EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

Anonymous gateway-oriented password-based authenticated key exchange based on RSA

Fushan Wei*, Chuangui Ma and Qingfeng Cheng

Abstract

A gateway-oriented password-based authenticated key exchange (GPAKE) is a three-party protocol, which allows a client and a gateway to establish a common session key with the help of an authentication server. To date, most of the published protocols for GPAKE have been based on Diffie-Hellman key exchange. In this article, we present the first GPAKE protocol based on RSA, then prove its security in the random oracle model under the RSA assumption. Furthermore, our protocol can resist both *e*-residue and undetectable on-line dictionary attacks. Finally, we investigate whether or not a GPAKE protocol can achieve both client anonymity and resistance against undetectable on-line dictionary attacks by a malicious gateway. We provide an affirmative answer by adding client anonymity with respect to the server.

Preprint submitted to EURASIP JWCN October 16, 2011 to our basic protocol.

Keywords: RSA, password-based authentication, gateway, anonymity, random oracle

1. Introduction

1.1. Password-based authenticated key exchange

Password-based authenticated key exchange (PAKE) protocols allow users to securely establish a common key over an insecure open network only using a low-entropy and human-memorable password. Owing to the low entropy of passwords, PAKE protocols are susceptible to so-called *dictionary* attacks [1]. Dictionary attacks can be classified into three types [1]: on-line, off-line, and undetectable on-line dictionary attacks. In on-line dictionary attacks, an adversary first guesses a password, and tries to verify the password using responses from a server in an on-line manner. On-line password guessing attacks can be easily detected, and thwarted by counting access failures. In off-line dictionary attacks, an adversary tries to determine the correct password without the involvement of the honest parties based on information obtained during previous executions of the protocol. Thus, the attacker can freely guess a password and then check if it is correct without limitation in the number of guesses. The last type is undetectable on-line dictionary at-2 tacks, where a malicious insider tries to verify a password guess in an on-line manner. However, a failed guess cannot be detected by the honest client or the server. The malicious insider participates in the protocol legally and un-detectably many times to get sufficient information of the password. Among these attacks, on-line dictionary attack is unavoidable when low-entropy pass-words are used, the goal of PAKE protocols is to restrict the adversary to on-line dictionary attacks only. In other words, off-line and undetectable on-line dictionary attacks should not be possible in a PAKE protocol.

In 1992, Bellovin and Merritt first presented a family of password protocols known as encrypted key exchange (EKE) protocols [2] which can resist dictionary attacks. They also investigated the feasibility of implementing EKE using three different types of public-key cryptographic techniques: RSA, ElGamal, and Diffie-Hellman key exchange. They found that RSA-based PAKE in their protocol is not secure against *e*-residue attacks [2,3], and pointed out that EKE is only suitable for implementation using Diffie-Hellman key exchange. From then on, lots of PAKE protocols based on Diffie-Hellman have been proposed [1,2,4-9]. While the approach of designing PAKE protocols with RSA is far from maturity and perfection. In 1997, Lucks presented a scheme called OKE (open key exchange) [10] which is based on RSA. It was later found to be insecure against

* Correspondence: weifs831020@163.com
Department of Information Research, Zhengzhou Information, Science and Technology Institute, Zhengzhou 450002, China

a variant of *e*-residue attacks because of MacKenzie et al. [11]. Furthermore, the authors modified OKE and proposed the first secure RSA-based PAKE protocol SNAPI. Since SNAPI protocol required that the RSA public exponent should be a larger prime than RSA modular, it is not practical. Later, Zhang proposed PEKEP and CEKEP protocols [12], which allow using both large and small prime numbers as RSA public exponents. To resist the *e*-residue attack, PEKEP protocol needs multiple RSA encryptions, and it is not very efficient. In 2007, Park et al. presented another efficient RSA-EPAKE protocol [13] which can resist the *e*-residue attack based on number-theoretic techniques. Unfortunately, as pointed by Youn et al. [14], RSA-EPAKE is insecure against a *separation attack*. Though the attack can be easily avoided by limiting the number of failed trials, an adversary can get remarkably much information of the password from single trial. Therefore, the separation attack is still a threatening attack against RSA-EPAKE protocol.

1.2. Related work

In 2005, Abdalla et al. [4] put forward the first gateway-oriented password-based authenticated key exchange (GPAKE) protocol among a client, a gateway, and an authentication server. The client and the server initially share a common password for authentication, but the session key is generated between the client and the gateway via the help of the server. In addition to the usual notion of semantic security of the session key, two additional security goals, namely key privacy with respect to honest-but-curious server and pass-word protection with respect to malicious gateway, are considered to capture dishonest behaviors of the server and the gateway, respectively. In 2006, Byun et al. [8] showed that the GPAKE protocol proposed by Abdalla et al. [4] was vulnerable to an undetectable on-line dictionary attack. A malicious gateway can iteratively guess a password and verify its guess without being detected by the server. They also proposed a countermeasure for the attack by exploiting MAC of keying material sent to the authentication server from the client. In 2008, Shim [15] showed that Byun's countermeasure was still insecure against the same undetectable on-line dictionary attack contrary to the claim in [8] that it was. In addition, Shim also designed its enhanced version (S-GPAKE) using a symmetric encryption algorithm to overcome the attack. Nevertheless, Yoon et al. [16] pointed out that the S-GPAKE protocol was inefficiently and incorrectly designed. Recently, Abdalla et al. [6] presented an anonymous variant of the original GPAKE protocol [4] with similar efficiency. They proposed a new model having stronger security which captured all the security goals in a single security game. The new security model also allowed corruption of the participants. They proved the security of the new protocol in the enhanced security model. However, partially owing to client anonymity, the new protocol is still subjected to undetectable on-line dictionary attacks. It is quite interesting to ask whether there exists a GPAKE protocol which can achieve both client anonymity and resistance against undetectable on-line dictionary attacks.

1.3. Our contribution

In this article, we investigate GPAKE protocol based on RSA. We first propose an efficient RSA-based GPAKE protocol. The new protocol involves three entities. The client and the server share a short password while the client and the gateway, respectively, possess a pair of RSA keys. However, all the RSA public/private keys are selected by the entities rather than distributed by a certificate authentication center, so no public-key infrastructure is needed. To resist *e*-residue attacks, the client uses the public key *e* of an 80-bit prime. The proposed protocol can be resistant to *e*-residue attacks and provably-secure under the RSA assumption in the random oracle model.

To achieve previously mentioned requirements, the authenticators and the final session key in the proposed protocol rely on different random numbers. In this way, the authenticators between the client and the server will leak no information of the password to the gateway, and the session key established between the client and the gateway is private to the server. Furthermore, standard techniques in threshold-based cryptography can also be used to achieve threshold version of the proposed protocol. It is worth pointing out that our protocol does not require public parameters. The client and the server only need to establish a shared password in advance and do not need to establish other common parameters such as generators of a finite cyclic group. This is appealing in environments where clients have insufficient resources to authenticate public parameters.

We also investigate whether or not a GPAKE protocol can achieve both client anonymity and resistance against undetectable on-line dictionary attacks by a malicious gateway. These two requirements seem to contradict each other (it seems that the server needs to know who the user is in order to resist undetectable on-line dictionary attacks). Nevertheless, this can be reconciled by saying that a server learns whether it is interacting with a user that belongs to a defined set of authorized users, but nothing more about which user it is in that set. We provide an affirmative answer to the above question by adding client anonymity to our GPAKE protocol based on RSA.

The remainder of this article is organized as follows. In Section 2, we recall the communication model and

some security definitions of GPAKE protocols. In Section 3, we present our protocol and show that the new protocol is provably-secure under the RSA assumption in the random oracle model. We show in Section 4 how to add client anonymity to the basic scheme using symmetric private information retrieval (SPIR) protocols [17]. We conclude this article in Section 5.

