IEICE TRANS. INF. & SYST., VOL.E99–D, NO.12 DECEMBER 2016

PAPER Special Section on Parallel and Distributed Computing and Networking

A Peer-to-Peer Content-Distribution Scheme Resilient to Key Leakage*

Tatsuyuki MATSUSHITA[†], **Shinji YAMANAKA**[†], ***Senior Members***, ***and*** **Fangming ZHAO**[†a)], ***Nonmember***

SUMMARY Peer-to-peer (P2P) networks have attracted increasing attention in the distribution of large-volume and frequently accessed content. In this paper, we mainly consider the problem of key leakage in secure P2P content distribution. In secure content distribution, content is encrypted so that only legitimate users can access the content. Usually, users (peers) cannot be fully trusted in a P2P network because malicious ones might leak their decryption keys. If the redistribution of decryption keys occurs, copyright holders may incur great losses caused by free riders who access content without purchasing it. To decrease the damage caused by the key leakage, the individualization of encrypted content is necessary. The individualization means that a different (set of) decryption key(s) is required for each user to access content. In this paper, we propose a P2P content distribution scheme resilient to the key leakage that achieves the individualization of encrypted content. We show the feasibility of our scheme by conducting a large-scale P2P experiment in a real network.

key words: *peer-to-peer, content distribution, key-leakage resilience, individualization, multiple encryption*

1. Introduction

1.1 Background and Motivation

Although many content distribution services are available to consumers, copyright holders are always concerned about the problem of online piracy. For instance, Stop Online Piracy Act (SOPA) and Protect IP Act (PIPA) bills in Congress argue that online piracy is a huge problem, one which costs the U.S. economy between $200 and $250 billion per year, and is responsible for the loss of 750,000 American jobs [17]. Legally, the unauthorized redistribution of copyrighted content is copyright infringement and hence a copyright holder of the content can claim damages against a user who committed the piracy [6], [23].

To protect digital content from various malicious users/attackers, the encryption-based content distribution system Fig. 1(a) is proposed and implemented so far. In this kind of system, the content provider encrypts the content in such a way that only legitimate users can access it. This access control is realized by giving a decryption key only to each legitimate user in advance and encrypting the content with the corresponding encryption key. End-user devices such as PC monitors will access unencrypted content in the end and, for instance, users can capture it using their camcorders. Therefore, some malicious users can redistribute the content itself, and in some cases that all the contents are encrypted using the same key, malicious users can just redistribute the decryption key to free riders.

In the case that the content itself is redistributed by malicious users as described above, both legal and technical measures are available and can help to mitigate adverse effects. Legally, the unauthorized redistribution of copyrighted content is copyright infringement and hence a copyright holder of the content can claim damages against a user who committed the piracy [6], [23]. Technically, audio/video fingerprinting (e.g., [15]) and digital watermarking (e.g., [7]) can be used to detect unauthorized content and then identify who redistributed the content. For example, Gao et al. [14] proposed an asymmetric fingerprint scheme which can be implemented to trace the malicious user by identifying his unique ID (as a fingerprint) from the content, then for example, a revocation process can be executed later. The existence of these measures deters users from the redistribution of content.

On the other hand, the reality is that no satisfactory technical measure against the redistribution of decryption keys has been taken (we will discuss the problem in Sect. 1.2 in detail). Besides, as far as we know, the legal status of the redistribution of decryption keys is still unclear. The leakage of decryption keys has become a problem (e.g.,[1], [29]). Therefore, we focus on the problem of the key leakage in this paper.

In recent years, P2P digital content distribution, e.g. [10], [13], have attracted increasing attention in terms of several desirable features such as adaptation, self-organization, load-balance, high availability, low cost, scalability, etc. Because of the features, it is expected that large-volume and frequently accessed content (e.g., a popular high-definition (HD) movie) can be efficiently distributed using a P2P network. In this paper, we consider the distribution of such content and discuss a secure distribution scheme which is resilient to the key-leakage threat utilizing the P2P network.

1.2 Challenging Issues

Even if several security solutions for solving the key-leakage problem have been proposed, we find that some significant characteristics are still unsatisfied in the practical

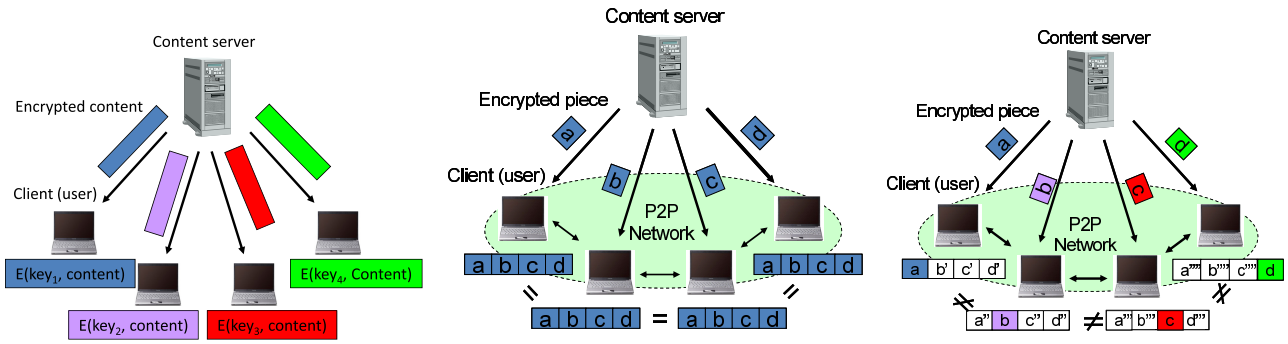
Manuscript received January 8, 2016.
Manuscript revised May 21, 2016.
Manuscript publicized August 25, 2016.

[†]The authors are with Toshiba Corporation, Kawasaki-shi, 212–8585 Japan.

*This is a full version of the paper presented at WISA 2011[21].

a) E-mail: fangming.zhao@toshiba.co.jp (Corresponding author)

DOI: 10.1587/transinf.2016PAP0018



(a) Centralized (server-client) system (b) Decentralized (P2P) system (c) Our approach

Fig. 1 Types of content distribution systems and our approach

content distribution service.

A famous one is the revocation of decryption keys (e.g., [24]). Revocation means that the content provider can make the decryption key(s) of a malicious user useless without confiscating it (them), and therefore prevent free riders from decrypting other content. However, in reality, the revocation is not an effective measure for the following pragmatic reason. Suppose that a licensing body makes a DRM specification and a manufacturing company produces DVD players that conform to the DRM specification after both of them make a license agreement, which specifies conditions covering production and sales of the DVD players. In the agreement, it is usually specified that, in the case of the key leakage from the DVD player, the licensing body can immediately revoke the leaked key. But in reality the deadline when the revocation will be imposed is waived for some period of time (e.g., one month). This grace period is necessary for both of them. The licensing body has to carefully verify that the leaked key is that of the DVD player. If the licensing body falsely incriminates an innocent consumer, it may give rise to a serious problem such as a lawsuit. The licensing body also needs time to inquire about the cause of the leakage to the manufacturer, in order to prevent similar trouble with other products of the company. The manufacturer needs time to find out the cause and get rid of it. Therefore, a grace period is inevitable from a practical point of view. Since free riders can keep using the leaked key during the grace period, local piracy can escalate into global piracy. As long as the legal status of the redistribution of decryption keys is unclear, the copyright holder might not be able to claim damages against the piracy. Hence, we cannot take the technical measure in which any free rider can use leaked decryption keys, even though we can identify a user who leaked decryption keys using e.g., a fingerprint scheme such as [14].

Another one is the individualization of encrypted content. We mean by this terminology that a different (set of) decryption key(s) is required for each user to decrypt an encrypted version of the downloaded content. As illustrated in Fig. 1(a), a content server sends $E(key_i, content)$ to each client in a one-to-one manner, where $E(key_i, content)$ denotes a ciphertext in which content, *content*, is encrypted with a key, key_i, using a symmetric-key encryption algorithm, E. Each user is given a distinct key, key_i, and therefore can access the content by decrypting the ciphertext. It is clear that the individualization of encrypted content is achieved in this system. However, the content server has to incur a heavy transmission burden proportional to the number of users. This scheme is still unsuitable for the large-scale distribution of HD content. Consequently, a technical solution is desirable where the leaked key is useless to free riders and the burden of the content server is reduced.

