An Efficient and Secure Event Signature (EASES) Protocol for Peer-to-Peer Massively Multiplayer Online Games [⋆]

Mo-Che Chan[1], Shun-Yun Hu[2], and Jehn-Ruey Jiang[3]

[1] National Central University, Chung-Li, Taiwan 32054.
chrysler@acnlab.csie.ncu.edu.tw
[2] National Central University, Chung-Li, Taiwan 32054. syhu@yahoo.com
[3] National Central University, Chung-Li, Taiwan 32054. jrjiang@csie.ncu.edu.tw

Abstract. In recent years, *massively multiplayer online games* (MMOGs) have become very popular by providing more entertainment and sociability than single-player games. In order to prevent cheaters to gain unfair advantages in peer-to-peer (P2P)-based MMOGs, several cheat-proof schemes have been proposed by using digital signatures. However, digital signatures generally require large amount of computations and thus may not be practical to achieve real-time playability. We propose an *Efficient and Secured Event Signature* (EASES) protocol to efficiently sign discrete event messages. Most messages need only two hash operations to achieve non-repudiation and event agreement. The computation, memory, and bandwidth consumptions of EASES are low, which makes it applicable to P2P-based MMOGs.

1 Introduction

Multiplayer online games are a rapidly growing segment of Internet applications in recent years. By providing more entertainment and sociability than single-player games, it is fast becoming a major form of digital entertainment. Earlier multiplayer games adopt client-server architectures where all players establish connections with the server to send and receive *event updates*. However, a single server cannot support a large numbers of concurrent players at the same time, so server-clusters were subsequently introduced to enable *massively multiplayer online games* (MMOGs) where concurrent users may reach into the range of hundreds of thousands [1]. Current MMOGs adopt server-cluster architectures to provide better scalability than earlier architectures. However, a server-cluster has only limited amount of resources such as CPU and bandwidth at a given time and poses as a single point of failure. A distributed approach to MMOGs thus may be more scalable if millions of concurrent users were to be supported. A number of recent *peer-to-peer* (P2P) *virtual environment* (VE) [2–9] research thus seeks to further improve the scalability of existing MMOGs. The key is

[⋆] This research was supported in part by the National Science Council of the Republic of China under the grant NSC95-2221-E-008-048-MY3 to correctly and efficiently maintain the topology of all participating peers by solving a *neighbor discovery problem* [9]. Unlike earlier fully-connected P2P systems where the number of connections grow exponentially with users. In scalable P2P-based VEs, it is unnecessary to establish contacts between every pairs of player nodes but only with those that are within a player's visibility range.

Although P2P-based MMOG may provide better scalability, by distributing server-side workload to clients, new issues such as maintaining consistency and ensuring fairness are introduced [2,10,11]. In server-based MMOGs, *game states* (e.g. user status, experience points, equipments, world items) are maintained and updated via *game logic* executions on trusted servers. When a player makes an action such as running, shooting, turning, etc., an event message is sent to the server, which subsequently processes the events and updates the game states. The server then periodically sends updated states to relevant players to keep their local copies of the game states synchronized with the server's. Under such event processing model, the server maintains all the information to ensure a global ordering of event executions and fair gameplay. However, when we turn to a P2P approach of MMOG, game states maintenance and game logic execution may be distributed to each individual players, creating opportunities for players to cheat.

A player may gain various advantages by cheating in multiplayer games. It is especially attractive as gaining valuable items or winning over other players is central to the gameplay value in commercial MMOGs. Valuable virtual items can even be sold in exchange for real-world money, which increases the motivations for cheating. Therefore, cheat-prevention is very important to game designers.

