Weaknesses in a leakage-resilient authenticated key transport protocol

Qiang Tang and Chris J. Mitchell
Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
{qiang.tang, c.mitchell}@rhul.ac.uk

10th June 2005

Abstract

In this paper we demonstrate the existence of a number of weaknesses in a leakage-resilient authenticated key transport protocol due to Shin, Kobara and Imai. The weaknesses imply that the protocol cannot achieve the security goals claimed by its designers. We also propose an enhanced protocol which is immune to some of these vulnerabilities.

1 Introduction

Recently, Shin, Kobara and Imai proposed a leakage-resilient authenticated key transport protocol [2] (referred to as the RSA-AKE protocol) to be used in an client-server environment, where

1. A client C, who can only remember a password, wishes to communicate with several servers.
2. C has insecure devices with very restricted computing power and built-in memory capacity. The servers have significant computing power, but they might be compromised.
3. Neither a PKI (Public Key Infrastructure) nor a TRM (Tamper-Resistant Module) is available.

Shin, Kobara and Imai claimed that the RSA-AKE protocol is provably secure in the random oracle model [2]. However, we show that the RSA-AKE

protocol suffers from potential security problems in the intended environment of use. Using a proof of equality of discrete logarithms scheme, we propose an enhanced protocol which is immune to some of these vulnerabilities.

The rest of this paper is organised as follows. In Section 2 we review the the RSA-AKE protocol. In section 3 we demonstrate weaknesses in the RSA-AKE protocol. In Section 4, we propose an enhanced RSA-based authenticated key transport protocol. In the final section, we conclude this paper.

2 Review of the RSA-AKE protocol

Suppose a client C shares password pw with server S_i ($i \geq 1$). Without loss of generality, we suppose that C possesses identity ID_C, and S_i possesses identity ID_{S_i}. Additionally, f is a full-domain hash function [3], and $h_i (1 \leq i \leq 4) : \{0,1\}^* \rightarrow \{0,1\}^k$ are four different hash functions, where k is a security parameter. Throughout this paper, $f(x_1, \cdots, x_n)$ and $h_i(x_1, \cdots, x_n)$ $(1 \leq i \leq 4)$ represent computing the hash value on the concatenation of messages x_j, $1 \leq j \leq n$.

At the initialisation stage, S_i generates its RSA public/private key pair (e, N) and (d, N), and sends (e, N) to C. C registers a password verifier $p_{i1} = \alpha_{i1} + pw \bmod N$ at S_i, where α_{i1} is randomly selected from Z_N. C stores α_{i1} and (e, N) on some insecure device such a PDA, which is not necessarily securely protected and may leak the stored information. S_i stores p_{i1} and (d, N) in its database, which is also not necessarily securely protected and may leak the stored information (both p_{ij} and (d, N)). Finally, C and S_i also store a counter j, initially set to 1.

In the j-th ($j \geq 1$) execution of the RSA-AKE protocol, C and S_i perform as follows.

1. C first computes the password verifier $p_{ij} = \alpha_{ij} + pw \bmod N$. Note that the values of α_{ij}, $j = 1, 2, \cdots$, are defined recursively – see step 3 below. Then C chooses a random $x \in Z_N^*$ and computes $W = f(j, p_{ij})$, $y = x^e \bmod N$, and $z = y \cdot W \bmod N$. Finally, C sends ID_C, j, z to S_i.

2. S_i first checks whether j is the correct counter value. If the check succeeds, S_i computes $y' = z \cdot W^{-1} \bmod N$, $x' = (y')^d \bmod N$, and $V_{S_i} = h_1(ID_C, ID_{S_i}, j, z, p_{ij}, x')$, and then sends ID_{S_i}, V_{S_i} to C. Otherwise, S_i terminates the proitocol execution.

3. After receiving S_i and V_{S_i}, C first checks whether the following equa-

tion is valid:

$$V_{S_i} = h_1(ID_C, ID_{S_i}, j, z, p_{ij}, x)$$

If the check succeeds, C computes and sends V_C to S_i, where

$$V_C = h_2(ID_C, ID_{S_i}, j, z, p_{ij}, x)$$

Otherwise, C terminates the protocol execution.

C computes the session key as $SK_{ij} = h_3(ID_C, ID_{S_i}, j, z, p_{ij}, x)$, and replaces the stored data α_{ij} with $\alpha_{i(j+1)}$:

$$\alpha_{i(j+1)} = \alpha_{ij} + h_4(ID_C, ID_{S_i}, j, z, p_{ij}, x) \bmod N$$

C sets the counter value to $j + 1$.