2. Security model

In this section, we recall the security model for GPAKE protocols introduced in [4]. We will prove security of our protocol in this model. We refer the reader to [4] for more details.

2.1. Overview

A GPAKE protocol allows a client to establish an authenticated session key with a gateway via the help of an authentication server. The password is shared between the client and the server for authentication. It is assumed that the communication channel between the gateway and the server is authenticated and private, but the channel connecting the client to the gateway is insecure and under the control of an adversary.

The main security goal of the GPAKE protocol is to securely generate a session key between the client and the gateway without leaking information about the password to the gateway. To achieve this goal, Abdalla et al. [4] defined three security notions to capture dishonest behaviors of the client, the authentication server, and the gateway, respectively. The first one is *semantic security* of the session key, which is modeled by a Real-Or-Random (ROR) game; the second one is *key privacy* with respect to the server, which entails that the session key established between the client and the gateway is unknown to the passive server; and the last one is *server password protection* against a malicious gateway, which means that the gateway cannot learn any information about the client's password from the authentication server.

Protocol participants

The participants in a gateway-oriented password-based key exchange are the client $C \in \mathcal{C}$, the gateway $G \in \mathcal{G}$, and the authentication server $S \in \mathcal{S}$. We denote by $\mathcal{U}$ the set of all the participants (i.e., $\mathcal{U} = \mathcal{C} \cup \mathcal{G} \cup \mathcal{S}$) and by U a non-specific participant in $\mathcal{U}$.

Long-lived keys

Each client $C \in \mathcal{C}$ holds a password pw_c. Each server $S \in \mathcal{S}$ holds a vector of passwords $pw_S = \langle pw_C \rangle_{C \in \mathcal{C}}$ with an entry for each client. pw_c and pw_s are also called the long-lived keys of client C and server S, respectively.

2.2. Security Model

The security model we adopted here is the ROR model of Abdalla et al. [5]. The adversary's capabilities are modeled through queries. During the execution, the adversary may create several concurrent instances of a participant. Let U^i denote the instance i of a participant U. The list of oracles available to the adversary is as follows:

- $Execute(C^i, G^j)$: This query models passive eavesdropping of a protocol execution between a client instance C^i and a gateway instance G^j. At the end of the execution, a transcript is given to the adversary, which logs everything an adversary could see during the execution.
- $Send(U^i, m)$: This query models an active attack against the client or gateway instance U^i, in which the adversary may intercept a message and then modify it, create a new one, or simply forward it to the intended recipient. Instance U^i executes as specified by the protocol and sends back its response to the adversary.
- $Test(U^i)$: This query is used to measure the semantic security of the session key of instance U^i, if the latter is defined. If the key is not defined, return the undefined symbol $\perp$. Otherwise, return either the session key held by instance U^i if $b = 1$ or a random key of the same size if $b = 0$, where b is a hidden bit chosen uniformly at random at the beginning of the experiment defining the semantic security of session keys.

In the ROR model, the adversary can ask *Test* queries for all the sessions. All the *Test* queries will be answered using the same random bit b that was chosen at the beginning of the experiment. In other words, the keys returned by the *Test* oracle are either all real or all random. However, in the random case, the same random key is returned for two partnered instances (see the notion of partnering below). The goal of the adversary is to guess the value of the hidden bit b used to answer *Test* queries. The adversary is said to be successful if it guesses b correctly.

It should be noted that *Reveal* oracle exists in the Find-Then-Guess (FTG) model is not available to the adversary in the ROR model. However, since the adversary in FTG model is restricted to asking only a single query to the *Test* oracle, the ROR security model is actually stronger than the FTG security model. Abdalla et al. demonstrated that proofs of security in the ROR model can be easily translated into proofs of security in the FTG model. For more details, refer to [5].

2.3. Security notions

We give the main definitions in the following. The definition approach of partnering uses session identifications and partner identifications. The session identification is

the concatenation of all the messages of the conversation between the client and the gateway instances before the acceptance. Two instances are partnered if they hold the same non-null session identification.

Definition 1. *A client instance C^i and a gateway instance G^j are said to be partnered if the following conditions are met: (1) both C^i and G^j accept; (2) both C^i and G^j share the same session identification; (3) the partner identification for C^i is G^j and vice versa; (4) no instance other than C^i and G^j accepts with a partner identification equal to C^i or G^j.*

The adversary is only allowed to perform tests on *fresh* instances. Otherwise, it is trivial for the adversary to guess the hidden bit b. The freshness notion captures the intuitive fact that a session key is not trivially known to the adversary.

Definition 2. *An instance of a client or a gateway is said to be fresh in the current protocol execution if it has accepted.*

Semantic security

Consider an execution of the key exchange protocol $\mathcal{P}$ by the adversary $\mathcal{A}$ in which the latter is given access to *Execute, Send* oracles, as well as to *Test* oracle calls to fresh instances. The goal of the adversary is to guess the value of the hidden bit b used by the *Test* oracle. Let *Succ* denote the event in which the adversary successfully guesses the hidden bit b used by *Test* oracle.

Definition 3. *The advantage of an adversary $\mathcal{A}$in violating the AKE semantic security of the protocol $\mathcal{P}$in the ROR sense, when passwords are uniformly drawn from a dictionary $\mathcal{D}$, is defined as*

$$Adv_{\mathcal{P},\mathcal{D}}^{ake-ror}(\mathcal{A}) = 2 \cdot \Pr[Succ] - 1.$$

The advantage function of the protocol $\mathcal{P}$is defined as

$$Adv_{\mathcal{P},\mathcal{D}}^{ake-ror}(t, R) = max\{Adv_{\mathcal{P},\mathcal{D}}^{ake-ror}(\mathcal{A})\},$$

where maximum is over all $\mathcal{A}$with time-complexity at most t and using resources at most R (such as the number of oracle queries).

We have the following definition of semantic secure GPAKE protocol, which is the same as in [4].

Definition 4. *A GPAKE protocol $\mathcal{P}$is said to be semantically secure if the advantage $Adv_{\mathcal{P},\mathcal{D}}^{ake-ror}(t, R)$is only negligibly larger than $kn/|\mathcal{D}|$, where n is number of active sessions, and k is a constant.*

Note that $k = 1$ is the best one can hope for since an adversary that simply guesses the password in each of the active sessions has an advantage of $n/|\mathcal{D}|$.

Key privacy

In GPAKE protocols, the session key between the client and the gateway is established with the help of the server. In order to reduce the amount of trust one puts into the server, we require that the session key should be even indistinguishable to an honest but curious server who knows all the passwords of the clients. The notion of key privacy with respect to the server was first introduced in [5] to capture this security requirement.

To define the notion of key privacy, we consider a server which knows all the passwords of the clients, and behaves in an honest but curious manner. We give the server access to all the oracles, but restricts the server to testing session keys generated by two oracles. To achieve this aim, we use a new type of *TestPair* oracle which was first introduced in [5]. The *TestPair* oracle is defined as follows:

- *TestPair*(C^i,G^j): If the client instance C^i and the gateway instance G^j do not share the same key, then return the undefined symbol $\perp$. Otherwise, return either the session key established between C^i and G^j if $b = 1$ or a random key of the same size if $b = 0$, where b is a hidden bit chosen uniformly at random at the beginning of the experiment defining the key privacy of session keys.