To prevent cheating, games may adopt cryptographic techniques. Several cheat types [11] occur when the actions of non-cheating players are known in advance by cheaters, who can then respond unfairly to their advantages (i.e. similar to in a real-life bridge game, if certain players deal their cards only after knowing what cards the opponents will choose, then they could have unfair advantages). Cryptographic techniques have been proposed to prevent players' actions from being known by others before each player submits the final decisions [10]. A potential countermeasure is thus the adoption of a secured event updating protocol that includes 1) a *commitment scheme* to ensure that players do not change their actions after decisions are made; and 2) a *digital signature scheme* to ensure that players cannot deny the actions they have done. If the commitments are digitally signed, we can also prevent impersonation and dodging [12,13]. Commitment and digital signatures together therefore provide a fair environment for games even when players do not trust each other [12, 13], as signatures prevent any identity forging and commitments prevent a player from changing the action decisions that are already made. However, there exists some time-consuming exponential operators in public-key cryptography for signing signatures, so significant computational power is required to sign and verify all event updates.

In existing cryptography schemes, when a large continuous document needs to be signed, a *hash function* can be used instead of digital signatures to reduce the computations. The document is first hashed into a much smaller *digest*, which is then signed by a digital signature. The digest is just one short message, but it can be used to characterize the original message. As it is very hard to find another document that can produce the same digest, signing the digest can be seen as signing the original document. *Unforgeability* and *verifiability* are the two requirements of digital signatures currently achievable only by public-key cryptography. As digital signature is necessary to achieve *non-repudiation* (i.e. undeniableness of user actions) due to its unforgeability and verifiability, some event update protocols [11, 14] have been proposed to use digital signatures to sign every event messages to avoid cheating in P2P-based MMOGs. A player cannot repudiate his/her signature after signing a message under these two properties. However, we cannot treat the the many discrete event messages in MMOGs as a large document. How to sign many discrete event updates so that unforgeability and verifiability can still be achieved without too much computation is thus our main concern. We propose an *Efficient and Secured Event Signature* (EASES) protocol to efficiently sign many event update messages. As we will show, EASES has low computation costs and consumes little memory or bandwidth. It is thus applicable to P2P-based MMOGs.

In the following sections, we first review related work for cheat-proof mechanisms in Section 2 and present a message transformation model in Section 3. We propose the efficient one-time signature scheme in Section 4, and evaluate its security and performance in Section 5. The paper is concluded in Section 6.

2 Related Work

Cheating in multiplayer games have been described by several papers. Yan [15] examines several security requirements that impact the design of online games by using a simple client-server bridge game. Kirmse and Kirmse [16] present the security goals for online games in areas such as the protection of sensitive information and the provision of a fair playing field. Smed et al. [17] describe issues in multiplayer games such as network resources, communication architectures, scalability and security. Besides the descriptions of cheats, several cheat-proof mechanisms have also been proposed. GauthierDickey et al. proposed the *New Event Ordering* (NEO) protocol [11] to improve on the work of Baughman and Levine [10] and Cronin et al. [18]. NEO claims to prevent common protocol-level cheats with low latency, such that the adversaries cannot gain any advantages by modifying messages between players. Corman et al. [14] later show that NEO cannot prevent all cheats as claimed, and present an improvement called *Secure Event Agreement* (SEA). However, both protocols are not practical due to the excessive use of cryptographic signatures. As shown by the NESSIE project [19], digital signatures consume much more computations than hash functions and symmetric encryptions. However, the use of digital signatures is unavoidable to prevent users from denying their behaviors. A key question thus is *how to use* *digital signature only minimally while achieving the same cheat-proof properties.* As we seek to present a more efficient protocol based on NEO and SEA, the two protocols are first examined below.

2.1 Description of NEO

GauthierDicky et al. propose NEO to avoid cheating by adding a voting mechanism to compensate for packet loss in the environment [11]. NEO divides the time into fixed-length *rounds*, where every player sends an event update in each round. The event update of NEO is given in Formula (1), where $\{\}_{K_A^r}$ represents encryption, $S_A()$ is the signature function, U_A^r is the update from player A for round r, K_A^{r-1} is A's key for the update from round $r-1$, and V_A^{r-1} is a bit vector for voting. The voting is used to form a consensus on whether a given player has sent an update within a round, in order to determine if an update by that player should be accepted. When each player acts, an event update is signed and encrypted before sending to other players. The encrypted event update and its signature serve as a *commitment*, which is revealed in the next round when the player sends the key. So players cannot modify their own actions after the commitments are sent (i.e. after they have learned of others' actions).

$$M_A^r = \{S_A(U_A^r)\}_{K_A^r}, K_A^{r-1}, S_A(V_A^{r-1}) \tag{1}$$

However, Corman et al. show there are some problems in NEO (e.g. an attacker can replay updates for another player, or send different updates to different opponents [14]) and present an improvement called SEA.

