

In-Device Proxy Re-encryption Service for Information-Centric Networking Access Control

Kalika Suksomboon, Atsushi Tagami, Anirban Basu
KDDI Research Inc., Japan
{ka-suksomboon, tagami, basu}@kddi-research.jp

Jun Kurihara
Zettant, Japan
kurihara@ieee.org

Abstract—In Information Centric Networking (ICN), the security, privacy and access control mechanisms are deployed directly in the network layer. However, the gap between security in the network and application layers creates a vulnerable space for cyber attacks from inside a device. We adopt a proxy re-encryption technique to facilitate a flexible ICN access control specific to applications. Unlike the existing work in which the proxy re-encryption is done in the network, we propose an In-device Proxy Re-Encryption Service (IPRES) for filling this security gap and for efficiency in both access control management and resource usage. We present the architecture and protocol of IPRES incorporating principles from security management specific to each application. Our evaluation shows that IPRES improves the state of the art by closing the security gap while it introduces acceptable operational and memory overheads at the device level.

Index Terms—Access control; Information Centric Networking (ICN); Proxy re-encryption; Security.

I. INTRODUCTION

In *Information-Centric Networking (ICN)*, unlike in IP networks, the publishers cannot always verify the entities requesting accesses to the content since the content may be cached in intermediate routers and it is difficult to transfer the verification logic to those routers. Several research studies (e.g., [1], [2]) have illustrated that a semi-trusted entity performing proxy re-encryption is a good candidate to handle the content-sharing-based access control because it allows the publisher to enforce the access control policy on the disseminated content through the semi-trusted proxy. On the other hand, the proxy has no right to access the content but it is responsible for transforming content that is encrypted with the *Publisher Key* to one encrypted with the authorized *Application Key (App Key)*. In addition, the revocation is easier since there is no need to revoke the disseminated content from the network. Instead, the publisher merely interacts with the proxy to update the re-encryption keys. The typical proxy re-encryption schemes for ICN [1], [2] are performed by third parties in the network and are based on the asymmetric cryptography.

The goal of this paper is to present an ICN-specific access control mechanism in which an authorized application is allowed to access the content while the same content can be cached in the device for reuse by other applications. We introduce an *In-device Proxy Re-Encryption Service (IPRES)* as an application programming interface (API) working together with an ICN forwarder and applications in a device. There

are two main advantages of using an in-device proxy over the proxy in the third party entity [2]; (1) a user can easily manage the access control policy via the proxy in his/her device, and (2) a user can reap the benefits of reusing the nearest-replicas in edge caches and the in-device cache.

The contributions of this paper are summarized as follows. We extend the original idea of an in-device proxy service from our previous preliminary work in [3] to present a new naming scheme, the re-encryption function, the key management and the proxy service that facilitate various applications to interact with the encrypted content and the proxy in the device. We evaluate the security of the IPRES and its performance in terms of the operational and memory overheads. What distinguishes our work from the existing approaches is that our proposed in-device service incorporates principles from access control management specific to an individual application and is generic enough to support proxy re-encryption constructions based on both asymmetric keys (e.g., [4], [5]) and symmetric keys (e.g., [6], [7]).

The rest of paper is structured as follows. Section II presents the threat model. Section III presents the IPRES architecture and protocol. Section IV evaluates the IPRES. We conclude the paper in Section V.

II. THREAT MODEL

Let publisher P publish a set of content objects M , where $m \in M$. The publisher encrypts content m with his/her keys $\mathcal{K}_P$. Let $\mathcal{V}(m)$ be the set of authorized applications for content m . Specifically, each application $v \in \mathcal{V}(m)$ has its key(s) (i.e., *App Key* $\mathcal{K}_v$) to access content m , while an unauthorized application or an adversary $u \notin \mathcal{V}(m)$ cannot access content m with its keys (i.e., *App Key* $\mathcal{K}_u$). We assume that an application cannot access any information across the other applications, but it may be able to intercept information from the storage of the device. Entity $u \notin \mathcal{V}(m)$ is said to be *spyware* if entity u covertly collects various information stored in a consumer's device and sends the collected information to another adversarial entity outside the device without the consumer's consent. Let *eavesdrop* be an event that an adversary $u \notin \mathcal{V}(m)$, said to be an unauthorized entity, collects data packets containing ciphertexts of m from the network layer. Hence, the access control mechanism is secure if no adversary $u \notin \mathcal{V}(m)$ can access any content $m \in M$ while *eavesdrop* occurs.