Consider an execution of the key exchange protocol $\mathcal{P}$ by an adversary $\mathcal{A}$ with access to all the passwords held by the server as well as to the *Execute, Send*, and *TestPair* oracles. Let *Succ* denote the event in which the adversary is successful in guessing the hidden bit b used by *TestPair* oracle. The advantage of an adversary $\mathcal{A}$ in violating the key privacy of the protocol $\mathcal{P}$ in the ROR sense ($Adv_{\mathcal{P},\mathcal{D}}^{ake-kp}(\mathcal{A})$) and the advantage function of $\mathcal{P}(Adv_{\mathcal{P},\mathcal{D}}^{ake-kp}(t, R))$, when passwords are uniformly drawn from a dictionary $\mathcal{D}$, can be defined as in Definition 3.

Definition 5. *A GPAKE protocol $\mathcal{P}$is said to achieve key privacy if the advantage $Adv_{\mathcal{P},\mathcal{D}}^{ake-kp}(t, R)$ is negligible.*

Server password protection

One of the security goals of GPAKE protocol is to prevent the gateway from learning the client's password that is stored in the server. If the adversary interacts q times with the server, then the probability that it can distinguish the true password from a random one in the dictionary should be only negligibly larger than $q/|\mathcal{D}|$. However, this does not rule out the possibility of undetectable on-line dictionary attacks by a malicious gateway. A malicious gateway can iteratively guess a password and verify its guess until it finds the correct password. To resist such attacks, we consider a malicious gateway $\mathcal{A}$ who guesses a password and verifies its guess by interacting with the server. If a failed guess will

not be detected by the server, then we say the malicious gateway is successful. Let $Adv_{\mathcal{P},\mathcal{D}}^{ake-uoda}(\mathcal{A})$ denotes the success probability of the gateway.

Definition 6. *A GPAKE protocol $\mathcal{P}$can resist undetectable on-line dictionary attacks if $Adv_{\mathcal{P},\mathcal{D}}^{ake-uoda}(\mathcal{A})$is negligibly larger than $kn/|\mathcal{D}|$, where n is number of active sessions, and k is a constant.*

3. Our GPAKE protocol based on RSA

In this section, we describe our GPAKE protocol based on RSA, and present its security results.

3.1. Description

Define hash functions H_1,H_2, $H_3 : \{0,1\}^* \rightarrow \{0, 1\}^k$, and $H : \{0,1\}^* \rightarrow Z_n$, where k is a security parameter, e.g., k = 160. We assume that H_1,H_2,H_3, and H are independent random functions in the following.

The protocol runs among a client, a gateway, and an authentication server. Its description is given in Figure 1. The client and the authentication server initially share a lightweight string pw, the password, uniformly drawn from the dictionary $\mathcal{D}$. The client has generated a pair of RSA keys n, e, and d, where n is a large positive integer equal to the product of two primes of the same size, e is an 80-bit prime relatively prime to $\varphi(n)$, and d is a positive integer such that $ed \equiv 1 \bmod \varphi(n)$. The gateway also has generated a pair of RSA keys n',e', and d', where n' is a large positive integer equal to the product of two primes of the same size, e' is a positive integer relatively prime to $\varphi(n')$, and d' is a positive integer such that $e'd' \equiv 1 \bmod \varphi(n')$. The channel connecting the gateway to the authentication server is assumed to be authenticated and private. The protocol proceeds as follows:

1. The client C sends her public key (n, e) and a random number $r_1 \in \{0, 1\}^k$ to the gateway G, and G just forwards the message and her RSA public key (n', e') to the authentication server.
2. The authentication server S verifies if e is an 80-bit prime, and n is an odd integer. S may also verify that the integer n is large enough, e.g., $n > 2^{1023}$. If e is not an 80-bit prime or n is not an odd integer, S rejects; otherwise, S selects three random numbers $x_1, x_2 \in Z_n^*$, and $r_2 \in \{0, 1\}^k$. S then computes $y_1 = x_1^e \bmod n$ and $y_2 = x_2^e \bmod n$, S also computes $w = H(pw, x_2, C, G, n, e, n', e', r_1, r_2, y_2)$ and checks whether $gcd(w, n) = 1$. If $gcd(w, n) = 1$, S computes $z = y_1 \cdot w \bmod n$ and sends (r_2,z, y_2) to the gateway. Upon receiving (r_2,z, y_2), G sends (n', e', r_2,z, y_2) to C.
3. Upon receiving (n', e', r_2, z, y_2) from G, C verifies if n' is an odd integer and n' is large enough, e.g., $n' > 2^{1023}$. C selects a random number $b_1 \in Z_{n'}^*$. C then decrypts $x_2 = y_2^d \bmod n$, computes w using her password pw and x_2, then checks if w and n are relatively prime. If $gcd(w, n) = 1$, C decrypts $x_1 = (w^{-1} \cdot z)^d \bmod n$, computes $c_1 = b_1^{e'} \bmod n'$. Finally, C computes $\mu = H_1(x_1, C, G, n, e, n', e', r_1, r_2, y_2, z, c_1)$ and sends (c_1,μ) to G. Upon receiving (c_1,μ), G selects a random number $b_2 \in Z_n^*$, computes $c_2 = b_2^e \bmod n$, sends (c_1,c_2, μ) to S.
4. S checks whether μ is valid or not. If μ is valid, S computes her authenticator $\eta = H_2(x_1, C, G, n, e, n', e', r_1, r_2, y_2, z, c_1, c_2)$. Finally, S sends η to G.
5. Upon receiving η, G decrypts $b_1 = c_1^{d'} \bmod n'$, sets the session key $sk = H_3(b_1, b_2, ID)$, where ID is the concatenation of all the exchanged messages. G sends η and c_2 to C.
6. C checks whether η is valid or not. If valid, C decrypts $b_2 = c_2^d \bmod n$ and sets the session key to be $sk = H_3(b_1, b_2, ID)$, where ID is the concatenation of all the exchanged messages.

In RSA-based protocols, security against e-residue attacks [3] has to be considered. To void such an e-residue attack, we adopt the approach of [18] and require the public key of the client is an 80-bit prime. However, [18] is basically a two-factor protocol, and their main concern is security against replacement attacks. Hence, in this context, we still briefly prove the security against e-residue attacks of our protocol. Suppose the adversary $\mathcal{A}$ generates the RSA parameter (n,e), where e is an 80-bit prime and $gcd(e, \varphi(n)) = e$. Upon receiving (n, e), the authentication server S randomly chooses $x_1, x_2 \in Z_n^*$, computes $y_1 = x_1^e \bmod n$ and $y_2 = x_2^e \bmod n$, then S calculates w using the password pw and x_2. Finally, S sends (r_2, z, y_2) back to the adversary, where $z = y_1 \cdot w \bmod n$. To mount an e-residue attack, first of all, the adversary should correctly find out the committed value x_2. Since $y_2 = x_2^e \bmod n$, which is equivalent to $e \cdot ind_g x_2 \equiv ind_g y_2 \bmod \varphi(n)$. The congruence has exactly e solutions because $gcd(e, \varphi(n)) = e$ and $e | ind_g y_2$. The success probability that the adversary correctly find out the committed value is $1/e$, which is negligible since e is an 80-bit prime.

Remark 1 To resist e-residue attacks, we require that the client use the public key e of an 80-bit prime, the server needs to test the primality for the 80-bit prime. However, there is no restriction on the gateway's public key e'. This is because the gateway's public key is only used to establish the session key and has nothing to do with the password.

