Efficient and Leakage-Resilient Authenticated Key Transport Protocol Based on RSA

SeongHan Shin, Kazukuni Kobara, and Hideki Imai

Institute of Industrial Science, The University of Tokyo,
4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, Japan
shinsh@imailab.iis.u-tokyo.ac.jp, {kobara,imai}@iis.u-tokyo.ac.jp
http://imailab-www.iis.u-tokyo.ac.jp/imailab.html

Abstract. Let us consider the following situation: (1) a client, who communicates with a variety of servers, remembers only one password and has *insecure* devices with very-restricted computing power and built-in memory capacity; (2) the counterpart servers have enormous computing power, but they are not perfectly secure; (3) neither PKI (Public Key Infrastructures) nor TRM (Tamper-Resistant Modules) is available. Our main goal of this paper is to provide its security against the leakage of stored secrets as well as to attain high efficiency on client's side. For those, we propose an efficient and leakage-resilient RSA-based Authenticated Key Establishment (RSA-AKE) protocol suitable for the above situation whose authenticity is based on password *and* an additional stored secret. The RSA-AKE protocol is provably secure in the random oracle model where an adversary is given the stored secret of client and the RSA private key of server. In terms of computation costs, the client is required to compute only one modular exponentiation with an exponent e ($e \geq 3$) in the protocol execution. We also show that the RSA-AKE protocol has several security properties and efficiency over the previous ones of their kinds.

1 Introduction

Since the discovery of public-key cryptography by Diffie and Hellman [7], one of the most important research topics is to design a practical and provably secure protocol for realizing secure channels. In the 2-party setting (e.g., a client and a server), this can be achieved by an authenticated key establishment (AKE) protocol at the end of which the two parties share a common session key to be used for subsequent cryptographic algorithms. Typically, for mutual authentication it requires some information that can be a (high-entropy) secret key (e.g., [4, 19]) to be shared between the parties or a private key corresponding to a public key (e.g., [8, 19, 20]) owned by the parties.

In practice, the high-entropy keys may be substituted by human-memorable passwords chosen from a relatively small dictionary size. Owing to the usability of passwords, password-based AKE protocols have been extensively investigated where a client remembers a short password and the corresponding server holds the password or its verification data. However, there are existing two major attacks on passwords: on-line and off-line dictionary attacks. The on-line dictionary attack is a series of exhaustive search for a secret performed on-line, so that an adversary can sieve out possible secret candidates one by one communicating with the target party. In contrast, the off-line dictionary attack is performed off-line massively in parallel by simply guessing a secret and verifying the guessed secret with recorded transcripts of a protocol. While on-line attacks are applicable to all of the password-based protocols equally, they can be prevented by letting a server take appropriate intervals between invalid trials. But, we cannot avoid off-line attacks by such policies, mainly because the attacks can be performed off-line and independently of the server, resulting in many password-based protocols insecure [21].

At first sight, it seems paradoxical to design a secure AKE protocol with passwords. For that, Bellovin and Merritt opened the door by showing that a combination of symmetric and asymmetric (public-key) cryptographic techniques can provide insufficient information for an adversary to verify a guessed password and thus defeats off-line dictionary attacks [1]. By asymmetric cryptographic techniques, we can roughly divide AKE protocols into two categories: authenticated key agreement (e.g., incorporating the Diffie-Hellman protocol) and authenticated key transport (e.g., using RSA) ones. When it comes to the lower-power computing devices (especially, on client's side), RSA-based AKE protocols would be preferable to the Diffie-Hellman based ones since with an encryption exponent e to be a small prime (e.g., $e = 3$) the computation costs will be drastically decreased. In the next section, we revisit the previous AKE protocols (using password and RSA) from a point of view of how much the leakage of stored secrets affect on its security of each protocol.

1.1 Previous Works

Bellovin and Merritt first proposed Encrypted Key Exchange (EKE) protocols (including the RSA-based EKE) in [1] that was very influential and became the basis for what we call Password-Authenticated Key Exchange (PAKE) protocols[1]. In PAKE protocols, a client is required to remember his/her password *only* (without any device) whereas the counterpart server has its verification data that should be stored securely. In other words, if the stored secret (or, password verification data) of the server is leaked out, the password eventually can be retrieved through off-line dictionary attacks, simply by verifying password candidates one by one using the verification data. When implementing with the RSA function, RSA-based PAKE protocols have to verify whether a server's RSA public key (e, N) is correct or not (i.e., $\gcd(e, \varphi(N)) = 1$) due to the lack of PKI (Public Key Infrastructures). This yields extra computation costs or communication overheads.

Contrary to the PAKE protocols, Lomas et al., introduced an AKE protocol, resistant to off-line dictionary attacks, where a client remembers his/her

[1] A complete list of such protocols can be found in Jablon's research link [17].

password and holds a server's public key in advance whereas the corresponding server has password verification data and its private key both of which should be stored securely [15]. This type of AKE protocols were further studied by Gong [9] and formalized by Halevi and Krawczyk [11]. However, the leakage of one of the stored secrets (the verification data or the private key) may cause a serious problem enough to break its security of the AKE protocol. For example, the leakage of the verification data makes possible for an adversary to retrieve the password through off-line dictionary attacks and thus to impersonate the client. With the leaked private key, an adversary can impersonate the server so that she can get the password through off-line dictionary attacks as well.

Other AKE protocols based on PKI can be found in SSL/TLS (Secure Socket Layer/Transport Layer Security) [10, 13] and SSH (Secure SHell) [12] where a client remembers his/her password and holds a server's public key whereas the corresponding server has password verification data and its private key both of which should be stored securely. The difference from the above AKE protocols is that the parties first establish a secure channel with the server's public key and then the client sends the password for authentication through the secure channel. Note that the client must verify the server's certificate via CRL (Certificate Revocation Lists) or OCSP (Online Certificate Status Protocol), before running the actual protocol, which entails additional computation and communication costs. As for the leakage of the stored secrets, the same discussion of the above paragraph can be done.

