A more efficient secure event signature protocol for massively multiplayer online games based on P2P

Dapeng Li\textsuperscript{1,a}, Liang Hu\textsuperscript{1,b}, and JianFeng Chu\textsuperscript{1,c}

\textsuperscript{1}College of Computer Science and Technology, Jilin University, Changchun, 130012, China

\textsuperscript{a}lidp14@mails.jlu.edu.cn, \textsuperscript{b}hul@jlu.edu.cn, \textsuperscript{c}chujf@jlu.edu.cn

Keywords: MMOGs, event signature, hash-chain, peer-to-peer, discrete logarithms.

Abstract. In recent years, Massively multiplayer online games (MMOGs) have been increasingly popular. Considering the limits of the traditional architectures, MMOGs based on the peer-to-peer networks have many advantages: scalability, computation cost, reliability and concurrency. Researchers have made improvements on this protocol to enhance its security since Chan et al. proposed an efficient and secure event signature (EASES) protocol, however, no one has solved this problem in essence. In 2013, Yuan et al. presented a higher secure scheme, unfortunately, its efficiency is lower than others due to bilinear pairing. In this paper, we propose a more efficient and secure protocol, the security of which is just guaranteed by the Diffie-Hellman problem. Meanwhile, it is concluded that the efficiency of ours also behaves better after the comparison with other existing protocols.

Introduction

MMOGs allow a multitude of players to act together concurrently in a virtual game world over the Internet\cite{1}. Many games such as Word of Warcraft, EVE Online, and Final Fantasy XI have shown that MMOGs are a thriving business industry\cite{2}. Players can play against each other or play in teams to complete some missions, which provides more entertainment than single-player games and attracts many people of different age, gender, and background to join\cite{3}. Currently the prevalent game architecture are client-server architectures, in which main functionalities of the virtual environment such as user identification and state management are achieved on the server\cite{4}. However, this architecture has the two limitations: low scalability and high costs of bandwidth and computation. When thousands of players are online simultaneously, MMOGs can produce huge network traffic and processing loads\cite{5}, making the server under great pressure. To solve this issue, the game company would over-provision a large number of servers\cite{6,7} at an unpredictable expense. A possible solution to the limitations is the use of peer-to-peer architectures\cite{8}. In 2004, an architecture using the peer-to-peer networking model to host MMOGs was proposed by Knutsson et al\cite{9}. The difference between them is that computation cost and network load in peer-to-peer architectures are distributed among peers, achieving high scalability\cite{10} and low cost. That is, peer-to-peer architectures are capable of solving problems prevalent in traditional architectures.

Although peer-to-peer architectures provide considerable support for scalability of MMOGs and other advantages over traditional client-server games, they still have to face a few key challenges\cite{11,12}. Security is probably the most important one of these challenges, which is caused by the fact that players’ communications are out of control of the server. Furthermore, a malicious player may take advantage of this point to attack others, so that it is vital for MMOGs to prevent cheating.

Traditional cryptographic techniques preventing cheating are to sign each update message of each round with players’ public and private key pairs, which requires a large amount of computation when players send and verify event update messages. For efficiency, Chan et al.\cite{13} proposed an efficient and secure event signature (EASES) protocol to sign a sequence of event update messages by applying one-time signature and the hash-chain keys. In EASES protocol, only the first signature is signed by the public-key cryptography and the following update messages are signed by hash-chain keys with hash function, which immensely reduces the computation cost compared to digital signatures. Chan et al. proposed another protocol called dynamic EASES based on the basic
The dynamic EASES protocol omits the pre-generation of hash-chain keys to reduce the costs of memory usage and preparation time. However, both methods do not solve the security problem existing in P2P-based MMOGs because the attacker can still forge event update messages except the first message signed by public-key cryptography. In 2009, Li et al. [14] made an improvement on the EASES protocol by adding a timestamp in sending event update messages. But this method is not practical. Later Li et al. proposed another scheme to prevent the replay attack by adding a unique game session number, while the forgery attack still exists[15]. In 2013, Yuan et al. proposed a new secure event signature protocol[16]. His protocol based on the discrete algorithms and bilinear pairing overcomes the forgery attack and replay attack, providing higher security service than other protocols introduced above. Owing to bilinear pairing requiring more computation cost, the efficiency is also higher than above protocols.