III. IPRES ARCHITECTURE DESIGN

A. IPRES architecture

Our architecture is based on Named Data Networking (NDN) [8], which is driven by two packet types, i.e., an *interest packet* and a *data packet*. An interest packet identifies the requested content name, and it is responded by a data packet whose name corresponds to that content name.

We design an IPRES architecture as shown in Fig. 1 based on the following principles. (1) No modification to the original forwarder: this principle makes the IPRES general and applicable to the other ICN architectures. (2) The re-encryption key specific to each authorized entity: this gives the publisher more granularity (than device-level access control) in granting and revoking access to the content. (3) IPRES as a service function in a device: this facilitates key handling, so that the application developer does not need to modify the application to deal with the key management specific to the desired access control policy. (4) Proxy in a device: the re-encryption function is placed in the device, so that it helps utilizing the copies of encrypted content in the caches in the network and the device.

The main functions in the IPRES architecture are presented as follows.

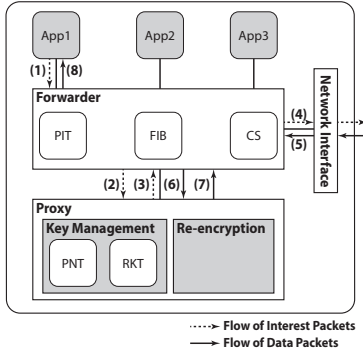


Fig. 1. IPRES architecture (numbers on arrows present the sequence of packet flows).

1) *Naming scheme*: To assist a forwarder to classify which packet requires a proxy service, the additional prefixes are added to the name in a packet. The name structure of the IPRES protocol is composed of four components: an access control (AC) prefix (/AC/), an application identifier (App ID) prefix (/AppID/), the content name (/CN/) and the segmentation (re-encryption key and nonce /RK/Nonce/). The AC prefix identifies which content requires the access control, while the App ID prefix identifies which application requests the content. Note that an App ID is unique. Two segmentations, /RK/ and /Nonce/, are added after the content name /CN/, where the term /RK/ specifies the packet for the re-encryption key, and the term /Nonce/, a unique random number (e.g., SHA256 digest), is used to prevent the interest aggregation and cache hit at intermediate routers towards the publisher. Note that CN can be any name structure not limit to the hierarchical one.

2) *Key management*: The key management in a proxy maintains two data structures: a Pending Name Table (PNT) and a Re-encryption Key Table (RKT). The PNT is almost

identical to the PIT. The PNT contains two tuples to keep track of interest packets. Each PNT entry records the content name carried in the interest packet and its App ID, while the content name and the App ID are extracted from the name prefixes in the interest packet. Similar to the PIT, the PNT allows the requests for the same name from the different applications to share the same entry by adding the new App ID to the existing one. The PNT entry is removed when the data packet satisfies a PNT entry. The RKT contains three-tuple entries, indexed by Content Name and App ID, to store the re-encryption key. Once the data packet carries the re-encryption keys, the RKT entry is created corresponding to the re-encryption keys. On the other hand, when the data packet carries the content data, it is matched to the associated re-encryption keys in the RKT. The entries in the RKT are updated when the re-encryption keys are revoked or expired.

3) *Re-encryption*: A re-encryption process is called by the proxy once the data packet arriving at the proxy requires access control. The proxy re-encrypts the payload in the data packet with the corresponding re-encryption key. Due to the space limit, we omit the detail of signaling messages for the re-encryption process. However, it can be referred as in [3].

4) *Proxy service*: The proxy service includes the following functions: (1) name modification for the appropriate route control, (2) key management considering message flows, and (3) message decryption. Algorithms 1 and 2 present the services for processing interest packets and data packets, respectively. Let d be a packet (either an interest packet or a data packet) and $N(d)$ be the name carried in packet d . When the proxy service receives an interest packet, `doInterest` procedure in Algorithms 1 is called. When the proxy service receives a data packet, `doData` procedure in Algorithms 2 is called.