1.2 Motivation

The previous password-based AKE protocols have been designed to be secure against an active adversary who controls the communications and usually based on the assumption that the stored secrets would not leak out. However, the leakage of stored secrets is a more practical risk rather than breaking a well-studied cryptographic hard problem. TRM (Tamper-Resistant Modules) of course may be one of the ways to reduce the probability of leakage, but they cannot prevent the damage caused by the leakage as well as it is still hard to make a perfect TRM with low cost. In the password-based AKE protocols, the leakage of stored secrets may occur more serious catastrophe in the following multiple server scenario: a client who would have access to a lot of different servers registered the same password to them for authentication. In this scenario either an adversary or a dishonest server administrator who finds out the password with the leaked stored secrets from one server can impersonate the client to the other remaining servers! The other motivation comes from the fact that all of the previous RSA-based AKE protocols couldn't achieve perfect forward secrecy, if an RSA key pair is fixed, without incorporating the Diffie-Hellman protocol.

1.3 Overview of Our Contributions

Let us consider the following situation for unbalanced wireless networks where a client holds some *insecure* devices (e.g., mobile phones or PDAs) with very-restricted computing power and built-in memory capacity, on the other hand, the counterpart server has enormous computing power but is *not perfectly secure* against various attacks (e.g., virus or hackers). In addition, neither PKI nor TRM is available. In this paper, we propose an efficient and leakage-resilient RSA-based AKE (RSA-AKE) protocol, suitable for the above situation, whose authenticity is based on the client's password *and* an additional stored secret both of which makes possible to extend to the multiple sever scenario with only one password. That is, the respective leakage of stored secret(s) from a client and servers doesn't reveal any information on the password. That implies the client need not change his/her password even if stored secrets are leaked out from either the client or servers.

We also prove its security of the RSA-AKE protocol in the random oracle model under the notion of LR-AKE security where an adversary is given the stored secret of client and the RSA private key of server. Though our protocol is a password-based AKE one, we can avoid even on-line dictionary attacks as long as the leakage of stored secret from client does not happen.

In the RSA-AKE protocol, the client is required to compute only one modular exponentiation with an exponent e ($e \geq 3$) and the remaining computation costs if the pre-computation is allowed are one modular multiplication and some negligible operations. This is because we provide perfect forward secrecy, when the RSA key pair is fixed, by taking a way to update each stored secret of client and server every session. Doing so, the RSA-AKE protocol becomes efficient mainly in aspects of both computation and communication costs.

Organization. This paper is organized as follows. In section 2, we introduce the security model and security definitions. In Section 3, we propose an efficient and leakage-resilient RSA-based AKE (RSA-AKE) protocol, followed by its security proof and discussion in Section 4. Section 5 is devoted to comparison with the previous RSA-based AKE protocols in aspects of security properties and efficiency.

2 Security Model and Definitions

In this section we introduce the security model (based on [6] but extended considering "Leak" queries) and security definitions for the notion of LR-AKE security.

We denote by $\mathcal{C}$ and $\mathcal{S}$ two parties that participate in a protocol P. Each of them may have several *instances* called oracles involved in distinct, possibly concurrent, executions of P where we denote $\mathcal{C}$ (resp., $\mathcal{S}$) instances by $\mathcal{C}^I$ (resp., $\mathcal{S}^J$), or by U in case of any instance. Let us show the capability of adversary $\mathcal{A}$ each query captures:

- $\mathsf{Execute}(\mathcal{C}^I, \mathcal{S}^J)$: This query models passive attacks, where the adversary gets access to honest executions of P between $\mathcal{C}^I$ and $\mathcal{S}^J$ by eavesdropping.
- $\mathsf{Send}(U, m)$: This query models active attacks by having $\mathcal{A}$ send a message to instance U. The adversary $\mathcal{A}$ gets back the response U generates in processing the message m according to the protocol P.

- $\mathsf{Reveal}(U)$: This query handles the misuse of a session key by any instance U. The query is only available to $\mathcal{A}$ if the instance actually "holds" a session key and the latter is released to $\mathcal{A}$.
- $\mathsf{Leak}(U)$: This query handles the leakage of "stored" secrets by any instance U. The query is available to $\mathcal{A}$ since stored secrets might be leaked out due to a bug of the system or physical limitations.
- $\mathsf{Test}(U)$: This oracle is used to see whether the adversary can obtain some information on the challenge session key, by giving a hint on the latter. The Test-query can be asked at most once by the adversary $\mathcal{A}$ and is only available to $\mathcal{A}$ if the instance U is "fresh" in that the session key is not *obviously* known to the adversary. This query is answered as follows: one flips a (private) coin $b \in \{0, 1\}$ and forwards the corresponding session key SK ($\mathsf{Reveal}(U)$ would output) if $b = 1$, or a random value except the session key if $b = 0$.

The goal of the adversary is to break the privacy of the session key (a.k.a., semantic security) in the context of executing P. We denote LR-AKE advantage, by $\mathsf{Adv}_P^{\mathsf{lr-ake}}(\mathcal{A}) = 2\Pr[b = b'] - 1$, as the probability that $\mathcal{A}$ can correctly guess the value of b. The protocol P is said to be LR-AKE secure if $\mathcal{A}$'s advantage is negligible for any adversary $\mathcal{A}$ with polynomial running time t. We formally define the LR-AKE security; this will be necessary for stating meaningful results about our protocol (compared to the other protocols) in Section 5.

Definition 1 (LR-AKE Security) *A protocol P is said to be LR-AKE secure if, when adversary $\mathcal{A}$ asks q_s queries to* Send *oracle and passwords are chosen from a dictionary of size D, the adversary's advantage $\mathsf{Adv}_P^{\mathsf{lr-ake}}(\mathcal{A}) = 2\Pr[b = b'] - 1$ in attacking the protocol P is bounded by*

$$O(q_s/D) + \epsilon(k), \tag{1}$$

for some negligible function $\epsilon(\cdot)$ [2] in the security parameter k. The first term represents the fact that the adversary can do no better than guess a password during each query to Send *oracle.*

The LR-AKE security notion captures the intuitive fact that the protocol P is secure against on-line and off-line dictionary attacks *even if* the leakage of stored secrets from the involving parties happens.

