A Novel Stochastic-Encryption-Based P2P Digital Rights Management Scheme

Majing Su*, Hongli Zhang*, Xiaojiang Du† and Qiong Dai
*School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China.
† Dept. of Computer and Information Sciences, Temple University, Philadelphia, PA, USA.
e-mail: {sumajing, zhl}@pact518.hit.edu.cn, dux@temple.edu

***Abstract*—Digital right protection in P2P systems is attracting more and more attentions. In this paper, we present a new stochastic-encryption-based Digital Rights Management (DRM) scheme for P2P content delivery networks. The files are encrypted such that unpaid users cannot access the plaintext content. We exploit the random characteristics of P2P to increase the key space, which can defense collusion attacks. We add piece validation policy during a download process to prevent poisoning attacks. In our scheme, peers make a payment after downloading, and this prevents user loss due to download failures (caused by the dynamics of P2P). Our scheme does not have frequent user authentications or state maintenance. Analysis and simulation experiments show that our scheme can defend against collusion attacks and poisoning attacks with a fairly high probability.**

***Index Terms*—P2P; copyright protection; DRM**

I. Introduction

Peer-to-Peer (P2P) networks provide us an effective way of file sharing. Large digital contents such as movies, games and software can be delivered fast over P2P networks. However, illegal sharing copyrighted materials also bring piracy problems. P2P users can download copyrighted content without authorization by copyright owners. These abuses lead to economic dispute and legal matters. Copyright owners have been fighting against piracy via both legal and technical measures. Lots of websites such as Mininova [1] have been forced to delete pirated content or even shut down due to copyright infringement. However, users seek new approaches (i.e. using Private Tracker) to share content for free. As a result, copyrighted content providers are unwilling to deliver their products by P2P systems. This not only leads to a great deal of financial loss to the content providers but also prevent the legal commercial use of P2P technologies.

Many possible counter-measures have been proposed to solve piracy problem in P2P network. Some piracy detection systems [2]-[4] identify copyright infringement by monitoring user traffic and behaviors. However, deep package inspection may invade users' privacy. Another type of approaches is based on user identity authentication [11][12]. Typically, each paid peer is given a unique ID, and it only uploads data to peers with valid IDs. However, paid peers may share content with unpaid users (referred to as *collusion attack*). Moreover, continuous authentication costs a large overhead.

Cryptographic technologies [5]-[10], especially encryption [5]-[7], have been widely used in designing copyright-protected P2P mechanisms. Zhang *et al.* [5] discuss some intuitive approaches of applying traditional symmetric key and public-key algorithms in P2P networks. The main concern is how to defend against collusion attack and keep the efficiency of P2P. In addition, if peers verify content only after downloading all the pieces and decrypting them, an attacker can easily deploy a *poisoning attack* (uploading fake pieces to peers). However, unfortunately, this is not considered in most existing encryption-based schemes [6][7].

On the other hand, P2P is open, anonymous and scalable. These characteristics make it hard to prevent piracy and require the DRM schemes to be high scalable and efficient. Besides, P2P is dynamic, and some pieces may be missing after a period of time. Thus users may fail to download the entire file after payment, which hurts users' interest.

Observing these issues, in this paper, we propose an novel digital right protection scheme for P2P content delivery networks. We exploit the random characteristics of P2P systems to increase the key space. The decryption key sequence of the content for each peer is different from others, thus unauthorized users could not access the content even if some colluders share their keys. We modify the piece validation policy to prevent the poisoning attacks. Peers does not need to maintain massive state nor frequently verify users' identity, which improves the efficiency. Analysis and experiments show that our scheme is secure to multiple types of attacks.

The remainder of this paper is organized as follows. In Section II, we present our copyright protection scheme. Then we analyze its security in Section III, and evaluate its performance in Section IV. We compare our scheme with some related work in Section V. Our work is concluded in Section V.

II. Our Copyright Protection Scheme

A. Design Overview

In our scheme, large files are broken into several pieces and each piece is encrypted with several different keys. According to our measurement, a peer usually download pieces of one file from multiple peers. Therefore, even if pieces transferred between two peers are encrypted with same keys, with high probability they will get all pieces from different peers. Consequently, the sequence of decryption keys for all the pieces are different. This is the basic idea of our approach. Besides, we attach a signed piece hash with each encrypted piece. Peers can verify the piece by checking the signature and hash.

