphertext because they are using a pseudorandom function, which is common to all consumers and producers instead of using a random value in the ElGamal

encryption scheme. Because of this kind of pseudorandom function, their scheme does not guarantee the unlinkability between ciphertexts and consumers in the intermediate nodes, which are located between proxies. As a result, there remains a room for an attacker to link the packets' names with the consumers even though the names are encrypted. On the other hand, we use a random value r, which makes the PEKS ciphertext more secure to protect the name in the packet. Assume that an attacker has public key of the producer, P_{pub}, which is [g, $h = g^{\alpha}$], and the PEKS ciphertext as shown in Equation (5).

$$[g^r, H_2(t)] = [g^r, e(H_1(N_j), h^r)] = [g^r, e(H_1(N_j), g^{\alpha r})] \quad (5)$$

Because both α and r are unknown to the attacker, it becomes difficult for him to get $H_1(N_j)$. It is proven in [22] that computing N_j is difficult under Bilinear Diffie–Hellman Assumption. Because of this property, PEKS_based NDN does not allow the attacker to link between ciphertexts.

(*b*) Non-Usage of Master Secret Key (MK) and Derivation of Secret Keys (DS): PrivICN also uses a Key Management Server, which holds a master secret key X_{MK} for the whole domain. After proxy re-encryption, ciphertexts become as they are encrypted by using X_{MK}. The usage of a single master secret key is not necessary in PEKS_based NDN. In PrivICN, each proxy key X_{proxy} is generated as $X_{proxy} = X_{MK} - X_{client}$. One can figure out to get to know X_{proxy} and X_{client}, e.g., collusion of a proxy and a client to derive X_{MK}, which means that one can decrypt the ciphertexts of re-encryption. In PEKS_based NDN, there is no such kind of key generation and one cannot derive secret key.

(*c*) Assumption of Secure Secret-value Delivery and Management (SD): However, it is necessary to disseminate trapdoors to the network. It is assumed that the dissemination of trapdoors is done securely and trapdoors are securely stored at routers. The assumption of secret value delivery is on-demand while PrivICN needs to deliver secret value periodically and ANDāNA only disseminates during the setup.

Efficiency: The issues relating to efficiency such as Name Lookup (NL), and Fully supported cache ability on all routers in NDN network (C) are discussed follow:

(*a*) Name Lookup (NL): One of the advantages of same ciphertexts in PrivICN is that string matching can be executed for encrypted names to lookup tables. By virtue of that, their work has a faster name lookup property. On the other hand, the application of the PEKS scheme to encrypt NDN name has slower name lookup as presented in Section 5. It is obvious that PEKS-based NDN strategy becomes slower when there are several nodes between producers and consumers.

(*b*) Fully supported cache ability on all routers in NDN network (C): As shown in Figure 12, when the cache hit ratio increases, the download time decreases which means that PEKS-based NDN also provides one of the good features of NDN, i.e., in-network caching. However, ANDāNA uses encrypted encapsulation of original content, using two symmetric keys generated by the consumer for two Anatomizing Routers (AR). The ARs encrypt whole packets with the use of these keys. This fact causes the caching mechanism cannot be fully used.

Future Work: In this paper, we do not discuss about content protection and signature. To have privacy-aware content retrieval, these two features should be included. To protect content by using the same scheme is one of our future works.

(*a*) Signature: Whenever a consumer generates PEKS ciphertexts C_{1j}, they become different. The producer needs to sign the data packet before sending out, which is composed of name and data content. If a consumer generates PEKS ciphertext C_{2j} of the same plaintext name again

and creates another interest packets, it differs from the previously created interest packet. Moreover, when there is a cache hit in CS of a router, it needs to reply the data packet which is stored in the CS. The router should reply the data packet with the new name C_2 which means the router needs to change the data packet by replacing the name C_{1j} into the new name C_{2j} and to generate a new signature. Therefore, adding such kind of feature to PEKS_based NDN is also one of our future work.

(*b*) Packet overhead and table size: To reduce the downside of applying PEKS scheme to NDN, we input multiple name components to PEKS scheme to reduce packet overhead and table size. It seems that inputting the whole name to the PEKS scheme is better, but we showed that this is always not an optimal choice. There are several number of name component inputs to PEKS scheme which have lower average download time comparing to inputting the whole name as shown in Section 5. Note that the simulation result is valid only under the same situation we simulate and we are not claiming any concrete method to reduce the packet overhead and table size. It still shows that there is a room to reduce these values by appropriately selecting the number of name components fed into the PEKS scheme. To determine the optimal number of component inputs to the PEKS scheme without affecting the other operation such test process executions are not yet decided. Therefore, reducing the packet overhead and table size without affecting other operations in PEKS-based NDN is also one of our future works.