Algorithm 1 Proxy service for an interest packet

```

1: procedure doInterest( $d$ )
2:   ( $app\_id, content\_name$ )  $\leftarrow$  Extract( $N(d)$ )
3:    $N(d) \leftarrow content\_name$ 
4:   if ExistInPNT( $N(d)$ ) then
5:     add  $app\_id$  into the matched PNT entry
6:     discard packet  $d$ 
7:   else
8:     create a new entry with ( $content\_name, app\_id$ )
9:     forward packet  $d$ 
10:  end if
11: end procedure

```

B. Revocation scheme

In ICN, the revoked requesters are allowed to access the old content. Hence, the lazy revocation scheme [9], in which key updating is delayed until new content under a new name is encrypted, is suitable. The cryptographic keys used to perform operations on the new content need to be changed as well.

Algorithm 2 Proxy service for a data packet

```

1: procedure DoData( $d$ )
2:   if not ExistInPNT( $N(d)$ ) then
3:     discard packet  $d$ 
4:   return
5: end if
6:  $app\_ids \leftarrow \text{SearchInPNT}(N(d))$ 
7: evict the matched PNT entry
8: for all  $app\_id$  in  $app\_ids$  do
9:    $name \leftarrow \text{Concat}(app\_id, N(d))$ 
10:  if  $d$  carries a key then
11:    insert the key into RKT
12:    generate and send Ack packet for  $name$ 
13:  else
14:    if ExistInRKT( $app\_id, N(d)$ ) then
15:       $\mathcal{RK} \leftarrow \text{SearchInRKT}(app\_id, N(d))$ 
16:       $d' \leftarrow \text{ReEncrypt}(\mathcal{RK}, d)$ 
17:       $N(d') \leftarrow name$ 
18:      forward packet  $d'$ 
19:    else
20:      discard packet  $d$ 
21:      send Notification packet for  $name$ 
22:    end if
23:  end if
24: end for
25: end procedure

```

IV. EVALUATION

In this section, we analyze the security and the performance of IPRES to validate its viability for ICN.

A. Security against eavesdroppers

To illustrate the limitation of the adversary in IPRES, we consider the following cases: (1) the adversary who can compromise an intermediate router, and (2) the adversary (i.e., malware/spyware) who can compromise the proxy and sends the eavesdropped information out of the device.

We assume that a publisher encrypts content $m \in M$ with his/her key $\mathcal{K}_P$, and only entities in set $\mathcal{V}(m)$ are allowed to access content $m \in M$. First, if adversary $u \notin \mathcal{V}(m)$ can compromise an intermediate router, the adversary is able to observe the content and the re-encryption key passing through the router. Suppose that u eavesdrops the encrypted content $c = E_{\mathcal{K}_P}(m)$ from the intermediate router. However, adversary u learns nothing about content m because $D_{\mathcal{K}_u}(c) \neq m$. Also, even if the adversary learns the re-encryption keys, applying re-encryption on any of the encrypted content simply transforms the content into another ciphertext, which the adversary cannot decrypt. Unless the publisher generates a re-encryption key for the adversary (e.g., if an authorised entity turns into an adversary), the adversary cannot learn the content of the encrypted data flowing through the routers.

Second, let adversary $u \notin \mathcal{V}(m)$ be a spyware stored in a device. If u can compromise the proxy, u is able to eavesdrop all re-encryption keys stored in the RKT and the

TABLE I
EXECUTION TIMES OF PROXY RE-ENCRYPTION SCHEMES.

Process	Time (ms)				
	IB-PRE [2]	NTRURenCrypt [4]			
	SS512	$N = 1171$		$N = 1499$	
Message size	128-bit	128-bit	256-bit	128-bit	256-bit
Key generation	1.0602	9.5449	9.6042	14.3931	14.4119
Encryption	5.6800	0.1292	0.1320	0.1485	0.1421
Re-encryption	1.0279	0.5637	0.6067	0.7513	0.7675
Decryption	4.7796	0.5145	0.5578	0.7060	0.7151

encrypted content c . Suppose that adversary u succeeds in sending the eavesdropped information to adversary $u' \notin \mathcal{V}(m)$ outside the device, adversary u' may try to access the encrypted content $c = E_{\mathcal{K}_P}(m)$ by using the re-encryption keys $\mathcal{RK}_{P \rightarrow v}$ for all $v \in \mathcal{V}(m)$, i.e., $c' = E_{\mathcal{RK}_{P \rightarrow v}}(c)$. The adversary u' learns nothing about the content m because the decryption of any of these re-encrypted ciphertexts with the key(s) of the adversary u' is $D_{\mathcal{K}_{u'}}(c') \neq m$.