Remark 2 In case of $n > 2^{1023}$, the computational load for generating an 80-bit prime is less than for a single RSA decryption, and the computational load for the primality test of an 80-bit prime is less than for a single RSA encryption with an 80-bit exponent. Hence, our

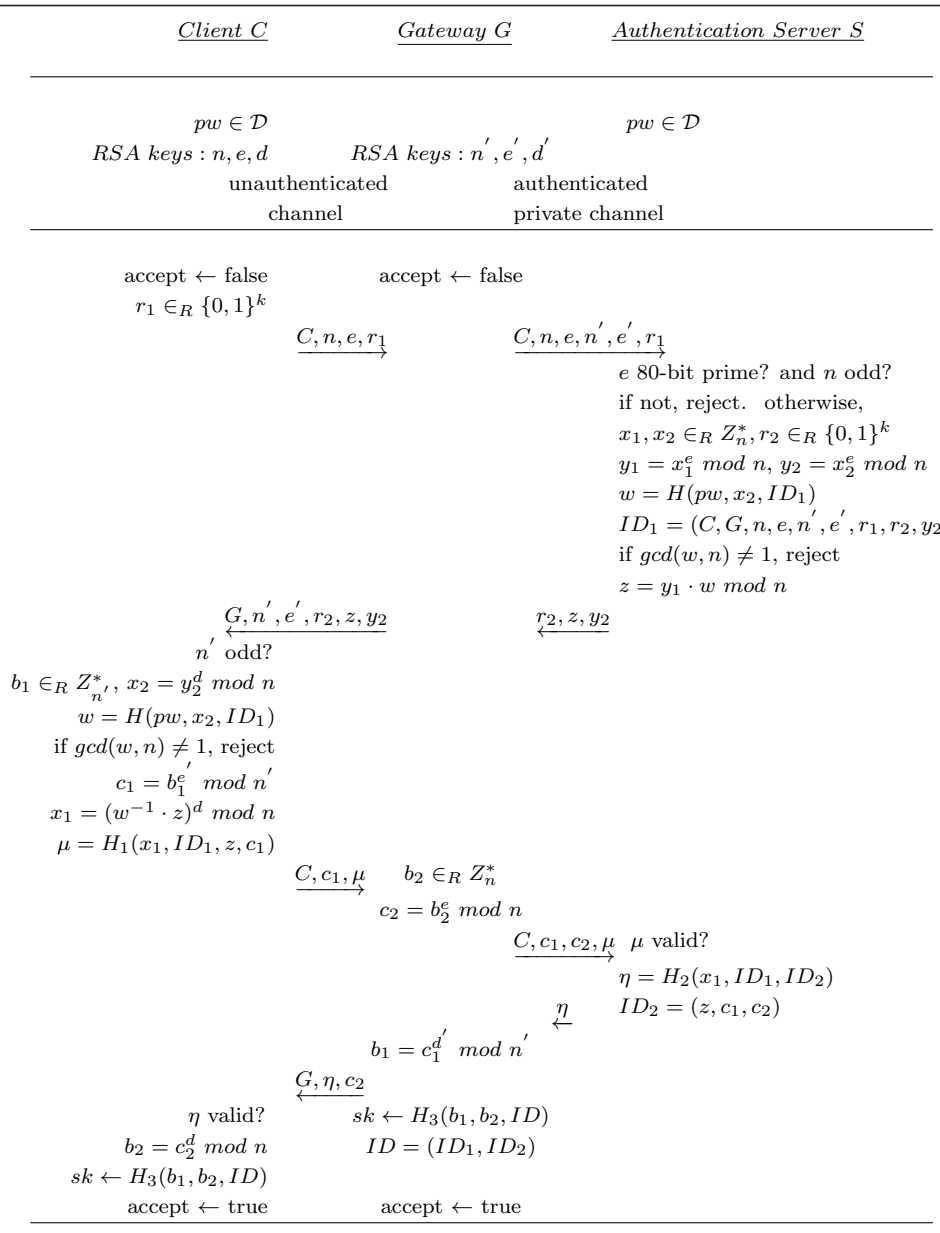

Figure 1 Gateway-oriented password-authenticated key exchange protocol based on RSA.

protocol is quite efficient in computation cost. Furthermore, if we exclude perfect forward secrecy from consideration, we need not to generate them in each session, this further improves the efficiency of our protocol.

3.2. Security

In this section, we prove the security of our protocol within the formal model of security given in Section 2. In our analysis, we assume the intractability of the RSA problem.

RSA assumption [13]

Let l be the security parameter of RSA. Let key generator GE define a family of RSA functions, i.e., $(e, d, n) \leftarrow GE(1^l)$, where n is the product of two primes of the same size, gcd $(e, \varphi(n)) = 1$, and $ed = 1 \bmod \varphi(n)$. For any probabilistic polynomial-time algorithm $\mathcal{C}$ in running time t, the following probability

$$Adv_{\mathcal{C}}^{rsa}(t) = \Pr\begin{pmatrix} x^e = c \mod n : (e, d, n) \leftarrow GE(1^l), \\ c \in_R \{0,1\}^l, x \leftarrow \mathcal{C}(1^l, c, e, n) \end{pmatrix}$$

is negligible. In the following, we use $Adv^{rsa}(t)$ to denote $\max_C\{Adv_C^{rsa}(t)\}$, where the maximum is taken over all the polynomial-time algorithms of running time t.

Semantic security

As the following theorem states, our protocol is a secure gateway-oriented password-based key exchange protocol as long as the RSA problem is intractable. The proof of security assumes $\mathcal{D}$ to be a uniformly distributed dictionary and of size smaller than 2^k. The proof of Theorem 3.1 can be found in Appendix A.

Theorem 3.1. *Let $\mathcal{A}$be an adversary which runs in time t and makes Q_{send}, $Q_{send} \le |D|$, queries of type Send to different instances. Then, the adversary's advantage in attacking the semantic security of the proposed protocol is bounded by*

$$Adv_{\mathcal{P},\mathcal{D}}^{ake-ror}(\mathcal{A}) \le \frac{2Q_{send}}{|\mathcal{D}|} + (3Q_{send} + 2Q_{execute})Adv^{rsa}(O(t)) + \frac{(2Q_{send} + Q_{execute})Q_{oh}}{\phi(n)} + \frac{Q_{send}}{2^{k-1}} + \frac{Q_{send}}{2^{79}},$$

where $Q_{execute}$ denotes the number of queries of type Execute, and Q_{oh} denotes the number of random oracle calls.

Key privacy

As the following theorem shows, our protocol achieves the goal of key privacy as long as the RSA problem is intractable.

Theorem 3.2. *Let $\mathcal{A}$be an adversary which runs in time t and makes $Q_{execute}$ queries of type Execute to different instances. Then, the adversary's advantage in attacking the key privacy of the proposed protocol is bounded by*

$$Adv_{\mathcal{P},\mathcal{D}}^{ake-kp}(\mathcal{A}) \le Q_{execute}Adv^{rsa}(O(t)).$$

The proof of Theorem 3.2 is similar to the proof of Lemma A.1 in Appendix A. The only difference is that in this case the adversary knows the passwords of all the clients. However, this only brings negligible advantage to the adversary since the authenticators and the session keys rely on different random numbers. In order to distinguish the session key from random numbers chosen from $\{0, 1\}^k$, the adversary still needs to break RSA. We omit the proof of Theorem 3.2 for simplicity.

Server password protection

As is shown by the following theorem, a malicious gateway cannot do much better than eliminating one password from the list of possible candidates with each interaction with the server. As a result, after q interactions with the server, the advantage of a malicious gateway would be only negligibly larger than $q/|\mathcal{D}|$. Furthermore, a failed guess of the malicious gateway will be detected by the authentication server. A malicious gateway cannot iteratively guess a password and verify its guess without being detected. Hence, our protocol can resist undetectable on-line dictionary attacks. The proof of Theorem 3.3 can be found in Appendix B.