2.2 Description of SEA

Corman et al. [14] present an improvement of NEO as described in Formula (2), where $H()$ is a hash function, U_A^r is the update from player A for round r, n^r is a random value for round r, $SessID$ is the session ID to prevent replaying this message in a different session or with a different group of players, ID_A is the unique identity of player A, and Vh_A^{r-1} is the update from player A for round $r-1$ that includes a hash of the update.

$$\begin{cases} Commit_A^r = H(U_A^r, n^r, SessID, ID_A), \\ M_A^r = S_A(Commit_A^r, U_A^{r-1}, Vh_A^{r-1}, n^{r-1}, r) \end{cases} \tag{2}$$

In order to achieve better performance and remove potential issues with key tampering and selection, the SEA protocol replaces encryption with a cryptographic hash function as the commitment method. When commitment is done by encryption, the distribution, protection and selection of keys must be carefully considered, or problems would arise from poorly designed key management schemes. However, if the commitment is made via hash functions, the above key-related issues can be avoided.

Signing the entire message is used to authenticate the message creator, and the hash serves to commit the player to the message sent in the next round. Every player thus is forced to accept the actions already submitted. Player signatures can also be checked to validate the players.

3 Message Transformation Model

We seek to achieve both scalability and fairness for P2P-based MMOGs, the proposed event signature protocol thus should be usable independent of the underlying network topologies. Our method can also be treated simply as a more efficient signature scheme. As public-key cryptosystem requires a large amount of computations and time, to improve the efficiency, we compute the message's digest by a hash function before signing the message. By using a digest, not only existential-forage is prevented, computation time is also greatly saved. The original signature and its message is described in Formula (3).

$$\begin{cases} M & \text{the full message,} \\ H(M) & \text{digest of } M, \\ S_{sk}(H(M)) & \text{signature of message digest.} \end{cases} \tag{3}$$

Although the above method is efficient for one message, it cannot sign many discrete event updates across different time periods (i.e. rounds) as required in a game scenario. We therefore try to find an efficient signature protocol usable under such environment. The proposed efficient signature for discrete messages adapts to different networks such as fully-connected or scalable P2P topologies, and is described in Formula (4).

$$\begin{cases} M_1, M_2, \ldots, M_i, \ldots, M_n & \text{discrete messages,} \\ S_{sk}(M_i) & \text{signatures for message } M_i. \end{cases} \tag{4}$$

To prevent other attacks such as eavesdropping using a network sniffer, authenticated key exchange protocol [20] can be used to agree the session keys to encrypt data between every pair of transmission parties. Adding a counter number to the original message before encryption can also prevent eavesdropping, modification and fabrication attacks. However, this topic is beyond our discussion.

4 The Proposed Scheme

One-time signature was proposed in 1981 [21] to authenticate remote users on a computer network. In order to sign the discrete event updates efficiently, we propose a one-time signature variant to achieve the same effects as the regular signature scheme.

There are four phases in EASES: 1) Every player generates the keys for signing event updates in the *initialization phase*. 2) Every player signs his/her event update in the *signing phase*. 3) After the event updates are received from other players, each player can verify these event updates in the *verification phase*. 4) In the *re-initialization phase*, players can re-generate new signature keys when the keys have been used up. The notations are listed in Table 1.