The aforementioned cases imply that the IPRES is secure against eavesdroppers.

B. Performance analysis

1) *Operational overhead*: To validate the applicability of performing proxy re-encryption in a device, we first quantify the operational overhead of re-encryption function by running IB-PRE [2] and the post-quantum NTRURenCrypt [4] as examples of proxy re-encryption schemes on a Ubuntu 16.04 (Linux kernel: 3.13.0-65-generic) running on a machine with a Intel Core i7-2600 @3.4 GHz processor and 8 GB memory.

We examined the execution times of the IB-PRE (the first construction only) using the open source code¹ with pairing group SS512 — a symmetric curve group with 512-bit base field, and the system parameter $SP_u = 2048$ to achieve a security level equivalent to RSA with a 1024-bits key. We modified the code from the original NTRURenCrypt² in terms of binary data to ternary polynomial conversion. The experimental results of the execution times shown in Table I were run repeatedly 1,000 times per setting. The difference in performance times for the same NTRU settings in the table should be ignored. This is because the encryption parameters for NTRU set to N polynomial coefficients allowing for 256-bits security can encrypt a maximum of 186-bytes (for $N = 1171$) and 248-bytes (for $N = 1499$) of data (including additional the bytes for the bytes to ternary polynomial conversion). The results in Table I show that processing proxy re-encryption in a device is possible and acceptable in practice.

Second, we analyze the operational overhead of PNT and RKT lookups for the proxy service. The complexity of PNT lookup is $O(n \log n)$ by the longest prefix matching algorithm and the complexity of RKT lookup is $O(n^2)$ for the exact matching algorithm, where n is a size of PNT/RKT table.

2) *Memory overhead*: In IPRES, a proxy has to maintain two data structures, i.e., PNT and RKT. Suppose that publisher P encrypts content $m \in M$ by publisher's key $\mathcal{K}_P$, and let

¹See: <https://github.com/nikosft/IB-PRE>.

²See: <https://github.com/cygnusv/ntrurencrypt/>.

$\mathcal{RK}_{P \rightarrow v}$ be a re-encryption key specific to application $v \in \mathcal{V}(m)$. Define I_v and N_m the size of App ID of application v and the size of the name of content object m , respectively. We consider the worst case when all applications $v \in \mathcal{V}(m)$ in a device requesting the re-encryption keys for content m . The maximum size of a PNT entry is calculated by

$$\sigma_{PNT} = \sum_{v \in \mathcal{V}(m)} I_v + N_m. \quad (1)$$

Note that the size of PNT depends on the number of entries and the lifetime of each entry. Denote p_m a probability that the PNT entry of content m is pending during time τ , where $0 \leq p_m \leq 1$. Hence, the storage overhead of PNT during time τ is equal to $\sum_{m \in M} p_m \sigma_{PNT}$.

The size of RKT is proportional to the size and the number of the re-encryption keys. The size of RKT is calculated by

$$\sigma_{RKT} = \sum_{m \in M} \sum_{v \in \mathcal{V}(m)} I_v + N_m + |\mathcal{RK}_{P \rightarrow v}|, \quad (2)$$

where $|\mathcal{RK}_{P \rightarrow v}|$ is the size of re-encryption key.

Since a device has to carry encrypted keys $C(k)$ for the requested content m for at least the lifetime of content, the memory overhead of IPRES is given by

$$S_{IPRES} = \sum_{m \in M} p_m \sigma_{PNT} + \sigma_{RKT} + \sum_{m \in M} p_m (N_m + |C(k_m)|). \quad (3)$$

Benchmark scheme: We compare IPRES against IB-PRE [2]. Due to the space limit, we omit the detail of mathematical equation of IB-PRES overhead and consider only the first construction of IB-PRE³, where the re-encryption keys are stored in the KS.