Theorem 3.3. *Let $\mathcal{A}$be a malicious gateway which runs in time t and makes Q_{send} queries of type Send to server instances. Then, the advantage of the malicious gateway in violating the resistance to undetectable on-line dictionary attacks of the proposed protocol is bounded by*

$$Adv_{\mathcal{P},\mathcal{D}}^{ake-uoda}(\mathcal{A}) \le \frac{Q_{send}}{|\mathcal{D}|} + \frac{Q_{send}}{2^k} + \frac{Q_{send}}{2^{80}}.$$

4. Adding client anonymity

Anonymity is one of the most important security goals of protocols on public networks. Many of the privacy problems that arise out of Internet use can be solved using anonymous Internet connections such that a client's actions are unlinkable. Implementing anonymity of clients not only protects their personal information but also reduces the chances of attacks based on impersonation. In this section, we show how to add client anonymity to our protocol.

The basic idea is same as Abdalla et al.'s [6]. We assume that there are many gateways, but the authentication server is unique. In order to add client anonymity, we try to hide the client identity to the authentication server using SPIR [17] protocols. An SPIR protocol allows a client to retrieve an item from a server in possession of a database without revealing which item they are retrieving, and it also allows for the restricting of the number of items a given client may retrieve. When the gateway receives an authorization request from a client, the gateway can run an SPIR protocol with the authentication server, such that the server does not know the real identity of the client and the gateway only gets the answer to the actual client. More precisely, the authentication server can be seen as a dynamic database. For each authorization request, the authentication server computes the answers for all the possible clients, and the gateway retrieves the one it is interested in. At the end of the SPIR protocol, the authentication server does not know which answer the gateway gets and the gateway will not get more than the number of the values it is allowed to retrieve.

Our RSA-based GPAKE can be efficiently implemented with any good SPIR protocol. Specifically, we assume that each client owns a password indexed by i, and the server manages a database of size N, which contains all the passwords for each client. In order to introduce anonymity to the protocol in Section 3, we do as follows: upon receiving of a *Send-query* with input (C_j, n, e, r_1), the gateway conceals the real identity of the client and sends (n, e, n', e', r_1) to the server. Upon

receiving (n, e, n', e', r_1), the server dynamically generates a database by computing the answers for each message (C_i, n, e, n', e', r_1), and thus for all the possible clients C_i, since it does not know which one is interacting with the gateway. More precisely, the server chooses $r_2 \in \{0,1\}^k, x_1 \in Z_n^*$, and for each C_i, the server also chooses $x_{2i} \in Z_n^*$, computes $y_1 = x_1^e \bmod n$ and $y_{2i} = x_{2i}^e \bmod n$. The dynamic database consists of all the blocks $Bi = (r_2, z_i, y_{2i})$, where $z_i = y_1 \cdot wi \bmod n$ and $w_i = H(pw_i, x_{2i}, C_i, G, n, e, n', e', r_1, r_2, y_{2i})$. Then, the gateway runs the SPIR protocol to get the correct B_j, while preserving the anonymity of the client. The remains are the same as the proposed GPAKE protocol except that the values μ and η are computed as $H_1(x_1, G, n, e, n', e', r_1, r_2)$ and $H_2(x_1, G, n, e, n', e', r_1, r_2, c_1, c_2)$, respectively.

It is worth pointing out that achieving client anonymity, our protocol still can resist the undetectable on-line dictionary attack in the sense that a failed guess of the malicious gateway will be detected by the server. To impersonate a client successfully, the malicious gateway needs recover y_1 using the guessed password of the victim client and then obtains x_1 by decrypting y_1. If the guessed password is not correct, then the value μ is not valid and the server will detect the attack, and then some measures should be taken to protect the passwords of the clients.

5. Conclusion

In this article, we investigate the design of RSA-based GPAKE protocols. First, we develop a new GPAKE protocol using RSA public-key cryptosystem. The proposed protocol is secure against e-residue attacks. Then, we provide a formal security analysis of our protocol under the RSA assumption and the random oracle model. We also show that our protocol is secure against undetectable on-line dictionary attacks. Finally, we investigate whether or not such a protocol can achieve both client anonymity and resistance to undetectable on-line dictionary attacks. We give an affirmative answer by adding client anonymity to our basic protocol.

Appendix A. Proof of Theorem 3.1

We prove Theorem 3.1 using similar techniques as described in [19]. We define a series of hybrid experiments. In each experiment, we modify the way session keys are chosen for instances involved in protocol execution. We start by choosing random session keys for instances for which the *Execute* oracle is called. Then, we continue to choose random session keys for instances for which the *Send* oracle is called. These instances are gradually changed over five hybrid experiments and in the last experiment, all the session keys are selected uniformly at random. Thus, the adversary $\mathcal{A}$ cannot distinguish them from random numbers. We denote these hybrid experiments by P_0, P_1, ..., P_4 and by $Adv(\mathcal{A}, P_i)$ the advantage of $\mathcal{A}$ when participating in experiment P_i.

Experiment P_0

This describes the real adversary attack. During the attack, the adversary $\mathcal{A}$ makes a number of oracle calls (*Send*, *Execute*, and *Test*) as specified in Section 2. In addition, the adversary $\mathcal{A}$ has access to four independent random oracles

$$H : \{0,1\}^* \to Z_n, H_1, H_2, H_3 : \{0,1\}^* \to \{0,1\}^k.$$

Each random oracle H_i(or H) maintains a list of input-output pairs (q_0, r_0), (q_1, r_1)···. On a new input q, H_i (or H) checks if q was queried before. If there exists q_i in the list such that $q = q_i$, then the random oracle returns the corresponding r_i as its reply. If q is not in the list, the random oracle chooses a random number r, returns r as its reply and adds the pair (q, r) to its list. It is clear that $Adv(\mathcal{A}) = Adv(\mathcal{A}, P_0)$.

Experiment P_1

In this experiment, the *Execute* oracle is modified so that the session keys of instances for which *Execute* is called are selected uniformly at random, that is, if the oracle *Execute* (C^i, G^j) is called, then the session key sk is set equal to a random number selected from $\{0,1\}^k$, rather than the output of the random oracle H_3. The following lemma shows that modifying the *Execute* oracle in this way affects the advantage of $\mathcal{A}$ by a negligible value.

Lemma Appendix A.1

For every polynomial-time adversary $\mathcal{A}$making $Q_{execute}$ oracle calls of type Execute,

$$|Adv(\mathcal{A}, P_1) - Adv(\mathcal{A}, P_0)| \le 2Q_{execute}Adv^{rsa}(O(t)) + Q_{execute}Q_{oh}/\phi(n),$$

where Q_{oh} denotes the number of random oracle calls, and t is the running time of $\mathcal{A}$.

Proof. We prove this lemma by showing how any advantage that $\mathcal{A}$ has in distinguishing P_1 from P_0 can be used to break RSA. In experiment P_0, the session key is the output of the random oracle H_3 on the input (b_1, b_2, ID), where ID is the concatenation of all the exchanged messages. If the adversary does not know b_1 and b_2, she cannot distinguish the output of H_3 from a random number uniformly selected from $\{0,1\}^K$. Hence, the adversary $\mathcal{A}$ can distinguish P_1 and P_0 if and only if $\mathcal{A}$ can recover the integers b_1 and b_2. Let $p_{b_1}(p_{b_2})$ denote the probability that $\mathcal{A}$ recovers the integer b_1 (b_2).

For a easier analysis, we let the adversary win if the adversary recovers the integer b_2. To bound p_{b_2}, we consider the following two games G_1 and G_2.

Game G_1

The adversary $\mathcal{A}$ carries out an honest execution between the instances C^i and G^j as the protocol description. When the game ends, the adversary $\mathcal{A}$ outputs her guess of the integer b_2.

Game G_2

This game is similar to game G_1 except that we use private oracles when we compute w, μ, and η.

