Original version appears in the Proc. of ASIACRYPT 2001. LNCS 2248 (December 9 - 13, 2001, Gold Coast, Australia)

Efficient and Mutually Authenticated Key Exchange for Low Power Computing Devices *

Duncan S. Wong and Agnes H. Chan

College of Computer Science, Northeastern University,
Boston, MA 02115, U.S.A.
{swong, ahchan}@ccs.neu.edu

December 4, 2001

Abstract. In this paper, we consider the problem of mutually authenticated key exchanges between a low-power client and a powerful server. We show how the Jakobsson-Pointcheval scheme proposed recently [15] can be compromised using a variant of interleaving attacks. We also propose a new scheme for achieving mutually authenticated key exchanges. The protocol is proven correct within a variant of Bellare-Rogaway model [3, 4]. This protocol gives the same scalability as other public-key based authenticated key exchange protocols but with much higher efficiency and fewer messages. It only takes 20 msec total computation time on a PalmPilot and has only three short messages exchanged during the protocol.

1 Introduction

The goal of a mutually authenticated key exchange protocol (MAKEP) between two communicating parties is to provide them with some assurance that they know each other's true identity and at the same time to have the two parties end up sharing a common key known only to them. This common key, also known as session key, can then be used to provide privacy and data integrity during the session. In this paper, we focus our attention on the design and analysis of MAKEPs for the two parties in which one of them is strictly limited in both computational power and memory capacity while the other is as powerful as a conventional desktop computer. We call the low-power party as the client and the powerful one as the server. Such a low-power client could be a Personal Digital Assistant (PDA), a cellular phone or a smart card in real applications. A powerful server could be a base station or the security center of a wireless network.

Although there is a long history of designing MAKEPs and many protocols have been proposed for various kinds of distributed systems, they seldom designed for such an unbalanced system setup. For symmetric-key based MAKEPs, [21, 22, 3, 4, 16], two communicating parties share a long-lived key or have a third

* This work was sponsored by the U. S. Air Force under contract F30602-00-2-0518.

party involved during runtime. In the first case, each party has to maintain a set of distinct keys for communicating with different parties. In the later case, a centralized trusted party is required to be present whenever the protocol is executed. Hence key management and scalability are two major issues when deploying the schemes in practice. For public-key based MAKEPs, [9, 7, 1, 11], on the other hand, high computational complexity is required on both communicating parties. For example, a 512-bit modular exponentiation on a 16MHz Palm V requires over one minute of pure computation as shown in [27].

Recently several schemes [26, 15] have been proposed for systems with unbalanced compututional power. In [26], we proposed two MAKEPs which attain efficiency on the client side and provide scalability for most systems. However, the schemes do not give as much scalability as a pure public-key based MAKEP does. In [15], Jakobsson and Pointcheval proposed a MAKEP which improves on efficiency by using precomputation. However, the protocol does not scale well and is susceptible to a variant of interleaving attacks as shown in Sect. 3.

In this paper, we propose a new scheme which not only gives the same scalability as other public-key based schemes do, but also requires only three short messages in a single protocol run. It takes only 20 msec computation time during a run of the protocol in the case where the client is a 16MHz PalmPilot. We also show that it is secure within a variant of Bellare-Rogaway model [3, 4] and provides a reasonable amount of forward secrecy.

The remainder of the paper is organized as follows. In Sect. 2, we describe some notations used throughout this paper. Then we present a variant of interleaving attacks and show how it can be used to compromise the MAKEP proposed in [15]. In Sect. 4, we introduce a new protocol referred to as the Client-Server MAKEP. A formal security analysis of the protocol is given in Sect. 5 and the performance is examined in Sect. 6. We conclude with some discussions on other properties of the protocol in Sect. 7.

2 Preliminaries

Let $\mathcal{E}_K$ and $\mathcal{D}_K$ denote the encryption and decryption transformations under the symmetric key K respectively. For public key systems, each entity A has a public key PK_A and a private key SK_A. For simplicity of notation, we use $\mathcal{E}_{PK_A}$ to denote the public key encryption. Sig_{TA} is a *secret* signing algorithm and Ver_{TA} is the *public* verification algorithm of a trusted authority TA. A certificate $Cert_A$ of an entity A is denoted by

$$Cert_A = <\mathrm{ID}_A,\ m,\ Sig_{\mathrm{TA}}(\mathrm{ID}_A, m)>,$$

where ID_A uniquely identifies A, and m is some message being certified. Usually m is A's public key with some other information such as a serial number and an expiration date. By $< X >$, we mean an appropriate encoding of X. A *nonce*, denoted by $r \leftarrow \{0,1\}^l$, is a l-bit random number. We use $x \in_R S$ to denote that x is chosen uniformly at random from the set S.

Throughout the paper, we assume that G is a cyclic group of prime order q and g is a generator of G. For simplicity, we only consider the case where G is a subgroup of $\mathbb{Z}_p^*$, the multiplicative group of the integers modulo a prime p. However the discussion applies equally well to any group of prime order in which the discrete logarithm problem is computationally intractable. We also assume that the domain parameters (g, p, q) are publicly known. This assumption can be dropped when the information is exchanged via certificates. An example of a certificate which contains the domain parameters will be given in Sect. 4.

3 A Variant of Interleaving Attacks — Hijacking Attack

As described in [26], a challenge-response based MAKEP should have two primitive elements. The first one is two challenge-response pairs which provide the mutual authentication for both participating parties. The second one is the 'binding' of each party's encrypted secret session key contribution to the corresponding challenge number sent by its partner. The bindings have to be stringent enough to guarantee the *freshness* of the session key and to counteract various attacks. These notions are later formalized in Definition 2. To illustrate their importance, we consider a recently proposed protocol [15] for mutual authentication and key exchange between a low-power client and a powerful server. The protocol is shown in Fig. 1, where H_0, H_1 and H_2 denote some cryptographic hash functions.