In this paper, we propose a new scheme which not only are secure but also makes communications process more efficient. Our protocol does not need to use public-key cryptography to sign any event update messages, and only hash operations are required when update messages are transmitted, enhancing the whole efficiency of our protocol.

The remainder of this paper is structured as follows. In Section 2 we introduce some preliminaries. We then review previous protocols and point out their main problems in Section 3. In Section 4, we propose an improved protocol and analyze its security and performance in Section 5. Section 6 is the conclusion.

Preliminaries

Discrete Logarithms. Let $G_1$ and $G_2$ be two groups of prime order $q$, let $e: G_1 \times G_1 \rightarrow G_2$ be a bilinear pairing, and let $g$ be a generator of $G_1$. The discrete logarithm (DL) problem can be expressed as follows:

Given $P, g \in G_1$, find $n \in \mathbb{Z}_q$ such that $p = g^n$.

Diffie-Hellman problem [17]. We suppose Alice and Bob wish to agree on a common secret key. They first need to agree on a large prime number $p$ and an integer $g$ with $2 \leq g \leq p - 2$. The prime $p$ and the primitive root $g$ can be publicly known. The specific agreement process is as follows.

- Alice chooses an integer $a \in \{0, 1, ..., p - 2\}$ randomly. She computes $A = g^a \mod p$ and sends the result $A$ to Bob, but she keeps the exponent $a$ secret.
- Bob chooses an integer $b \in \{0, 1, ..., p - 2\}$ randomly. He computes $B = g^b \mod p$ and sends the result to Alice. He also keeps his exponent $b$ secret.
- Alice computes $B^a \mod p = g^{ab} \mod p$ and Bob computes $A^b \mod p = g^{ab} \mod p$. Then the common key is $K = g^{ab} \mod p$.

Bilinear Pairing. Let $G_1$ be a cyclic additive group generated by $P$, whose order is a prime $q$, and $G_2$ be a cyclic multiplicative group with the same order $q$. A bilinear pairing is a map $e: G_1 \times G_1 \rightarrow G_2$ with the following properties:

- Bilinearity: $e(P^a, Q^b) = e(P^{a}, Q^{b}) = e(P, Q)^{ab}$ where $P, Q \in G_1$ and $a, b \in \mathbb{Z}_q^*$.
- Non-degenerative: There exists $P, Q \in G_1$ such that $e(P, Q) \neq 1$.
- Computable: There exists an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in G_1$.

Previous Protocols Analysis

In this section we will introduce two previous protocols: the EASES protocol and the protocol of Yuan et al., and then analyze the problems existing in them. The dynamic EASES and protocols of
Li et al. are similar to the basic EASES in essence, so that they are not introduced here. First, we define some notations mentioned in protocols' equation in Table 1. Now let us first introduce the EASES protocol in detail, which is the basis of other protocols.

Table 1. Explanation of notations

<table>
<thead>
<tr>
<th>notation</th>
<th>meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_i^r$</td>
<td>one-time signature player$_i$ generates in the rth round</td>
</tr>
<tr>
<td>$U_i^r$</td>
<td>event update message player$_i$ sends in the rth round</td>
</tr>
<tr>
<td>$x</td>
<td>y$</td>
</tr>
<tr>
<td>$H(x)$</td>
<td>hash operation of message $x$</td>
</tr>
<tr>
<td>$S_{sk}(x)$</td>
<td>signature operation of message $x$ with secret key $sk$</td>
</tr>
<tr>
<td>$D_{pk}(x)$</td>
<td>decryption operation with corresponding key $pk$</td>
</tr>
<tr>
<td>$\delta_i^r$</td>
<td>signature signed by the rth one-time signature of player$_i$</td>
</tr>
<tr>
<td>$\Delta_i$</td>
<td>signature signed by secret key of player$_i$</td>
</tr>
</tbody>
</table>

The EASES protocol. In 2008, Chan et al. proposed a novel idea that just uses one-time signature and a series of hash-chain keys to achieve the same cheat-proof properties of digital signatures. In specific realization player$_i$, first chooses a random number as master key $MK_i$ to compute a series of signature keys. Then only the first one-time signature key will be signed by player$_i$'s private key, and other keys are based on the relationship of the hash-chain keys. So minimal digital signature operations enhance the protocol's efficiency. The EASES protocol has four phases: initialization phase, signature phase, verification phase and re-initialization phase. Actually, the last phase is not necessary and not described in our paper.

