Cryptanalysis of a Group Key Transfer Protocol Based on Secret Sharing*

Junghyun Nam[1], Moonseong Kim[2], Juryon Paik[3], Woongryul Jeon[3], Byunghee Lee[3], and Dongho Won[3,**]

[1] Department of Computer Engineering, Konkuk University, Korea
jhnam@kku.ac.kr
[2] Information and Communications Examination Bureau, Korean Intellectual Property Office, Korea
moonseong@kipo.go.kr
[3] Department of Computer Engineering, Sungkyunkwan University, Korea
wise96@ece.skku.ac.kr,{wrjeon,bhlee,dhwon}@security.re.kr

Abstract. Group key establishment protocols allow a set of communicating parties to establish a common secret key. Due to their significance in building a secure multicast channel, a number of group key establishment protocols have been suggested over the years for a variety of settings. Among the many protocols is Harn and Lin's group key transfer protocol based on Shamir's secret sharing. This group key transfer protocol was designed to work in the setting where a trusted key generation center shares a long-term secret with each of its registered users. As for security, Harn and Lin claim that their protocol prevents the long-term secret of each user from being disclosed to other users. But, we found this claim is not true. Unlike the claim, Harn and Lin's protocol cannot protect users' long-term secrets against a malicious user. We here report this security problem with the protocol and show how to address it.

Keywords: Security, key establishment protocol, group key transfer, secret sharing, replay attack.

1 Introduction

Key establishment protocols allow two or more communicating parties to establish their common secret key called a *session key*. Establishment of session keys is one of the fundamental cryptographic operations and provides a typical way of building secure communication channels over insecure public networks. Traditionally, protocols which can be run by an arbitrary number of parties are called group (or conference) key establishment protocols, in contrast to protocols

* This work was supported by Priority Research Centers Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2011-0018397).

** Corresponding author.

T.-h. Kim et al. (Eds.): FGIT 2011, LNCS 7105, pp. 309–315, 2011.

which can be run only by two or three parties. In the group setting, a session key is also called a *group key*. Key establishment protocols are often classified into two types: key agreement protocols and key transfer protocols. Key agreement protocols require each participant to contribute its part to the final form of the session key, whereas key transfer protocols allow one trusted entity to generate the session key and then transfer it to all participants.

The first priority in designing a key establishment protocol is placed on ensuring the security of the protocol. Even if it is computationally infeasible to break the cryptographic algorithms used, the whole system becomes vulnerable to all manner of attacks if the keys are not securely established. But the experience shows that the design of secure key establishment protocols is notoriously difficult. Over the last decades, a number of protocols have been found to be insecure years after they were published [5,4,2,1]. Thus, key establishment protocols must be subjected to a thorough scrutiny before they can be deployed into a public network which might be controlled by an adversary.

This work is concerned with the security of the group key transfer protocol designed recently by Harn and Lin [3]. We use HL to refer to Harn and Lin's protocol. The protocol HL employs Shamir's secret sharing [6] to achieve information-theoretically secure distribution of session keys. Accordingly, the security of HL does not depend on any unproven assumption about computational hardness (as far as confidentiality of session keys is concerned). HL assumes a trusted key generation center (KGC) who provides key distribution service to its registered users. During registration, KGC issues each user a long-term secret which should be kept privately by the user. One of the security claims made for HL is that the long-term secret of each user cannot be learned by other users. But, it turns out that this claim is not true. The truth is that HL is vulnerable to a replay attack whereby a malicious user, who is registered with KGC, can readily obtain the long-term secret of any other registered user. In the current work, we reveal this security vulnerability of HL and then suggest a countermeasure against the replay attack.

2 Harn and Lin's Group Key Transfer Protocol

This section reviews Harn and Lin's group key transfer protocol HL [3]. The protocol HL consists of three phases: system initialization, user registration, and key distribution.

System Initialization. KGC randomly chooses two safe primes p and q (i.e., p and q are primes such that $p' = \frac{p-1}{2}$ and $q' = \frac{q-1}{2}$ are also primes) and computes $n = pq$. n is made publicly known.

User Registration. Each user is required to register at KGC to subscribe the key distribution service. During registration, KGC shares a secret (x_i, y_i) with each user U_i where $x_i, y_i \in \mathbb{Z}_n^*$.

Key Distribution. This phase constitutes the core of the protocol and is performed whenever a group of users $U_1, \ldots, U_t$ decide to establish a common session key.

Step 1. A designated user of the group, called the initiator, sends a key distribution request to KGC. The request carries the list of participating users $\langle U_1, \ldots, U_t \rangle$.