Comparison: Consider the first scenario in Fig 2. The results show that the memory overhead of IPRES is scaled by p_m . The second scenario aims to show the comparison of the scalability of IPRES against IB-PRE. Suppose that there are a million subscribers, $|U| = 1e6$, in the network and the number of devices $d = w|U|$, where w represents the proportion of subscribers (or applications) per device and $0 \leq w \leq 1$. The results in Fig. 3 shows that IPRES requires lower memory storage per device than that of IB-PRE, where $w < 0.5e-3$. We also note that IPRES can be considered as a distributed proxy in a device while IB-PRE is a centralized one in a network. The results in Figs. 2 and 3 show that the memory size to maintain the PNT, RKT and re-encryption keys is either shrunk or scaled by the number of content packets and applications being requested. To support downloading 1000 items/period of time, 1 MByte memory for IPRES can handle that matter.

V. CONCLUSION AND FUTURE WORK

We proposed IPRES architecture and its protocol that not only efficiently handles the keys and access control rights for ICN in an application layer, but also efficiently reuses replicas. The IPRES architecture composes of three building

³See Section 4.2 in [2]: the second construction consumes the memory overhead more than that of the first one

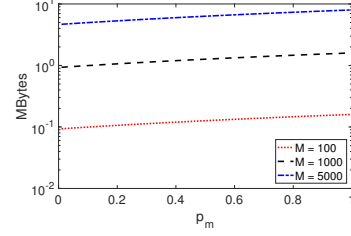


Fig. 2. Memory overhead of IPRES in a device (varies $0 < p_m \leq 1$ and $M = 100, 1000, 5000$).

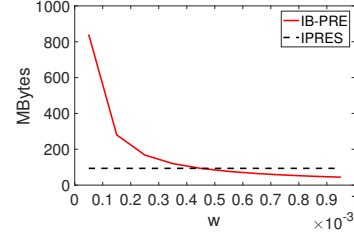


Fig. 3. Comparison of memory overhead of IPRES and IB-PRE per device ($p_m = 1$, $M = 1000$ and $5e-4 \leq w \leq 1e-3$).

boxes: 1) re-encryption key, 2) key management, and 3) proxy service. To leverage in-network and in-device caching, IPRES places the re-encryption function in the user's device. Our analysis shows that the IPRES is efficient and secure against eavesdroppers. IPRES improves the state of the art by closing the security gap and achieving a better trade-off between in-network caching and in-device operational overhead.

ACKNOWLEDGEMENTS

This work is partly supported by the Horizon2020/NICT EU-JAPAN ICN2020 project, Grant Agreement No. 723014 and Contract No. 184.

REFERENCES

- [1] C. A. Wood and E. Uzun, "Flexible end-to-end content security in CCN," in *Proc. of IEEE CCNC*, 2014, pp. 858–865.
- [2] N. Fotiou and G. C. Polyzos, "Securing content sharing over ICN," in *Proc. of ACM ICN'16*, September 2016, pp. 176–185.
- [3] K. Suksomboon, A. Tagami, A. Basu, and J. Kurihara, "IPRES : In-device proxy re-encryption service for secure ICN," in *Proc. of ACM ICN'17*, 2017, pp. 176–177.
- [4] D. Nuñez, I. Agudo, and J. Lopez, "NTRUReEncrypt: An efficient proxy re-encryption scheme based on NTRU," in *Proc. of the 10th ACM Symposium on Information, Computer and Communications Security*, 2015, pp. 179–189.
- [5] G. Ateniese, K. Benson, and S. Hohenberger, "Key-private proxy re-encryption," in *Fischlin M. (eds) Topics in Cryptology: CT-RSA 2009. CT-RSA 2009. Lecture Notes in Computer Science*. Springer, 2009, pp. 279–294.
- [6] A. Syalim, T. Nishide, and K. Sakurai, "Realizing proxy re-encryption in the symmetric world," *Informatics Engineering and Information Science*, no. 251, pp. 259–274, 2011.
- [7] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan, "Key Homomorphic PRFs and Their Applications," *Canetti R., Garay J.A. (eds) Advances in Cryptology : CRYPTO 2013. Lecture Notes in Computer Science*, vol. 8042, no. 2, pp. 410–428, 2013.
- [8] L. Zhang, V. Jacobson, S. Diego, P. Crowley, S. Louis, and L. Wang, "Named data networking," *ACM SIGCOMM CCR*, vol. 44, no. 3, pp. 66–73, 2014.
- [9] M. Backes, C. Cachin, and A. Oprea, "Secure key-updating for lazy revocation," in *Proc. of 11th ESORICS*, 2006, pp. 1–20.