7. Conclusions

In this paper, we discussed name privacy in NDN. NDN uses human-readable name and information about content may be exposed. Protecting names prevents preserves this kind of exposure as well as exposure to consumers' interests by looking at the names. It also protects consumers' privacy. To protect the name privacy, we developed a strategy by using public key encryption with keyword search. Our proposed PEKS-based NDN strategy provides higher privacy preservation, there is no master secret key which can be used to derive private keys and PEKS_based NDN still allows in-network caching. While this work has a certain amount of latency, it is still useful for specific applications which need high protection of the name. In the future, we will improve the download time of the content with protection and signature in PEKS-based NDN Strategy.

Author Contributions: Conceptualization, K.T.K.; Methodology, K.T.K.; Software, K.T.K. and H.H.H.; Validation, K.T.K. and M.M.; Formal analysis, K.T.K., H.H.H. and M.M.; Writing—original draft preparation, K.T.K.; Writing—review and editing, K.T.K and M.M.; Visualization, K.T.K and H.H.H.; Supervision, M.M.; Construction, K.T.K. and M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: First author would like to thanks Japan International Cooperation Agency (JICA) for funding the scholarship to study at Kanazawa University.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Saxena, D.; Raychoudhury, V.; Suri, N.; Becker, C.; Cao, J. Named Data Networking: A survey. *Comput. Sci. Rev.* **2016**, *19*, 15–55. [CrossRef]
2. Zhang, L.; Afanasyev, A.; Burke, J.; Jacobson, V.; Claffy, K.C.; Crowley, P.; Papadopoulos, C.; Wang, L.; Zhang, B. Named data networking. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 66–73. [CrossRef]
3. Named Data Networking: Executive Summary. Available online: https://nameddata.net/project/execsummary/ (accessed on 4 April 2020).