Fig.1 shows the overview of our scheme. We employ an *agent server* (*AS*) to perform delivery, which has a website, a key generation clients and several distribution clients. The

978-1-4673-6432-4/15/$31.00 ©2015 IEEE

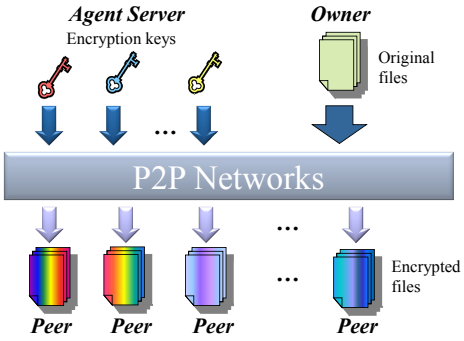

Fig. 1. Sketch of our scheme

website is in charge of publishing content index information, handling peer registration and bills, and delivering decryption keys. The key generation client generates keys for encryption and signature. The distribution clients are responsible for encrypting pieces and distributing them to peers. The agent server is trustable to the owner. To simplicity, we only use one agent server to illustrate system. In fact, the owner can deploy multiple agent servers to improve performance and robustness.

We use a group of cryptographic algorithms to perform piece operations:

- $KGen(n)$ are a group of functions performed by the *AS* to generate a public key U_f and a private key R_f, a group of n-bit encryption keys $\boldsymbol{EK}$=(k_1,k_2,...,k_m) and the corresponding decryption keys $\boldsymbol{DK}$=(Dk_1,Dk_2,...,Dk_m). U_f and R_f are used to encrypt/decrypt key information and verify/sign the piece hash. $\boldsymbol{EK}$ and $\boldsymbol{DK}$ are used to encrypt and decrypt pieces.
- $Enc(P_x, k_i)$ is an encryption function performed by the *AS* to encrypt the x-th plaintext piece P_x with key k_i, obtaining a cipher piece C_iP_x.
- $Dec(C_iP_x, Dk_i)$ is a decryption function performed by peers to get the plaintext P_x by decrypting the cipher piece C_iP_x with decryption key Dk_i.
- $RKGen(k_i, k_j)$ is a function performed by the *AS* to generate a transfer key tk_{ij} from k_i and k_j.
- $REnc(C_iP_x, tk_{ij})$ is a re-encryption function used by peers to re-encrypt the cipher piece C_iP_x (P_x encrypted with k_i) with a transfer key tk_{ij}, which generates a new cipher piece C_jP_x (the same as P_x encrypted with k_j).
- $Hash(X)$ is a hash function to compute a digest of X.
- $Sign(X, R_f)$ is a signature function to sign X with R_f.

Many existing crypto algorithms may be used as above functions as long as they are secure enough (e.g., resilience to brute-force attacks and chosen-plaintext attacks). Besides, to maintain the efficiency of P2P network, these algorithms should be efficient and easy to implement. In this paper, we use SHA-1 to compute hash and RSA algorithm to sign. We employ the ElGamal cryptosystem [13] to encrypt and decrypt pieces, which is widely used in secure transmission. To generate keys, the *AS* chooses a primer p, a primitive element α of Z_p^*, and an integer a ($0 < a < p-1$), and computes $\beta = \alpha^a$. Then it randomly chooses a group of integers $k_1, k_2, ..., k_m$ ($0 < k_1, k_2, ..., k_m < p-1$) for encryption. The corresponding decryption key of k_i is $((\alpha^{k_i})^a)^{-1} \bmod p$.

In our scheme, only p and the re-encryption keys generated from two encryption key are public; while (α, a, β) are secret.

For each plaintext piece P_x and an encryption key k_i, the cipher is calculated as equation (1). Correspondingly, the decryption algorithm is in equation (2). The transfer key generation algorithm performed is in equation (3) and the re-encryption algorithm performed is in equation (4).

$$C_iP_x = Enc(P_x, k_i) = P_x\beta^{k_i} \bmod p \quad (1)$$

$$Dec(C_iP_x, Dk_i) = P_x\beta^{k_i}((\alpha^{k_i})^a)^{-1} \bmod p = P_x \quad (2)$$

$$tk_{ij} = RKGen(k_i, k_j) = \beta^{k_j - k_i} \bmod p \quad (3)$$

$$\begin{aligned} C_jP_x = REnc(C_iP_x, k_{ij}) &= P_x\beta^{k_i}\beta^{k_j-k_i} \bmod p \\ &= P_x\beta^{k_j} \bmod p = Enc(P_x, k_i) \end{aligned} \quad (4)$$