1.3 Our Contributions

In this paper, compared with the problems of the centralized system Fig. 1(a), we consider realizing individualization of encrypted content to a large scale P2P HD content distribution system to deal with the key-leakage problem. Content is divided into fragments called pieces and a user receives the encrypted pieces not only from the content server but also from other users. Since the transmission burden of the content server is shifted to the users, the problem described above is solved in this type of system. To endow a P2P content distribution scheme with both the individualization of encrypted content and the key-leakage resilience, we also propose a scheme in which each piece is multiply encrypted when transmitted via each user, as illustrated in Fig. 1(c). Sets of decryption keys for the same piece are designed to be different from one another. Therefore, the influence of the leakage of the decryption keys is minimized.

We analyze and point out that a previous scheme [34] achieves the individualization of encrypted content, but it has no key-leakage resilience. Our proposed P2P content distribution scheme achieves both the individualization of encrypted content and the key-leakage resilience. The proposed light-weight piece encryption scheme reduces the decryption cost for a user client. Finally, to show the feasibility of our scheme, we implement our scheme in a large-scale network to analyze both the communication cost and the computation cost. The experiment result shows our scheme are practical in a real P2P network.

2. System Model and Requirements

We first introduce the entities (See Fig. 2) involved in our scheme, and then we identify the threat model and security requirements.

2.1 Overall System

- *A content provider* provides content for a content server. For simplicity, we suppose that it also works as a content distribution service provider.
- *A content server* (a.k.a. an initial seeder) receives content from the content provider. It divides the content into multiple pieces and encrypts them. Then, it transmits the encrypted pieces to users. For simplicity, we suppose that it also works as a BitTorrent tracker [30], [31].
- *A personal-information management server* (a *PM* server for short) receives and manages the personal information of all users. It issues a registration certificate stating that a user has registered his personal information with it, to the user. It is managed by, e.g., the content provider, which is independent of a content distribution system such as BitTorrent [30].
- *A key-management server* (a *KM* server for short) holds all keys assigned to the content server and users. It gives each user a unique key, which we call a user key. Each user uses it for generating an encryption key for a piece. It also computes decryption keys for the encrypted pieces and gives them to users on request.
- *Users* subscribe to the content distribution service by registering to the *PM* server and the *KM* server. They upload/download encrypted pieces in a P2P network.

A general use case of our scheme is described as follows.

1. At first, a user registers his personal information to the *PM* server for subscribing the content service. The *PM* server issues a registration certificate which includes a pair of secret/public keys and a digital signature. A third party can verify the certificate from its signature using the public key.
2. Then, the user can acquire a content's metadata, i.e. from the homepage of this service. The metadata involves detail information of the content and the P2P network, the IP addresses of the content server and the *KM* server.
3. The user then registers to the *KM* server using his certificate. If the certificate is verified, the *KM* server generates a unique user ID and a key for encrypting content pieces for that user. A user may skip this *KM* registration process once the user has registered before.
4. The *KM* server has a unique pair of signing key and verification key. The verification key is used to verify a signature made by the *KM* server using the singing key. This verification key is included in a public-key certificate which must be distributed to all users. (Details are described in Sect. 3.2.)
5. Using the metadata, a user downloads content from the content server and other online users. Each piece is encrypted multiply in the transmission route. A one-time random number is used to achieve the individualization of encrypted content.
6. After all encrypted pieces are downloaded, the user has to obtain their decryption keys ($\mathcal{K}_i$ for user i) from the *KM* server by sending the following information: user ID, content ID, route information and random numbers. The computation of $\mathcal{K}_i$ is performed at the *KM* server. (Details are described in Sect. 3.1)

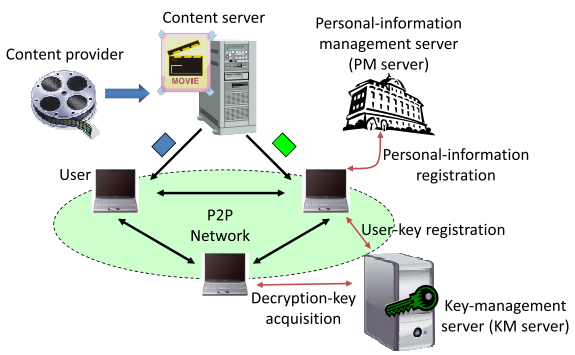


Fig. 2 Overall system

2.2 Threat Model and Security Requirements

We assume that users are not trusted but the other entities are trusted. We consider the following attacks in this paper.

- A malicious user might leak his decryption keys, which are received from the KM server, to free riders.
- A malicious user might alter a received encrypted piece and upload the altered piece to other users.

The latter is called a pollution attack. If it happens, the content quality is degraded owing to altered pieces.

The requirements for secure P2P content distribution in our system are as follows:

- Personal information, a user key, and decryption keys should be protected from eavesdroppers who intercept communications (1) between the PM server and a user and/or (2) between the KM server and a user.[†]
- Content should be protected from eavesdroppers who monitor communications among users.
- Suppose that a malicious user leaks his decryption keys. In this case, the leaked decryption keys should not be usable for free riding.
- Colluding users cannot compute any decryption keys other than those they have.
- A user should be able to detect a pollution attack.

[†]We suppose that the communications are protected by e.g., TLS [32].

3. Proposed Scheme

First, we show the piece-encryption scheme. Second, we present an integrity-verification scheme that copes with a pollution attack.

3.1 Piece Encryption

We explain how a piece is multiply encrypted during the piece downloading phase (see Fig. 3). We define k_A, k_B, k_C, k_I as the user keys of users, A, B, C, and a content server, I, respectively.

First, I generates a random number, r_I, and computes an encryption key, W_I, as follows:

$$W_I = H(k_I \| r_I),$$

where $H, \|$ denote a hash function and concatenation, respectively. In this paper, we suppose that (1) SHA-256 [11] is used as H, (2) the length of W_i is 128 bits, and (3) the output of H is appropriately truncated, i.e., the lengths of k_i, r_i are 128 bits each, and W_i is the lower 128 bits of $H(k_i \| r_i)$. Note that a different random number, r_i, is used each time an entity, i, (the content server or a user) encrypts a piece.

Next, I encrypts a piece (plaintext), *piece*, and then sends a 3-tuple of data, (X_1, X_2, X_3), to A.

$$\begin{aligned}(EP_1 =)X_1 &= Enc(W_I, Tr(=1), piece),\\ X_2 &= ID_I,\\ X_3 &= r_I,\end{aligned}$$

where ID_i, Tr denote ID of an entity, i, and hop counter, respectively. In other words, Tr means that the piece has traversed Tr nodes. The value of Tr can be computed by counting how many IDs are included in X_2. We explain *Enc* below. Note that I encrypts each of the pieces with a different encryption key.

Definition 1 (*Enc* algorithm):
Input: W an encryption key, Tr the number of times piece transfer occurs, Y a piece to be encrypted. Y will be divided into multiple sub-pieces, Y_i, where $1 \le i \le t$ (t means the number of sub-pieces in Y).
Output: an encrypted data, Y'.