Table 1. Notations

$Player_i$	each player in the game, where $i \in \{1..m\}$
$H(x)$	hash operator with input message x
OSK_i^j	$player_i$'s j^{th} one-time signature key
$S_{sk}(x)$	message signing x by secret key sk
$x\|y$	concatenation of the message x and y
δ_i^j	signature signed by $player_i$'s j^{th} OSK
Δ	signature signed by secret key sk

4.1 Initialization Phase

Before starting the game, each player must first generate his/her one-time signature keys. The operation of generating a list of one-time signature keys can be done before a user logins the game, or before the first event update is sent.

Basis: For each player $player_i$, an unpredictable random one-time master key MK_i is first picked to compute a series of the player's one-time signature keys $OSK_i^n = H(MK_i)$, where n is a system parameter that specifies the maximum number of times for signing updates by each player. Choosing larger n may save computation time during the later updating stage, but it may also increase the time to compute the one-time signature keys.

Induction: Every player subsequently computes the one-time signature keys $OSK_i^j = H(OSK_i^{j-1})$, where $j \in 1..n-1$ is a one-time signature key.

After these one-time signature keys (i.e. a *hash chain*) are generated, every player signs the first one-time signature key by the player's secret key $\Delta = S_{sk}(OSK_i^1)$. Those keys are then stored in the players' computer.

$$
\begin{aligned}
\text{Master key} &= MK_i \\
OSK_i^n &= H(MK_i) \\
OSK_i^{n-1} &= H(OSK_i^n) \\
OSK_i^{n-2} &= H(OSK_i^{n-1}) \\
.... &= .... \\
.... &= .... \\
OSK_i^2 &= H(OSK_i^3) \\
OSK_i^1 &= H(OSK_i^2) \\
\Delta_i &= S_{sk}(OSK_i^1)
\end{aligned}
$$

Fig. 1. Generating OSKs in the initialization phase

4.2 Signing Phase

The player computes the signature of the event updates $\delta_i^j = H(OSK_i^j|M_i^j)$, where M_i^j is the $player_i$'s j^{th} event update. During each round, each player sends an event update and its signature $M|\delta_i^j$ to other players. Note that if the situation requires (e.g. to re-initialize another list of one-time signature keys), one should not use up all the keys, but to at least reserve the last two keys OSK_i^n and OSK_i^{n-1}. Each round, the signature of the event update $player_i$ has to send is shown in Formula (5).

$$\begin{cases} \Delta_i, \delta_i^1 = H(OSK_i^1|M_i^1) & \text{the first round,} \\ \delta_i^j = H(OSK_i^j|M_i^j) & \text{the following rounds.} \end{cases} \tag{5}$$

For instance, $player_i$ sends the signature $\delta_i^1 = H(OSK_i^1|M_i^1)$ in the first round. In the second round, $player_i$ sends $\delta_i^2 = H(OSK_i^2|M_i^2), M_i^1, OSK_i^1$, and Δ_i. In subsequent rounds, $player_i$ sends $\delta_i^j = H(OSK_i^j|M_i^j), M_i^{j-1}$, and OSK_i^{j-1}.

4.3 Verification Phase

When each player receives other players' event updates, it is necessary to verify those messages. Each player receives $\delta_i^1 = H(OSK_i^1|M_i^1)$ in the first round, and then receives and verifies the signature Δ_i in the second round. Subsequently, each player receives and verifies $H(OSK_i^j|M_i^j) \stackrel{?}{=} \delta_i^j$ in the $(j+1)^{th}$ round. Additionally, the signature Δ_i also needs to be verified when the first update is received.

4.4 Re-initialization Phase

The signature keys have to be re-generated when they are used up. One basic method is for players to re-execute the initialization phase to generate new signature keys. However, a more efficient way exists if we assume that the last two keys are reserved: a player first generates the new one-time signature keys $NewOSK_{1..n}$ and then signs the head of the new signature key by the previous signature key $\Delta_{reinit} = H(OSK_i^{n-1}|NewOSK_i^1)$.

5 Evaluation

5.1 Security Analysis

Unforgeability and verifiability are the two fundamental requirements for digital signatures. Unforgeability means that no one can generate a legal signature for a specific message except the signer. Signer is the only one who keeps the correct private key for generating a legal signature verifiable by the corresponding public key. Verifiability means that every one can verify whether a digital signature is legal. We thus evaluate EASES according to these two requirements.