A – low-power client		**B** – powerful server
$(a,\ g^a)$		$(y,\ g^y)$
Precomputation		
$x,\ t \in_R \mathbb{Z}_q \backslash \{0\}$		
$X = g^x$; $K = (g^y)^x$		
$\sigma = H_0(g^y,\ X,\ K)$		
$R = H_2(g^y,\ X,\ K)$		
$r = H_1(g^t)$	$\xrightarrow{X,\ r}$	
		$K = X^y$
		$R' = H_2(g^y,\ X,\ K)$
		$0 \le e < 2^k$
	$\xleftarrow{R',\ e}$	
$R' \stackrel{?}{=} R$		
$d = t - ea \bmod q$	$\xrightarrow{d}$	
		$r \stackrel{?}{=} H_1(g^d (g^a)^e)$
		$\sigma = H_0(g^y,\ X,\ K)$

Fig. 1. Jakobsson-Pointcheval MAKEP

In the protocol, A is a low-power client and B is a powerful server. Each of them has a public key pair. A's private key is a which is chosen randomly in $\mathbb{Z}_q^*$ and g^a is her public key. It is assumed that A knows B's authentic public key, and vice versa. Before the protocol begins, A precomputes a session key σ from K where K is a secret to be shared with B using the Diffie-Hellman key exchange technique [9]. (This is done through X in the first message from A to B in the protocol.) She also precomputes the expected response of B which is denoted as R and a value r which is intended to be used for client authentication. Now when A runs the protocol with B, A authenticates B (server authentication) by checking if the incoming message (second message of the protocol) contains a value denoted by R' which is equal to R. We note that B's response R' 'binds' the challenge number X of A to his computed secret K. For client authentication, B chooses an independent random number e and sends it to A. A then computes a value d and sends it back to B. After B receives it, he then computes $H_1(g^d(g^a)^e)$ and compares it with the value of r received from A. We note that B's challenge number e has never been bounded to any secret value or any previous messages of the current session. This oversight creates a vulnerability for the protocol.

We now show that this protocol is vulnerable to an attack pictured in Fig. 2 where E denotes an adversary, S1 and S2 denote two parallel sessions to be established with B.

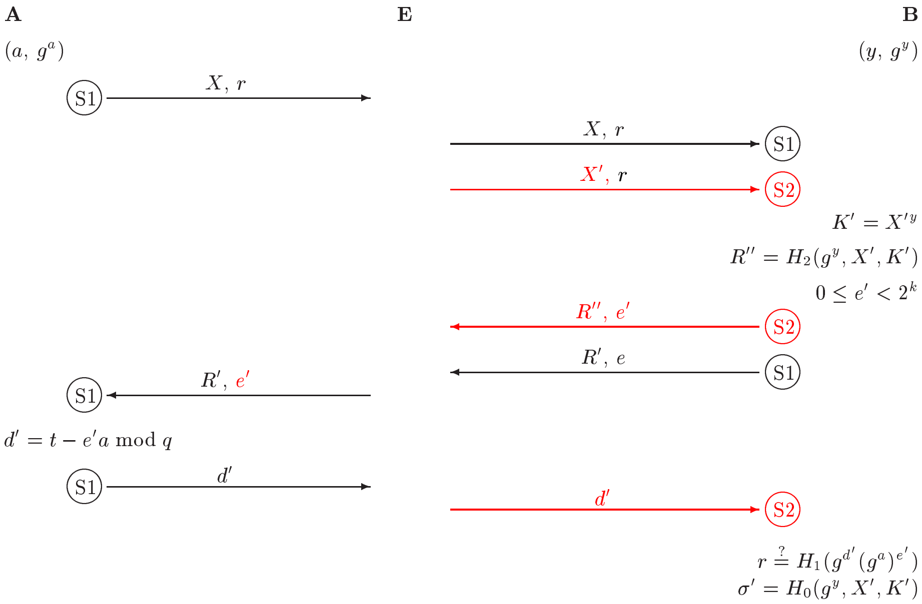

Fig. 2. An attack on Jakobsson-Pointcheval MAKEP

In this attack, we assume that E is an active adversary who is 'sitting' in between A and B and can intercept and inject messages. We also assume that B accepts multiple session connections from different instances of a single party simultaneously. In the figure, each message is associated with a session denoted by a circle containing the session number. For example, the message $< X,\ r >$ from A to E indicates that the message is for session S1. Session S1 begins when A sends the first message $< X,\ r >$ to B but it is also eavesdropped by E. E immediately creates a new message which contains an integer $X' \in G$ and r. E then sends this message to B alleging that it comes from another instance of A. B thinks that A wants to establish two sessions, denoted by S1 and S2, with him at the same time. He then sends back two messages, one for session S1 and the other for session S2. After receiving the two messages from B, E constructs a message which contains the correct response R' for session S1 and the challenge number e' of B for session S2. This message is sent to A by E alleging that it comes from B. Since A thinks that she is establishing only one session with B, she believes that the message must be the response of B for session S1. A then verifies the response and computes an outgoing message d'. However this message is relayed to B as the response of A for session S2. Meanwhile session S1 remains incomplete and will finally be terminated by B after a timeout.

Through this attack, the client authentication is compromised. A believes that she has established a secure session S1 with B sharing a secret key $K = X^y$, while B believes that he has established a different secure session S2 with A sharing a different secret key $K' = X'^y$. Furthermore E can choose randomly a value $x' \in \mathbb{Z}_q \backslash \{0\}$ and computes the pair $(X',\, K')$ as $g^{x'}$ and $(g^y)^{x'}$ respectively. Hence she can decrypt all the encrypted messages sent from B to A.

One possible patch to the authentication flaw is to modify the computation of d and the verification process to the following:

$$d = t - h(e, K)a \bmod q$$
$$r \stackrel{?}{=} H_1(g^d (g^a)^{h(e,K)})$$

where h is a cryptographic hash function. Another possible solution is to change the computations of R and R' to $H_2(g^y, X, K, e)$. The tradeoff is that A can no longer precompute R and this may affect the performance of the protocol. We call this attack a 'hijacking attack' because an adversary E can hijack A's conversations with B and impersonate her in a *new* run of the protocol. In fact, hijacking attack can be seen as a variant of interleaving attacks [5, 10, 19].