B. Key Algorithms

1) Piece Encapsulation. Pieces are encapsulated to prevent collusion attacks and poisoning attacks. For each piece (say P_x) and a key (say k_i), the *AS* generates a encapsulated piece using Algorithm 1, and caches their Ek, CP, EH in a *cipher table*. Each key is only used once per piece. The *key information* Ek_i (line 3 in Algorithm 1) is used by *AS* to find the appropriate decryption key for piece P_x. We use a sequence number i instead of k_i to prevent brute-force attack and reduce storage cost. $nonce_{ix}$ is a timestamp used to prevent replay attack (replaying an old Ek_i to a piece). We encrypt i and $nonce_{ix}$ with the public key U_f so they are secure.

Algorithm 1 Piece Encapsulation

Input: P_x, k_i, R_f, U_f
Output: Encapsulated piece $EC(P_x; k_i)$
Procedures:
1: $C_iP_x = Enc(P_x, k_i)$
2: select a nonce $nonce_{ix}$
3: $Ek_i = Encrypt(i||nonce_{ix}, U_f)$
// Encrypt the key information with public key U_f //
4: $H_{ix} = Hash(Ek_i||C_iP_x)$ // calculate a digest of $Ek_i||C_iP_x$ //
5: $EH_{ix} = Sign(H_{ix}, R_f)$
//Sing the digest with private key R_f //
6: $EC(P_x; k_i) = Ek_i||C_iP_x||EH_{ix}$
// join Ek_i, C_iP_x and EH_{ix} to form a encapsulated piece //

We attach a modified piece hash EH_{ix} (line 5 in Algorithm 1) to C_iP_x to prevent replay attack and poisoning attack (replacing Ek_i or C_iP_x with a fake one). The hash of Ek_i and C_iP_x is signed by the private key R_f, which can be verified by peers after downloading the piece. To verify a piece $EC(P_x; k_i)$, a peer decrypts EH_{ix} with U_f and compares it with $Hash(Ek_i||C_iP_x)$. If identical, the piece is valid. Otherwise, the piece is invalid.

2) Piece Re-encryption. In practice, cipher pieces on peers with more connections have larger chances to be downloaded. This increases the probability that different peers may share the same decryption keys. To address this issue, we use a re-encryption algorithm (Algorithm 2) at each peer, which transfers a piece encrypted with k_i to a piece encrypted with k_j. To ensure the confidentiality, peers should not be able to obtain plaintext content during re-encryption.

Algorithm 2 Piece Re-encryption

Input: $EC(P_x; k_i)$
Output: $EC(P_x; k_j)$
Procedures:
$Ek_i = GetEK(EC(P_x; k_i))$
// get key information from $EC(P_x; k_i)$ //
$C_iP_x = GetCP(EC(P_x; k_i))$ // get cipher from $EC(P_x; k_i)$ //
$Reply = Re\text{-}encryptionKeyReuqest(Ek_i, x, AS)$
// request a re-encryption key for piece x from *AS*//
if $Reply == \{Ek_j, EH_{jx}, tk_{ij}\}$ **then**
  $C_jP_x = REnc(C_iP_x, tk_{ij})$
  $EC(P_x; k_j) = Ek_j||C_jP_x||EH_{jx}$
else
  // $Reply$ = "nonce invalid", $EC(P_x; k_j)$ //
  $EC(P_x; k_j) = Extract(Reply)$
  // extract new piece from reply message //
end if

To re-encrypt a piece P_x, the peer first sends a request to the *AS* that contains the key information Ek_i of P_x. When received the request, the *AS* decrypts Ek_i with its private key R_f to get i and $nonce_{ix}$. If the $nonce_{ix}$ is valid, the *AS* locates the key k_i according to i and randomly chooses another key k_j (different from k_i). Then it computes tk_{ij} using the $RKGen(k_i, k_j)$ algorithm, looks up Ek_j and EH_{jx} in cipher table (or computes if not exist), and sends tk_{ij}, Ek_j, EH_{jx} to the peer. If the $nonce_{ix}$ is invalid, the *AS* replies a "nonce invalid" message and a new encapsulated piece of P_x.

