Comments on mutual authentication and key exchange protocols for low power wireless communications

Siaw-Lynn Ng and Chris Mitchell

***Abstract*— In [1], Shim describes "unknown key-share" attacks on the two protocols, server-specific MAKEP and linear MAKEP, proposed by Wong and Chan in [2]. In this letter we point out an error in one of the attacks and demonstrate further undesirable properties in the protocols of Wong and Chan.**

***Index Terms*— Mutual authentication, key exchange.**

I. Introduction

In [2], Wong and Chan proposed two mutual authentication and key exchange protocols (MAKEPs), namely server-specific MAKEP and linear MAKEP. They are designed to be used for establishing secure communications between a low-power wireless device (client) and a powerful base station (server).

In [1], Shim described "unknown key-share" attacks on the two protocols. An unknown key-share attack on an authenticated key agreement protocol is an attack whereby an entity A ends up believing it shares a key with an entity B, and although this is in fact the case, B mistakenly believes the key is instead shared with another entity $E \neq A$. Here we point out that, while the attack on server-specific MAKEP works, the attack on linear MAKEP does not achieve the goals of an unknown key-share attack. In addition, we will demonstrate further limitations of the two protocols. Specifically, server-specific MAKEP allows the choice of the session key to be entirely under the control of the server, while in linear MAKEP, the authentication of the client to the server and the security of the public key scheme may be compromised in certain implementations.

II. The protocols and their limitations

A. Server-specific MAKEP

This protocol eliminates the use of public-key cryptographic operations at the client side and replaces them with symmetric key operations. Before running the protocol with a server B, the client A first obtains a certificate from a trusted authority TA:

$$\text{Cert}_A^B = \langle \text{ID}_A, E_{PK_B}(K_A),\ \text{Sig}_{TA}(\text{ID}_A, E_{PK_B}(K_A))\rangle,$$

where K_A is A's long-live symmetric key. Inside the certificate K_A is encrypted under B's public key. It is assumed that A and TA have authentic copies of PK_B. The protocol actions are as follows, with r_A, r_B representing nonces chosen by A and B respectively:

1. $A \to B$: $E_{K_A}(r_A)$, Cert_A^B
2. $B \to A$: $E_{K_A}(r_A, r_B, \text{ID}_B)$
3. $A \to B$: $E_{K_A}(r_B)$

The session key is computed to be $\sigma = r_A \oplus r_B$, which includes contributions from both party with the aim that no single party has full control over the selection of the session key.

However, this aim is not achieved: the server B can *always* ensure that the session key is its choice σ, by putting $r_B = \sigma \oplus r_A$.

It was pointed out in [1] that this protocol is susceptible to an unknown key-share attack, if an attacker E is able to obtain the certificate

$$\text{Cert}_E^B = \langle \text{ID}_E, E_{PK_B}(K_A),\ \text{Sig}_{TA}(\text{ID}_E, E_{PK_B}(K_A))\rangle.$$

We refer the reader to [1] for details. It was also pointed out in the same paper that by including ID_A in the encrypted message in step 2, this attack can be prevented. We note, however, that in this improved protocol, the selection of the session key is still completely under the control of the server B. This problem can be avoided by replacing $E_{K_A}(r_A)$ in the first message by $h(r_A)$ where h is a one-way hash function, replacing r_A in the second message by $h(r_A)$, and including r_A in the encrypted message of the last step.

B. Linear MAKEP

This protocol is designed to allow each client to communicate with as many servers as it wants without inducing any scalability problems, and also to prevent any server impersonating its own clients.

Let p be a prime such that the discrete logarithm problem in $\mathbf{Z}_p$ is intractable. Let $g \in \mathbf{Z}_p^*$ be a primitive element. The client A randomly chooses a sequence of $2n$ integers $a_1, a_2, \ldots, a_{2n}$ in $\mathbf{Z}_{p-1}$ as its secret keys. The corresponding sequence of public keys is $g^{a_1}, g^{a_2}, \ldots, g^{a_{2n}}$ in $\mathbf{Z}_p^*$. For each pair of public keys $(g^{a_{2i-1}}, g^{a_{2i}})$, $1 \leq i \leq n$, a certificate is obtained from the TA:

$$\text{Cert}_A^i = \langle \text{ID}_A, g^{a_{2i-1}}, g^{a_{2i}},\ \text{Sig}_{TA}(\text{ID}_A, g^{a_{2i-1}}, g^{a_{2i}})\rangle.$$

The actions of the i-th run of linear MAKEP are as follows, again with r_A, r_B representing nonces chosen by A and B respectively from $\mathbf{Z}_{p-1}$. The session key σ is computed to be $r_A \oplus y$.

Manuscript received July 18, 2003; revised September 4, 2003.
The authors are with the Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX, U.K.