To ensure our protocol is secure against such an attack, we adopt the communication model where the server accepts multiple sessions from the same client in parallel and there is an active adversary. Details of the model will be given in Sect. 5.1. Furthermore, this attack highlights the importance of the authenticity of protocol flow, a notion formalized in [3] as *Matching Conversations*, which will be summarized in Sect. 5.2.

4 The Protocol

In this section, we propose a new scheme which is designed for mutually authenticated key exchange between a low-power client and a powerful server and we call it the Client-Server MAKEP. In this scheme, each party has a long-lived public key pair. For the powerful server B, we use SK_B and PK_B to denote the corresponding private and public keys respectively. Our description of the protocol can apply to any public key cryptographic algorithm but in practice, one that requires less memory and can do efficient encryption is preferred since the protocol requires the client to do one public-key encryption. We assume that the public key of the server is publicly known. For the low-power client A, $a \in_R \mathbb{Z}_q \backslash \{0\}$ is the private key and g^a is the public key. A also has a certificate obtained from the Trusted Authority (TA). The certificate is given by

$$Cert_A =< \mathrm{ID}_A,\ g,\ p,\ q,\ g^a,\ Sig_{\mathrm{TA}}(\mathrm{ID}_A, g, p, q, g^a) >$$

As described in Sect. 2, the domain parameters (g, p, q) can be removed from the certificate if they are publicly known. The protocol is illustrated in Fig. 3 and is described as follows.

Client-Server MAKEP

1. A selects $r_A \leftarrow \{0,1\}^k$, $b \in_R \mathbb{Z}_q \backslash \{0\}$ and computes $x = \mathcal{E}_{PK_B}(r_A)$ and $\beta = g^b$.
2. A sends $Cert_A$, β and x to B.
3. B checks $Cert_A$ by running Ver_{TA} and verifies that $1 < \beta < p$ and $\beta^q \equiv 1 \pmod p$. If any check fails, B terminates the protocol run with failure. Otherwise, B decrypts x and obtains r_A.
4. B selects $r_B \leftarrow \{0,1\}^k$, computes $\mathcal{E}_{r_A}(r_B,\ \mathrm{ID}_B)$ and sends it to A. B also computes $\sigma = (r_A \oplus r_B)||\mathrm{ID}_B$ and destroys r_A, r_B from his memory.
5. A decrypts the incoming message under r_A and checks if the decrypted message contains a proper coding of ID_B with some number. If the check fails, A terminates the protocol run with failure. Otherwise, A denotes the number as r_B.
6. A computes $\sigma = (r_A \oplus r_B)||\mathrm{ID}_B$ and $y \equiv ah(\sigma) + b \pmod q$ where $h : \{0,1\}^* \rightarrow \mathbb{Z}_q \backslash \{0\}$ is a cryptographic hash function. Then A sends y to B. A also computes $K = H(\sigma)$ as the session key and accepts the connection. She destroys r_A, r_B and σ from her memory.
7. B verifies if $g^y \equiv (g^a)^{h(\sigma)}\beta \pmod p$. If it is false, B terminates the protocol run with failure. Otherwise, B computes $K = H(\sigma)$ as the session key and accepts the connection. He also destroys σ from his memory.

In step 3, B verifies that $1 < \beta < p$ and $\beta^q \equiv 1 \pmod p$. This process is called public key validation [7]. It is a very important security measure in practice for protecting the system from several subtle attacks such as small subgroup attacks [18, 17] and identity element attack [7]. $H : \{0,1\}^* \rightarrow \{0,1\}^k$ is a hash function instantiating a public random oracle [2]. It is also called a key derivation function [14] here because it is used to derive the session key K from the

A		**B**
$(a,\ g^a)$		$(PK_B,\ SK_B)$
$r_A \leftarrow \{0,1\}^k$		
$x = \mathcal{E}_{PK_B}(r_A)$		
$b \in_R \mathbb{Z}_q \backslash \{0\}$		
$\beta = g^b$		
	$\xrightarrow{Cert_A,\ \beta,\ x}$	
		$r_B \leftarrow \{0,1\}^k$
	$\xleftarrow{\mathcal{E}_{r_A}(r_B,\ \mathrm{ID}_B)}$	
$\sigma = (r_A \oplus r_B)\vert\vert \mathrm{ID}_B$		$\sigma = (r_A \oplus r_B)\vert\vert \mathrm{ID}_B$
$y = ah(\sigma) + b \bmod q$		
	$\xrightarrow{y}$	
		$g^y \stackrel{?}{\equiv} (g^a)^{h(\sigma)}\beta$
$K = H(\sigma)$		$K = H(\sigma)$

Fig. 3. Client-Server MAKEP

shared secret σ. One reason for doing this is to destroy the algebraic relationships between the session key K and the nonces (r_A, r_B). Another reason is to mix together *strong* bits and potential *weak* bits of σ where weak bits are certain bits of information about σ that can be correctly predicted with non-negligible advantage.

5 Security Analysis

To prove that the protocol described above is secure, we use a variant of the Bellare and Rogaway's model [3, 4]. The approach we take closely follows the approach of [6].

A distributed system in our model has a set I_C of clients and a set I_S of servers. I_C (or I_S) is a set of identities which defines the clients (or servers) who can participate in the protocol. The cardinalities of these two sets may be any two polynomial functions of a security parameter, k. The adversary E is *not* in I_C or I_S.