After the peer receives response containing the transfer key tk_{ij} from the *AS*, it re-encrypts the cipher C_iP_x with tk_{ij} using the re-encryption algorithm $REnc(C_iP_x, tk_{ij})$, and caches the encrypted piece $EC(P_x, k_j)$. If the response is a new encapsulated piece of P_x, the peer caches it instead.

C. Main Work flow

1) Content Publishing. To delivery a content, the owner uploads the original files and content information to *AS* in a secure way. Then the *AS* perform $KGen(n)$ to generate keys (U_f, R_f, $\boldsymbol{EK}$, $\boldsymbol{DK}$) for the content, and publish the content index information (*file name*, *infohash*, U_f, etc.) on its website. The *AS* breaks the content into N pieces. For each piece P_x, the *AS* calculates and caches T_{AS} different encapsulated pieces using Algorithm 1. The purpose of caching T_{AS} copies of each piece is to serve concurrent requests.

2) Piece Delivery. Peers join the *swarm* (group of peers sharing same content) and sequentially request pieces from *AS* and other peers. Fig. 2 illustrates the piece transmission among peers. The *AS* returns an undistributed encapsulated piece for each request, then it chooses a different key to encrypt that piece and caches the cipher. Once downloaded a piece, a peer verify it immediately. The peer should discard the invalid piece and download the piece again.

Each peer tracks the upload times $utimes$ of each encapsulated piece. When received a piece request, if $utimes$ is no more than a threshold T_P, the peer returns the encapsulated piece it has downloaded. Otherwise, the peer must generate a new encapsulated piece using Algorithm 2 and send it to the remote peer. Peers perform re-encryption up to T_r times. After that, a peer responds a corresponding encapsulated pieces randomly chosen from its cipher table.

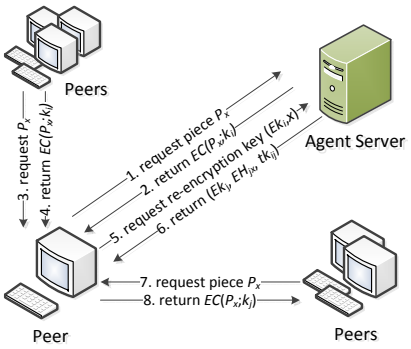

Fig. 2. Piece transmission among peers

3) File Decryption. In our scheme, peers pay for decryption keys when finished downloading all pieces of the content. The peer logs into the *AS* and pays for the content, then uploads the Ek sequence of pieces. The *AS* checks the Ek sequence to generate a response message (decryption keys or piece number of all incorrect pieces) using Algorithm 3 and returns the results to the peer. After receiving response, the peer verifies the signature. If the response is the decryption key sequence, the peer decrypts each encrypted piece with its corresponding key using $Dec(C_iP_x, Dk_i)$ (e.g., decrypts C_iP_x with Dk_i). If the response is the piece number of incorrect pieces, the peer should re-download these pieces from *AS* and then upload the Ek sequence, until the content is successfully decrypted.

Algorithm 3 Decryption Key Generation

Input: Ek sequence, R_f
Output: Dk sequence SEQ or incorrect piece number IPN
Procedures:
SEQ = null, IPN = null, $all\_correct$ = true
for each Ek_i in Ek sequence **do**
  $i||nonce_{ix} = Decrypt(Ek_i, R_f)$
  if $Check(nonce_{ix}) ==$ "$correct$" **then**
    $Dk_i = GetKey(i)$ // find the i-th decryption key//
    SEQ = $SEQ||Dk_i$ // join Dk_i to SEQ //
  else
    IPN = $IPN||x$ // mark the incorrect piece number//
    $all\_correct$ = false
  end if
end for
if $all\_correct$ == true **then**
  return SEQ
else
  return IPN
end if

During above three process, data transmission between the owner and the *AS*, peers and *AS* is protected by existing security mechanisms such as SSL, VPN, etc. The purpose of peers re-downloading incorrect pieces from *AS* is to ensure that all paid peers can download the entire content correctly and decrypt it successfully. The *AS* is required to serve for a long term before the content index is deleted.

III. Security Analysis

Usually, with partially decrypted data, an attacker cannot tell whether the decryption is successful; neither can the attacker access the content, especially to the audio or video

files. Hence, *we define a successful decryption as decryption of the entire content*. In this section, we evaluate our scheme by analyzing the resistance against some potential attacks.