4. After receiving V_C, S_i first checks whether the following equation is valid:

$$V_C = h_2(ID_C, ID_{S_i}, j, z, p_{ij}, x')$$

If the check succeeds, S_i computes the session key as

$$SK_{ij} = h_3(ID_C, ID_{S_i}, j, z, p_{ij}, x'),$$

and replaces the password verifier p_{ij} with $p_{i(j+1)}$:

$$p_{i(j+1)} = p_{ij} + h_4(ID_C, ID_{S_i}, j, z, p_{ij}, x') \bmod N$$

S_i sets the counter value to $j+1$. Otherwise, S_i terminates the protocol execution as a failure.

3 Weaknesses in the RSA-AKE protocol

Shin, Kobara and Imai [2] claim that the RSA-AKE protocol is provably secure in the random oracle model based on the assumption that the RSA problem is computationally infeasible. They also claim that an adversary cannot determine the correct password through off-line dictionary attacks, even if she knows the client's secret and the server's RSA private key, because generating the valid client's authenticator after computing z, or generating the valid server's authenticator, fall into the category of online dictionary attacks.

However, we show that the RSA-AKE protocol suffers from the following potential security problems. It should be noted that the first vulnerability is outside the scope of the security model in [2].

1. Observe that p_{ij} is the only secret used for authentication in the j-th run of the RSA-AKE protocol. So, if the attacker has compromised S_i and obtained p_{ij}, then he can successfully impersonate C to S_i in the subsequent protocol executions without the need to have access to pw. If this occurs, the legitimate client will no longer be able to authenticate himself, because the password verifier held by S_i will change. However, if the legitimate client authenticates himself before the attacker uses the stolen p_{ij}, then the attacker cannot launch the above attack because the stolen password verifier p_{ij} will no longer be valid.

   This attack means that leakage of p_{ij} from S_i may enable an attacker to mount an impersonation attack. Hence the RSA-AKE protocol does not appear to be suitable for use in environments where the server is not securely protected.

2. Shin, Kobara and Imai point out that measures should be adopted to restrict an attacker from replacing the RSA public key (e, N) on the client's device; otherwise they show that an eth-residue attack can be mounted. They also propose a means to thwart the eth-residue attack if the attacker does succeed in replacing the RSA public key (e, N) with (e', N'). However, we show below that, in some extreme circumstances, more serious vulnerabilities exist in this case.

   Suppose, for example, that the attacker has obtained α_{ij} and replaced the RSA public key (e, N) with (e', N'), where $e' = \phi(N')$, just before the j-th execution of the RSA-AKE protocol. In this case, the attacker can exhaustively search for the password using the intercepted message z. This is because $x^{e'} \bmod N' = 1$ for every x (since $e' = \phi(N')$), and hence $z = f(j, pw + \alpha_{ij}) \bmod N'$. That is, the only unknown used to compute z is pw. The measures proposed in [2] do not eliminate this vulnerability.

3. Shin, Kobara and Imai suggest that C can use the same password pw with a number of servers S_i $(i \geq 1)$. However, it is potentially dangerous to do this. Suppose the client shares the same password with m servers S_i $(1 \leq i \leq m)$. Then an attacker can successfully guess the password with a probability p by mounting n/m dictionary attacks in parallel at each server S_i $(1 \leq i \leq m)$, while he would need to mount n dictionary attacks against one specific server in order to achieve the same goal. This attack means that the client might need to change his password much more frequently (if m is very large) in order to prevent undetected dictionary attacks.

   This attack is of particular concern in environments where m is large and the client chooses the password from a small password set, e.g. based on personal preferences.

4 Enhanced authenticated key transport protocol

In the enhanced authenticated key transport protocol described below, we make the same assumptions as in [2], except that we suppose C shares password pw_i with server S_i ($i \geq 1$), where $pw_i \neq pw_j$ if $i \neq j$. We also assume that all the operations are securely protected during the initialisation stage.

4.1 Description of the enhanced protocol

In the initialisation stage, S_i generates its RSA public/private key pair (e, N) and (d, N). In order to eliminate the second weakness, we could require that e is a fixed value ($e = 2^{16} + 1$) or, alternatively, $e \leq N/2$. S_i also generates two large primes p and q, where $p = 2q + 1$ and $q > N$. Server S_i then generates a generator g_1 of a multiplicative subgroup of order q in Z_p^*; S_i also computes a second generator g_2, where $g_2 = (g_1)^{x^*} \bmod p$ and x^* is randomly selected from Z_q^*. Finally, S_i sends (e, N), p, g_1, and g_2 to C. C registers a password verifier $(g_2)^{p_{i1}} \bmod p$ at S_i, where $p_{i1} = \alpha_{i1} + pw \bmod q$ and α_{i1} is randomly selected from Z_q.

At the end of the initialisation stage, C stores p, g_1, g_2, α_{i1}, (e, N), and a counter j (initially set to 1) on some insecure device such a PDA, which is not necessarily securely protected and may leak the stored information. S_i stores p, g_1, g_2, $(g_2)^{p_{i1}}$, (d, N), and a counter j (initially set to 1) in its database, which is also not necessarily securely protected and may leak the stored information.