3 An RSA-Based AKE (RSA-AKE) Protocol

Before presenting an RSA-based AKE (for short, RSA-AKE) protocol, we will start by giving some preliminary notations to be used. Let k and l denote the security parameters, where k can be thought of as the general security parameter

[2] Denote with $\mathbb{N}$ the set of natural numbers and with $\mathbb{R}^+$ the set of positive real numbers. We say that a function $\epsilon : \mathbb{N} \rightarrow \mathbb{R}^+$ is *negligible* (in k) if and only if for every polynomial $P(k)$ there exists an $n_0 \in \mathbb{N}$ such that for all $n > n_0$, $\epsilon(k) \leq 1/P(k)$.

for hash functions (say, 160 bits) and l ($l > k$) can be thought of as the security parameter for RSA (say, 1024 bits). We define the RSA function by $\mathsf{RSA}_{N,f}(w) \equiv w^f \bmod N$ for all $w \in \mathbb{Z}_N^\star$. Let D be a dictionary size (cardinality) of passwords (say, 36 bits for alphanumerical passwords with 6 characters). Let $\{0,1\}^\star$ denote the set of finite binary strings and $\{0,1\}^k$ the set of binary strings of length k. If A is a set, then $a \overset{R}{\leftarrow} A$ indicates the process of selecting a at random and uniformly over A. Let "$||$" denote the concatenation of bit strings in $\{0,1\}^\star$.

Let us define secure one-way hash functions (e.g., SHA-1). While $\mathcal{G} : \{0,1\}^\star \rightarrow \mathbb{Z}_N^\star \backslash \{1\}$ denotes a full-domain hash (FDH) function, the other hash functions are denoted $\mathcal{H}_j : \{0,1\}^\star \rightarrow \{0,1\}^k$ for $j = 1, 2, 3$ and 4. Here $\mathcal{G}$ and $\mathcal{H}_j$ are distinct random functions one another. Let $\mathcal{C}$ and $\mathcal{S}$ be the identities of client and server, respectively, with representing each ID $\in \{0,1\}^\star$ as well.

3.1 The RSA-AKE Protocol

We consider the following scenario where a client is communicating with many disparate i servers[3]. We especially focus on unbalanced wireless networks where the client has *insecure* devices (e.g., mobile phones or PDAs) with very-restricted computing power but some memory capacity itself, on the other hand, each server has its database and enormous computing power enough to generate a pair of (public and private) keys of RSA and to perform the RSA decryption function, when e is a small prime number. The choice of RSA key pair $((e, N), (d, N))$ is in general left to the implementations. However, in order to speed-up computation of $\mathsf{RSA}_{N,e}$, e should be chosen to be a small prime with a small number of 1's in its binary representation (e.g., $e = 3$ or $2^{16} + 1$). In addition, neither PKI nor TRM is available at all. Here we propose an efficient and leakage-resilient RSA-based AKE (RSA-AKE) protocol suitable for the above-mentioned situation. The whole protocol is illustrated in Fig. 1.

[Initialization]. During the initialization phase, client $\mathcal{C}$ registers a verification data, computed by a secret and his password, to one of different servers $\mathcal{S}_i$ ($i \geq 1$). At first, server $\mathcal{S}_i$ sends its RSA public key (e, N), which is generated from $\mathsf{RSAKeyGen}(1^l)$, to the client. The latter picks a secret value α_{i1} randomly chosen in $\mathbb{Z}_N^\star$ and registers securely a verification data p_{i1} to server $\mathcal{S}_i$:

$$p_{i1} \equiv \alpha_{i1} + \alpha_0 \bmod N \tag{2}$$

and sets the term $\alpha_0 = pw$ where pw is the client's password[4]. Since both α_{i1} and p_{i1} are in the set of the same length, each of α_{i1} and p_{i1} is a share of $(2, 2)$-threshold secret sharing scheme for α_0 [18].

Then client $\mathcal{C}$ remembers his password *pw and* additionally stores the secret value α_{i1} and the RSA public key (e, N) on insecure devices (e.g., mobile devices

[3] For simplicity, we assign the servers consecutive integer $i \geq 1$ where $\mathcal{S}_i$ can be regarded as i-th server.

[4] The password pw is drawn from password space $\mathbb{D}_{\mathsf{Password}}$ according to a certain probability distribution.

Client $\mathcal{C}$ (Mobile Device)		Server $\mathcal{S}_i$ $(i \geq 1)$
[Initialization]		
	$\xleftarrow{(e,N)}$	$(e,N),(d,N) \leftarrow \mathsf{RSAKeyGen}(1^l)$
$\alpha_{i1} \overset{R}{\leftarrow} \mathbb{Z}_N^\star$, $p_{i1} \equiv \alpha_{i1} + pw \bmod N$	$\xrightarrow{p_{i1}}$	
$\boxed{1, \alpha_{i1}, (e,N)}$		$\boxed{1, p_{i1}, (d,N)}$
[j-th Protocol Execution $(j \geq 1)$]		
$\boxed{j, \alpha_{ij}, (e,N)}$		$\boxed{j, p_{ij}, (d,N)}$
$p_{ij} \equiv \alpha_{ij} + pw \bmod N$		
$W \leftarrow \mathcal{G}(j, p_{ij})$		$W \leftarrow \mathcal{G}(j, p_{ij})$
$x \overset{R}{\leftarrow} \mathbb{Z}_N^\star$, $y \equiv x^e \bmod N$		
$z \equiv y \cdot W \bmod N$	$\xrightarrow{\mathcal{C}, j, z}$	
		If j is incorrect, then reject.
		Otherwise, $y' \equiv z \cdot W^{-1} \bmod N$
		$x' \equiv (y')^d \bmod N$,
	$\xleftarrow{\mathcal{S}_i, V_{\mathcal{S}_i}}$	$V_{\mathcal{S}_i} \leftarrow \mathcal{H}_1(tmp\|\|x')$.
If $V_{\mathcal{S}_i} \neq \mathcal{H}_1(tmp\|\|x)$, then reject.		
Otherwise, $V_{\mathcal{C}} \leftarrow \mathcal{H}_2(tmp\|\|x)$	$\xrightarrow{V_{\mathcal{C}}}$	
$SK_{ij} \leftarrow \mathcal{H}_3(tmp\|\|x)$,		If $V_{\mathcal{C}} \neq \mathcal{H}_2(tmp\|\|x')$, then reject.
$\alpha_{i(j+1)} = \alpha_{ij} + \mathcal{H}_4(tmp\|\|x)$,		Otherwise, $SK_{ij} \leftarrow \mathcal{H}_3(tmp\|\|x')$,
and accept.		$p_{i(j+1)} = p_{ij} + \mathcal{H}_4(tmp\|\|x')$,
		and accept.
$\boxed{j+1, \alpha_{i(j+1)}, (e,N)}$		$\boxed{j+1, p_{i(j+1)}, (d,N)}$