Unforgeability. One cannot claim that he has signed M_i^j to get the signature δ_i^j unless he can present OSK_i^j. No one can show that he knows the signing key

OSK_i^j for the current message M_i^j before the original signer reveals it in the next round. The cryptographic hash function also has the following secured properties: for any given hashed value, it is computationally infeasible to find its pre-image due to the one-way property of hash functions. For any given message x, it is computationally infeasible to find another message that has the same hash value, such that $H(x) = H(x')$. Moreover, strong collision resistance property exists in some hash functions, meaning that it is computationally infeasible to find any pair of (x, y) such that $H(x) = H(y)$. EASES is unforgeable since it adopts a secured hash function that has the properties mentioned above. With unforgeability, non-repudiation can be achieved in EASES.

Verifiability. Cryptographic hash function is a public standard that can be installed and executed on every player's computers, so that everyone can re-compute the hash value of a given signature key OSK_i^j and message M_i^j to verify if it equals to the received signature $H(OSK_i^j|M_i^j) \overset{?}{=} \delta_i^j$.

5.2 Performance Analysis

Computational cost. The major benefit of the proposed scheme is computational efficiency. There is at least one signature operator and one verification operator for each event update in the existing schemes (e.g. RSA, DSA, etc.). In comparison, there are only three hash operators for each event update in EASES: two for signing and one for verifying, and one traditional signature operator during the initialization phase in EASES. The approximate CPU cycles for some cryptographic functions are listed in Table 2.

Table 2. Approximate CPU cycles for some cryptographic functions [14, 19]

Primitive type	Example	Clock cycles
Hash function	SHA-1	15/byte + 1040
Symmetric encryption	AES	25/byte + 504
Digital signature	RSA-PSS	42,000,000

Memory consumption. EASES requires all players to prepare a block of memory to store the list of one-time signature keys. It exact size depends on the hash code length $L(H)$ and the chosen n. For example, if SHA-1 is used and n = 1000, then each player will need 1000 * 192 = 192,000 *bits* = 24,000 *bytes* of memory to store the one-time signature keys. The consumed space is not needed in other existing digital signature schemes.

Bandwidth consumption. Except for the first update, EASES needs to transfer the one-time signature for every event update, with the size of a hash code. This is much shorter than the size of traditional digital signatures. For example, SHA-1 uses 192 bits for each hash code, whereas 1024-RSA uses 1024 bits for each signature.

5.3 Comparisons

We briefly compare EASES against NEO [11] and SEA [14], both of which adopt traditional digital signatures. There are two signature operators in NEO and one signature operator in SEA. Each event update in NEO is given in Formula (1), and Formula (2) describes the update in SEA. With EASES, the signature operators used in each event update can be replaced by just two hash operators.

5.4 Discussions

The proposed scheme in this paper is a signature protocol that can be used to sign many discrete messages efficiently. It is thus inherently topology independent. In fact, the event update protocol can be adopted to any topology easily [3], by requiring the protocol be used between every pairs of connected players.

However, network topology may change constantly in P2P-based MMOGs. Since existing signature schemes sign each event update independently, the event updates can be sent to different targets directly even when the topology has changed. In contrast, EASES adopts a sequence of hash values, signatures thus are not independent to event updates. If a new target emerges, all one-time signatures (i.e. sequence of hash values) and event updates up to the current (i.e. j^{th}) event have to re-sent to the new target. An alternative is for the player to regenerate a new hash chain and sign its head by executing the initialization phase, treating a new target as a newly joined player. EASES therefore cannot fully eliminate traditional digital signatures, but it can be used to reduce the signature operators used.