Initialization Phase. a) Player$_i$ first chooses a random number as the master key $MK_i$ and then computes a series of one-time signature keys $K_i^n = H(MK_i)$ and $K_i^{r-1} = H(K_i^r)$. The computation process is shown in Fig. 1.

$$MK_i \xrightarrow{H(MK_i)} K_i^n \xrightarrow{H(K_i^n)} K_i^{r-1} \xrightarrow{H(K_i^{r-1})} K_i^0$$

Fig. 1. Production process of hash chain keys.

b) Sign the first one-time signature key $K_i^0$ by player$_i$'s private key and broadcast the signature $\Delta_i = S_{sk}(K_i^0)$ to others. Note that the hash-chain keys are used in the reverse order from $K_i^0$ to $K_i^n$. Because of the one-way property of hash functions, it is impossible to figure out $K_i^1$ to $K_i^n$ from $K_i^0$.

Signature Phase. In the first round, player$_i$ sends the message $\delta_1^i = H(K_i^1 | U_i^1), \Delta_i, K_i^0$ to other players. In the rth round, player$_i$ sends the message $\delta_i^r = H(K_i^r | U_i^r), U_i^{r-1}, K_i^{r-1}$ to other players. Eq. 1 shows the messages sent by player$_i$.

$$\begin{align*}
\delta_1^i &= H(K_i^1 | U_i^1), \Delta_i, K_i^0 \\
\delta_i^r &= H(K_i^r | U_i^r), U_i^{r-1}, K_i^{r-1}
\end{align*}$$

Verification Phase. Player$_j$ who receives the event update messages from player$_i$ should do the following operations to verify them.

a) In the first round, player$_j$ should verify the legitimation of $K_i^0$. If the equation $K_i^0 = D_{pk}(\Delta_i)$ holds, it declares $K_i^0$ is legitimate.
b) In the rth round, player\textsubscript{j} first verify the legitimation of \(K_{i}^{r-1}\) by the equation \(K_{i}^{r-2} = H(K_{i}^{r-3})\) where \(K_{i}^{r-2}\) is received in the former round. Then \(\delta_{i}^{r-1} = H(K_{i}^{r-1} \mid U_{i}^{r-1})\) is verified to see whether the update message is modified or not. If all verification passes, it shows the update message can be accepted as valid.

A main problem residing in this protocol is that the receiver cannot know whether the messages are modified or not. We know \(K_{i}^{0}\) is signed by player\textsubscript{i}'s private key and it can be verified by receivers. Consequently, \(K_{i}^{1}, ..., K_{i}^{n}\) can also be verified because the relationship of the hash-chain keys. However, the attacker can modify the verification message such as the hash value to satisfy event update messages that have been modified by himself. Receivers then know nothing about what has happened.

Suppose player\textsubscript{k} is an attacker who wants to forge the update messages sent to player\textsubscript{j} from player\textsubscript{i}. In the first round, player\textsubscript{i} sends the first message \(\delta_{i}^{1} = H(K_{i}^{1} \mid U_{i}^{1}), \Delta_{i}, K_{i}^{0}\) to player\textsubscript{j}. Player\textsubscript{k} can intercept it and record \(K_{i}^{0}\). When player\textsubscript{i} sends the second message \(\delta_{i}^{2} = H(K_{i}^{2} \mid U_{i}^{2}), U_{i}^{1}, K_{i}^{1}\) to player\textsubscript{j}, player\textsubscript{k} can intercept it and record \(K_{i}^{1}\). Player\textsubscript{k} then replaces \(U_{i}^{1}\) with a new update message \(U_{i}^{r}\), computes \(H(K_{i}^{1} \mid U_{i}^{r})\) and sends message \(\delta_{i}^{3} = H(K_{i}^{1} \mid U_{i}^{r}), \Delta_{i}, K_{i}^{0}\) to player\textsubscript{j}. In the same way, when player\textsubscript{i} sends the rth message \(\delta_{i}^{r} = H(K_{i}^{r} \mid U_{i}^{r}), U_{i}^{r-1}, K_{i}^{r-1}\) to player\textsubscript{j}, player\textsubscript{k} can intercept it, record \(K_{i}^{r-1}\) and send a forged message \(\delta_{i}^{r} = H(K_{i}^{r-1} \mid U_{i}^{r-1}), U_{i}^{r-2}, K_{i}^{r-2}\) to player\textsubscript{j}. Finally, player\textsubscript{j} receives all the forged messages \(U_{i}^{r}, ..., U_{i}^{r}\) and do not know that the attacker player\textsubscript{k} has modified event update messages.