Fig. 1. The initialization and j-th protocol execution of RSA-based AKE (RSA-AKE) protocol where $tmp = \mathcal{C}||\mathcal{S}_i||j||z||p_{ij}$ and the enclosed values in rectangle represent stored secrets of client and server, respectively. The first flow of the protocol execution comprises a key exchange (concretely, key transport), followed by authenticators that are just the hashed values easily computable by both parties. Both of them check the received authenticator prior to accepting the session key.

or smart cards) which may happen to leak the secret α_{i1} and the key (e, N) eventually. The server $\mathcal{S}_i$ also stores the verification data p_{i1} and its RSA private key (d, N) on its databases both of which may be leaked out. Finally, they set a counter j as 1.

[The *j*-th Protocol Execution]. When client $\mathcal{C}$ wants to share an authenticated session key securely with server $\mathcal{S}_i$, they run the j-th ($j \geq 1$) execution of the RSA-AKE protocol as follows. At the start of the j-th protocol execution, client $\mathcal{C}$ and server $\mathcal{S}_i$ hold $(j, \alpha_{ij}, (e, N))$ and $(j, p_{ij}, (d, N))$, respectively, where $p_{ij} \equiv \alpha_{ij} + pw \bmod N$. The client $\mathcal{C}$ should recover the verification data p_{ij} by adding the secret value α_{ij} stored on devices with the password pw kept in his mind. Then the client chooses a random value x (as a keying material) from $\mathbb{Z}_N^\star$ and sends $(\mathcal{C}, j, z)$ to server $\mathcal{S}_i$, after calculating z using a mask generation function as the product of an encryption of x under the RSA public key (e, N) with a full-domain hash of (j, p_{ij}). If the received counter j is correct, the server divides this encrypted value by a hash of the counter and its verification data p_{ij}, and then decrypts the resultant value under its RSA private key (d, N) so as to obtain the keying material x that is used to compute its authenticator $V_{\mathcal{S}_i}$ and a session key SK. Upon receiving $(\mathcal{S}_i, V_{\mathcal{S}_i})$ from the server, client $\mathcal{C}$ computes his authenticator $V_{\mathcal{C}}$ and a session key SK_{ij}, as long as the authenticator $V_{\mathcal{S}_i}$ is valid, and sends $V_{\mathcal{C}}$ to server $\mathcal{S}_i$. Of course, if $V_{\mathcal{S}_i}$ is not the case, client $\mathcal{C}$ wipes off all the temporal data including the keying material and then terminates the protocol. If the authenticator $V_{\mathcal{C}}$ is valid, server $\mathcal{S}_i$ actually computes a session key SK_{ij} that will be used for their subsequent cryptographic algorithms.

At the end of the j-th protocol execution, client $\mathcal{C}$ refreshes the secret value α_{ij} to a new one $\alpha_{i(j+1)}$ for $(j+1)$-th session:

$$\alpha_{i(j+1)} = \alpha_{ij} + \mathcal{H}_4(\mathcal{C}||\mathcal{S}_i||j||z||p_{ij}||x).$$

In the same way, server $\mathcal{S}_i$ also refreshes the verification data p_{ij} to a new one: $p_{i(j+1)} = p_{ij} + \mathcal{H}_4(\mathcal{C}||\mathcal{S}_i||j||z||p_{ij}||x')$. Finally, client $\mathcal{C}$ stores $(j+1, \alpha_{i(j+1)}, (e, N))$ on his devices and server $\mathcal{S}_i$ stores $\big(j+1, p_{i(j+1)}, (d, N)\big)$ on its databases for the next session.

Only if client $\mathcal{C}$ inputs the right password pw and the corresponding secret value α_{ij} to server $\mathcal{S}_i$ *and* server $\mathcal{S}_i$ uses the right verification data p_{ij} and its RSA private key (d, N), they can generate the correct authenticators, share the same session key and refresh each stored secret to a new one all of which are derived from the keying material x. Without any leaked secret the probability of guessing the other's keying material is $1/2^l$.

4 Security Proof for the RSA-AKE Protocol

In this section we show the RSA-AKE protocol of Fig. 1. is provably secure in the random oracle model[5], under the assumption that inverting an RSA problem is

[5] To analyze the security of certain cryptographic constructions Bellare and Rogaway introduced an idealized security model called the random oracle model [3]. Random oracles are used to model cryptographic hash functions such as SHA-1 which produce a random value for each new query. Note that security in the random oracle model is only heuristic: it does not imply security in the real world. Nevertheless, the random oracle model is a useful tool for validating natural cryptographic constructions.

hard. Informally speaking, 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, since generating the valid client's authenticator after computing z *or* generating the valid server's authenticator falls into on-line dictionary attacks (which can be easily prevented and detected).

4.1 Security Proof

In order to simplify the security proof, we only consider the first two flows of the j-th protocol execution (unilateral authentication of $\mathcal{S}$ to $\mathcal{C}$) [6]. This is due to the well-known fact that the basic approach in folklore for adding authentication to an AKE protocol is to use the distributed Diffie-Hellman key or the keying material to construct a simple "authenticator" for the other party [2, 6]. Therefore, the security proof with unilateral authentication can be extended to one with mutual authentication by simply adding the authenticator of $\mathcal{C}$ (the third flow) as in Fig. 1. However, this entails a more complicated proof.