4. Afanasyev, A.; Burke, J.; Refaei, T.; Wang, L.; Zhang, B.; Zhang, L. A Brief Introduction to Named Data Networking. In Proceedings of the MILCOM 2018—2018 IEEE Military Communications Conference (MILCOM), Los Angeles, CA, USA, 29–31 October 2018; pp. 1–6. [CrossRef]
5. Majed, A.-Q.; Wang, X.Y.B. Name Lookup in Named Data Networking: A Review. *Information* **2019**, *10*, 85. [CrossRef]
6. Fotiou, N.; Polyzos, G. Name-Based Security for Information-Centric Networking Architectures. *Future Internet* **2019**, *11*, 232. [CrossRef]
7. Bernardini, C.; Marchal, S.; Asghar, M.R.; Crispo, B. PrivICN: Privacy-preserving content retrieval in information-centric networking. *Comput. Netw.* **2019**, *149*, 13–28. [CrossRef]
8. Edith, N.; Börje, O.; Gene, T.; Ersin, U. Information-centric Networking and Security (Dagstuhl Seminar 16251). *Dagstuhl Rep.* **2016**, *6*, 49–61. [CrossRef]
9. DiBenedetto, S.; Gasti, P.; Tsudik, G.; Uzun, E. ANDaNA: Anonymous Named Data Networking Application. *arXiv* **2011**, arXiv:1112.2205. Available online: https://arxiv.org/pdf/1112.2205.pdf (accessed on 30 July 2020).
10. Afanasyev, A.; Shi, J.; Zhang, B.; Zhang, L.; Moiseenko, I.; Yu, Y.; Shang, W.; Li, Y.; Mastorakis, S.; Huang, Y.; et al. *NFD Developer's Guide*, 10 ed. Available online: https://named-data.net/wp-content/uploads/2016/03/ndn-0021-diff-5..6-nfd-developer-guide.pdf (accessed on 30 July 2020).
11. Zhang, Z.; Yu, Y.; Zhang, H.; Newberry, E.; Mastorakis, S.; Li, Y.; Afanasyev, A.; Zhang, L. An Overview of Security Support in Named Data Networking. *IEEE Commun. Mag.* **2018**, *56*, 62–68. [CrossRef]
12. Hamdane, B.; Serhrouchni, A.; Fadlallah, A.; Fatmi, S.G.E. Named-Data security scheme for Named Data Networking. In Proceedings of the 2012 Third International Conference on The Network of the Future (NOF), Gammarth, Tunisia, 21–23 November 2012; pp. 1–6. [CrossRef]
13. Bouk, S.H.; Ahmed, S.H.; Kim, D. Hierarchical and Hash Based Naming with Compact Trie Name Management Scheme for Vehicular Content Centric Networks. *Comput. Commun.* **2015**, *71*, 73–83. [CrossRef]
14. Bertino, E.; Nabeel, M. Securing named data networks: Challenges and theway forward. In Proceedings of the ACM Symposium on Access Control Models and Technologies, SACMAT, Indianapolis, IN, USA, 13–15 June 2018; pp. 51–59. [CrossRef]
15. Yu, Y. Public Key Management in Named Data Networking. Available online: https://pdfs.semanticscholar.org/2cd3/2e33b4683e30d945c4e87bb546d8d93c8807.pdf (accessed on 30 July 2020).
16. Yu, Y.; Afanasyev, A.; Clark, D.; claffy, k.; Jacobson, V.; Zhang, L. Schematizing Trust in Named Data Networking. In Proceedings of the 2nd ACM Conference on Information-Centric Networking, San Francisco, CA, USA, 30 September–2 October 2015. [CrossRef]
17. Signature in NDN. Available online: https://named-data.net/doc/NDN-packet-spec/0.3/signed-interest.html (accessed 30 July 2020).
18. Named Data Networking: Motivation and Details. Available online: http://nameddata.net/project/archoverview/. (accessed on 26 March 2020).
19. Song, D.X.; Wagner, D.; Perrig, A. Practical techniques for searches on encrypted data. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 14–17 May 2000; pp. 44–55. [CrossRef]
20. Curtmola, R.; Garay, J.; Kamara, S.; Ostrovsky, R. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In Proceedings of the 13th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 30 October–3 November 2006; pp. 79–88. [CrossRef]
21. Li, S.; Li, M.; Xu, H.; Zhou, X. Searchable Encryption Scheme for PersonalizedPrivacy in IoT-Based Big Data. *Sensors* **2019**, *19*, 2327. [CrossRef]
22. Boneh, D.; Di Crescenzo, G.; Ostrovsky, R.; Persiano, G. Public Key Encryption with Keyword Search. In *Advances in Cryptology–EUROCRYPT 2004*; Cachin, C.; Camenisch, J.L., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 506–522.
23. Huang, Q.; Li, H. An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks. *Inf. Sci.* **2017**, *403*, 1–14. [CrossRef]
24. Guo, X.; Chen, C.; Zhang, M.J.; Ngaboyindekwe, A.; Cao, L.C. Privacy-aware transmission scheme based on homomorphic proxy re-encryption for NDN. *Int. J. Sec. Netw.* **2018**, *13*, 58–70. [CrossRef]

25. Martinez-Julia, P.; Gomez-Skarmeta, A.F. Using identities to achieve enhanced privacy in future content delivery networks. *Comput. Electr. Eng.* **2012**, *38*, 346–355. [CrossRef]
26. Chaabane, A.; De Cristofaro, E.; Kaafar, M.A.; Uzun, E. Privacy in Content-Oriented Networking: Threats and Countermeasures. *ACM SIGCOMM Comput. Commun. Rev.* **2013**, *43*, 25–33. [CrossRef]
27. ndn-cxx: NDN C++ Library with eXperimental eXtensions. Available online: https://github.com/named-data/ndn-cxx (accessed on 21 July 2020).
28. Named Forwarding Daemon. Available online: https://github.com/named-data/NFD. (accessed on 21 July 2020).
29. The Pairing-Based Cryptography Library. Available online: https://crypto.stanford.edu/pbc/ (accessed on 3 March 2020).
30. Freeman, D.; Scott, M.; Teske, E. A Taxonomy of Pairing-Friendly Elliptic Curves. *J. Cryptol.* **2010**, *23*, 224–280. [CrossRef]
31. Named Data Networking: NDN Interest Packet Format Specification. Available online: https://named-data.net/doc/NDN-packet-spec/current/interest.html (accessed on 30 July 2020).
32. Akinwande, O. Interest Forwarding in Named Data Networking Using Reinforcement Learning. *Sensors* **2018**, *18*, 3354. [CrossRef] [PubMed]