- If $Tr = 1$, then compute Y' as follows:

$$Y' = E(W, Y),$$

where E denotes that Y is encrypted with W. We consider AES-CBC is E in this paper.
- Else if $2 \le Tr \le t+1$, then compute Y' as follows:

$$\begin{aligned}Y' &= Y_1 \| Y_2 \| \cdots \| Y_{Tr-2} \| E(W, Y_{Tr-1}) \|\\ &\quad Y_{Tr-1} \oplus Y_{Tr} \| \cdots \| Y_{Tr-1} \oplus Y_t,\end{aligned}$$

where $\oplus$ denotes XOR.
- Else if $Tr \ge t+2$, then compute Y' as follows:

$$\begin{aligned}Y' &= Y_1 \| Y_2 \| \cdots \| Y_{z-1} \| E(W, Y_z) \| Y_{z+1} \| \cdots \| Y_t,\\ z &= \begin{cases} Tr - 1 \bmod t & (Tr \bmod t \ne 1)\\ t & (Tr \bmod t = 1)\end{cases}.\end{aligned}$$

Based on the Definition 1 above, when receiving (X_1, X_2, X_3), A increments Tr by one, generates a random number, r_A, and computes (X'_1, X'_2, X'_3) as follows:

$$\begin{aligned}(EP_2 =)X'_1 &= Enc(W_A, 2, X_1),\\ X'_2 &= X_2 \| ID_A = ID_I \| ID_A,\\ X'_3 &= X_3 \| r_A = r_I \| r_A,\\ W_A &= H(k_A \| r_A).\end{aligned}$$

Then, A sends (X'_1, X'_2, X'_3) to B. As mentioned above, a distinct random number is generated each time A computes the encryption key, W_A. Therefore, EP_3, which is sent to C, is different from EP_2. To upload EP_3 to C, A computes and sends (X''_1, X''_2, X''_3) to C.

$$\begin{aligned}(EP_3 =)X''_1 &= Enc(W'_A, 2, X_1),\\ X''_2 &= X_2 \| ID_A = ID_I \| ID_A,\\ X''_3 &= X_3 \| r'_A = r_I \| r'_A,\\ W'_A &= H(k_A \| r'_A),\end{aligned}$$

where $r_A \ne r'_A$.

Suppose that a piece to be encrypted is $Y = Y_1 \| Y_2 \| \cdots \| Y_t$ and the size of each sub-piece, Y_i, is the same. In our experiments, we set $t = 8$. First, the content server encrypts a piece in a conventional way (when $Tr = 1$). Secondly, a user encrypts Y_{Tr-1} and performs XOR of each subsequent sub-piece and Y_{Tr-1} when $Tr \le t+1$. Thanks to the proposed encryption algorithm in Definition 1, both the individualization of encrypted content and the key-leakage resilience are achieved. The reason for introducing the partial encryption for $Tr \ge 2$ is that we can reduce the decryption cost for a user. The reason for introducing the XOR operation is that, for example, we can prevent decrypting of EP_2 if the only leaked key is K_I. Lastly, a user just encrypts a sub-piece when $Tr \ge t+2$, since the above concern is reduced.

For ease of understanding, we give a concrete example of the piece encryption in the case that the encrypted piece hops among I, A, and B as illustrated in Fig. 3. First, I computes EP_1 as follows:

$$EP_1 = Enc(W_I, 1, piece),$$

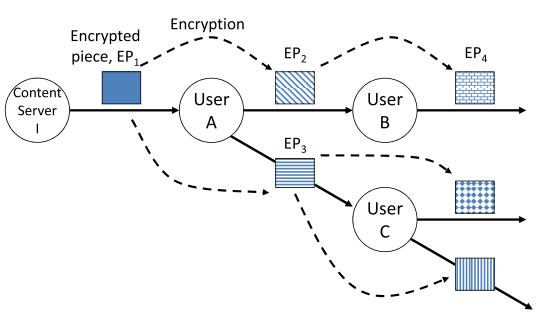


Fig. 3 Sketch of the flow of an encrypted piece

$$= E(W_I, piece).$$

Suppose that $EP_1 = EP_{1,1}||EP_{1,2}||\cdots||EP_{1,t}$. Secondly, A computes EP_2 as follows:

$$EP_2 = E(W_A, EP_{1,1})||EP_{1,1} \oplus EP_{1,2}||\cdots||EP_{1,1} \oplus EP_{1,t}.$$

Note that, for another transmit route to C, A generates EP_3 using another key, W'_A.

$$EP_3 = E(W'_A, EP_{1,1})||EP_{1,1} \oplus EP_{1,2}||\cdots||EP_{1,1} \oplus EP_{1,t}.$$

It is intractable to discover EP_1 even if B and C collude.

Thirdly, B computes EP_4 as follows:

$$\begin{aligned} EP_4 &= E(W_A, EP_{1,1})||E(W_B, EP_{1,1} \oplus EP_{1,2}) \\ &\quad ||(EP_{1,1} \oplus EP_{1,2}) \oplus EP_{1,1} \oplus EP_{1,3}||\cdots \\ &\quad ||(EP_{1,1} \oplus EP_{1,2}) \oplus EP_{1,1} \oplus EP_{1,t} \\ &= E(W_A, EP_{1,1})||E(W_B, EP_{1,1} \oplus EP_{1,2}) \\ &\quad ||EP_{1,2} \oplus EP_{1,3}||\cdots||EP_{1,2} \oplus EP_{1,t}. \end{aligned}$$

We present a method of protecting X_2 and X_3 in Sect. 3.2.

We explain the key management mechanism of the KM server. After receiving the key request from the user, the KM server authenticates the user and then checks if the total number of pieces is correct and also confirms the validity of the following items for each piece: $Tr \le Tr_{\max}$, user IDs included in route information, and the uniqueness of each user ID included in route information. After confirming all of these items, the KM server computes $\mathcal{K}_i$ for the key request, appends its digital signature to $\mathcal{K}_i$, and sends them to the user, i.

If B wants to get the decryption keys for EP_2, B sends (ID_B, CID, X'_2, X'_3) to the KM server, where CID denotes a content ID. When receiving (ID_B, CID, X'_2, X'_3), the KM server computes the decryption key as follows:

$$\begin{aligned} W_I &= H(k_I||r_I), \\ W_A &= H(k_A||r_A). \end{aligned}$$

where the KM server knows all of the user keys. Likewise, B makes a request for a set of the decryption keys, $\mathcal{K}_B$, for all of the encrypted pieces of the content, and obtains $\mathcal{K}_B$. Then B verifies the signature and decrypts the encrypted content with $\mathcal{K}_B$. It is obvious that a user can decrypt the encrypted pieces (and therefore play back the content) if all of the W_i's used in encrypting the pieces are obtained.

3.2 Integrity Verification

An attacker might alter encrypted pieces to disrupt the P2P content distribution. In order to detect a pollution attack, the integrity verification of (encrypted) pieces is necessary. We show how the integrity verification is integrated into the proposed piece-encryption scheme. Let $Sig(sk, M)$ denote a signature in which a message, M, is signed with a signing key, sk, using e.g., ECDSA [4]. Let $sk_i, vk_i, cert_i$ be a signing key, a verification key corresponding to sk_i, and a public-key certificate of an entity, i, and ID_i, vk_i be included in $cert_i$. We reuse the same notations used in Sect. 3.1 and omit describing the same part of the protocol as in Sect. 3.1.

The content server, I, sends (X_1, X_2, X_3) to the user, A.

$$\begin{aligned} X_2 &= cert_I, \\ X_3 &= r_I||Sig(sk_I, X_1||X_2||r_I). \end{aligned}$$

When receiving (X_1, X_2, X_3), A verifies (1) $cert_I$ with a (public) verification key of the KM server, vk_{KM}, which can be used for verifying all certificates $cert_i$ of all users in the transmission route, and (2) $Sig(sk_I, X_1||X_2||r_I)$ with vk_I included in $cert_I$. If both verifications succeed, A transmits (X'_1, X'_2, X'_3) to the user, B.

$$\begin{aligned} X'_2 &= X_2||cert_A = cert_I||cert_A, \\ X'_3 &= X_3||r_A||Sig(sk_A, X'_1||X'_2||X_3||r_A). \end{aligned}$$

Otherwise, A discards (X_1, X_2, X_3).

Likewise, when a user, U, receives a 3-tuple of data, (Z_1, Z_2, Z_3), U verifies both the public-key certificate and the signature that have been appended most recently. If both verifications succeed, U stores (Z_1, Z_2, Z_3). Otherwise, U discards it. If U transfers the encrypted piece to another user, V, then U encrypts Z_1 using Enc, appends $cert_U$ to Z_2, generates and appends a signature of the entire data to Z_3, and then sends the resulting data, (Z'_1, Z'_2, Z'_3), to V.