6 Conclusion

In this paper, an efficient and secured event signature (EASES) protocol for P2P-based MMOGs is proposed. This protocol has the non-repudiation property inherited from traditional signature schemes. Furthermore, the proposed scheme requires much less computation, which makes it applicable to P2P-based MMOGs. The security of EASES is shown to possess unforgeability and verifiability as traditional digital signatures. By signing a hash chain, event commitment can be implicitly achieved. The computation, memory and bandwidth consumptions of EASES are also shown to be low. We have also shown how EASES may be adapted to non-fully-connected and dynamically changing network topologies such as P2P-based MMOGs.

References

1. Woodcock, B.S.: An analysis of MMOG subscription growth (2006) http://www.mmogchart.com.
2. Knutsson, B., Lu, H., Xu, W., Hopkins, B.: Peer-to-peer support for massively multiplayer games. In: Proc. IEEE Infocom. (2004)

3. GauthierDickey, C., Lo, V., Zappala, D.: Using n-trees for scalable event ordering in peer-to-peer games. In: Proc. of the international workshop on Network and operating systems support for digital audio and video, ACM Press New York, NY, USA (2005) 87–92
4. Keller, J., Simon, G.: Solipsis: A massively multi-participant virtual world. In: Proc. of PDPTA. (2003) 262–268
5. Morillo, P., Moncho, W., na, J.O., Duato, J.: Providing Full Awareness to Distributed Virtual Environments Based on Peer-To-Peer Architectures. (June 2006)
6. Lee, J., Lee, H., Ihm, S., Gim, T., Song, J.: Apolo: Ad-hoc peer-to-peer overlay network for massively multi-player online games. Technical report, KAIST Technical Report, CS-TR-2005-248 (December 2005)
7. Douglas, S., Tanin, E., Harwood, A., Karunasekera, S.: Enabling massively multi-player online gaming applications on a p2p architecture. In: Proc. IEEE Intl. Conf. Information and Automation. (December 2005) 7–12
8. Bharambe, A., Pang, J., Seshan, S.: Colyseus: A distributed architecture for multiplayer games. In: Proc. ACM/USENIX NSDI. (2006)
9. Hu, S., Chen, J., Chen, T.: VON: a scalable peer-to-peer network for virtual environments. IEEE Network **20**(4) (2006) 22–31
10. Baughman, N., Levine, B.: Cheat-proof playout for centralized and distributed online games. In: Proc. of IEEE Infocom. (2001)
11. Dickey, C., Zappala, D., Lo, V., Marr, J.: Low latency and cheat-proof event ordering for peer-to-peer games. In: Proc. of ACM International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Kinsale, County Cork, Ireland. (2004) 134–139
12. Kamada, M.: A fair dynamical game over networks. In: Proc. of the 2004 International Conference on Cyberworlds (CW'04). (2004) 141–146
13. Kamada, M., Kurosawa, K., Ohtaki, Y., Okamoto, S.: A network game based on fair random numbers. IEICE Transactions on Information and Systems **88**(5) (2005) 859–864
14. Corman, A., Douglas, S., Schachte, P., Teague, V.: A secure event agreement (SEA) protocol for peer-to-peer games. In: First International Conference on Availability, Reliability and Security. (2006)
15. J., Y.: Security design in online games. In: Proc. of the 19th Annual Computer Security Applications Conference. (2003) 286–295
16. Kirmse, A., Kirmse, C.: Security in online games. Game Developer **4**(4) (1997) 20–8
17. Smed, J.: Aspects of networking in multiplayer computer games. The Electronic Library **20**(2) (2002) 87–97
18. Cronin, E., Filstrup, B., Jamin, S.: Cheat-proofing dead reckoned multiplayer games. In: Proc. of Application and Development of Computer Games. (2003)
19. Preneel, B., Van Rompay, B., Ors, S., Biryukov, A., Granboulan, L., Dottax, E., Dichtl, M., Schafheutle, M., Serf, P., Pyka, S.: Performance of optimized implementations of the NESSIE primitives (2003) http://www.cosic.esat.kuleuven.ac.be/nessie/deliverables/.
20. The MIT Kerberos Team: The network authentication protocol http://web.mit.edu/Kerberos/.
21. Lamport, L.: Password authentication with insecure communication. Communications of the ACM **24**(11) (1981) 770–772