**The Protocol of Yuan et al.** In 2013, Yuan et al. proposed a higher secure protocol whose security basis are the discrete logarithms and bilinear pairing. However, the efficiency of this protocol is lower because of bilinear pairing.

**Initialization Phase.** a) Player\textsubscript{i} chooses a master key \(x\) and computes a series of one-time signature keys \(K_{i}^{n} = H(x), K_{i}^{n-1} = H(K_{i}^{n})\).

b) Compute \(P_{pub} = g^{x}\), where \(g\) is the generator of cyclic additive group. Player\textsubscript{i} then broadcasts \(P_{pub}\) to other receivers.

c) Sign the first one-time signature with his or her private key and broadcast \(\Delta_{i} = S_{ym}(ID_{i} \mid K_{i}^{n})\), where \(ID_{i}\) denotes the identity of player\textsubscript{i}.

**Signature Phase.** In the first round, player\textsubscript{i} sends the message \(\delta_{i}^{1} = H(K_{i}^{1} \mid U_{i}^{1} \mid ID_{i} \mid gno \#)^{x}, ID_{i}, \Delta_{i}, K_{i}^{0}, gno \#\) to other players. \(gno \#\) is not a duplicate value, and different session has different \(gno \#\). In the rth round player\textsubscript{i} sends message \(\delta_{i}^{r} = H(K_{i}^{r} \mid U_{i}^{r} \mid ID_{i} \mid gno \#)^{x}, ID_{i}, U_{i}^{r-1}, K_{i}^{r-1}, gno \#\) to other players. Eq. 2 shows the messages sent by player\textsubscript{i}.

\[
\begin{align*}
\delta_{i}^{1} &= H(K_{i}^{1} \mid U_{i}^{1} \mid ID_{i} \mid gno \#)^{x}, ID_{i}, \Delta_{i}, K_{i}^{0}, gno \# \\
\delta_{i}^{r} &= H(K_{i}^{r} \mid U_{i}^{r} \mid ID_{i} \mid gno \#)^{x}, ID_{i}, U_{i}^{r-1}, K_{i}^{r-1}, gno \#
\end{align*}
\]

**Verification Phase.** In the first round, player\textsubscript{j} should verify the equation \(ID_{i} \parallel K_{i}^{0} = D_{pk}(\Delta_{i})\). If the equation holds, it declares \(K_{i}^{0}\) is legitimate. In the rth round, player\textsubscript{j} first verifies the legitimation of \(K_{i}^{r-1}\) by the equation \(K_{i}^{r-2} = H(K_{i}^{r-3})\). Then player\textsubscript{j} verifies \(e(\delta_{i}^{r-1}, g) = e(H(K_{i}^{r-1} \mid U_{i}^{r-1} \mid ID_{i} | U_{i}^{r-1} \mid K_{i}^{r-1} \mid gno \#)\).
gno #), $P_{pub}$) to see whether the event update message is modified or not, where $\delta_{t}^{t\rightarrow t}$ is received in the former round. If all verification passes, it shows the update message is integrated.