Since $cert_U$, which includes vk_U, is signed by the KM server, V can verify, using vk_{KM} and vk_U, that (1) the received data was generated by the legitimate user, U, and (2) it has not been altered since the time it was generated by U. Note that V does not need to know vk_U before he receives (Z'_1, Z'_2, Z'_3), since V does not have to verify that he is communicating with U but just needs to verify that (1) and (2) are satisfied. This is suitable for the P2P network, in which a user might not know in advance who will communicate with him.

The above method is sufficient to detect a pollution attack as long as U behaves honestly. But we should also consider a case in which U is malicious. Suppose that U is a legitimate but malicious user. The malicious user, U, can choose any data as Z'_1 as long as U follows the above protocol in computing Z'_2 and Z'_3. Such (Z'_1, Z'_2, Z'_3) can pass the above verification for V. To cope with this attack, we adopt the following method.

Suppose that a user, V, receives (1) an encrypted piece, c_i, and (2) a public-key certificate, $cert_{U_j}$, and a signature, Sig_{U_j}, that have been appended to c_i most recently. We denote by Sig_{U_j} a signature generated by a user, U_j, according to the above protocol. After downloading $c_1, \ldots, c_N$, V executes the following procedures. We analyze the efficiency of this scheme in Sect. 4.2.

1. Receive a set of decryption keys, $\mathcal{K}_V$, for the content from the KM server.
2. Check if the decryption keys are valid by verifying with vk_{KM} a signature appended to $\mathcal{K}_V$ by the KM server.
3. Repeat the following procedures for $1 \le i \le N$.

   3-a Decrypt c_i with the corresponding keys in $\mathcal{K}_V$, and

obtain a (potentially altered) piece, m'_i.

3-b (Let m_i denote a correct piece.) Compute $H(m'_i)$ and check if $H(m'_i)$ is equal to $H(m_i)$ obtained from the content server, in the same way as in the original BitTorrent system. If $H(m'_i) = H(m_i)$, then determine that the piece is not altered. Otherwise, discard c_i.

4. Performance Analysis

In this section, we present an experimental evaluation of our scheme. Regarding a P2P platform, our implementation is based on BitTorrent. We show that (1) content is distributed efficiently by conducting a large-scale content distribution experiment in Sect. 4.1, (2) the encryption and decryption costs for a user are acceptable in Sect. 4.2, (3) the burden for a key-management server to distribute decryption keys to users is also acceptable in Sect. 4.3, and (4) the key-leakage resilience is achieved in Sect. 4.4.

4.1 Performance Evaluation of P2P Content Distribution

Our P2P content distribution experiments are conducted on, StarBED [22], [27], a large-scale network testbed.

4.1.1 StarBED – Experimental Environment

StarBED is the large-scale network testbed of the National Institute of Information and Communications Technology Hokuriku Research Center in Ishikawa, Japan. This experiment environment is a cluster-based testbed.

We use actual nodes (PCs) for our content distribution experiments. Each node runs on the Intel Pentium 4 processor at 3.2 GHz with 2 GB of RAM. Each node also has two network devices. As depicted in Fig. 4, one network device is connected to a management network and the other network device is connected to an experiment network. Each node is controlled by a management node in the management network and content distribution is done in the experiment network. One node acts as a content server, which has

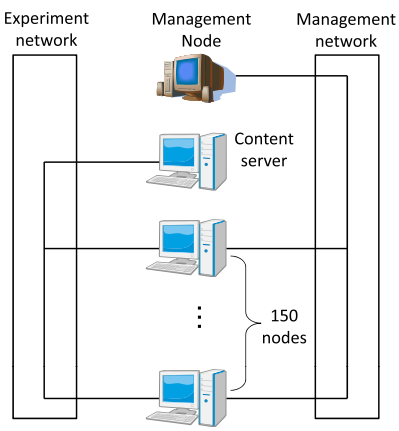


Fig. 4 StarBED experimental environment

all of the pieces of content. The other 150 nodes behave as clients. We also use LibTorrent C++ library [19] for implementing our client software. We use the original BitTorrent tracker [31] and implement it to the content server.

4.1.2 Parameters of Experiments

We describe six parameters of the experiments: the number of nodes, frequency of re-entry to the network, the maximum number of times piece transfer occurs, waiting time after downloading is finished, the number of clients per node, and the size of data. We illustrate how our scheme works when we change each parameter.

The number of nodes. In our experiments, we run up to 150 nodes. The management node controls all of the nodes in the management network. Note that we run plural clients on one node in some experiments. Clients upload/download data pieces to/from one another.

Frequency of re-entry to the network. The frequency of re-entry to the network denotes the number of times one client node enters the experiment network in one experiment. If this parameter is set to 50, for instance, all clients do the following steps 50 times:

1. Generate a new client ID and a user key,
2. Enter the experiment network,
3. Upload/download pieces,
4. Finish downloading (and continue uploading during pre-determined waiting time),
5. Escape from the experiment network,
6. Save log data and clean up downloaded data.

A re-entered client is assigned a new different ID, and so this client acts as a newly joined client. In order to avoid a case in which most of the clients enter the network at the same time, we introduce a random delay, which is at most 10 [sec], between Step 1 and Step 2.

The configured limitation of piece transfer times. As explained in Sect. 3.1, a piece is multiply encrypted as it is transferred through clients. The number of times a piece is encrypted increases as the number of times a piece is transferred, Tr, grows. In the case of the excessive increase in the number of times of encryption, the following three problems arise: (1) The KM server has to incur the heavier load of generating the decryption keys for the encrypted pieces. (2) The decryption cost for a client increases. (3) (Recall that (Z_1, Z_2, Z_3) denotes a 3-tuple of data transferred among users.) The size of $Z_2 \| Z_3$ increases as Tr grows, though the size of Z_1, which stays constant, is much larger than that of $Z_2 \| Z_3$ in most cases. In order to avoid these problems, $Tr_{\max}$, which is defined as the limitation number of Tr, needs to be set. At the same time, we should not set $Tr_{\max}$ to a too small value. If the configured $Tr_{\max}$ is too small, most of the clients try to connect to the content server, and then the downloading time grows owing to the substantially increased congestion.

Waiting time after downloading is finished. This waiting time is defined between Step 4 and Step 5. For ex-

ample, when this parameter is set to 0, a client leaves the network immediately after downloading the data is completed, even if other clients are downloading encrypted pieces from this client. A longer waiting time means that the client contributes to more other clients by uploading the encrypted pieces to them.

The number of clients per node. Since a node is a PC, we can run one or more clients on it. In some experiments, we run five clients on one node. For example, in Exp. 4-2 and Exp. 4-4, the total number of clients in the experiment is 37,500, whereas it is 7,500 in Exp. 4-1 and Exp. 4-3.

The size of data. We generate two random data for our distribution experiment, 195-MB data and 1954-MB data. In most of the experiments, we use 195-MB data. Throughout the paper, we set the size of each piece to 1 MB.

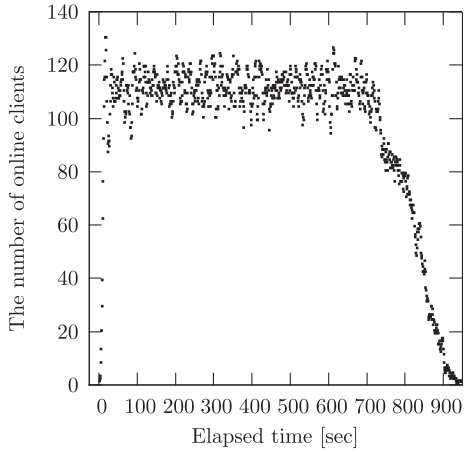


Fig. 5 The number of online clients in Exp. 3-3

4.1.3 Experimental Results

All parameters and the results of the experiments are shown in Appendix A. We explain our experimental results from three viewpoints: contribution analysis, the number of times piece transfer occurs, and downloading time.