1. $A \rightarrow B$: Cert_A^i
2. $B \rightarrow A$: r_B
3. $A \rightarrow B$: $x = E_{PK_B}(r_A)$,
   $y = a_{2i-1}(x \oplus r_B) + a_{2i} \bmod (p-1)$
4. $B \rightarrow A$: $E_\sigma(x)$

Note that when B receives x and y in step 3, B determines whether

$$(g^{a_{2i-1}})^{x \oplus r_B} \, g^{a_{2i}} \overset{?}{\equiv} g^y \pmod p \tag{1}$$

before proceeding to decrypt x to obtain r_A.

The unknown key-share attack described in [1] involves the attacker E obtaining

$$\begin{aligned}\mathrm{Cert}_E^i &= \langle \mathrm{ID}_E, (g^{a_{2i-1}})^c, (g^{a_{2i}})^c, \\ &\quad \mathrm{Sig}_{TA}(\mathrm{ID}_E, (g^{a_{2i-1}})^c, (g^{a_{2i}})^c)\rangle\end{aligned}$$

where $c \in \mathbf{Z}_{p-1}$, from A's public keys.

When A initiates linear MAKEP and sends Cert_A^i to B, E intercepts and replaces it with Cert_E^i. When B sends r_B to E in reply, E forwards it to A who computes x using B's public key, computes y, and sends x, y to B. Again, E intercepts x, y and sends x, $y' = yc$ to B. Now, B verifies that x and y' are correct according to Cert_E^i and Equation (1). If so B computes session key $\sigma = r_A \oplus y'$ and sends $E_\sigma(x)$ to E, which E forwards to A. At this point however, the attack fails. This is because the last message A expects is $E_{r_A \oplus y}(x)$, not $E_{r_A \oplus y'}(x)$, which E cannot construct. Hence this “attack” described in [1] does not achieve the goals of an unknown key-share attack.

However, there are some more serious weaknesses of this protocol.

Firstly, the authentication of A to B relies on the asymmetric encryption method having certain properties that are not stated in [2], that is, a randomly chosen value should not decipher correctly. Consider the following attack, where E has observed a successful run of the protocol and wishes to impersonate A to B. If E, pretending to be A, sends the old certificate to B, B will respond with a new nonce r'_B. Now E lets $x' = x \oplus r_B \oplus r'_B$, where x and r_B are the values from the intercepted run of the protocol, and sends (x', y) to B. As specified in the protocol, B determines whether x' and y match using Equation (1). This will hold since $x' \oplus r'_B = x \oplus r_B$. Thus B will now decrypt x'. In most cases this will fail, but, for example, if RSA was used naively then it will work, and B will believe it is communicating with A. This is an instance of a successful unknown key-share attack: E succeeds in misleading B, without necessarily obtaining the session key. As pointed out by Shim in [1], such an attack *will* be detected if key confirmation is performed. Otherwise the protocol should include a requirement on the encryption function and also a test by B as to whether x' decrypts correctly – only after this can B be sure that it is communicating with A.

Another weakness in the protocol is the use of the secrets a_{2i-1} and a_{2i} in a linear equation for computing y, without hiding the coefficient $x \oplus r_B$.

It is not quite clear how these pairs of “public keys” are used, but if they are used more than once then an eavesdropper E can simply obtain (r_B, x, y) in the first run, (r'_B, x', y') in a subsequent run, compute $(y - y')/(x \oplus r_B - x' \oplus r'_B)$ to get a_{2i-1}, and then get a_{2i}. After that E can impersonate A to any other servers. A straightforward inclusion of r_A in the third message (using $x \oplus r_A \oplus r_B$ instead of $x \oplus r_B$ in y) would prevent an eavesdropper from launching such an attack, but would still allow the server B to impersonate its client A. Hence treating the secret keys associated with Cert_A^i as long-term reusable secrets leads to a breach in security. This is a serious vulnerability, since some devices may be prone to “reset” attacks, where counters can be reset after a power failure, or some implementations may reuse certificates and keys due to the expensive overhead of updating the limited storage of key-signature sets. To prevent this, either servers will have to check with each other that Cert_A^i is never reused, which seems infeasible, or A should delete the private keys as soon as they have been used. It would appear, then, that a secure implementation of linear MAKEP, while efficient in terms of computation, would incur a significant overhead in communication load, thereby potentially making it impractical.

III. Conclusion

In this letter we have pointed out an error in the attack proposed by Shim in [1] on one of Wong and Chan's MAKEPs ([2]). We have also shown further limitations of these protocols and suggested improvements.

Acknowledgment

The authors would like to thank the anonymous referees for their suggestions and comments.

References

[1] Kyungah Shim. “Cryptanalysis of mutual authentication and key exchange for low power wireless communications”. *IEEE Communications Letters*, Vol 7, No 5, 2003, pp 248–250.

[2] D. S. Wong and A. H. Chan. “Mutual authentication and key exchange for low power wireless communications”. *Proc. IEEE MILCOM 2001*, Vol 1, 2001, pp 39–43.