Let $p_{b_2}(G_1)$ denote the probability that $\mathcal{A}$ makes a correct guess of b_2 in game G_1. Likewise, $p_{b_2}(G_2)$ denote the probability that $p_{b_2} = p_{b_2}(G_1)$ makes a correct guess of b_2 in game G_2. It is clear that $\mathcal{A}$. Let $AskH$ denote the event that $\mathcal{A}$ queries random oracle H on (pw, x_2, C, G, n, e, n', e', r_1, r_2, y_2). Let $AskH_{1,2}$ denote the event that $\mathcal{A}$ queries random oracle H_1 on (x_1, C, G, n, e, n', e', r_1, r_2, y_2,z, c_1) or H_2 on (x_1, C, G, n, e, n', e', r_1, r_2, y_2, z, c_1, c_2), while $AskH$ does not happen.

Then, we have

$$|p_{b_2}(G_1) - p_{b_2}(G_2)| \le \Pr[AskH] + \Pr[AskH_{1,2}],$$
$$p_{b_2}(G_1) \le \Pr[AskH] + \Pr[AskH_{1,2}] + p_{b_2}(G_2).$$

Let Q_{oh} denote the number of random oracle calls to H_1 and H_2 by $\mathcal{A}$ In the following, we bound the probabilities of events $AskH$ and $AskH_{1,2}$, and also show that $p_{b_1}(G_2) \le Adv^{rsa}(O(t))$.

Given RSA public key (n, e) and integer $c \in_R Z_n$, we construct an efficient algorithm $\mathcal{C}$ to decrypt c as follows: algorithm $\mathcal{C}$ runs the adversary $\mathcal{A}$ exactly as in game G_2 except that when simulate the authentication server, $\mathcal{C}$ first chooses two random numbers x, $x' \in Z_n^*$, computes $y_2 = x^e \cdot c \bmod n$, and set z to be $z = x'^e \cdot c \cdot w \bmod n$, where w is uniformly chosen from Z_n^*. Finally, when simulate the gateway, $\mathcal{C}$ set c_2 to be c. If event $AskH$ happens, which means $\mathcal{A}$ queries random oracle H on (pw, x_2, C, G, n, e, n', e', r_1, r_2, y_2), where $x_2^e = x^e \cdot c \bmod n$, then we can decrypt c by $x_2/x \bmod n$. If event $AskH$ does not happen, then z is a random number from $\mathcal{A}$'s view. $\mathcal{A}$ can select a random number $x' \in Z_n^*$ as her guess on x_1 and verifies the correctness of x' by comparing μ (or η). Then,

$$\Pr(AskH) = Adv_{\mathcal{C}}^{rsa}(O(t)) \le Adv^{rsa}(O(t)),$$
$$\Pr(AskH_{1,2}) = Q_{oh}/\phi(n).$$

Similarly, if $\mathcal{A}$'s output (denoted by b_2) in game G_2 is correct, then b_2 is the decryption of c.

$$p_{b_2}(G_2) = Adv_{\mathcal{C}}^{rsa}(O(t)) \le Adv^{rsa}(O(t)), p_{b_2} \le 2Adv^{rsa}(O(t)) + Q_{oh}/\phi(n).$$

Assume that $\mathcal{A}$ makes $Q_{execute}$ oracle calls of type *Execute* in the hybrid experiment P_1, then

$$|Adv(\mathcal{A}, P_1) - Adv(\mathcal{A}, P_0)| \le 2Q_{execute}Adv^{rsa}(O(t)) + Q_{execute}Q_{oh}/\phi(n).$$

Before we present the experiments P_2, P_3, and P_4, we describe *Send* oracles which an active adversary $\mathcal{A}$ uses.

- $Send_0(C^i)$: the instance C^i selects a pair of RSA public/private keys e, d, n, and a random number $r_1 \in \{0, 1\}^k$. It returns C, n, e, and r_1 to the adversary $\mathcal{A}$.
- $Send_1(G^j, C, n, e, r_1)$: the instance G^j selects a pair of RSA public/private keys (e', d', n'), sends (C, n, e, n', e', r_1) to the server. G^j obtains (r_2, z, y_2) as the reply of the server. It returns (n', e', r_2, z, y_2) to the adversary $\mathcal{A}$.
- $Send_2(C^i, n', e', r_2, z, y_2)$: the instance C^i verifies if n' is big enough, i.e., $n' > 1023$. Then, C^i selects a random number $b_1 \in Z_{n'}^*$, and decrypts $x_2 = y_2^d \bmod n$, then computes w using her password pw and x_2, checks if w and n are relatively prime. If $gcd(w, n) = 1$, C^i decrypts $x_1 = (w^{-1} \cdot z)^d \bmod n$, computes $c_1 = b_1^{e'} \bmod n'$. Finally, C^i computes $\mu = H_1(x_1, C, G, n, e, n', e', r_1, r_2, y_2, z, c_1)$ and returns (c_1, μ) to the adversary $\mathcal{A}$.
- $Send_3(G^j, c_1, \mu)$: the instance G^j selects a random number $b_2 \in Z_n^*$, computes $c_2 = b_2^e \bmod n$, sends (c_1,c_2,μ) to S. G^j obtains η as the reply of the server. It decrypts $b_1 = c_1^{d'} \bmod n'$, sets the session key $sk = H_3(b_1, b_2, ID)$, where ID is the concatenation of all the exchanged messages. It returns η and c_2 to the adversary $\mathcal{A}$.
- $Send_4(C^i, \eta, c_2)$: the instance C^i checks whether η is valid or not. If η is invalid, it rejects. Otherwise, it decrypts $b_2 = c_2^d \bmod n$, and computes $sk = H_3(b_1, b_2, ID)$, where ID is the concatenation of all the exchanged messages.

A message is said to have been oracle-generated if it was output by an instance; otherwise, it is said to have been adversarially-generated. A message generated by instance U^i is said to have been U^i-oracle-generated.

Experiment P_2

In this experiment, an instance G^j receives a C^i-oracle-generated message (C, n, e, r_1) in a $Send_1$ oracle call. If both C^i and G^j accept, they are given the same random session keys $sk \in \{0, 1\}^k$, and if G^j accepts but C^i does not accept, then only G^j receives a random session key, and no session key is defined for C^i.

Lemma Appendix A.2

For every polynomial-time adversary $\mathcal{A}$making Q_{send} oracle calls of type Send to different instances,

$$|Adv(\mathcal{A}, P_2) - Adv(\mathcal{A}, P_1)| \le 2Q_{send}Adv^{rsa}(O(t)),$$

where t is the running time of $\mathcal{A}$.

Proof. Assume that G^j returns (G, n', e', r_2, z, y_2) to the adversary according to the description of the protocol

after receiving a C^i-oracle-generated message (C, n, e, r_1) in a $Send_1$ oracle call. Since the RSA public key (e, n) was generated by C^i, not by $\mathcal{A}$, the private key d is not known to $\mathcal{A}$. As shown in the proof of Lemma A.1, the probability for $\mathcal{A}$ to recover the random number x_1 is upper bounded by $Adv^{rsa}(O(t))$. Hence, except for a probability as small as $Adv^{rsa}(O(t))$, G^j has received a C^i -oracle-generated message in a $Send_3$ oracle when G^j accepts. Similarly, if C^i accepts, then it has received a G^j-oracle-generated message in a $Send_4$ oracle call. If both C^i and G^j accept, then they share the same session key which is equal to the output of the random oracle H_3 on (b_1, b_2, ID), where ID is the concatenation of all the exchanged messages. Hence, the modification of the session keys of C^i and G^j affects the adversary's advantage by a value as small as $Adv^{rsa}(O(t))$. Since $\mathcal{A}$ makes Q_{send} oracle calls of type $Send$ to different instances, $\mathcal{A}$'s advantage in distinguishing between P_2 and P_1 is upper bounded *by* $Q_{send}Adv^{rsa}(O(t))$.