(1) Contribution Analysis.

In a P2P system, data pieces are provided not only from the content server but also from participant nodes (clients). Therefore, a P2P system can distribute data more efficiently than a server-client system. We analyze the contribution of each client to our system, such as how many pieces are uploaded from clients. i.e. In the table of Appendix A, "Ratio of uploading to downloading" means the average ratio (x/y) of the number of pieces that a node transferred to other nodes, x, to that of pieces that the node received, y. In Exp. 1-1, when a small number of client is online, nearly (or, less than) half of the pieces are uploaded from clients. In Exp. 3-1, 3-2, 3-3, 3-4 where $Tr_{\max}$ is set higher, the "Ratio of uploading to downloading" is also growing. This means much more pieces are uploaded from clients. From these results, it follows that the number of online clients and a value of $Tr_{\max}$ should not be set too small. Therefore, the cost of the content server is reduced in comparison. (However, some exceptions also exist in our experiment, e.g. in Exp. 1-5 and Exp. 2-1, where a higher $Tr_{\max}$ does not achieves higher "Ratio of uploading to downloading". We will discuss and explain the reason later in paragraph *(3) The Measured Maximum of Piece Transfer*)

In Fig. 5, we present a tendency analysis of the number of online clients using an example of Exp. 3-3. The number suddenly rises to around 110 at first, then it is generally constant in the middle, and at last it gradually drops to zero. Since the number of nodes is 150 and the number of clients per node is 1, about 40 clients are offline constantly. This behavior indicates that most (about 73%) of the clients work (or, contribute) during Step 2, Step 3 and Step 4.

(2) The Average Number of Times Piece Transfer Occurs.

Our results shows the transmission burden of the content server is directly related to this parameter, Tr. In Exp. 1-1, 1-5, 2-1, and 2-3, where the number of online clients is very small (e.g. Tr=1.7), because the encrypted piece is transferred at least once by the content server, over half of the transfers have to be performed by the content server herself and the load of the content server is thought to be very high, similar to that of a server-client system. When the number of the online clients is relatively large (e.g., Exp. 1-2, 1-3, and 1-4), the average number of Tr also increases. This means instead of downloading pieces directly from the content server, because a lot of clients downloaded pieces from other user clients, pieces transfer occurs more frequently between user clients. According to the experiment results analysis above, we can infer that a higher average piece transfer time represents a lower ratio of piece transfer from the server, then we say the transmission burden of the content server is shifted to the user clients in the P2P network.

To analyze the influence from the $Tr_{\max}$, we set $Tr_{\max}$ to a different smaller value (i.e. $5 \sim 30$) in Exp. 3-1,..., 3-4. The result shows Tr settles at around 60% of $Tr_{\max}$. When $Tr_{\max}$ is set to an extremely large number, e.g., 254 in Exp. 4-2, Tr falls down. The reason can be considered that lots of node can get pieces directly from its neighborhoods nearby. This trend is evident in experiments for both data sizes, 195 MB and 1954 MB. Therefore, this parameter also impacts the decryption cost. (We discuss the decryption cost in Sect. 4.2)

(3) The Measured Maximum of Piece Transfer.

This result shows the max hop count we have measured in each experiment. We can compare this value, Tr_{meas}, with another result, $Tr_{\max}$, to show whether the configured limitation really worked in each experiment. Such a comparison is important in a tendency analysis of the average number of Tr among several experiments. For example, in Exp. 1-5 and Exp. 2-1, the tendency of "Ratio of uploading to downloading" does not follow our analysis above, "a higher $Tr_{\max}$ achieves higher the Ratio of uploading to downloading". The reason can be explained as that the $Tr_{\max}$ configured

in both Exp. 1-5 and Exp. 2-1 did not really work because a small number of online clients is not able to transmit the pieces too many times, and then, $Tr_{\text{meas}} < Tr_{\max}$ in both experiments . In such a case, since the "Ave. no. of transfers" of Tr in Exp. 1-5 and Exp. 2-1 are *1.8* and *1.7*, we could explain that a higher average hop number (or, average Tr) also achieves higher the Ratio of uploading to downloading" when the setting of $Tr_{\max}$ does not work. This case also implies that a higher "average hop number" means a bigger chance for a user to upload pieces to others.

(4) Downloading Time.

We analyze the factors that impact the average downloading time. Our experiment results show the time is mainly influenced by the following three factors/parameters: Limitation of pieces transmission ($Tr_{\max}$), Client PC resources (CPU, memory, etc.) and Waiting time.

Compare the Exp. 2-2 where $Tr_{\max} = 254$ and Exp. 2-5, 3-1,..., 3-4, where $Tr_{\max}$ are small, the downloading time increases as we reduce the $Tr_{\max}$. A smaller $Tr_{\max}$ also influences the decryption cost for users. The average downloading time in Exp. 4-1 ($Tr_{\max} = 254$) and Exp. 4-3 ($Tr_{\max} = 20$) is longer than that in Exp. 2-2 ($Tr_{\max} = 254$) and Exp. 3-3 ($Tr_{\max} = 20$), though the total number of clients is the same. This is because PC resources (CPU, memory, etc.) are shared by 5 clients in one node. The process of reading from and writing to a hard disk of such a node increase the time. In Exp. 3-3 (no waiting time) and Exp. 6-6 (where 20-second waiting time is introduced), the average downloading time increases from 8.7 to 11.1 [sec]. Besides, Exp. 5-2 and Exp. 5-4 also show that average downloading time is reduced by introducing waiting time. From these results, it follows that the downloading process is speeded up if users who finished downloading contribute to other ones.

4.2 Performance Analysis of the User Client

To show the feasibility of our scheme on the client side, we give an analysis of the computation cost (i.e. encryption, decryption) and communication cost (i.e. keys transmission).

As described in Sect. 3.2, a user, U, performs two signature verifications, one piece encryption using Enc, and one signature generation in re-encrypting an encrypted piece. We estimate the time needed from the time U received (Z_1, Z_2, Z_3) until the completion of generating (Z'_1, Z'_2, Z'_3). We refer to the following benchmarks on the Intel Core 2 processor at 1.83 GHz in [8]: 2.88 [msec] for a 256-bit ECDSA signing operation, 8.53 [msec] for a 256-bit ECDSA verifying operation, and 109 [MB/sec] for a 128-bit AES-CBC operation. From these benchmarks, it follows that the estimated time is 21.1 ($= 8.53 \times 2 + (1/8)/0.109 + 2.88$) [msec], where U encrypts $1/t$ ($= 1/8$) of the encrypted piece regardless of a value of Tr. Since the total downloading time (shown in Appendix A) overshadows the piece re-encrypting time, the computation cost is acceptable.

One might be concerned about the computation cost for the user client, since it seems to explode as the number of times piece transfer occurs increases. We numerically show that this is not true if we set $Tr_{\max}$ to an appropriate value. For $Tr_{\max} = 20$, we measured the time required for the repeated procedures (3-a and 3-b in Sect. 3.2) using our decryption software respectively running on: (1). Intel Pentium 4 processor at 2.4 GHz with 1 GB of RAM; on average it takes 174.2 [msec] per piece; (2). Intel Pentium D processor at 3.4 GHz with 2 GB of RAM; on average it takes 63.6 [msec] per piece. Both results are measured in the worst case where each piece is encrypted 20 times. These results imply that it is possible to play back the content while decrypting the encrypted pieces (depending on the cost of decoding e.g., MPEG audio and video data).

Regarding the communication cost for receiving the decryption keys ($|\mathcal{K}_U|$) from the KM server, we also calculate the size of decryption keys in the worst case. For a 25,000 MB content, when the size of each piece is 1 MB, $Tr_{\max} = 20$, and the size of each decryption key, $|key|$=128 bits, the size: $|\mathcal{K}_U| < 7.63$ MB. It follows that the cost for receiving decryption keys stays acceptable if we set $Tr_{\max}$ to an appropriate value.