A. Brute-Force Attack

Let N be the number of pieces of the entire content. Let ω be the average number of *distributed versions* (different encrypted versions of a piece delivered in P2P system) of each piece. Therefore, the key space size is ω^N. In P2P, a file usually can be divided into hundreds of pieces, and most pieces at AS and hot peers will be request by lots of peers, thus ω is usually larger than 1. Hence the key space is large enough to defense brute force attack. For instance, if $\omega = 2$, $N = 500$, the key space is $2^{500} \approx 3.27e + 150$. Suppose an attacker tries to decrypt with 1 billion key per second, the time of searching decryption key is about 1.04e+134 years.

B. Collusion Attack

We mainly analyze two types of collusion attacks to demonstrate the security of our scheme as follows.

1) Sharing-key Attack. Colluders share their decryption key sequences to pirates. In this case, a pirate can decrypt the content only if its decryption key sequence is the same as the colluders'. According to the well-known birthday paradox, theoretically, the probability $P(\omega^N, \chi)$ of two peers having the same decryption keys (referred to as *key conflict*) is shown in equation (5), where χ is the number of peers in a swarm.

$$P(\omega^N, \chi) = 1 - \frac{\omega^N!}{(\omega^N - \chi)!\omega^{N\chi}} \approx 1 - e^{-\chi(\chi-1)/2\omega^N} \quad (5)$$

According to the equation (5), in a typical size swarm, a very huge number of colluders are required to achieve a successful collusion. Take a simple situation ($\omega = 2$ and $N = 50$) as an example, to achieve $P(\omega^N, \chi) = 0.001$, about χ=5e+13 peers are required. In addition, given certain level of χ, the larger N and ω are, the smaller $P(\omega^N, \chi)$ is, therefore, even if some colluders publish their decryption key sequences, almost no pirates can decrypt all of their own pieces.

2) Selecting-key Attack. Colluders upload key information Ek and corresponding decryption key Dk. A pirate can decrypt the content if he can find Dk for all the downloaded pieces by comparing its own Ek with the keys uploaded by colluders. For a χ-scale swarm, let γ be the percentage of colluders, then the number of colluders in the swarm is $N_c = \gamma\chi$. *Distribution rate* $\lambda = \omega/\chi$ is the ratio of the number of distributed versions and swarm size. In theory, the probability that pirates successfully decrypt the content is:

$$P_{success} = [1 - (1 - \frac{1}{\omega})^{N_c}]^N = [1 - (1 - \frac{1}{\lambda\chi})^{\lambda\chi}]^N \quad (6)$$

Figure 3 shows $P_{success}$ varying with λ and γ on some typical N and χ. According to equation (6) and Fig. 3, for a typical swarm $\chi > 100$ and $N > 500$, when $\lambda > 0.1$ and $\gamma < 0.5$, the probability of pirate successful decryption is less than 0.2%. When $\lambda > 0.2$, even if the percentage of colluders is close to 100% (γ=0.99 in Fig. 3), the probability that a pirate can decrypt its content is still close to zero. This indicates that: to a given swarm, the larger ω is, the more likely that each peer get different decryption keys. A large N (e.g., $N > 1000$) ensures that $P_{success}$ is small. N is determined by the owner and can be large. To prevent collusion attacks, we need to increase λ by setting a small T_p and a large T_r. We suggest set T_p as 2-3 and T_r as 3-5.

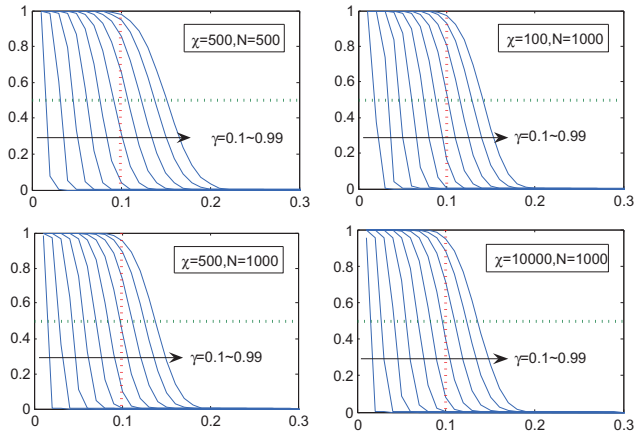

Fig. 3. Probability of successful collusion. The x-axis is λ and the y-axis is the $P_{success}$

C. Poisoning Attack.

In our scheme, hash and signature are used to prevent poisoning attack. Malicious attackers can upload invalid Ek, CP and EH to other peers. Peers compute the hash of $Ek||CP$, and compare it with the hash embedded in EH. If the two hashes are different, this piece is regarded as falsified. In the case that EH is falsified by the attacker, peers can always find the signature invalid when verifying it with U_f. Therefore, our scheme is resilience to the poisoning attack.