Here we assert that the two-flows $\mathsf{RSA\text{-}AKE}$ protocol distributes semantically-secure session keys and provides unilateral authentication for the server $\mathcal{S}$ under the one-wayness of RSA. Since the security of the $\mathsf{RSA\text{-}AKE}$ protocol only depends on the password, the LR-AKE security can be proven with some adjustments and changes of the proof for the RSA-based PAKE protocol [6][7].

Theorem 1 (LR-AKE/UA Security) *Let P be the two-flows* $\mathsf{RSA\text{-}AKE}$ *protocol, where passwords are chosen from a dictionary of size D. For any adversary $\mathcal{A}$ within a polynomial time t, with less than q_s active interactions with the parties (*Send*-queries), q_p passive eavesdroppings (*$\mathsf{Execute}$*-queries) and q_l leakages (*Leak*-queries), and asking q_g and q_h hash queries to $\mathcal{G}$ and any $\mathcal{H}_i$ respectively,* $\mathsf{Adv}_P^{\mathsf{lr-ake}}(\mathcal{A}) \le 4\varepsilon$ *and* $\mathsf{Adv}_P^{\mathsf{S-auth}}(\mathcal{A}) \le \varepsilon$*, with ε upper-bounded by*

$$(q_{\mathcal{C}} + 3q_{\mathcal{S}})/D + 6q_l \cdot \mathsf{Succ}_{RSA}^{\mathsf{ow}}\left(q_h^2, t + 2q_h^2\tau_{rsa}\right) + \frac{q_{\mathcal{C}}}{2^{k_1}} + \frac{2q_{\mathcal{C}} + (q_g + q_h)^2}{2^{l+1}} \ , \quad (3)$$

where $q_{\mathcal{C}}$ and $q_{\mathcal{S}}$ denote the number of $\mathcal{C}$ and $\mathcal{S}$ instances involved during the attack (each upper-bounded by $q_p + q_s$), k_1 is the output length of $\mathcal{H}_1$, l is the security parameter, and τ_{rsa} is the computational time needed for $\mathsf{RSA}_{N,d}$.

Due to the lack of space, we omit the security model, formal definitions and the proof but those will appear in the full version. Here we justify the main terms in the security result. Some ways for the adversary $\mathcal{A}$ to break the protocol are: (1) guess a password and makes an on-line trial with respect to $\mathcal{C}^I$ and $\mathcal{S}^J$ involved during the attack. The $\mathsf{RSA\text{-}AKE}$ protocol is secure against on-line dictionary attacks since the advantage of $\mathcal{A}$ essentially grows with the ratio of

[6] For the sake of brevity, we omit the index i in the proof.

[7] The security reduction to the one-wayness of RSA is based on [5] where a challenge RSA problem is included in the answer of many hash queries so that the adversary is useful to the simulator with greater probability.

interactions to the number of passwords. Hence the term $(q_{\mathcal{C}} + 3q_{\mathcal{S}})/D$; (2) use the authenticator $V_{\mathcal{S}}$ to check the correct password. But this requires the ability to compute $\mathsf{RSA}_{N,d}(z \times W^{-1})$. Hence the term $6q_l \cdot \mathsf{Succ}^{\mathsf{ow}}_{RSA}(\cdot,\cdot)$; (3) send a correct authenticator $V_{\mathcal{S}}$, but being lucky. Hence the term $q_{\mathcal{C}}/2^{k_1}$. Additional negligible terms come from very unlikely collisions. All the remaining kinds of attacks need some information about the password.

In the proof, the simulator has to "guess" which hash query to $\mathcal{G}$ will be used by the adversary to produce the correct bit b, resulting in a factor of q_l (the number of Leak-queries) in the success probability to invert the RSA encryption function.

4.2 Discussion

The only restriction on an adversary is that she cannot replace an RSA public key (e, N), stored on a client's devices, with a different one (e', N'). If it is possible, the adversary can store a fake RSA public key (e', N') such that $(e, N) \neq (e', N')$ and $\gcd(e', \varphi(N')) \neq 1$. As a result, the RSA encryption function $\mathsf{RSA}_{N',e'}(x) \equiv x^{e'} \bmod N'$ is no longer a permutation on $\mathbb{Z}^{\star}_{N'}$ which maps an element $x \in \mathbb{Z}^{\star}_{N'}$ to the set of e'-residues (a proper subset of $\mathbb{Z}^{\star}_{N'}$). Since the adversary knows the factorization of N', it is easy to check whether an element $x \in \mathbb{Z}^{\star}_{N'}$ is e'-residues or not. This is generally called e-residue attack that is one of the off-line dictionary attacks.

However, one has to notice the intrinsic distinction of adversary's behavior in the RSA-AKE and PAKE protocols. In the latter, an adversary is a kind of network adversary who can impersonate the involving parties and control all of the communications for e-residue attacks. On the other hand, an adversary, who is willing to perform e-residue attacks in the RSA-AKE protocol, should first steal a client's device, change an RSA public key and return back to the client. Then the adversary impersonates the corresponding server so as to narrow down the password candidates. (We call this "replacement attack" for convenience.)

One of the possible ways to thwart e-residue attacks in the RSA-AKE protocol is as follows. If the client has noticed the leakage of stored secrets $(\alpha_j, (e, N))$ or couldn't run the protocol within a fixed number of trials, he runs one of the RSA-based PAKE protocols (e.g., [6, 22]) by using p_j instead of the password pw. After establishing a secure channel, the client refreshes the secret α_j to α_{j+1} and stores the correct RSA public key. In this case, the adversary cannot mount e-residue attacks successively since its security now depends on the refreshed secret α_{j+1} as well as the latter is completely independent from α_j. In order to continue the e-residue attacks, the adversary should steal the device to get α_{j+1}, change the RSA public key, return back to the client, and then impersonate the server again.