4.3 Performance Evaluation of the Key Management Mechanism

In our scheme, because the computational burden (or cost) is shifted to the KM server for managing decryption keys, one might worry about the costs of the KM server in the system. We implement the key management mechanism to a KM server using Java. Our application, which runs on the Intel Xeon processor X3320 at 2.50 GHz with 4 GB of RAM, performs the key distribution service. To simulate multiple virtual nodes that simultaneously access the KM server to get their decryption keys, we use a load generator, which is based on WebLOAD [33] and runs on the Intel Core 2 Duo processor P8400 at 2.26 GHz with 1 GB of RAM.

As shown in Fig. 6, a user first sends a key request to the KM server. We use ECDSA as the signature algorithm. When the KM server receives the key request, it first authenticates the user and then it verifies the signature of the ticket. If both processes are successful, it generates a set of decryp-

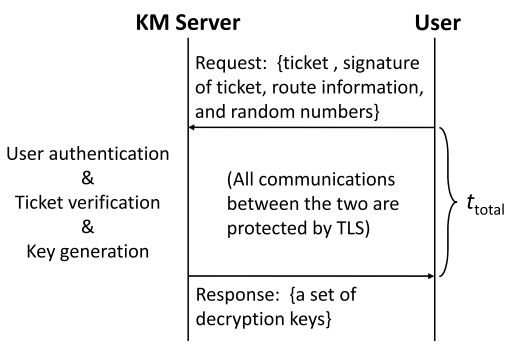


Fig. 6 Key distribution protocol. We measure processing time, t_{total}, in the experiment.

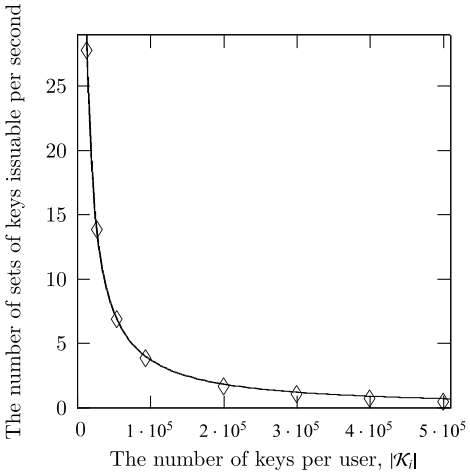


Fig. 7 Key distribution capacity of the KM server

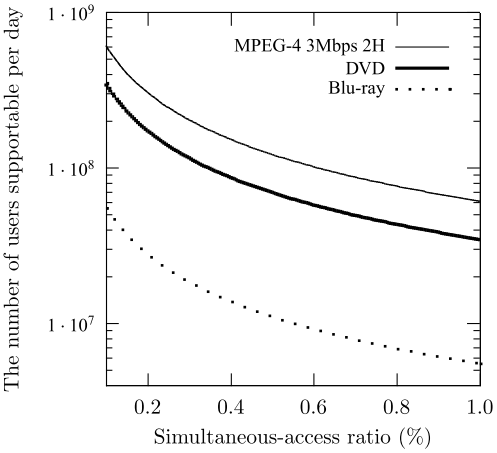


Fig. 8 Service capacity

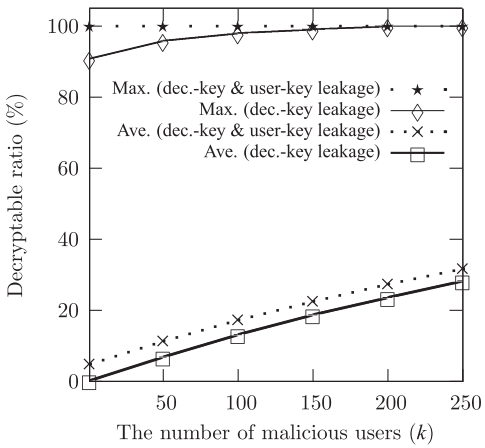


Fig. 9 Key-leakage resilience in Exp. 3-3

tion keys and sends them to the user. We choose eight kinds of key requests whose sizes are 675, 1,350, 2,700, 4,700, 10,000, 15,000, 20,000, and 25,000 MB [†]. For $Tr = 20$, we measure processing time, t_{total}, for completing each key request. Note that t_{total} includes the overhead due to TLS. Figure 7 shows the key distribution capability of the KM server. Let S be the number of users that the KM server can support per day, i.e., service capacity of the KM server. Let R be the simultaneous-access ratio, which means R % of users simultaneously access the KM server to receive their keys. From the results shown in Fig. 7, we can estimate $S = 24 \times 60^2 \times t_{\text{total}}^{-1}/(R/100)$ on the assumption that accesses from users are dispersed throughout a day. In Fig. 8, we show the service capacity when $0.1 \le R \le 1$. For examples, when $R = 1.0$, the KM server can support 61.3, 34.8, and 5.5 million users per day for MPEG-4 content, DVD content, and Blu-ray Disc content, respectively. Moreover, we can achieve much more service capacity using several KM servers in parallel. Consequently, these results show that the key management mechanism in our scheme is efficient and practical for large-scale content distribution.

4.4 Key-Leakage Resilience Analysis

We explain which user can get a free ride in our scheme. First, we consider a case in which there is one malicious user and focus on a single piece. Suppose that, in Fig. 3, B leaks a set of his decryption keys, $\mathcal{K}_B$. Then A can decrypt EP_1 (and therefore obtain the plaintext piece) with the leaked key, W_I ($\subseteq \mathcal{K}_B$). A free rider (in this case, A) can decrypt an encrypted piece (EP_1) he received with leaked keys only if an encrypted piece (EP_2) a malicious user (B) received has traversed the free rider. On the other hand, C cannot decrypt EP_3 with any leaked key because $W'_A \nsubseteq \mathcal{K}_B$.

Suppose that (1) k malicious users, $i_1, \ldots, i_k$, leak sets of their decryption keys, $\mathcal{K}_{i_1}, \ldots, \mathcal{K}_{i_k}$ and (2) a free rider, F, has downloaded N encrypted pieces, $c_1, \ldots, c_N$, of content where a set of decryption keys for each c_j is $\mathcal{K}_{F,j}$. If $\mathcal{K}_{F,j} \subseteq \cup_{\ell=1}^{k} \mathcal{K}_{i_\ell}$, F can decrypt c_j with the leaked keys. Therefore, F can obtain D % of the content if $|\{j \mid \mathcal{K}_{F,j} \subseteq \cup_{\ell=1}^{k} \mathcal{K}_{i_\ell}\}|/N = D/100$. Figure 9 shows the experimental results for the influence of the key leakage in Exp. 3-3. We mean by decryptable ratio what percentage of encrypted pieces of content can be decrypted by a free rider with leaked keys. The solid lines show the maximum and average decryptable ratios when k malicious users leak their decryption keys. In this case, the decryptable ratio is formulated as D.

Malicious users might also leak their user keys to free riders, though the leaked user keys can immediately be traced back. The dotted lines show the maximum and average decryptable ratios when k malicious users leak their user keys in addition to their decryption keys. In this case, the decryptable ratio is formulated as $D' = 100 \times |\{j \mid \mathcal{K}_{F,j} \subseteq \cup_{\ell=1}^{k} (\mathcal{K}_{i_\ell} \cup \mathcal{K}'_{i_\ell})\}|/N$, where $\mathcal{K}'_{i_\ell}$ denotes a set of decryption keys that are derived from i_ℓ's user key.

We found in either case, a free rider can only decrypt a low percentage of the encrypted pieces on average, whereas

[†]2,700-MB, 4,700-MB, and 25,000-MB content can be considered to be MPEG-4 content (3 Mbps, 2 hours), DVD content, and Blu-ray Disc content [5], respectively.

only a few free riders can decrypt almost 100% of the encrypted pieces. Since the quality of content is greatly degraded unless the decryptable ratio becomes high, these results verify that the proposed scheme is resilient to the key leakage in the presence of hundreds of malicious users.