IV. Simulation Experiments

In real P2P networks, there are some hot peers, which makes the probability of key conflict larger than the theoretical value. To examine this, we implement our scheme in a simulation environment. We deploy χ peers and a *AS* with 5 distribution clients in a server. There are 5% hot peers who have large bandwidth and no connection limit. Other peers have a connection limit of 50. All peers apply the tit-for-tat peer selection policy and rarest-first piece selection policy. We use $M = 0.2\chi$ 1024-bit encryption keys, and the piece number N is set to 1000.

We plot the results of sharing-key attack in Fig.4. It shows that none of the peer pairs shares more than 40% of the key sequence, that is, all peers have different key sequences. Thus the collusion of sharing-key does not work. This confirms the theoretical analysis in subsection III-B. Besides, the trend of the data in Fig.4 implies that: the larger the swarm is, the smaller the average proportion of shared key sequence is; in a large swarm (e.g., $\chi > 2000$), most peers would share less than 10% of the key sequence with others.

To evaluate the resistance against selecting-key attack, we randomly select a group of colluders from the peers and estimate the probability of successfully decryption for different percentages of colluders. In our simulations, χ is set to 50, 100, 500, 1000, and 2000, and γ is set from 0.1 to 0.99. The

TABLE I
PROBABILITY OF SUCCESSFULLY COLLUSION

Swarm size χ	Percentage of colluders γ									
	0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**0.99**
50	0	7.06E-257	6.65E-174	1.29E-114	3.89E-85	4.96E-61	1.73E-46	6.11E-28	1.28E-14	0.059
100	0	5.97E-245	2.14E-151	1.85E-111	1.37E-77	8.05E-56	2.20E-42	1.68E-20	4.48E-11	0.237
500	0	7.45E-255	1.29E-187	2.47E-141	1.03E-105	1.55E-73	1.36E-50	2.07E-32	1.97E-14	0.045
1000	0	5.34E-276	5.28E-199	8.10E-146	1.87E-105	1.80E-76	1.09E-52	1.77E-31	3.07E-15	0.073
2000	0	1.17E-290	9.36E-205	1.59E-150	7.01E-111	2.25E-80	1.08E-55	4.01E-34	3.77E-16	0.046

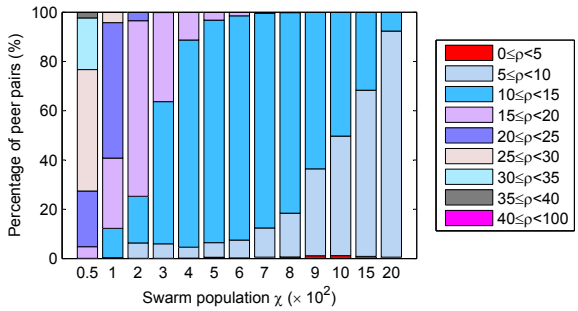

Fig. 4. Percentage of peer pairs that share certain proportions ρ of the key sequence under different swarm populations

results are shown in Tab. I, which indicates that the probability of successful collusion is very low even when the percentage of colluders reaches 90%.

We also measure the computational overhead of the cryptographic functions using a PC (CPU: 2.60GHz, memory: 2GB). Results shows that for a typical size piece, it takes less than 1s to perform encryption and re-encryption, e.g., about 0.3s for a 2MB piece. This is affordable for both the AS and peers in their uploading and downloading process.

V. Related Work and Comparison

There are lots of studies on copyright protection in P2P systems. For example, Zhang *et al.* [5] present a piece-level encryption-based DRM scheme for BitTorrent-like systems. Chen *et al.* [6] redesign Zhang's [5] system to avoid the performance bottleneck of trackers. Lou *et al.* [11] poison the detected pirates with fake blocks to protect copyrighted content. Qiu *et al.* [7] propose a decentralized authorization scheme for DRM in P2P based on proxy re-encryption mechanism. Other technologies such as Diffie-Hellman group key[8], fingerprinting[9], digital watermarking [10], are also used in design DRM mechanism in P2P systems.