Step 2. KGC broadcasts the participant list $\langle U_1, \ldots, U_t \rangle$ in response to the request.

Step 3. Each user U_i, for $i = 1, \ldots, t$, sends a random challenge $r_i \in \mathbb{Z}_n^*$ to KGC.

Step 4. KGC randomly selects a session key k and constructs by interpolation a t-th degree polynomial $f(x)$ passing through the $(t+1)$ points: $(x_1, y_1 \oplus r_1), \ldots, (x_t, y_t \oplus r_t)$ and $(0, k)$. Next, KGC selects t additional points $P_1, \ldots, P_t$ that lie on the polynomial $f(x)$. KGC then computes $\beta = h(k, U_1, \ldots, U_t, r_1, \ldots, r_t, P_1, \ldots, P_t)$, where h is a one-way hash function, and broadcasts $\langle \beta, r_1, \ldots, r_t, P_1, \ldots, P_t \rangle$ to the users. All computations with respect to $f(x)$ are performed modulo n.

Step 5. Each U_i constructs the polynomial $f(x)$ from the $(t+1)$ points: $P_1, \ldots, P_t$ and $(x_i, y_i \oplus r_i)$. Then U_i recovers the session key $k = f(0)$ and checks the correctness of β in the straightforward way. U_i aborts if the check fails.

Since the above protocol HL focuses on protecting the keying material broadcasted from KGC to users, Harn and Lin also present (in Remark 2 of [3]) how HL can be extended to provide user authentication and key confirmation. Let HL$^+$ be the extended version of HL. HL$^+$ is constructed from HL by revising Steps 3 and 4 to achieve user authentication and by adding Steps 6 and 7 to achieve key confirmation.

Step 3 (of HL$^+$). Each user U_i, for $i = 1, \ldots, t$, selects a random challenge $r_i \in \mathbb{Z}_n^*$, computes $\alpha_i = h(x_i, y_i, r_i)$, and sends $\langle \alpha_i, r_i \rangle$ to KGC.

Step 4 (of HL$^+$). KGC checks the correctness of each α_i in the straightforward way. KGC aborts if any of the checks fails. Otherwise, KGC continues with Step 4 of HL.

Step 6. Each U_i sends $\gamma_i = h(x_i, y_i, k)$ to KGC.

Step 7. After receiving all γ_i's, KGC sends $\delta_i = h(x_i, y_i, k, U_1, \ldots, U_t)$ to U_i for $i = 1, \ldots, t$.

All other parts (including the phases of system initialization and user registration) remain unchanged between HL and HL$^+$.

3 Replay Attack

The fundamental security goal of a key establishment protocol is to ensure that no one other than the intended users can compute the session key. In the cases of HL and HL$^+$, this goal can be achieved only when the secrecy of every (x_i, y_i) is guaranteed. As soon as (x_i, y_i) is disclosed, all the protocol sessions that U_i participates become completely insecure. It is thus crucial that x_i's and y_i's must not be revealed under any circumstances.

Harn and Lin claim that their protocols prevent the secret (x_i, y_i) of each U_i from being disclosed to other users, either insiders or outsiders (Theorem 3

of [3]). However, we found that this claim is wrong. Suppose that a malicious registered user U_j has a goal of finding out U_i's secret (x_i, y_i). Then U_j can achieve its goal by mounting the following attack against the protocol HL^+.

Step 0. As a preliminary step, the adversary U_j eavesdrops on a protocol session, where U_i participates, and stores the message $\langle \alpha_i, r_i \rangle$ sent by U_i in Step 3 of the session.

U_j then initiates two concurrent sessions S and S′ of the protocol alleging that the participants of both sessions are U_i and U_j. Once KGC responds with the participant list $\langle U_i, U_j \rangle$ in Step 2 of each session, U_j performs Step 3 of the sessions while playing dual roles of U_j itself and the victim U_i.

Step 3 of S. U_j sends the eavesdropped message $\langle \alpha_i, r_i \rangle$ to KGC as if the message is from U_i. But, U_j behaves honestly in sending its own message; U_j selects a random $r_j \in \mathbb{Z}_n^*$, computes $\alpha_j = h(x_j, y_j, r_j)$, and sends $\langle \alpha_j, r_j \rangle$ to KGC.

Step 3 of S′. U_j replays the messages $\langle \alpha_i, r_i \rangle$ and $\langle \alpha_j, r_j \rangle$. That is, U_j sends $\langle \alpha_i, r_i \rangle$ as U_i's message and sends $\langle \alpha_j, r_j \rangle$ as its own message.