5. Related Work and Discussion

Many P2P content distribution schemes (e.g., [10], [13], [30]) have been proposed. The survey in [3] is helpful for grasping the literature of P2P content distribution. Content protection in a P2P network is studied in [12], [18], [20], [25], [26], [28]. Key management is discussed in [20], [26]. And a fingerprint based scheme is proposed in [14]. With regard to other properties, availability, file authenticity, and anonymity are extensively studied in the literature. This is the case also for a content delivery network (e.g., [2]).

To the best of our knowledge, previous schemes, except that of [34], in a P2P content distribution system do not achieve the individualization of encrypted content. In [34] the problem of the key leakage is dismissed as insoluble by making an unrealistic assumption. Their implementation of players (software players, set-top boxes, etc.) is idealized and it is assumed that no decryption keys are leaked from any players. This assumption does not hold true in reality for software players. As implied in [1], not all of the players are implemented robustly enough to ensure protection from the key leakage in the real world. Therefore, we should make a realistic assumption that decryption keys might be leaked.

The scheme of [34] is based on BitTorrent [30]. Let m_i $(1 \leq i \leq N)$ be the i-th piece of content and g, p be parameters in the standard ElGamal encryption scheme [9]. Calculations are done over $\mathbb{Z}_p^*$. Each user, P_j, generates a pair of secret and public keys, (s_j, g^{s_j}). A tracker, which is a server that keeps track of which users are online, generates secret random numbers, $r_{1,j}, \ldots, r_{N,j}$, for all of the users. At the initial stage of distribution, the tracker encrypts m_i with $r_{i,j}, g^{s_j}$ (in response to P_j's request) and sends the resulting ciphertext, $m_i g^{r_{i,j} s_j}$, to P_j. As users have encrypted pieces, they upload/download encrypted pieces to/from one another. Suppose that P_j wants to upload a ciphertext of m_i to another user, P_k. First, P_j makes a request for re-encryption to the tracker. Secondly, the tracker sends a re-encryption key, $RK_{j\to k}$, back to P_j, where $RK_{j\to k} = g^{r_{i,k} s_k - r_{i,j} s_j}$. Thirdly, P_j re-encrypts his ciphertext for P_k by calculating $(m_i g^{r_{i,j} s_j}) RK_{j\to k} = m_i g^{r_{i,k} s_k}$, and sends it to P_k. Lastly, P_k decrypts the received ciphertext with their decryption key, $(s_k, g^{r_{i,k}})$, where $g^{r_{i,k}}$ is given by the tracker after P_k pays for the content. Since the ciphertext for P_j and that for P_k are different (i.e., $m_i g^{r_{i,j} s_j} \neq m_i g^{r_{i,k} s_k}$), this scheme supports the individualization of encrypted content. Unfortunately, the scheme is vulnerable to the leakage of decryption keys. Suppose that P_k is compromised and a set of its decryption keys, $(s_k, g^{r_{1,k}}, \ldots, g^{r_{i,k}}, \ldots, g^{r_{N,k}})$, is leaked. Because any user, P_ℓ, who just follows the above protocol and has their ciphertext, $m_i g^{r_{i,\ell} s_\ell}$, can obtain $RK_{\ell\to k}$ from the tracker, P_ℓ can get $m_i g^{r_{i,k} s_k} (= (m_i g^{r_{i,\ell} s_\ell}) g^{r_{i,k} s_k - r_{i,\ell} s_\ell})$ by re-encrypting their ciphertext. Therefore, P_ℓ can obtain m_i by decrypting the re-encrypted ciphertext with the leaked decryption key, $(s_k, g^{r_{i,k}})$. This means that anyone can play back the content without purchasing it and that just a single set of leaked decryption keys suffices for the free ride. We also consider the decryption cost in the scheme of [34]. Suppose that (1) content consists of N pieces, (2) the length of each piece, $|m_i|$, is the same, (3) m_i consists of n sub-pieces, (4) the length of each sub-piece is the same as $|p|$, i.e., $n = |m_i|/|p|$. A user has to execute the standard ElGamal decryption nN times in order to decrypt the content. This is impractical in most cases: if the sizes of content, m_i, and p are 25,000 MB (e.g., Blu-ray Disc content), 1 MB, and 1024 bits, respectively, then it holds that $nN = 25,600,000$. In this case, it takes more than 7 hours to decrypt the content if the time required for decryption of a single sub-piece (i.e., 1024-bit ElGamal decryption) is 1 [msec].

We explain how to cope with a threat that a valid but malicious user redistributes the decrypted content itself. Gao et al.[14] applied an asymmetric fingerprinting scheme [16] to content protection in the P2P network. In their scheme, each user gets a slightly different fingerprinted content as follows: A distributor (a content server in our scheme) encrypts the content in such a way that each user decrypts the same encrypted content with a different key. Every user has to leave his node ID (user ID in our scheme) into the content in order to decrypt it, since his decryption key corresponds to his own node ID and the results of decryption vary with the decryption key. Assuming that the maximum number of malicious users in a coalition is less than a predetermined value, redistributed content can be traced back to at least one of the colluders.

Fortunately, their scheme can easily be integrated into ours in the following way. We omit describing the same part of the proposed scheme. A content server, I, devides the content into pieces, $piece_1, \ldots, piece_\ell$. According to the encryption algorithm of [14], I encrypts $piece_j$ with e_j for $1 \leq j \leq \ell$, where $\{e_1, \ldots, e_\ell\}$ denotes the encryption key in the scheme of [14] and $\{e_1, \ldots, e_\ell\}$ is given by the KM server. Let $piece'_j$ be the result of the encryption of $piece_j$. Using the method as described in Sect. 3.1, I encrypts $piece'_j$ and sends a 3-tuple of data, (X_1, X_2, X_3), to a user, A. After a user, B, gets all of the encrypted pieces, B obtains a set of the decryption keys, $\mathcal{K}_B$, and the decryption key in the scheme of [14] from the KM server. By the decryption with $\mathcal{K}_B$, B gets $piece'_1, \ldots, piece'_\ell$. Finally, B obtains a fingerprinted content using the decryption algorithm of [14].

6. Conclusions

We proposed a P2P content distribution scheme that achieves both the individualization of encrypted content and the key-leakage resilience. From the experiments on a real and large-scale P2P network, we showed that the proposed scheme works on a large-scale network and provides the key-leakage resilience.

Acknowledgments

We would like to thank the National Institute of Information and Communications Technology for providing StarBED. We would also like to thank Satoshi Ito, Toru Kambayashi, Taku Kato, Ryuichi Koike, Hideki Matsumoto, Hideaki Sato, Haruhiko Toyama, and Kentaro Umesawa for illuminating discussion of the research.