Definition 1. *A C-S MAKEP is a triple $P = (\Pi, \Psi, LL)$ of probabilistic polynomial-time computable functions (each with respect to its first argument). Π specifies how a honest client behaves; Ψ specifies how a honest server behaves; and LL specifies the initial distribution of clients' and servers' long-lived keys. The domain and range of these functions are briefly described as follows.*

$(m, \delta, \sigma) = \Pi(1^k, A, B, SK_A, PK_A, PK_B, conv, r)$ *where*

$1^k \in \mathbb{N}$ — *the security parameter;*

$A \in I_C$ — *client's identity;*
$B \in I_S$ — *(intended) server's identity;*
$SK_A \in \{0,1\}^*$ — *long-lived secret key of* A*.*
$PK_A \in \{0,1\}^* \cup \{*\}$ — *long-lived public key of* A*; (Note that the value "*" refers to that the client does not have a long-lived public key. This is the case when the client is using a symmetric encryption algorithm in the protocol.)*
$PK_B \in \{0,1\}^* \cup \{*\}$ — *long-lived public key of* B*;*
$conv \in \{0,1\}^*$ — *conversation so far;*
$r \in \{0,1\}^\omega$ — *random coin flips;*
$m \in \{0,1\}^* \cup \{*\}$ — *the next message sent to* B*;*
$\delta \in \{\mathbf{A}, \mathbf{R}, *\}$ — *the decision where* **A** *represents accept,* **R** *represents reject and* $*$ *refers to that no decision has been made so far;*
$\sigma \in \{0,1\}^* \cup \{*\}$ — *the* private *output; it is the session key when it is string-valued.*

$(m, \delta, \sigma) = \Psi(1^k, B, A, SK_B, PK_B, PK_A, conv, r)$ *where the notations are similarly defined as those for* Π*.*

$(SK, PK) = LL(1^k, t, r)$ *where* $t \in \{\mathbf{client}, \mathbf{server}\}$ *is the type of the entity.*

In our protocol, if the value t of LL is **client**, SK is denoted by a four-tuple (g, p, q, a) and PK is denoted by another four-tuple (g, p, q, b) where p and q are prime such that $q \mid p - 1$. The length of p is polynomial in k. g is an element in $\mathbb{Z}_p^*$ of order q. a is chosen randomly from $\mathbb{Z}_q \backslash \{0\}$ and $b = g^a \bmod p$. If t is **server**, the value returned by LL will be the public key pair of some asymmetric key encryption algorithm chosen by the protocol.

5.1 Communication Model (Adversarial Model)

ADVERSARY E All communication among interacting parties is under the control of an adversary. In particular, the adversary can read, inject, modify, delete, delay and replay messages in the model. The adversary can also start up entirely new "instances" of any of the parties at any time. Hence there could be multiple *sessions* engaged in the system at the same time. This model gives the adversary the capability to launch attacks such as reflection and interleaving attacks suggested in [20, 5, 10, 19] and the hijacking attack described in Sect. 3.

Formally, an adversary E is a probabilistic machine which equips with an oracle denoted by LL_{PK} and an *infinite* collection of oracles Π_{ij}^s and Ψ_{ji}^t, for $i \in I_C$, $j \in I_S$ and $s, t \in \mathbb{N}$. Oracle LL_{PK}, which will be described below, models the event that public keys of all entities (both clients and servers) are publicly known. Oracle Π_{ij}^s models the instance s of client i attempting to agree on a shared session key with server j. Oracle Ψ_{ji}^t models the instance t of server j attempting to agree on a shared session key with client i. The model uses oracle queries to capture E's attacks such that (1) E writes queries on a special tape; (2) the corresponding oracle reads the tape automatically; and (3) E gets back a response in unit time (i.e. oracle query is treated as one step in an algorithm).

RUNNING THE PROTOCOL A generic execution of a protocol between a client instance and a server instance is called a *run* of the protocol. We model one run of the protocol as conducting one experiment in the presence of an adversary E using a security parameter k. It is described as follows.

1. **Initialization**
   Toss coins for LL, E and all oracles Π_{ij}^{s} and Ψ_{ji}^{t}.
2. **Run** E
   E may make oracle queries and the queries are answered as described in Table 1.

	On query of	Return
1	(`SendClient`$, i, j, s, x$)	$\Pi^{m\delta}(1^k, i, j, SK_i, PK_i, PK_j, conv\Pi_{ij}^{s}, coins\Pi_{ij}^{s})$ And then set $conv\Pi_{ij}^{s} \leftarrow conv\Pi_{ij}^{s} \cdot x \cdot m$
2	(`SendServer`$, j, i, t, x$)	$\Psi^{m\delta}(1^k, j, i, SK_j, PK_j, PK_i, conv\Psi_{ji}^{t}, coins\Psi_{ji}^{t})$ And then set $conv\Psi_{ji}^{t} \leftarrow conv\Psi_{ji}^{t} \cdot x \cdot m$
3	(`RevealClient`$, i, j, s$)	$\Pi^{\sigma}(1^k, i, j, SK_i, PK_i, PK_j, conv\Pi_{ij}^{s}, coins\Pi_{ij}^{s})$
4	(`RevealServer`$, j, i, t$)	$\Psi^{\sigma}(1^k, j, i, SK_j, PK_j, PK_i, conv\Psi_{ji}^{t}, coins\Psi_{ji}^{t})$
5	(`CorruptClient`$, i, SK, PK$)	$< SK_i, PK_i, conv\Pi_{ij}^{s}, coins\Pi_{ij}^{s} >_{j,s}$ And then set $SK_i \leftarrow SK$; $PK_i \leftarrow PK$
6	(`CorruptServer`$, j, SK, PK$)	$< SK_j, PK_j, conv\Psi_{ji}^{t}, coins\Psi_{ji}^{t} >_{i,t}$ And then set $SK_j \leftarrow SK$; $PK_j \leftarrow PK$
7	(`RequestPublic`)	$< PK_i, PK_j >_{i,j}$
8	(`Test`$, i, j, s$)	Choose at random a bit θ. If $\theta = 0$ return $r \leftarrow \{0,1\}^k$; if $\theta = 1$ return $\Pi^{\sigma}(1^k, i, j, SK_i, PK_i, PK_j, conv\Pi_{ij}^{s}, coins\Pi_{ij}^{s})$.
9	Other queries (meaningless)	λ

Table 1. The queries which E can ask of its oracles

When the adversary makes `SendClient` or `SendServer` query, oracle Π_{ij}^{s} or Ψ_{ji}^{t} calculates the answer using the description of function Π or Ψ respectively. Hence E gets the message m, and the decision $\mathbf{A}/\mathbf{R}/*$ which means that she can "see" when an oracle accepts. When the adversary makes `RevealClient` or `RevealServer` query, she gets back the private output of the corresponding oracle. The most severe type of loss for a player is when the player's complete internal state becomes known to the adversary. To model this possibility, we allow `CorruptClient` and `CorruptServer` queries, from which E learns the internal state of a player, including the private key SK_i, and substitutes some new value SK for the player's long-lived key. From that point on, we assume that all other players will use the revised long-lived keys of the corrupted players. When E writes (`RequestPublic`) on its query tape, we think of that query as being answered by the oracle LL_{PK}. The return is simply all the public keys of both clients and servers.