KGC cannot detect any discrepancy since α_i and α_j are both valid. Note that KGC does not check for message replays. Hence, KGC will distribute the keying materials for the sessions. Let $f(x) = a_2x^2 + a_1x + k$ and $f'(x) = a_2'x^2 + a_1'x + k'$ be the polynomials constructed by KGC respectively in sessions S and S'. As soon as receiving the keying materials, U_j derives these polynomials as specified in Step 5 of the protocol. Now let

$$\begin{aligned} g(x) &= f(x) - f'(x) \\ &= (a_2 - a_2')x^2 + (a_1 - a_1')x + k - k'. \end{aligned}$$

Then, $g(x_i) = 0$ and $g(x_j) = 0$ since $f(x_i) = f'(x_i) = y_i \oplus r_i$ and $f(x_j) = f'(x_j) = y_j \oplus r_j$. This implies that x_i and x_j are the two roots of the quadratic equation $(a_2 - a_2')x^2 + (a_1 - a_1')x + k - k' = 0$. It follows that

$$(a_2 - a_2')x^2 + (a_1 - a_1')x + k - k' = (a_2 - a_2')(x - x_i)(x - x_j)..$$

Therefore,

$$x_i = x_j^{-1}(a_2 - a_2')^{-1}(k - k'). \tag{1}$$

Here, the computations are done modulo n. Once x_i is obtained as in Eq.. (1), y_i can be easily computed from $f(x_i) = y_i \oplus r_i$. The value of y_i is different depending on whether $y_i \oplus r_i < n$ or $y_i \oplus r_i \geq n$.

$$y_i = \begin{cases} f(x_i) \oplus r_i & \text{if } y_i \oplus r_i < n \\ (f(x_i) + n) \oplus r_i & \text{otherwise.} \end{cases}$$

α_i can serve as a verifier for checking which of the two cases is true. Using (x_i, y_i) obtained as above, U_j is able to complete the protocol without the attack being noticed.

The above attack assumes, for ease of exposition, that KGC allows for the key establishment between two parties. But, this assumption is not necessary.. If two-party key establishments are not allowed, U_j can collude with another malicious user U_k to mount a slight variant of the attack. Assume two concurrent sessions of the protocol, in both of which the participants are U_i, U_j and U_k. If U_j and U_k collude together and run the two sessions as in the attack above, they can construct a cubic polynomial $g(x) = (a_3 - a'_3)x^3 + (a_2 - a'_2)x^2 + (a_1 - a'_1)x + k - k'$ such that $g(x_i) = g(x_j) = g(x_k) = 0$. Then, with x_j and x_k in hand, the adversaries can compute x_i as

$$x_i = (-1)x_j^{-1}x_k^{-1}(a_3 - a'_3)^{-1}(k - k')$$

and thereby can determine y_i as above.

So far, we have seen the vulnerability of the protocol HL^+. As can be expected, the basic protocol HL also suffers from the same vulnerability. The attack against HL is essentially similar to the above attack and is provided in Appendix.

4 Countermeasure

The security failure of HL^+ (and HL) is attributed to one obvious flaw in the protocol design: the messages sent by users in Step 3 can be replayed in different protocol sessions. This flaw allows our adversary U_j to send the same random challenges twice and thereby to construct a quadratic polynomial $g(x)$ such that $g(x_i) = g(x_j) = 0$.. Fortunately, message replays can be effectively prevented if Steps 2 and 3 of the protocols are revised as follows:

Step 2 (revision). KGC selects a random $r_0 \in \mathbb{Z}_n^*$ and broadcasts it along with the participant list $\langle U_1, \ldots, U_t \rangle$.

Step 3 (revision). Each user U_i, for $i = 1, \ldots, t$, selects a random $r_i \in \mathbb{Z}_n^*$, computes $\alpha_i = h(x_i, y_i, r_i, r_0, U_1, \ldots, U_t)$, and sends $\langle \alpha_i, r_i \rangle$ to KGC..

The other steps of the protocols remain unchanged except that in Step 4 of HL, KGC has to check the correctness of α_i, for $i = 1, \ldots, t$, before starting to construct the polynomial $f(x)$. As a result of our modification, the protocols HL and HL^+ become identical except that HL^+ requires two additional steps (Steps 6 and 7) for key confirmation. With the modification applied, the message $\langle \alpha_i, r_i \rangle$ eavesdropped in a protocol session can no longer be replayed in any other sessions. Hence, our attacks are not valid against the improved protocols.