Our scheme is suitable for large file distribution over P2P networks. Comparing with [7][11], it does not have frequent identity authentication and state maintenance between peers during a download process. This makes it scalable and efficient. Different from [11], in our scheme, peers can join and leave randomly and pay after downloading, which protects users' interest. In addition, our scheme can prevent pollution attack, which is not considered in [6][7]. Our scheme is easy to implement and deploy as well.

VI. Conclusion

In this paper, we presented an encrypion-based digital right protection scheme for P2P content delivery networks. We encrypted pieces to ensure none can obtain plaintext content during the piece transmission. We exploited the random characteristic of P2P systems to increase the key space and prevent collusion attacks. We modified current piece hash scheme to prevent poisoning attack. In our scheme, users can pay after downloading the file, which protect users' interest against P2P download failures. Analysis and simulation results showed that our scheme can defend brute-force attack, collusion attacks and poisoning attack. Its computational overhead is acceptable.

Acknowledgment

This work was supported by the National Basic Research Program of China under Grant No. 2011CB302605, the National High-Tech Development 863 Program of China under Grant No. 2011AA010705, 2012AA012506, the National Natural Science Foundation of China under Grant No. 61402475, 61472164, and by the US National Science Foundation under grants CNS-0963578, CNS-1022552 and CNS-1065444.

References

[1] http://www.mininova.org/
[2] J. Mee, P.A. Watters, "Detecting and Tracing Copyright Infringements in P2P Networks," in *Proc. ICN/ICONS/MCL'06*, Morne, Mauritius, 2006.
[3] K.P. Chow, K.Y. Cheng, L.Y. Man, P.K.Y. Lai, L.C.K. Hui, C.F. Chong, *et al.*, "BTM-An Automated Rule-based BT Monitoring System for Piracy Detection," in *Proc. ICIMP*, 2007
[4] A. Sherman, A. Stavrou, J. Nieh, A. D. Keromytis, C. Stein, "Adding Trust to P2P Distribution of Paid Content," in *Proc. ISC'09*, Pisa, Italy, 2009, pp. 459-474.
[5] X. Zhang, D. Liu, S. Chen, Z. Zhang, and R. Sandhu, "Towards Digital Rights Protection in BitTorrent-like P2P Systems," in *Proc. SPIE/ACM Multimedia Comput. and Networking*, San Jose, CA, USA, 2008.
[6] Y. Y. Chen; J. K. Jan, Y. Y. Chi, M. L. Tsai, "A Feasible DRM Mechanism for BT-Like P2P System", in *Proc. IEEC'09*, Ternopil, Ukraine, 2009, pp. 323-327.
[7] Q. Qiu, Z. Tang, Y. Y. Yu, "A Decentralized Authorization Scheme for DRM in P2P File-Sharing Systems", in *Proc. IEEE CNCC 2011*, Las Vegas, NV, USA, 2011, pp. 136-140.
[8] Y. Chen, W. Wu, "An Anonymous DRM Scheme for Sharing Multimedia Files in P2P Networks", *Multimedia Toos and Applications*, vol. 69, April 2014, pp. 1041-1065.
[9] X. L. Li, S. Krishnan, N. W. Ma, "A Wavelet-PCA-Based Fingerprinting Scheme for Peer-to-Peer Video File Sharing", *IEEE Trans. Inf. Forensics. Security*, vol. 5, pp. 365-373, Sept. 2010
[10] D. Tsolis, S. Sioutas, A. Panaretos, I. Karydis, K. Oikonomou, "Decentralized Digital Content Exchange and Copyright Protection via P2P Networks," in *Proc. ICSS 2011*, Corfu, Greece, 2011, pp.1056-1061.
[11] X. S. Lou, K. Hwang, "Collusive Piracy Prevention in P2P Content Delivery Networks," *IEEE Trans. Comput.*, vol. 58, pp. 970-983, July 2009.
[12] J. Y. Sung, J. Y. Jeong; K. S. Yoon, "DRM Enabled P2P Architecture Advanced Communication Technology", in *Proc. ICACT 2006*, Phoenix Park, Korea, 2006, pp. 487-490.
[13] T.E. Gamal, "A Public Key Cryptosystem And Signature Scheme Based On the Discrete Logarithm," *IEEE Trans. Inf. Theory*, vol. 4, Jan. 1985.