We define that Π_{ij}^s is

1. *accepted* if $\Pi^\delta(1^k, i, j, SK_i, PK_i, PK_j, conv\Pi_{ij}^s, coins\Pi_{ij}^s) = \mathbf{A}$;
2. *opened* if there has been a `RevealClient` query;
3. *unopened* if it is not opened;
4. *corrupted* if there has been a `CorruptClient` query; and
5. *uncorrupted* if it is not corrupted.

Similar notations apply to the server instances.

BENIGN ADVERSARIES We use the term, *benign adversary*, to model a reliable channel. This is used to show that a protocol is 'well-defined' which means that the protocol provides the two communicating parties (two oracles) the same session key at the end of a protocol run (when the oracles have accepted). For every $(i, j, s, t) \in I_C \times I_S \times \mathbb{N} \times \mathbb{N}$, there exists an (i, j, s, t)-benign adversary which is deterministic and always performs a single run of the protocol between Π_{ij}^s and Ψ_{ji}^t by faithfully relaying flows between these two oracles.

5.2 Definition of Security

With such a powerful adversary described above, a C-S MAKEP is considered secure if:

1. The protocol provides an instance of a client Π_{ij}^s with some assurance that she is involved in a *real-time* communication with an instance of a server Ψ_{ji}^t, and vice versa.
2. No adversary can learn anything about a session key which is held by an uncorrupted, unopened but accepted client instance Π_{ij}^s (or respective server instance Ψ_{ji}^t) and the corresponding server instance Ψ_{ji}^t (or respective client instance Π_{ij}^s) is uncorrupted and unopened but not necessarily accepted.

Before we give a formal definition of security, we need two more tools described as follows.

MATCHING CONVERSATIONS Matching conversations [3] provide the necessary formalism to define the assurance provided to one player that she has been involved in a *real-time communication* with another player. The formal definition of matching conversations is given in Appendix A. Here, we describe the notion of $\texttt{No-Matching}^E(k)$. This notation is used to specify the event that when protocol P is run against adversary E, there exists an uncorrupted oracle which has accepted but there is no other oracle which has engaged in a matching conversation with this oracle.

PROTECTING FRESH SESSION KEYS The notion that no adversary can learn information about fresh session keys [3, 4] is formalized by using the polynomial indistinguishability approach. Specifically at the end of an experiment, the adversary should not be able to gain more than a negligible advantage on distinguishing the actual fresh session key from a random number sampled from $\{0,1\}^k$. This idea is formalized by using the type 8 query shown in Table 1. We make the following modifications to the experiment.

- The `Test` query must be the adversary's last meaningful query, and it must be asked of a fresh oracle.
- To answer the query, the oracle flips a fair coin $\theta \leftarrow \{0,1\}$. If $\theta = 0$, the oracle returns a key sampled at random from $\{0,1\}^k$. If $\theta = 1$, it returns the session key.
- The adversary's job is to guess θ and outputs a bit `Guess`.

Let $\texttt{Good-Guess}^E(k)$ be the event that $\texttt{Guess} = \theta$. Then we define

$$advantage^E(k) = 2 \cdot \Pr[\texttt{Good-Guess}^E(k)] - 1.$$

Now we give a formal definition of security which is modified from [6]:

Definition 2. *A C-S MAKEP $P = (\Pi, \Psi, LL)$ is secure if:*

1. *(Well-Defined Protocol) In the presence of the benign adversary on Π_{ij}^s and Ψ_{ji}^t, both oracles always accept holding the same session key, which is uniformly distributed over $\{0,1\}^k$;*

and for any adversary E:

2. *(Real-time Partner) If two uncorrupted oracles have matching conversations, then both oracles accept and hold the same session key;*
3. *(Authenticity of Flow) The probability of $\texttt{No-Matching}^E(k)$ is negligible;*
4. *(Protecting Fresh Session Key) $advantage^E(k)$ is negligible.*

Here we use the conventional definition of the negligible function, that is, a real-valued function $\epsilon(k)$ is *negligible* if for every $c > 0$ there exists a $k_c > 0$ such that $\epsilon(k) < k^{-c}$ for all $k > k_c$.

Before we can show that our scheme meets the conditions of Definition 2, we need to specify the cryptographic primitives on which the security of our scheme relies. The primitives used in Client-Server MAKEP are public key encryption scheme, symmetric key encryption scheme and discrete logarithm problem. Their security definitions are derived from [12, 13] and are given in Appendix A.

Theorem 1. *The Client-Server MAKEP (described in Sect. 4) is a secure C-S MAKEP provided that the discrete logarithm problem is intractable, a secure symmetric key encryption scheme exists and a secure public key encryption scheme exists.*

The proof of this theorem appears in Appendix B.

6 Performance

In Sect. 4, we present the protocol without specifying any particular public key cryptographic algorithm for both TA and the server B. The choice of which is up to the target applications by taking into consideration of their systems' capabilities and constraints. The performance evaluation given here is therefore based on the number of times the cryptographic operations have to be performed, the sizes of the messages, the total number of messages sent in each protocol run and the memory requirement. We also restrict our attention to the efficiency of the client side only. Throughout this section, we also use the measurement results given in [27] to estimate the speed of the protocol running on a 16MHz Palm V with Palm OS version 3.3.