More simple way against e-residue attacks is to exploit insecure devices in the practical point of view. Suppose that a client has a mobile phone on which a server's RSA public key (e, N) is stored, on the other hand, he also holds a memory card separately for the mobile phone where the fingerprint of (e, N) is kept. Of course, TRM is not needed at all. Whenever the client runs the RSA-AKE protocol with the server, he should at first do the integrity check for (e, N), as Halevi and Krawczyk's protocol [11] does[8], by inserting the memory card to the phone and confirming the correctness of (e, N).

5 Comparison

In this section we compare the RSA-AKE protocol of Section 3.1 with the previous AKE protocols using password and RSA. In order to be fair, we instantiate with the RSA function if a public key encryption is not specified in the relevant previous works. For simplifying its discussion, we omit additional computation and communication costs of SSL/TLS and SSH in order to verify the counterpart's certificate.

5.1 As for Security Properties

The RSA-AKE protocol may seem to be similar to Halevi and Krawczyk's protocol [11] since a client holds a server's RSA public key and remembers his password after the initialization phases of both protocols. However, in their protocol if an adversary changes the RSA public key (e, N), stored on the client's devices, with a one (e', N') generated by $\mathsf{RSAKeyGen}(1^l)$ of the adversary, she can discover the password through off-line dictionary attacks with only one interaction with the client.

As for several security properties, we show the comparative results in Table 1 and 2. For an easier comparison, the following three cases are considered.

- Case1: This is the case that an adversary gets the stored secret associated with the password from client $\mathcal{C}$.
- Case2: This is the case that an adversary gets the verification data associated with the password from server $\mathcal{S}$.
- Case3: This is the case that an adversary gets the RSA private key from server $\mathcal{S}$.

In the RSA-AKE protocol, Case1, Case2 and Case3 correspond to the leakage of $\alpha_{i(j+1)}$, $p_{i(j+1)}$ and (d, N), respectively. We can see in Table 1 that the RSA-AKE protocol guarantees semantic security of session keys even if Case1 and Case3 happen at the same time. In terms of semantic security against Case2, the key-establishment part of SSL/TLS and SSH in the public-key based user authentication mode is the only survivor simply because the password is not used for client's authentication but for protecting the client's private key.

In terms of security of password against Case1 and Case2 (in Table 2), we claim the following theorem:

[8] For the integrity check in their protocol, a client receives a public key from a server and then compares the key with one stored on devices. Unfortunately, their protocol is insecure against the "replacement attack" described above (refer to Section 5).

Table 1. Comparison of RSA-based AKE protocols in a situation where no perfect TRM is available.

Protocols		Client's possessions			Semantic security of session key against			
		PW[*1]	SS[*2]	PI[*3]	Case1	Case2	Case3	Case1∨ Case3
P	RSA-IPAKE [6]	$\surd$			secure	insecure	secure	secure
A	PEKEP [22]							
K	CEKEP [22]							
E	SNAPI [16]							
	SNAPI-X [16]	$\surd$		$\surd$[*4]				
MAKE[*5] [11]		$\surd$		$\surd$[*7]	secure	insecure	insecure	insecure
MA-DHKE[*6] [11]		$\surd$		$\surd$[*4,*7]	secure	insecure	insecure	insecure
SSL/TLS, SSH[*8]		$\surd$		$\surd$	secure	insecure	insecure	insecure
SSL/TLS, SSH[*9]		$\surd$	$\surd$	$\surd$	insecure[*10]	secure	insecure	insecure
RSA-AKE		$\surd$	$\surd$	$\surd$	secure	insecure	secure	secure[*11]

*1: Human-memorable secret (i.e., password)
*2: Stored secret: a secret value, a signing (decryption) key or a symmetric key
*3: Public information: a CA's verification key, an encryption key or its fingerprint
*4: Public parameters for the Diffie-Hellman protocol: let $\mathbb{G}$ be a finite, cyclic group of prime order q and g be a generator of $\mathbb{G}$
*5: Mutual Authentication and Key Exchange of Section 3.4
*6: Mutual Authentication and Diffie-Hellman Key Exchange of Section 3.4
*7: Cached server's RSA public key (e.g., $\mathcal{H}(e, N)$)
*8: Key-establishment part of SSL/TLS and SSH in the password-based user authentication mode
*9: Key-establishment part of SSL/TLS and SSH in the public-key based user authentication mode with a password-protected private key
*10: $E_{pw}(d)$ where d is an RSA private key and E is a symmetric encryption with pw as its key
*11: Theorem 1

Theorem 2 (Security of Password) *The password in the* RSA-AKE *protocol remains information-theoretically secure against off-line dictionary attacks even after either* Case1 *or* Case2 *happens.*

Proof. The proof is straightforward. First, we think of an adversary who obtains the stored secret $\alpha_{i(j+1)}$ of client $\mathcal{C}$ and is trying to deduce the password pw. Since $\alpha_{i(j+1)}$ is completely independent from pw, $\alpha_{i(j+1)}$ doesn't reveal any information about the password.

$$H(pw) = H\left(pw|\alpha_{i(j+1)}\right) \tag{4}$$

where $H(X)$ denotes the (Shannon) entropy of X and $H(X|Y)$ denotes the conditional entropy of X conditioned on Y. Second, we think of the security of password against an adversary who obtains the stored secret $p_{i(j+1)}$ of server

Table 2. Comparison of RSA-based AKE protocols in a situation where no perfect TRM is available (con't).