According to the bilinear pairing, we know
\[ e(\delta_{t}^{t\rightarrow t}, g) = e(H(K_{i}^{t\rightarrow t} | U_{i}^{t\rightarrow t} | ID_{i} | gno #)^{x}, g) = e(H(K_{i}^{t\rightarrow t} | U_{i}^{t\rightarrow t} | ID_{i} | gno #), g^{x}) = e(H(K_{i}^{t\rightarrow t} | U_{i}^{t\rightarrow t} | ID_{i} | gno #), P_{pub}), \]

and each player can verify event signature by the public parameter $P_{pub}$ but cannot re-compute it without the random number $x$. The only way to obtain it is to compute it by the public parameter $x^{pub} = g^{x}$, which means that its difficulty equals to solving the DL problem. So this protocol is much safer than previous protocols. Whereas, the security is achieved at the expense of its efficiency.

**Our Proposed Protocol**

**Design Principles.** We find in the EASES protocol all information to compute hash verification is directly transmitted in public form, and therefore it is easy for attackers to forge corresponding hash values of fake messages. This process is transparent to the receiver, for those fake update messages can pass his or her verification. To prevent this attack, some information must be transmitted in a private way. The protocol of Yuan et al. solves the security problem with discrete logarithms and bilinear pairing. However, just as what he says in paper, the efficiency is not good enough due to bilinear pairing. In order to enhance efficiency under the premise of security, we make any two players agree on a common secret key through key agreement protocol before sending the first update message. This secret key is used to sign later update messages for each round between them. Because it is not directly transmitted in the whole process, attackers cannot work out the key value with intercepted messages, ensuring good security. Moreover, only hash operations are required on both sides when update messages are sent, which enhances the whole efficiency.

**Our Protocol.** In our protocol, Diffie-Hellman key agreement protocol is selected to help any two players agree on a common secret key, which has been introduced in Section 2. To prevent negotiation process from being attacked by intermediary when they send their public keys to others, every player's public key will be put on central server when he or she joins the P2P networks. A sender can securely obtain a receiver's public key from central server, and then compute a secret key with his or her own private key. Simultaneously, the receiver can also figure out the same key in the similar way. Communications security of later update messages between them will be guaranteed by this common key. Our protocol has three phases: initialization phase, signature phase and verification phase. The specific process is as follows.

**Initialization Phase.** a) Suppose player$_{j}$ is one of receivers, player$_{i}$ first gets player$_{j}$'s public key $pk_{j} = g^{xj} \mod p$ from the central server and computes a secret key with his or her own private key. This common secret key between them denotes $sk(i, j) = pk_{j}^{xi} \mod p = g^{xi\times xj} \mod p$. $p$ and $g$ are publicly known for Diffie-Hellman key agreement protocol.

b) Player$_{i}$ chooses a random number as the master key $MK_{i}$ and computes a series of one-time signature keys $K_{i}^{0} = H(MK_{i})$ and $K_{i}^{n} = H(K_{i}^{n})$. The hash-chain keys are used in the reverse order from $K_{i}^{0}$ to $K_{i}^{n}$. Player$_{i}$ then sends $\Delta_{i} = H(K_{i}^{0} | sk(i, j))$ to player$_{j}$.

**Signature Phase.** When player$_{i}$ sends event update messages to player$_{j}$, he or she should do the following operations:

a) In the first round, player$_{i}$ sends the message $\delta_{i}^{1} = H(K_{i}^{1} | U_{i}^{1} | sk(i, j) | gno #), \Delta_{i}, K_{i}^{0}, gno # to player$_{j}$. gno# is the game session number.

b) In the $r$th round, player$_{i}$ sends the message $\delta_{i}^{r} = H(K_{i}^{r} | U_{i}^{r} | sk(i, j) | gno #), U_{i}^{r\rightarrow r}, K_{i}^{r\rightarrow r}, gno # to player$_{j}$. Eq. 3 shows the messages sent to player$_{j}$.
\[
\begin{align*}
\delta_i^t &= H(K_{i}^t | U_{i}^t | sk(i, j) | \text{gno} \#), \Delta_i, K_{i}^0, \text{gno} \# \\
\delta_i^r &= H(K_{i}^r | U_{i}^r | sk(i, j) | \text{gno} \#), U_{i}^{r-1}, K_{i}^{r-1}, \text{gno} \#
\end{align*}
\]

**Verification Phase.** First, \(player_j\) figures out \(sk(i, j)\) in the same way as \(player_i\) does. This value \(sk(i, j) = g^{x_{i}^{r_{x_{j}}}} \mod p\) is the same as \(player_i\’s.\)

(a) In the first round, \(player_j\) should verify the legitimacy of \(K_i^0\). He or she verifies it through the equation \(H(K_i^0 | sk(i, j)) = \Delta_i\). If the equation holds, \(player_j\) can be sure \(K_i^0\) is legitimate.