Speed The client is required to compute one public key encryption, one symmetric key decryption, one modular exponentiation, one modular multiplication, one modular addition, two hashes and two random number generations. On the average, SHA-1 only takes 0.9 msec to digest a 128-bit binary string on the Palm V. Therefore we can ignore the time taken for hashing in our evaluation. In practice, hash functions are also used to generate pseudo-random numbers. Thus their generation speed is comparable to that of hashing and can be ignored. For symmetric key decryption, both SSC2 [28] and ARC4 (Alleged RC4) only take about one millisecond each to decrypt a 256-bit ciphertext. Even for a block cipher, like Rijndael [8], it takes less than three milliseconds. The public key encryption is also doable if we choose a public-key cryptographic algorithm with very efficient encryption process. For example, it takes 710 msec to do a 512-bit RSA encryption when the value of the public exponent is three. If the Rabin cryptosystem is used, the encryption process will comprise one modular addition and one modular multiplication. By ignoring the overhead of doing any appropriate encoding of the plaintext, it takes only 110 msec to perform a 512-bit encryption.

We notice that in step 1 of the protocol described in Sect. 4, all the parameters can be prepared offline as precomputation. Hence only the following operations have to be done by the client during the runtime of the protocol:

1. one symmetric key decryption
2. one modular multiplication
3. one modular addition

We find that a 160-bit modular addition and multiplication can be done in 0.29 msec and 15 msec respectively on a 16MHz Palm V. Thus if k is 128, the length of ID_B is 128 bits and the length of q is 160 bits, the time taken to do the computation is less than 20 msec.

Network and Storage Efficiency There are only three messages exchanged in a single run of the protocol. If 1024-bit RSA is chosen to be the public key cryptographic algorithm for the TA, 512-bit RSA for the server, the length of p is 512 bits and the domain parameters are publicly known, the length of $Cert_A$

would be 208 bytes. The sizes of the three messages would be 336 bytes, 32 bytes and 20 bytes respectively. Therefore, this scheme is also very suitable for wireless communications. For storage, the client needs to store a, $Cert_A$, PK_{TA}, PK_B, ID_B, r_A, x, b, β and (g, p, q) if precomputation is applied. The total memory requirement for storing these parameters is 940 bytes. The actual memory requirement depends on the specific cryptographic algorithms and their parameters set in each target application. We notice that much less memory is required if G is the group of points of an elliptic curve over a finite field and an elliptic curve cryptosystem is used by both the TA and the server.

7 Concluding Remarks

Forward Secrecy It is clear that if the server's private key is compromised, then all the session keys from the earlier runs can be recovered from the transcripts. However, the corruption of the client may not help to reveal the session keys. Hence our scheme provides half forward secrecy [7]. Since the client may be a weak device while the server can be a strong and secure entity which support much stronger security measures than the client, we believe that forward secrecy on the client side is a much more important feature than that on the server side.

On the other side, consider that all previous sessions have been compromised, that is, the adversary E knows $\{\sigma_i\}_{0<i<n}$ where σ_i is the "algebraic" session key of the i-th session, E may not be able to reveal the client's private key, a, because of the unknown random values of b in these sessions. In brief, since r_A and r_B are nonces and no single party can control the values of the session keys, we assume that $\sigma_i \neq \sigma_j$ for $i \neq j$. Hence if h is a cryptographic hash function, the probability that $h(\sigma_i) = h(\sigma_j)$ is negligible. Similarly, since b is randomly chosen from $\mathbb{Z}_q \backslash \{0\}$, we assume that $b_i \neq b_j$ for $i \neq j$. If E wants to obtain a, E needs to solve the following equations:

$$y_i = ah(\sigma_i) + b_i \pmod{q} \qquad \text{for } 0 < i < n$$

with unknowns a and $\{b_i\}_{0<i<n}$. It can be seen that there are $(q-1)$ sets of possible solutions. Hence it is no easier than solving the discrete logarithm problem.

Similarly, to know the random values of b for a couple of sessions alone may not be enough to reveal the client's private key either. It is required that both b and $h(\sigma)$ of a particular session are known in order to reveal a. Alternatively, two sessions with the same value of b and having the values of $h(\sigma)$ known are also being able to compute the client's private key, namely $a \equiv (y_1 - y_2)/(h(\sigma_1) - h(\sigma_2)) \pmod{q}$.

Precomputation As we mentioned earlier, the scheme benefits from the precomputation technique to significantly reduce the computational requirement during the runtime of the protocol. In most of the applications, the client can conduct the precomputation during idle time. This technique also helps to reduce the peak power consumption by averaging out most of the computations over time.

Scalability The protocols proposed in [15, 26] are either server-specific or limited by the memory capacity and the precomputation overhead. Server-specific means that the client needs to pre-determine the server she wants to communicate and has to do some precomputations offline. In this scheme, the precomputation is optional and also applications have the flexibility to choose the extent of precomputation. For example, the client can pre-select the values of r_A and b and precompute β. During runtime, she only needs to compute x using a public key encryption before sending out the first message. As mentioned in Sect. 6, it can still be done efficiently if we choose a public-key cryptographic algorithm with very efficient encryption process. In this way, the values precomputed by the client do not depend on any specific server. Therefore it gives full scalability that other public-key MAKEPs provide but with higher efficiency and fewer messages.

Hash function h The hash function h specified in the Client-Server MAKEP is to output an integer in $\mathbb{Z}_q \backslash \{0\}$. As the client authentication is essentially the same as Schnorr's identification scheme [23, 24], the requirement of h can be loosed to output a binary string in $\{0, \cdots, 2^t - 1\}$ where 2^{-t} governs the success rate of an adversary to launch a *crooked* proof attack, which is described in the proof of Claim 3.