5 Concluding Remark

It is worth noting that polynomial interpolation over $\mathbb{Z}_n^*$ may fail, though the probability of failure is negligible. This is essentially because the multiplicative group $\mathbb{Z}_n^*$ is not closed under addition and subtraction while interpolation formulas include additive and subtractive terms. If an addition/subtraction operation in $\mathbb{Z}_n^*$ returns a value z such that $\gcd(z, n) \neq 1$ (i.e., $z = cp$ or $z = cq$ for

some integer c), then there will not exist the multiplicative inverse of z modulo n. The protocols HL and HL$^+$ fail if such a z happens to be a divisor in the interpolation formula. (Of course, the probability of this occurring should be negligible, because otherwise we have a polynomial-time factoring algorithm.) This correctness issue of the protocols can be addressed simply by replacing $\mathbb{Z}_n^*$ with a prime field $\mathbb{F}_p$ in which interpolation never fails. We believe that the replacement causes no security degradation.

References

1. Choo, K.-K.: Refuting the security claims of Mathuria and Jain (2005) key agreement protocols. International Journal of Network Security 7(1), 15–23 (2008)
2. Choo, K.-K.R., Boyd, C., Hitchcock, Y.: Errors in Computational Complexity Proofs for Protocols. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 624–643. Springer, Heidelberg (2005)
3. Harn, L., Lin, C.: Authenticated group key transfer protocol based on secret sharing. IEEE Transactions on Computers 59(6), 842–846 (2010)
4. Krawczyk, H.: HMQV: a High-Performance Secure Diffie-Hellman Protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005)
5. Pereira, O., Quisquater, J.-J.: A security analysis of the Cliques protocols suites. In: Proc. 14th IEEE Computer Security Foundations Workshop, pp. 73–81 (2001)
6. Shamir, A.: How to share a secret. Communications of the ACM 22(11), 612–613 (1979)

Appendix: Attack on the Protocol HL

Consider the protocol HL described in Section 2. We here show that HL is vulnerable to an attack whereby a registered user U_j can learn the long-term secret (x_i, y_i) of any other registered user U_i. The main idea of the attack is the same as that of the attack against HL$^+$. The attack works as follows:

1. The adversary U_j initiates two concurrent sessions S and S$'$ of the protocol alleging that the participants of both sessions are U_i and U_j.
2. KGC will respond with the participant list $\langle U_i, U_j \rangle$ in Step 2 of each session.
3. U_j performs Step 3 of the sessions while playing dual roles of U_j itself and the victim U_i.

   Step 3 of S. U_j selects a random $r_i \in \mathbb{Z}_n^*$ and sends it to KGC as if it is from U_i. In addition, U_j sends its own challenge $r_j \in \mathbb{Z}_n^*$ to KGC.

   Step 3 of S$'$. U_j replays the challenges r_i and r_j. That is, U_j sends r_i as U_i's challenge and sends r_j as its own challenge.
4. KGC will distribute the keying materials for the sessions. Let $f(x) = a_2x^2 + a_1x + k$ and $f'(x) = a_2'x^2 + a_1'x + k'$ be the polynomials constructed by KGC respectively in S and S'.

5. After receiving the keying materials, U_j recovers the two polynomials $f(x)$ and $f'(x)$ as specified in Step 5 of the protocol. Let

$$\begin{aligned} g(x) &= f(x) - f'(x) \\ &= (a_2 - a'_2)x^2 + (a_1 - a'_1)x + k - k'. \end{aligned}$$

Then since $g(x_i) = 0$ and $g(x_j) = 0$,

$$x_i = x_j^{-1}(a_2 - a'_2)^{-1}(k - k').$$

Given x_i, we should consider two different cases in calculating the value of y_i.

$$y_i = \begin{cases} f(x_i) \oplus r_i & \text{if } y_i \oplus r_i < n \\ (f(x_i) + n) \oplus r_i & \text{otherwise.} \end{cases}$$

6. U_j needs to decide whether $y_i \oplus r_i < n$ or $y_i \oplus r_i \geq n$. Note that the equations $f(x_i) = y_i \oplus r_i$ and $f'(x_i) = y_i \oplus r_i$ do not allow to determine which of the two cases is true. This is because both values of y_i satisfy the equations. But, knowledge of another polynomial $f''(x)$ such that $f''(x_i) = y_i \oplus \tilde{r}_i$, where $\tilde{r}_i \neq r_i$, would immediately reveal which one of the two values of y_i is correct. U_j can easily generate such a polynomial $f''(x)$ if he runs a new protocol session as above, but this time using a different challenge $\tilde{r}_i$ for U_i.

As is the case for HL^+, HL is also vulnerable to a colluding attack where two adversaries U_j and U_k collude together to learn U_i's secret (x_i, y_i). We here omit the description of the colluding attack on HL since it is clear from the attack above and the colluding attack on HL^+.