In the j-th ($j \geq 1$) execution of the enhanced protocol, C and S_i perform as follows.

1. C first computes $p_{ij} = \alpha_{ij} + pw \bmod q$, $t_1 = (g_1)^{p_{ij}} \bmod p$, and $t_2 = (g_2)^{p_{ij}} \bmod p$. Note that the values of α_{ij}, $j = 1, 2, \cdots$, are defined recursively – see step 3 below. Then C chooses a random $x \in Z_N^*$ and computes $W = f(j, t_2)$, $y = x^e \bmod N$, and $z = y \cdot W \bmod N$. Finally, C sends ID_C, j, z, t_1 to S_i.

2. S_i first checks whether j is the correct counter value. If the check succeeds, S_i computes $W = f(j, (g_2)^{p_{ij}} \bmod p)$, $y' = z \cdot W^{-1} \bmod N$, $x' = (y')^d \bmod N$, and $V_{S_i} = h_1(ID_C, ID_{S_i}, j, z, t_1, (g_2)^{p_{ij}} \bmod p, x')$, and then sends ID_{S_i}, V_{S_i} to C. Otherwise, S_i terminates the proitocol execution.

3. After receiving S_i and V_{S_i}, C first checks whether the following equation is valid:

$$V_{S_i} = h_1(ID_C, ID_{S_i}, j, z, t_1, t_2, x)$$

If the check succeeds, C computes the session key as

$$SK_{ij} = h_3(ID_C, ID_{S_i}, j, z, t_1, t_2, x),$$

and replaces the stored data α_{ij} with $\alpha_{i(j+1)}$:

$$\alpha_{i(j+1)} = \alpha_{ij} + h_4(ID_C, ID_{S_i}, j, z, t_1, t_2, x) \bmod q$$

C sets the counter value to $j + 1$.

4. C proves to S_i that $\log_{g_1}(t_1) = \log_{g_2}((g_2)^{p_{ij}})$ using the cryptographic primitive $PLOGEQ$ [1]. If the proof fails, S_i terminates the protocol execution as a failure.

5. C computes and sends V_C to S_i, where

   $$V_C = h_2(ID_C, ID_{S_i}, j, z, t_1, t_2, x)$$

   Otherwise, C terminates the protocol execution.

6. After receiving V_C, S_i first checks whether the following equation is valid:

   $$V_C = h_2(ID_C, ID_{S_i}, j, z, t_1, (g_2)^{p_{ij}}, x')$$

   If the check succeeds, S_i computes the session key as

   $$SK_{ij} = h_3(ID_C, ID_{S_i}, j, z, t_1, (g_2)^{p_{ij}}, x'),$$

   and replaces the password verifier $(g_2)^{p_{ij}}$ with $(g_2)^{p_{i(j+1)}} = (g_2)^{p_{ij}} \cdot g_2^t \bmod p$, where

   $$t = h_4(ID_C, ID_{S_i}, j, z, t_1, (g_2)^{p_{ij}}, x') \bmod q$$

   S_i sets the counter value to $j+1$. Otherwise, S_i terminates the protocol execution as a failure.

4.2 Security analysis

Because of the similarity between the enhanced protocol and the RSA-AKE protocol, we omit the general security discussion here. We stress that in the enhanced protocol, even if the attacker has obtained all the secret information from S_i, he still cannot impersonate C to S_i. This is because, in the $PLOGEQ$ mechanism [1], an attacker can only complete the proof if he knows p_{ij}; however, compromise of the server does not enable the attacker to compute p_{ij}.

Note that, if we assume that both α_{ij} and $(g_2)^{p_{ij}} \bmod p$ could be leaked and obtained by an attacker, then obviously this attacker could exhaustively compute the password. This vulnerability also exists in the RSA-AKE protocol. So both the RSA-AKE protocol and the enhanced protocol should not be used in environments where an attacker can obtain leaked information from both the client and the server.

5 Conclusions

In this paper we have demonstrated certain weaknesses in a leakage-resilient authenticated key transport protocol. We also proposed an enhanced protocol which is immune to some of these vulnerabilities.

6 Acknowledgements

The authors would like to express their deep appreciation for the valuable comments provided by SeongHan Shin.

References

[1] J. Camenisch, U. M. Maurer, and M. Stadler. Digital payment systems with passive anonymity-revoking trustees. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *ESORICS '96: Proceedings of the 4th European Symposium on Research in Computer Security*, pages 33–43. Springer-Verlag, 1996.

[2] S. Shin, K. Kobara, and H. Imai. Efficient and leakage-resilient authenticated key transport protocol based on RSA. In J. Ioannidis, A. D. Keromytis, and M. Yung, editors, *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA*, volume 3531 of *Lecture Notes in Computer Science*, pages 269–284. Springer-Verlag, 2005.

[3] D. Stinson. *Cryptography Theory and Practice*. CRC Press, Inc., second edition, 2002.