(b) In the \(r\)th round, \(player_j\) first verifies \(K_{i}^{r-2} = H(K_{i}^{r-1})\) to see if the signature key \(K_{i}^{r-1}\) is legitimate, where \(K_{i}^{r-2}\) is received in the \((r-1)\)th round. If the equation holds, \(player_j\) next uses \(K_{i}^{r-1}, U_{i}^{r-1}, sk(i, j), \text{gno} \#\) to re-compute a hash value to verify whether it equals to the received value \(\delta_i^{r-1}\) or not. If it holds, \(player_j\) can be sure that the update message is from \(player_i\) and has not been modified. Fig. 2 shows the main process of this protocol.

| Player_i | \(\delta_i^0, \Delta_i, K_i^0, \text{gno} \#\) | verify \(K_i^0\) by \(H(K_i^0 | sk(i, j)) = \Delta_i\) |
|---------|---------------------------------|---------------------------------|
| \(\delta_i^1, U_i^1, K_i^1, \text{gno} \#\) | verify \(U_i^1\) by \(H(K_i^1) = K_i^0\) and \(\delta_i^0 = H(K_i^0 | sk(i, j) | \text{gno} \#)\) |
| \(\delta_i^2, U_i^2, K_i^2, \text{gno} \#\) | verify \(U_i^2\) by \(H(K_i^2) = K_i^1\) and \(\delta_i^1 = H(K_i^1 | U_i^1 | sk(i, j) | \text{gno} \#)\) |
| \(\vdots\) | \(\vdots\) | \(\vdots\) |
| \(\delta_i^{r-1}, U_i^{r-1}, K_i^{r-2}, \text{gno} \#\) | verify \(U_i^{r-1}\) by \(H(K_i^{r-1}) = K_i^{r-2}\) and \(\delta_i^{r-2} = H(K_i^{r-2} | U_i^{r-2} | sk(i, j) | \text{gno} \#)\) |
| \(\delta_i^r, U_i^r, K_i^{r-1}, \text{gno} \#\) | verify \(U_i^r\) by \(H(K_i^r) = K_i^{r-1}\) and \(\delta_i^{r-1} = H(K_i^{r-1} | U_i^{r-1} | sk(i, j) | \text{gno} \#)\) |
| \(\vdots\) | \(\vdots\) | \(\vdots\) |

Fig. 1. The process of our protocol

**Security And Performance Analysis**

**Security Analysis.** Our protocol can prevent the forgery attack and replay attack. In the EASES protocol, event update messages are transmitted in the public form, so that it is possible for the attacker to construct new ones and forge fake signatures with update messages intercepted. For example, a sender sends two messages \(\delta_i^r = H(K_i^r | U_i^r)\), \(U_i^{r-1}, K_i^r\) in the \(r\)th round and \(\delta_i^{r+1} = H(K_i^{r+1} | U_i^{r+1}), U_i^r, K_i^r\) in the \((r+1)\)th round. The attacker can intercept both of them and get \(U_i^r, K_i^r\). Then he or she could generate a new signature \(\delta_i^{r+1} = H(K_i^r | U_i^r)\) with fake \(U_i^r\), and sends the messages \(\delta_i^{r+1} = H(K_i^r | U_i^{r+1}), U_i^{r-1}, K_i^{r+1}\) and \(\delta_i^{r+1} = H(K_i^{r+1} | U_i^{r+1}), U_i^r, K_i^r\) to the receiver. \(U_i^r\) will be accepted as valid. This process is transparent to the receiver. However, in our protocol, even if \(U_i^r, K_i^r\) and \(\text{gno} \#\) are public to the attacker when the \((r+1)\)th round message \(\delta_i^{r+1} = H(K_i^{r+1} | U_i^{r+1} | sk(i, j) | \text{gno} \#), U_i^r, K_i^r, \text{gno} \#\) is intercepted, the attacker cannot also forge a verification \(\delta_i^{r+1}\) for \(U_i^r\) because this secret key \(sk(i, j)\) is invisible. Therefore, the attacker's forged update message \(U_i^r\) cannot pass the receiver's verification. The forgery attack is prevented. Another way for the attacker
to try is to gain private key $x$ of the sender by his or her public key $pk = g^x \mod p$, with $g$ and $p$, whose difficulty equals to solving the Diffie-Hellman problem.