References

1. Ashar Aziz and Whitfield Diffie. A secure communcations protocol to prevent unauthorized access – privacy and authentication for wireless local area networks. *IEEE Personal Communications*, First Quarter 1994.
2. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, Fairfax, 1993. ACM.
3. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Proc. CRYPTO 93*, pages 232–249. Springer, 1994. Lecture Notes in Computer Science No. 773.
4. Mihir Bellare and Phillip Rogaway. Provably secure session key distribution – the three party case. In *Proc. 27th ACM Symp. on Theory of Computing*, pages 57–66, Las Vegas, 1995. ACM.
5. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 44–61. Springer, 1992. Lecture Notes in Computer Science No. 576.
6. Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In *Sixth IMA International Conference on Cryptography and Coding*, pages 30–45, 1997. Lecture Notes in Computer Science No. 1355.
7. Simon Blake-Wilson and Alfred Menezes. Authenticated Diffie-Hellman key agreement protocols. In *5th annual international workshop, SAC'98*, pages 339–361. Springer-Verlag, 1998. Lecture Notes in Computer Science No. 1556.
8. J. Daemen and V. Rijmen. AES Proposal: Rijndael. *AES Algorithm Submission*, Sep 1999. http://www.nist.gov/aes.

9. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, November 1976.
10. Whitfield Diffie, Paul C. Van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2(2):107–125, June 1992.
11. Alan O. Freier, Philip Karlton, and Paul C. Kocher. *The SSL Protocol Version 3.0*. INTERNET-DRAFT, Nov 1996. www.netscape.com/eng/ssl3/draft302.txt.
12. S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.
13. Shafi Goldwasser and Mihir Bellare. *Lecture Notes on Cryptography*. http://www-cse.ucsd.edu/users/mihir/papers/gb.html, August 1999.
14. IEEE. *P1363: Standard Specifications For Public Key Cryptography*, Nov 1999. Draft P1363 / D13.
15. Markus Jakobsson and David Pointcheval. Mutual authentication for low-power mobile devices. In P. Syverson, editor, *Proceedings of Financial Cryptography 2001*. Springer-Verlag, February 2001.
16. J. Kohl and C. Neuman. *The Kerberos Network Authentication Service (V5)*. IETF RFC1510, Sep 1993.
17. Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. An efficient protocol for authenticated key agreement. Technical Report CORR 98-05, University of Waterloo, 1998.
18. C. Lim and P. Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Proc. CRYPTO 97*, pages 249–263. Springer, 1997. Lecture Notes in Computer Science No. 1294.
19. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press LLC, 1997.
20. C. Mitchell. Limitations of challenge-response entity authentication. *Electronics Letters*, 25(17), Aug 1989.
21. R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.
22. Dave Otway and Owen Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21, Jan 1987.
23. C. P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 239–252. Springer, 1990. Lecture Notes in Computer Science No. 435.
24. C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3), 1991.
25. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Proc. EUROCRYPT 97*, pages 256–266. Springer, 1997. Lecture Notes in Computer Science No. 1233.
26. Duncan S. Wong and Agnes H. Chan. Mutual authentication and key exchange for low power wireless communications. In *IEEE MILCOM 2001 Conference Proceedings*, October 2001.
27. Duncan S. Wong, Hector Ho Fuentes, and Agnes H. Chan. The performance measurement of cryptographic primitives on palm devices. In *Proc. of the 17th Annual Computer Security Applications Conference*, Dec 2001.
28. Muxiang Zhang, Christopher Carroll, and Agnes H. Chan. The software-oriented stream cipher SSC2. *Fast Software Encryption Workshop 2000*, 2000.

A Definitions

For matching conversations, we use the same definition as given in [3]. Without loss of generality, we may assume the number of flows R in the protocol to be ood. Let E be an adversary. For any oracle Π_{ij}^{s} or Ψ_{ji}^{t}, its *conversation* can be captured by a sequence:

$$C = (\tau_1, \alpha_1, \beta_1),\ (\tau_2, \alpha_2, \beta_2), \cdots, (\tau_m, \alpha_m, \beta_m).$$

This sequence encodes that at time τ_1 the oracle was asked α_1 and responded with β_1; at time $\tau_2 > \tau_1$, the oracle was asked α_2 and answered β_2; finally, at time τ_m, it was asked α_m and answered β_m. At time τ_m, adversary E terminates without asking any more queries. If oracle Π_{ij}^{s} (or Ψ_{ji}^{t}) has $\alpha_1 = \lambda$, it is called an *initiator oracle*; otherwise it is called a *responder oracle*.

Definition 3 ([3]). *Let P be a R-flow protocol, where $R = 2\rho - 1$ is the number of flows. Run P in the presence of an adversary E and consider two oracles, an initiator oracle, and a responder oracle, that engage in conversations C and C' respectively.*

1. *C' is said to be a* matching conversation *to C if there exist $\tau_0 < \tau_1 < \cdots < \tau_{R-1}$ and $\alpha_1, \beta_1, \cdots, \beta_{\rho-1}, \alpha_\rho$ such that C is prefixed by:*

$$(\tau_0, \lambda, \alpha_1),\ (\tau_2, \beta_1, \alpha_2), \cdots, (\tau_{2\rho-2}, \beta_{\rho-1}, \alpha_\rho)$$

and C' is prefixed by:

$$(\tau_1, \alpha_1, \beta_1),\ (\tau_3, \alpha_2, \beta_2), \cdots, (\tau_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1}).$$

2. *C is said to be a* matching conversation *to C' if there exist $\tau_0 < \tau_1 < \cdots < \tau_R$ and $\alpha_1, \beta_1, \cdots, \beta_{\rho-1}, \alpha_\rho$ such that C' is prefixed by:*

$$(\tau_1, \alpha_1, \beta_1),\ (\tau_3, \alpha_2, \beta_2), \cdots, (\tau_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1}),\ (\tau_{2\rho-1}, \alpha_\rho, *)$$

and C is prefixed by:

$$(\tau_0, \lambda, \alpha_1),\ (\tau_2, \beta_1, \alpha_2), \cdots, (\tau_{2\rho-2}, \beta_{\rho-1}, \alpha_\rho).$$

If C is a matching conversation to C' and C' is a matching conversation to C, then the two oracles are said to have had matching conversations.

The following definitions are derived from [12, 13]. They are just briefly introduced here.