References

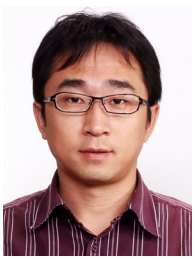
[1] Advanced Access Content System, "Response to reports of attacks on aacs technology," press on January 24, 2007 [Online]. Available: http://www.aacsla.com/news/.
[2] Akamai. [Online]. Available: http://www.akamai.com
[3] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," ACM Comput. Surv., vol.36, no.4, pp.335–371, 2004.
[4] ANSI X 9.62, "American national standard for financial services – public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA)," American National Standard Institute, 1998.
[5] Blu-ray Disc Association. [Online]. Available: http://www.blu-raydisc.com
[6] CNET, "Court orders Jammie Thomas to pay RIAA $1.92 million." [Online]. Available: http://news.cnet.com/8301-1023_3-10268199-93.html
[7] I. Cox, J. Kilian, T. Leighton, and T. Shamoon, "A secure, robust watermark for multimedia," in Proc. Information Hiding, ser. LNCS 1174, pp.185–206, Springer-Verlag, 1996.
[8] Crypto++ 5.6.0 Benchmarks. [Online]. Available: http://www.cryptopp.com/benchmarks.html
[9] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," IEEE Trans. Information Theory, vol.31, no.4, pp.469–472, 1985.
[10] eMule. [Online]. Available: http://www.emule-project.net
[11] FIPS PUB 180-2, secure Hash Standard, Federal Information Processing Standards Publication 180-2, Department of Commerce, National Institute of Standards and Technology (NIST), Information Technology Laboratory (ITL), 2002.
[12] L. Garcia, L. Arnaiz, F. Álvarez, J. Menéndez, and K. Grüneberg, "Protected seamless content delivery in P2P wireless and wired networks," IEEE Wireless Commun., vol.16, no.5, pp.50–57, 2009.
[13] Gnutella. [Online]. Available: http://rfc-gnutella.sourceforge.net/index.html
[14] H. Gao, W. Zeng, and Z. Chen, "Fingerprinting for Copyright Protection in P2P Context," 2010 International Symposium on Intelligence Information Processing and Trusted Computing (IPTC), IEEE, 2010.
[15] J. Haitsma and T. Kalker, "A highly robust audio fingerprinting system," Proc. 3rd International Conference on Music Information Retrieval (ISMIR), pp.107–115, 2002.
[16] D. Hu and Q. Li, "Asymmetric fingerprinting based on 1-out-of-n oblivious transfer," IEEE Commun. Lett., vol.14, no.5, pp.453–455, 2010.
[17] "How Much Do Music and Movie Piracy Really Hurt the U.S. Economy?." [Online]. Available: http://freakonomics.com/2012/01/12/how-much-do-music-and-movie-piracy-really-hurt-the-u-s-economy/
[18] T. Iwata, T. Abe, K. Ueda, and H. Sunaga, "A DRM system suitable for P2P content delivery and the study on its implementation," in Proc. 9th Asia-Pacific Conference on Communications (APCC 2003), vol.2, pp.806–811, 2003.
[19] LibTorrent. [Online]. Available: http://libtorrent.rakshasa.no
[20] X. Liu, H. Yin, C. Lin, and C. Du, "Efficient user authentication and key management for peer-to-peer live streaming systems," Tsinghua Science & Technology, vol.14, no.2, pp.234–241, 2009.
[21] T. Matsushita, S. Yamanaka, and F. Zhao, "A peer-to-peer content-distribution scheme resilient to key leakage," in Proc. WISA 2011, ser. LNCS 7115. Springer-Verlag, pp.121–135, 2011.
[22] T. Miyachi, K. Chinen, and Y. Shinoda, "StarBED and SpringOS: Large-scale general purpose network testbed and supporting software," in Proc. 1st International Conference on Performance Evaluation Methodologies and Tools. ACM Press, 2006.
[23] Motion Picture Association of America, "Missouri man sentenced to two years in federal prison for camcord movie theft." [Online]. Available: http://www.mpaa.org/resources/b50ae492-248e-4960-985e-1051efd1428a.pdf
[24] D. Naor, M. Naor, and J. Lotspiech, "Revocation and tracing schemes for stateless receivers," in Proc. CRYPTO 2001, ser. LNCS 2139. pp.41–62, Springer-Verlag, 2001.
[25] E. Palomar, J. Tapiador, J. Hernandez-Castro, and A. Ribagorda, "Secure content access and replication in pure P2P networks," Computer Communications, vol.31, no.2, pp.266–279, 2008.
[26] F. Qiu, C. Lin, and H. Yin, "EKM: An efficient key management scheme for large-scale peer-to-peer media streaming," in Proc. PCM 2006, ser. LNCS 4261. pp.395–404, Springer-Verlag, 2006.
[27] StarBED Project. [Online]. Available: http://www.starbed.org
[28] J.-Y. Sung, J.-Y. Jeong, and K.-S. Yoon, "DRM enabled P2P architecture," in Proc. 8th International Conference of Advanced Communication Technology (ICACT), vol.1, pp.487–490, 2006.
[29] TechEye, "DVDFab is still going claims shocked studios." [Online]. Available: http://www.techeye.net/business/dvdfab-is-still-going-claims-shocked-studios
[30] The BitTorrent Protocol Specification. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html
[31] The BitTorrent Tracker Source Code. [Online]. Available: http://download.bittorrent.com/dl/archive/BitTorrent-3.9.1.tar.gz
[32] Transport Layer Security (TLS) Protocol Version 1.0. [Online]. Available: http://www.ietf.org/rfc/rfc2246.txt
[33] WebLOAD. [Online]. Available: http://www.webload.org
[34] X. Zhang, D. Liu, S. Chen, Z. Zhang, and R. Sandhu, "Towards digital rights protection in BitTorrent-like P2P systems," in Proc. 15th SPIE/ACM Multimedia Computing and Networking, 2008.

Appendix A: Detailed Results of Large-Scale P2P Content Distribution Experiments

Table A· 1 This table shows the results of StarBED based large-scale P2P content distribution experiments

Exp. no.	No. of nodes	Re-entry frequency	Max. no. of transfers (Tr_{max})	Waiting time [sec]	No. of clients per node	Size of data (MB)	Ratio of uploading to downloading (%)	Measured max. no. of transfers (Tr_{meas})	Ave. no. of transfers	Ave. downloading time [sec]
1-1	10	10	254	0	1	195	46.7	10	1.8	8.2
1-2	50	10	254	0	1	195	89.8	67	10.0	8.8
1-3	100	10	254	0	1	195	94.5	93	21.3	9.5
1-4	150	10	254	0	1	195	96.2	100	26.2	9.7
1-5	10	50	20	0	1	195	47.5	12	1.8	8.2
1-6	50	50	20	0	1	195	89.6	20	8.2	9.3
1-7	100	50	20	0	1	195	93.4	20	11.0	10.7
2-1	10	50	254	0	1	195	45.6	14	1.7	8.1
2-2	150	50	254	0	1	195	96.5	254	55.7	9.6
2-3	10	500	254	0	1	195	45.8	17	1.7	8.1
2-4	150	500	254	0	1	195	97.1	254	60.7	9.6
2-5	150	50	15	0	1	195	94.2	15	9.4	12.1
3-1	150	50	5	0	1	195	78.2	5	3.1	47.9
3-2	150	50	10	0	1	195	91.3	10	6.3	16.3
3-3	150	50	20	0	1	195	95.1	20	12.0	11.1
3-4	150	50	30	0	1	195	95.8	30	16.4	10.4
4-1	30	50	254	0	5	195	96.7	227	38.0	27.8
4-2	150	50	254	0	5	195	99.4	254	89.6	28.9
4-3	30	50	20	0	5	195	95.7	20	11.9	27.5
4-4	150	50	20	0	5	195	98.9	20	14.9	30.9
5-1	50	50	254	0	1	1,954	93.1	141	11.0	77.0
5-2	150	50	20	0	1	1,954	97.2	20	11.9	90.6
5-3	150	50	20	0	5	1,954	100.2	20	13.1	616.0
5-4	150	50	20	20	1	1,954	97.3	20	11.8	82.2
6-1	150	50	254	2	1	195	96.7	254	47.9	9.0
6-2	150	50	254	4	1	195	96.9	254	47.4	8.7
6-3	150	50	254	8	1	195	96.7	226	41.6	8.7
6-4	150	50	20	5	1	195	95.5	20	11.7	9.3
6-5	150	50	20	10	1	195	95.7	20	11.8	8.9
6-6	150	50	20	20	1	195	97.9	20	13.0	8.7

Tatsuyuki Matsushita is working as a security architect at Industrial ICT Security Center, Industrial ICT Solutions Company, Toshiba Corporation. His research interests include information security and applied cryptography. He is a member of IEICE, IEEE.



Fangming Zhao is working as a security architect at Industrial ICT Security Center, Industrial ICT Solutions Company, Toshiba Corporation. His research interests include information security and applied cryptography. He is a member of CIGRE.

Shinji Yamanaka is currently working as a Security Architect at Advanced Meter Communication Systems Department, Fuchu Complex, Toshiba Corporation. His research interests include information security and applied cryptography, especially key management in large scale infrastructure systems. He is a member of IEICE.