In our protocol, each round has a different one-time signature key, and these keys can be verified by the relationship $K_i^{-1} = H(K_i')$, guaranteeing that messages replayed by an attacker are not accepted as valid. Simultaneously, $gno\#$ is not a duplicate value representing one-time session, and different session has different $gno\#$. It is used to prevent an attacker from collecting the messages that the normal player has sent in a whole session, and then replaying them to impersonate that player. Thus, the replay attack is also prevented by our protocol.

**Performance Analysis.** In the initialization phase, there are many time-consuming one-time operations for all protocols. Because they won’t be operated when event update messages are sent, we don’t consider them here.

<table>
<thead>
<tr>
<th>Protocols</th>
<th>Our protocol</th>
<th>Basic EASES</th>
<th>Protocol of Yuan et al.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature phase</td>
<td>$(N-1)hash$</td>
<td>$\text{Hash}$</td>
<td>$\text{Hash} + \text{1ep}$</td>
</tr>
<tr>
<td>Verification phase</td>
<td>$(N-1)hash$</td>
<td>$(N-1)hash$</td>
<td>$2(N-1)bp + (N-1)hash$</td>
</tr>
<tr>
<td>Total cost</td>
<td>$2(N-1)hash$</td>
<td>$N\text{hash}$</td>
<td>$2(N-1)bp + N\text{hash} + \text{1ep}$</td>
</tr>
</tbody>
</table>

Table 2 shows what operations a player need to do for each round, in which $N$ denotes the number of players, $\text{hash}$ denotes one hash operation, $bp$ denotes one bilinear pairing operation and $ep$ denotes one exponent operation. From the comparisons, we can see that in the signature phase, our protocol computes $N-1$ times hash operations instead of one like basic EASES, because any two players have a different secret key. And the protocol of Yuan et al. requires one hash operation and one exponent operation in this phase. When a player’s update message is verified in the verification phase, the basic EASES protocol and ours just need to compute one hash operation, however, besides that, Yuan et al.’s has to compute two additional more time-consuming pairing operations.

\[
\begin{align*}
    a &= \frac{2(N-1)hash}{N\text{hash}} \approx 2 \\
    b &= \frac{2(N-1)bp + N\text{hash} + \text{1ep}}{2(N-1)hash} \approx \frac{bp}{\text{hash}}
\end{align*}
\]

According to total cost and Eq. 4, the computation cost of our protocol is about twice that of the basic EASES protocol, and much less than Yuan et al.’s. In Intel i3 processor of Yuan et al. [16], executing 10 hash operations needs about 1ms and 200ms of a pairing operation. To make the result of comparison more clear, Fig. 3 shows the change of one player’s CPU time usage in each round for different protocols as the number of players increases. From this figure we can see the computation cost of Yuan et al. is far higher than the basic EASES protocol and ours.
Fig. 2. A comparison of total CPU time usage in different protocols.

Conclusions
In this paper, we first introduce several previous event signature protocols based on peer-to-peer architectures and then point out their problems about security and efficiency. To have a better efficiency with security, we propose an improved one with the Diffie-Hellman key agreement protocol. Our protocol has achieved the two basic requirements of unforgeability and verifiability, and is capable of preventing the forgery attack and replay attack. Moreover, when event update messages are transmitted, only hash operations are required, which makes the efficiency of our protocol close to the original one and better than Yuan et al.’s. Meanwhile, the security is guaranteed by the Diffie-Hellman problem. In the following work, we plan to improve this protocol considering its fair retransmission mechanism and better real-time.

Acknowledgements
This work was financially supported by the European Seventh Framework Program (FP7) (GA-2011-295222) and the National Sci-Tech Support Plan of China (2014BAH02F03).

References