Protocols		Security^{*1} of password		Extension^{*2}	Perfect forward secrecy (Case1 $\vee$ Case2 $\vee$ Case3)
		Case1	Case2		
P	RSA-IPAKE [6]	$\bigcirc$	X ($\triangle^{*3}$)	impossible	PFS can be achieved only if server $\mathcal{S}$ changes its RSA key pair every time.
A	PEKEP [22]				
K	CEKEP [22]				
E	SNAPI [16]				
	SNAPI-X [16]	$\bigcirc$	$\triangle^{*4}$		
MAKE [11]		$\bigcirc$	$\triangle^{*3}$	impossible	not achieved
MA-DHKE [11]		$\bigcirc$	$\triangle^{*3}$	impossible	achieved^{*5}
SSL/TLS, SSH		$\bigcirc$	X ($\triangle^{*3}$)	impossible	achieved^{*5}
SSL/TLS, SSH		$\triangle$	$\bigcirc$	possible	achieved^{*5}
RSA-AKE		$\bigcirc^{*6}$	$\bigcirc^{*6}$	possible^{*7}	achieved

*1: Security level against an adversary who either obtains client's devices (Case1) or intrudes severs (Case2) in order to retrieve the client's password in each case: $\bigcirc$ guarantees the security of password against both on-line and off-line dictionary attacks; $\triangle$ guarantees the security of password against on-line, but not off-line attacks; and X guarantees the security of password against neither on-line nor off-line attacks.
*2: Extension to the multiple server scenario with only one password
*3: A client registers password verification data computed with a particular one-way function of the password, $f(pw)$, to the server instead of pw. Doing this somewhat slows down off-line dictionary attacks of an adversary who obtained the server's database.
*4: $g^{\mathcal{H}_5(\mathcal{C}||\mathcal{S}||pw)}$ where $\mathcal{H}_4$ is a secure one-way hash function
*5: Due to the Diffie-Hellman protocol
*6: Information-theoretically secure
*7: The number of stored secrets α_{ij} grows linearly to the number of servers.

$\mathcal{S}_i$ and is trying to deduce the password pw. However, the adversary cannot get any information about the password, simply because $p_{i(j+1)}$ is one share of $(2,2)$-threshold (perfect) secret sharing scheme. As a result, the password is information-theoretically secure as a secret value of $(2,2)$-threshold secret sharing scheme. $\square$

Contrary to the RSA-AKE protocol, the other AKE protocols don't have the security of password since their stored secrets in either the client or the server(s) contain enough information to succeed in retrieving the relatively short password with off-line dictionary attacks.

One of the important security properties is perfect forward secrecy. According to [20], we informally say that a protocol P achieves perfect forward secrecy if the disclosure of "long-term" secrets of the involving parties does not compromise the semantic security of session keys from previous sessions (even though that compromises the authenticity and thus the security of new sessions). In the $\mathsf{RSA\text{-}AKE}$ protocol, perfect forward secrecy can be interpreted as follows: if an adversary is given with $\alpha_{i(j+1)}, p_{i(j+1)}$ and (d, N), such that $p_{i(j+1)} = \alpha_{i(j+1)} + pw$, in the $(i+1)$-th session, the adversary is trying to deduce the previous session key SK_{ij} for the i-th session[9]. In order to compute x

$$\begin{aligned} x &= \mathsf{RSA}_{N,d}(z/W) = \mathsf{RSA}_{N,d}(z)/\mathsf{RSA}_{N,d}\left(\mathcal{G}(j, \underline{p_{ij}})\right) \\ &= \mathsf{RSA}_{N,d}(z)/\mathsf{RSA}_{N,d}\left(\mathcal{G}(j, \underline{\alpha_{ij}} + pw)\right) \\ &= \mathsf{RSA}_{N,d}(z)/\mathsf{RSA}_{N,d}\left(\mathcal{G}(j, p_{i(j+1)} - \mathcal{H}_4(\cdot||\cdot||\cdot||\underline{p_{ij}}||\underline{x}))\right) , \end{aligned} \tag{5}$$

the adversary should know α_{ij} or p_{ij} both of which are completely independent from $\alpha_{i(j+1)}$ and $p_{i(j+1)}$ without x. Remember that α_{ij} and p_{ij} are uniformly distributed in $(\mathbb{Z}_N^\star)^2$. That is, the $\mathsf{RSA\text{-}AKE}$ protocol achieves perfect forward secrecy[10] even if server $\mathcal{S}_i$ would use the same RSA key pair for many sessions.

5.2 As for Efficiency

Since password-based AKE protocols have been motivated by the very practical implementations and widely used even in wireless networks, we analyze computation costs of client and communication overheads in the $\mathsf{RSA\text{-}AKE}$ protocol while comparing with those of each protocol execution, providing perfect forward secrecy, in Table 3. We denote by l (resp., k) the security parameter for the RSA function and the Diffie-Hellman protocol (resp., for the hash functions and random numbers). The number of modular exponentiations is a major factor to evaluate efficiency of a cryptographic protocol because that is the most power-consuming operation. So we count the number of modular exponentiations as computation costs of client $\mathcal{C}$. The figures in the parentheses are the remaining costs after pre-computation. For brevity, we denote by RSA-Exp. (resp., DH-Exp.) the number of RSA modular exponentiations with an exponent e (resp., the number of Diffie-Hellman modular exponentiations with an exponent of 160-bits long). In terms of communications overheads, the length of identities is excluded and $|\cdot|$ indicates its bit-length.

With respect to computation costs in the $\mathsf{RSA\text{-}AKE}$ protocol, client $\mathcal{C}$ is required to compute one modular exponentiation with an exponent e $(e \geq 3)$ and one modular multiplication. In particular, the remaining costs after pre-computation is only one modular multiplication and additional operations for modular additions and hash functions. On the other hand, MAKE protocol doesn't allow pre-computation and doesn't provide perfect forward secrecy. With respect to communication overheads in the $\mathsf{RSA\text{-}AKE}$ protocol, it requires a bandwidth of $(l + 2k)$ bits approximately.

[9] The adversary is in the game to distinguish the i-th session key given by Test oracle.

[10] For an RSA-based AKE protocol without incorporating the Diffie-Hellman protocol, it seems impossible to prove perfect forward security in a sense of [2, 14] since this kind of protocol is actually a key transport one.

Table 3. Comparison of RSA-based AKE protocols as for efficiency.