Definition 4. *A public key encryption scheme is a triple, $(\mathcal{G}, \mathcal{E}', \mathcal{D}')$, of probabilistic polynomial-time algorithms satisfying the following conditions:*

1. *key generation algorithm : $(e, d) \leftarrow \mathcal{G}(1^k)$ where k is the security parameter, e is the public key, and d is the corresponding private key.*

2. *encryption algorithm* : $c \leftarrow \mathcal{E}'_e(1^k, m)$ *where* $m \leftarrow \{0,1\}^l$ *is the message,* $c \in \{0,1\}^*$ *is the ciphertext and* l *is polynomial in* k.
3. *decryption algorithm* : $m \leftarrow \mathcal{D}'_d(1^k, c)$.

Definition 5. *A public key encryption scheme* $(\mathcal{G}, \mathcal{E}', \mathcal{D}')$ *is secure (polynomial time indistinguishable) if for every PPT algorithm* E *and for every polynomial* Q*, for all sufficiently large* k*,*

$$\begin{aligned} \Pr[E(1^k, e, m_0, m_1, c) = m \ \mid\ & (e,d) \leftarrow \mathcal{G}(1^k);\ m_0 \leftarrow \{0,1\}^k;\ m_1 \leftarrow \{0,1\}^k; \\ & m \leftarrow \{m_0, m_1\};\ c \leftarrow \mathcal{E}'_e(m)] \\ < \frac{1}{2} + \frac{1}{Q(k)} & \end{aligned}$$

B Proof of Theorem 1

Proof. (sketch) We prove the security of the protocol by establishing each condition of Definition 2.

Condition 1 and 2: The first two conditions follow immediately from the description of the Client-Server MAKEP and the assumption that H is a hash function instantiating a random oracle [2].

Condition 3: We prove this by contradiction. Assume that E is an arbitrary adversary and that $\Pr[\texttt{No-Matching}^E(k)]$ is non-negligible, then we show that certain cryptographic primitives which have assumed to be secure would be broken. We divide the proof into several subsections.

SERVER AUTHENTICATION

Consider the first two messages of the protocol shown in Fig. 3.

Claim 1 *If there exists a secure pair of symmetric key encryption scheme and public key encryption scheme, then upon receiving* $x = \mathcal{E}_{PK_B}(r_A)$*, only* B *can compute the second message,* $\mathcal{E}_{r_A}(r_B, \mathrm{ID}_B)$*; that is, for every PPT algorithm* E*, there is a negligible function* $\epsilon(k)$ *such that for sufficiently large* k*,*

$$\Pr[E(1^k, x, \mathrm{ID}_B, PK_B) = \mathcal{E}_{r_A}(r, \mathrm{ID}_B) \ \mid\ r_A \leftarrow \{0,1\}^k;\ x \leftarrow \mathcal{E}_{PK_B}(r_A)] \le \epsilon(k)$$

where r *is some random number of length* k.

This can be shown by contradiction. Suppose that on inputs 1^k, $\mathcal{E}_{PK_B}(r_A)$, ID_B and PK_B, E computes $\mathcal{E}_{r_A}(r, \mathrm{ID}_B)$ without asking an oracle of B (i.e. without knowing SK_B) where r is some random number of length k. We can construct a machine C to break the public key encryption scheme. The following is only a sketch of the complete proof. Formally we should simulate an adversary's point of view completely like the proofs given in [3, 4, 6].

C = "On inputs 1^k, PK_B, m_0, m_1 and $c = \mathcal{E}_{PK_B}(m)$ where $m \leftarrow \{m_0,\ m_1\}$:

1. We simulate E's view and answer all the oracle queries involved.
2. For an initiator oracle, set $x = c$ (i.e. $r_A = m$) and denote the identity of the corresponding responder as ID_B.
3. For the second query of the initiator oracle (here we think the query contains $\mathcal{E}_{r_A}(r, \mathrm{ID}_B)$), decrypt the query under m_0 and check if the decrypted message contains a proper coding of ID_B with some number.
4. If it is true, output m_0; otherwise decrypt the query under m_1 and check the validity of the decrypted message again.
5. If it passes, output m_1; otherwise give up.

Claim 2 *The first two messages of Client-Server MAKEP are sent in the correct order.*

Let the adversary E makes $Q(k)$ oracle calls. If we assume that B produces $\mathcal{E}_{r_A}(r_B, \mathrm{ID}_B)$ *before* A sends out $\mathcal{E}_{PK_B}(r_A)$, then the probability that B is queried with the correct value of $\mathcal{E}_{PK_B}(r_A)$ is at most $Q(k) \cdot 2^{-k}$ which is negligible.

CLIENT AUTHENTICATION

Claim 3 *Assuming the discrete logarithm problem is intractable, then only A can compute the correct pair* (β, y) *such that* $g^y \equiv (g^a)^{h(\sigma)}\beta \pmod p$.

The authentication mechanism is similar to Schnorr's identification scheme [23, 24]. Its security is based on the intractability of the discrete logarithm problem where the problem instance is $\log_g g^a$. To see the forgery probability of the client authentication, we consider an adversary E who impersonates A by choosing some b, guessing the correct value of $h(\sigma)$ (which may be obtained from the guessed value of r_B or σ instead) and sending $\beta = g^b(g^a)^{-h(\sigma)}$ and then $y = b$ to B. This is called *crooked proof attack*. The probability of success for this attack is $1/\Psi$ where $\Psi = \min(|\mathcal{H}|,\ 2^k)$ and $\mathcal{H}$ denotes the range of h. In the original papers, Schnorr showed that this success rate cannot be increased unless computing the discrete logarithm is easy. Detailed security analysis can be referred to the original papers as well as [25].

Claim 4 *A cannot send* y *out before B sends* $\mathcal{E}_{r_A}(r_B, \mathrm{ID}_B)$.

The proof is similar to that for Claim 2.

Condition 4: As all the messages are generated by the intended parties in the right order in a single run of the protocol, it is obvious that an adversary cannot obtain any information of r_A or r_B from the messages provided that there exists a secure public key encryption scheme and a secure symmetric key encryption scheme. □