Experiment P_3

In this experiment, an instance C^i receives a G^j-oracle-generated message (n', e', r_2, z, y_2) in a $Send_2$ oracle call, while the instance G^j has received a C^i -oracle-generated message (C, n, e, r_1) in a $Send_1$ oracle call. If both C^i and G^j accept, then they are given the same random session keys $sk \in \{0, 1\}^k$. It is clear that the advantage of $\mathcal{A}$ in P_3 is the same as its advantage in P_2.

Lemma Appendix A.3

For every polynomial-time adversary $\mathcal{A}$making Q_{send} oracle calls of type Send to different instances,

$$Adv(\mathcal{A}, P_3) = Adv(\mathcal{A}, P_2).$$

Experiment P_4

In this experiment, we consider an instance C^i (or G^j) that receives an adversarially-generated message in a $Send_2$ (or $Send_1$) oracle call. In this case, if C^i (or G^j) accepts, then the experiment is halted, and the adversary is said to have succeeded. This certainly improves the probability of success of the adversary.

Lemma Appendix A.4

For every polynomial-time adversary $\mathcal{A}$making Q_{send} oracle calls of type Send to different instances,

$$Adv(\mathcal{A}, P_3) = Adv(\mathcal{A}, P_4).$$

At this point, we have given random session keys to all the accepted instances that receive *Execute* or *Send* oracle calls. We next proceed to bound the adversary's success probability in P_4. The following lemma shows that the adversary's success probability in the experiment P_4 is negligible.

Lemma Appendix A.5

For every polynomial-time adversary $\mathcal{A}$making Q_{send} oracle calls of type Send to different instances, $Q_{send} \leq |D|$,

$$Adv(\mathcal{A}, P_4) \leq \frac{2Q_{send}}{|\mathcal{D}|} + 2Q_{send}Adv^{rsa}(O(t)) + \frac{2Q_{send}Q_{oh}}{\phi(n)} + \frac{Q_{send}}{2^{k-1}} + \frac{Q_{send}}{2^{79}},$$

where Q_{oh} denotes the number of random oracle calls, and t is the running time of $\mathcal{A}$.

Proof. Let Q_{send_1} and Q_{send_2} denote the number of $Send_1$ and $Send_2$ oracle calls made by the adversary in experiment P_4, respectively. We consider the following two cases:

Case 1: Consider an instance C^i receives an adversarially-generated message (n', e', r_2, z, y_2) in a $Send_2$ oracle. Assume that C^i returns (n, e, r_1) in a $Send_0$ oracle. After receiving (n', e', r_2, z, y_2), C^i first decrypts y_2 to obtain x_2, then queries the random oracle H on (pw, x_2, C, G, n, e, n', e', r_1, r_2, y_2) and receives w from H. Without lose of generality, we assume that $gcd(w, n) = 1$. Then, C^i computes $x_1 = (w^{-1} \cdot z)^d \bmod n$ and $c_1 = b_1^{e'} \bmod n'$, where $b_1 \in Z_{n'}^*$. C^i queries H_1 on (x_1, C, G, n, e, n', e', r_1, r_2, y_2, z, c_1) and returns the reply (denoted by μ) to the adversary $\mathcal{A}$. To succeed in this case, $\mathcal{A}$ must generate a number η which is equal to the output of the random oracle H_2 on (x_1, C, G, n, e, n', e', r_1, r_2, y_2, z, c_1, c_2). Without the knowledge of x_1, the probability for $\mathcal{A}$ to generate η is just 2^{-k}. Let p_{x_1} denote the probability that $\mathcal{A}$ can recover the integer x_1. The adversary's success probability in this case is bounded by

$$\Pr[Succ] \leq Q_{send_2}(p_{x_1} + 2^{-k}).$$

If z was selected by $\mathcal{A}$ at random from Z_n^*, then similar to the proof of Lemma A.1, we can prove that p_{x_1} is bounded by

$$p_{x_1} \leq Adv^{rsa}(O(t)) + \frac{Q_{oh}}{\phi(n)}.$$

Next, assume that z was generated by $\mathcal{A}$ as follows: $\mathcal{A}$ selected two random numbers $x_1, x_2 \in Z_n^*$, as well as a candidate password $pw' \in \mathcal{D}$, $\mathcal{A}$ queries the random oracle H on (pw', x_2, C, G, n, e, n', e', r_1, r_2, y_2) and receives the reply w, then $\mathcal{A}$ computed $z = x_1^e \cdot w \bmod n$. In this scenario, if $\mathcal{A}$ guesses the correct password $pw = pw'$, then $\mathcal{A}$ succeeds. If $\mathcal{A}$ guesses an invalid password $pw \neq pw'$, then z can be treated as a random number in Z_n^*. Hence, we have

$$p_{x_1} \leq \frac{1}{|\mathcal{D}|} + Adv^{rsa}(O(t)) + \frac{Q_{oh}}{\phi(n)}.$$

The adversary's success probability in Case 1 is upper bounded by

$$\Pr[Succ] \le \frac{Q_{send_2}}{|\mathcal{D}|} + Q_{send_2} Adv^{rsa}(O(t)) + \frac{Q_{send_2} Q_{oh}}{\phi(n)} + \frac{Q_{send_2}}{2^k}.$$

Case 2: Consider an instance G^j receives an adversarially-generated message (C, n, e, r_1) in a $Send_1$ oracle, where n is an odd integer, and e is odd prime. The instance G^j sends (C, n, e, n, e, r_1) to the server. The server replies (r_2, z) according to the protocol description. To succeed in this case, $\mathcal{A}$ must send back a number μ which is equal to the output of the random oracle H_1 on (x_1, C, G, n, e, n', e', r_1, r_2, z, c_1). Without the knowledge of x_1, the probability for $\mathcal{A}$ to generate μ is just 2^{-k}. Let p_{x_1} denote the probability that $\mathcal{A}$ can recover the integer x_1.

Note that (n, e) was generated by $\mathcal{A}$. If gcd (e, $\varphi(n)$) = 1, then $\mathcal{A}$ can compute $w = H$ (pw', x_2, C, G, n, e, n', e', r_1, r_2, y_2) using a guessing password pw'. Then, congruence $z = x_1^e \cdot w \bmod n$ has a unique solution because gcd (e, $\varphi(n)$) = 1. If $\mathcal{A}$ guesses the correct password $pw = pw'$, then $\mathcal{A}$ can obtain x_1 correctly. If $\mathcal{A}$ does not guess the correct password, then $\mathcal{A}$ will not succeed. On the other hand, if gcd (e, $\varphi(n)$) $\neq$ 1, since we require that e is an 80-bit prime, then the congruence $y_2 = x_2^e \bmod n$ has e solutions. In order to recover the correct x_1, the adversary needs to find out the correct x_2. As is shown in Section 3, the probability to find out the correct x_2 is $1/2^{80}$, which is negligible.