Protocols		Computation costs of client $\mathcal{C}$		Communication overheads
		DH-Exp.	RSA-Exp. with e	
P	RSA-IPAKE [6]		$m+1$ when $e \geq 3$, (m) ^{*1}	$(m+2)l+3k$
	PEKEP [22]		$n+1$ when $e \geq 3$, (n) ^{*2}	$2l+4k+\lvert e\rvert$
A	CEKEP [22]		$2n$ when $e \geq 3$, $(2n-1)$ ^{*3} or 2, (2) ^{*4}	$3l+6k+\lvert e\rvert+\lvert n\rvert$
K	SNAPI [16]		Primality test of large e and 1 ^{*5}, (Primality test of e)	$2l+4k+\lvert e\rvert$
E	SNAPI-X [16]	2, (2)	Primality test of large e and 1 ^{*5}, (Primality test of e)	$3l+4k+\lvert e\rvert$
MAKE [11]			1 when $e \geq 3$, (1)	$2l+3k+\lvert e\rvert$
MA-DHKE [11]		2, (1)	1 when $e \geq 3$, (1)	$4l+3k+\lvert e\rvert$
SSL/TLS, SSH		2, (1)	1 when $e \geq 3$, (1)	$3l+\lvert E\rvert$
SSL/TLS, SSH		2, (1)	1 when $e \geq 3$ and 1 RSA-Exp. with d, (1 RSA-Exp. with d) ^{*6}	$3l$
RSA-AKE			1 when $e \geq 3$, (0)	$l+2k$

*1: m is the system parameter
*2: $n = \lfloor \log_e N \rfloor$
*3: $n = \lceil \log_e \omega^{-1} \rceil$ where $0 < \omega \leq 2^{-80}$
*4: 2 modular exponentiations each having an exponent of $\lceil \log_e \omega^{-1} \rceil$ bits where $0 < \omega \leq 2^{-80}$
*5: 1 modular exponentiation with an exponent e having the following explicit requirements on e and N (in order to enforce the relative primality of e and $\varphi(N)$). One is to set e to be a prime, in the range of $2^l + 1 \leq e < 2^{l+1}$, greater than N. The other is to set e to be a prime such that $e \geq \sqrt{N}$ and $(N \bmod e) \nmid N$.
*6: 1 modular exponentiation with an exponent d

Acknowledgements

The authors appreciate anonymous reviewers for their helpful comments.

References

1. S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-based Protocols Secure against Dictioinary Attacks. In *Proc. of IEEE Symposium on Security and Privacy*, pages 72-84. IEEE Computer Society, 1992.
2. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In *Proc. of EUROCRYPT 2000*, LNCS 1807, pages 139-155. Springer-Verlag, 2000.
3. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proc. of ACM CCS '93*, pages 62-73, 1993.
4. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Proc. of CRYPTO '93*, LNCS 773, pages 232-249. Springer-Verlag, 1993.
5. M. Bellare and P. Rogaway. The Exact Security of Digital Signatures: How to Sign with RSA and Rabin. In *Proc. of Eurocrypt '96*, LNCS 1070, pages 399-416. Springer-Verlag, 1996.

6. D. Catalano, D. Pointcheval, and T. Pornin. IPAKE: Isomorphisms for Password-based Authenticated Key Exchange. In *Proc. of CRYPTO 2004*, LNCS 3152, pages 477-493. Springer-Verlag, 2004. The full version is available at `http://www.di.ens.fr/∼pointche/slides.php?reference=CaPoPo04`.
7. W. Diffie and M. Hellman. New Directions in Cryptography. In *IEEE Transactions on Information Theory*, Vol. IT-22(6), pages 644-654, 1976.
8. W. Diffie, P. van Oorschot, and M. Wiener. Authentication and Authenticated Key Exchange. In *Proc. of Designs, Codes, and Cryptography*, pages 107-125, 1992.
9. L. Gong. Optimal Authentication Protocols Resistant to Password Guessing Attacks. In *Proc. of IEEE Computer Security Foundation Workshop*, pages 24-29, 1995.
10. A. Frier, P. Karlton, and P. Kocher. The SSL 3.0 Protocol. Netscape Communication Corp., 1996. available at `http://wp.netscape.com/eng/ssl3/`.
11. S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. February 1999.
12. IETF (Internet Engineering Task Force). Secure Shell (secsh) Charter. `http://www.ietf.org/html. charters/secsh-charter.html`
13. IETF (Internet Engineering Task Force). Transport Layer Security (tls) Charter. `http://www.ietf.org/ html.charters/tls-charter.html`
14. J. Katz, R. Ostrovsky, and M. Yung. Forward Secrecy in Password-Only Key Exchange Protocols. In *Proc. of SCN 2002*, LNCS 2576, pages 29-44. Springer-Verlag, 2002.
15. M. Lamos, L. Gong, J. Saltzer, and R. Needham. Reducing Risks from Poorly Chosen Keys. In *Proc. of the 12th ACM Symposium on Operating System Principles*, ACM Operating Systems Review, pages 14-18, 1989.
16. P. MacKenzie, S. Patel, and R. Swaminathan. Password-Authenticated Key Exchange Based on RSA. In *Proc. of ASIACRYPT 2000*, LNCS 1976, pages 599-613. Springer-Verlag, 2000. The full version is available at `http://cm.bell-labs.com/who/philmac/bib.html`.
17. Phoenix Technologies Inc., Research Papers on Strong Password Authentication. available at `http://www.integritysciences.com/links.html`.
18. A. Shamir. How to Share a Secret. In *Proc. of Communications of the ACM*, Vol. 22(11), pages 612-613, 1979.
19. V. Shoup. On Formal Models for Secure Key Exchange. IBM Research Report RZ 3121, 1999.
20. S. B. Wilson, D. Johnson, and A. Menezes. Key Agreement Protocols and their Security Analysis. In *Proc. of IMA International Conference on Cryptography and Coding*, December 1997.
21. T. Wu. A Real-world Analysis of Kerberos Password Security. In *Proc. of Network and Distributed System Security Symposium*, February 1999.
22. M. Zhang. New Approaches to Password Authenticated Key Exchange based on RSA. In *Proc. of ASIACRYPT 2004*, LNCS 3329, pages 230-244. Springer-Verlag, 2004. Cryptology ePrint Archive, Report 2004/033, available at `http://eprint.iacr.org/2004/033`.