Hence, the adversary's success probability in Case 2 is bounded by

$$\Pr[Succ] \le \frac{Q_{send_1}}{|\mathcal{D}|} + \frac{Q_{send_1}}{2^k} + \frac{Q_{send_1}}{2^{80}}.$$

From the above analysis, it can be concluded that the adversary's success probability in experiment P_4 is upper bounded by

$$\begin{aligned}\Pr[Succ] &\le \frac{Q_{send}}{|\mathcal{D}|} + Q_{send_2} Adv^{rsa}(O(t)) + \frac{Q_{send_2} Q_{oh}}{\phi(n)} + \frac{Q_{send}}{2^k} + \frac{Q_{send_1}}{2^{80}} \\ &\le \frac{Q_{send}}{|\mathcal{D}|} + Q_{send} Adv^{rsa}(O(t)) + \frac{Q_{send_2} Q_{oh}}{\phi(n)} + \frac{Q_{send}}{2^k} + \frac{Q_{send}}{2^{80}}.\end{aligned}$$

Since $Q_{send} \le |D|$, we have $\frac{Q_{send}}{|\mathcal{D}|} \le 1$. Therefore,

$$\begin{aligned}Adv(\mathcal{A}, P_4) &= 2\Pr[Succ] - 1 \\ &\le \frac{2Q_{send}}{|\mathcal{D}|} + 2Q_{send} Adv^{rsa}(O(t)) + \frac{2Q_{send} Q_{oh}}{\phi(n)} + \frac{Q_{send}}{2^{k-1}} + \frac{Q_{send}}{2^{79}}.\end{aligned}$$

This completes the proof of Lemma A.5.

By combining Lemma A.1 to Lemma A.5, we get the announced result.

Appendix B. Proof of Theorem 3.3

Consider a malicious gateway $\mathcal{A}$ generates a message (C, n, e, n', e', r_1). The malicious gateway sends the message to the server. The server replies (r_2, z) according to the protocol description. To succeed in this case, the malicious gateway must send back a number μ which is equal to the output of the random oracle H_1 on (x_1, C, G, n, e, n', e', r_1, r_2, y_2, z, c_1). Otherwise, the on-line impersonation attack will be detected by the authentication server. Without the knowledge of x_1, the probability for $\mathcal{A}$ to generate μ is just 2^{-k}.

Let p_{x_1} denote the probability that the malicious gateway can recover the integer x_1.

Note that (n, e) was generated by $\mathcal{A}$. If gcd (e, $\varphi(n)$) = 1, then $\mathcal{A}$ can compute $w = H$ (pw', x_2, C, G, n, e, n', e', r_1, r_2, y_2) using a guessing password pw'. Then, congruence $z = x_1^e \cdot w \bmod n$ has a unique solution because gcd (e, $\varphi(n)$) = 1. If $\mathcal{A}$ guesses the correct password $pw = pw'$, then $\mathcal{A}$ can obtain x_1 correctly. If $\mathcal{A}$ does not guess the correct password, then $\mathcal{A}$ will not succeed. On the other hand, if gcd (e, $\varphi(n)$) $\neq$ 1, since we require that e is an 80-bit prime, then the congruence $y_2 = x_2^e \bmod n$ has e solutions. In order to recover the correct x_1, the adversary needs to find out the correct x_2. As is shown in Section 3, the probability to find out the correct x_2 is $1/2^{80}$, which is negligible.

Hence, the adversary's success probability in violating the resistance to undetectable on-line dictionary attacks is bounded by

$$Adv_{\mathcal{P},\mathcal{D}}^{ake-uoda}(\mathcal{A}) \le \frac{Q_{send}}{|\mathcal{D}|} + \frac{Q_{send}}{2^k} + \frac{Q_{send}}{2^{80}}.$$

Acknowledgements

The authors would like to thank the anonymous referees for their helpful comments. This study was supported by the National High Technology Research and Development Program of China (No. 2009AA01Z417) and Key Scientific and Technological Project of Henan Province (No. 092101210502).

Competing interests

The authors declare that they have no competing interests.

Received: 29 January 2011 Accepted: 10 November 2011
Published: 10 November 2011

References

1. Y Ding, P Horster, Undetectable on-line password guessing attacks. ACM Oper Syst Rev. **29**, 77–86 (1995). doi:10.1145/219282.219298
2. SM Bellovin, M Merritt, Encrypted key exchange: password-based protocols secure against dictionary attacks, in *IEEE Symp on Security and Privacy 1992*, 72–84 (1992)
3. S Patel, Number theoretic attacks on secure password schemes, in *Proc IEEE Symposium on Security and Privacy*, Oakland, CA (May 5-7, 1997)
4. M Abdalla, O Chevassut, PA Fouque, D Pointcheval, A simple threshold authenticated key exchange from short secrets, ASIACRYPT 2005, LNCS, Springer, Heidelberg **3788**, 566–584 (2005)
5. M Abdalla, P Fouque, D Pointcheval, Password-based authenticated key exchange in the three-party setting. PKC 2005, LNCS, Springer, Heidelberg **3386**, 65–84 (2005)
6. M Abdalla, M Izabachene, D Pointcheval, Anonymous and transparent gateway-based password-authenticated key exchange. CANS2008, LNCS, Springer, Heidelberg **5339**, 133–148 (2008)
7. M Abdalla, D Pointcheval, Interactive Diffie-Hellman assumptions with applications to password-Based Authentication. FC 2005, LNCS, Springer, Heidelberg **3570**, 341–356 (2005)

8. JW Byun, DH Lee, JI Lim, Security analysis and improvement of a gateway-oriented password-based authenticated key exchange protocol. IEEE Commun Lett. **10**(9), 683–685 (2006). doi:10.1109/LCOMM.2006.1714545
9. M Bellare, D Pointcheval, P Rogaway, Authenticated key exchange secure against dictionary attacks. EUROCRYPT 2000, LNCS, Springer, Heidelberg **1807**, 139–155 (2000)
10. S Lucks, Open key exchange: how to defeat dictionary attacks without encrypting public keys, in *Proc of Security Protocol Workshop. LNCS*, vol. 1361. Springer, Heidelberg 79–90 (1997).
11. P MacKenzie, S Patel, R Swaminathan, Password-authenticated key exchange based on RSA. SIACRYPT 2000, LNCS, Springer, Heidelberg **1976**, 599–613 (2000)
12. MX Zhang, New approaches to password authenticated key exchange based on RSA. ASIACRYPT 2004, LNCS, Springer, Heidelberg **3329**, 230–244 (2004)
13. S Park, J Nam, S Kim, D Won, Efficient password-authenticated key exchange based on RSA. CT-RSA 2007, LNCS, Springer, Heidelberg **4377**, 309–323 (2007)
14. TY Youn, YH Park, C Kim, J Lim, Weakness in a RSA-based password authenticated key exchange protocol. Inf Process Lett. **108**, 339–342 (2008). doi:10.1016/j.ipl.2008.06.002
15. KA Shim, Cryptanalysis and enhancement of modified gateway-oriented password-based authenticated key exchange protocol. IEICE Trans Fund. **E91-A**(12), 3837–3839 (2008). doi:10.1093/ietfec/e91-a.12.3837
16. Ej Yoon, KY Yoo, An optimized gateway-oriented password-based authenticated key exchange protocol. IEICE Trans Fund. **E93-A**(4), 850–853 (2010). doi:10.1587/transfun.E93.A.850
17. L Lincoln, Symmetric private information retrieval via ho-momorphic probabilistic encryption. PhD thesis. http://www.cs.rit.edu/7Elbl6598/thesis/Lincoln full document.pdf (2006)
18. S Shin, K Kobara, H Imai, An RSA-based leakage-resilient authenticated key exchange protocol secure against replacement attacks, and its extensions. IEICE Trans Fund. **E93-A**(6), 1086–1101 (2010). doi:10.1587/transfun.E93.A.1086
19. MX Zhang, New approaches to password authenticated key exchange based on RSA. http://eprint.iacr.org. Cryptology ePrint Archive, Re-

doi:10.1186/1687-1499-2011-162
Cite this article as: Wei *et al.*: **Anonymous gateway-oriented password-based authenticated key exchange based on RSA.** *EURASIP Journal on Wireless Communications and Networking* 2011 **2011**:162.