New Approaches to Password Authenticated Key Exchange based on RSA[*]

Muxiang Zhang
Verizon Communications, Inc.
40 Sylvan Road, Waltham, MA 02451, USA
muxiang.zhang@verizon.com

August 18, 2004

Abstract

We investigate efficient protocols for password-authenticated key exchange based on the RSA public-key cryptosystem. To date, most of the published protocols for password-authenticated key exchange were based on Diffie-Hellman key exchange. It seems difficult to design efficient password-authenticated key exchange protocols using RSA and other public-key cryptographic techniques. In fact, many of the proposed protocols for password-authenticated key exchange based on RSA have been shown to be insecure; the only one that remains secure is the SNAPI protocol. Unfortunately, the SNAPI protocol has to use a prime public exponent e larger than the RSA modulus n. In this paper, we present a new password-authenticated key exchange protocol, called *PEKEP*, which allows using both large and small prime numbers as RSA public exponents. Based on number-theoretic techniques, we show that the new protocol is secure against the *e-residue attack*, a special type of off-line dictionary attack against RSA-based password-authenticated key exchange protocols. We also provide a formal security analysis of PEKEP under the RSA assumption and the random oracle model. On the basis of PEKEP, we present a computationally-efficient key exchange protocol to mitigate the burden on communication entities.

Key words: password authentication, off-line dictionary attack, public-key cryptography

1 Introduction

The design of authentication and key exchange protocol is usually based on the assumption that entities either share or own some secret data (called keys) which are drawn from a space so large that an adversary can not enumerate, either on-line or off-line, all possible keys in the space. In practice, however, cryptographic keys may often be substituted by human-memorable passwords consisting of only six to ten characters. The consequence is the proliferation of the so-called exhaustive guessing or dictionary attacks against many password-based systems [27]. Password guessing attacks have been around for so long, it seems paradoxical that strong authentication using only small passwords would be possible. In 1992, Bellovin and Merritt [5] showed that such paradoxical protocols did exist. They presented a family of protocols known as *Encrypted Key Exchange*, or *EKE*. By using a combination of symmetric and asymmetric (public-key) cryptographic techniques, EKE provides insufficient information for an adversary to verify a guessed password and thus defeats off-line dictionary attacks. Following EKE, a number of protocols for password-based authentication and key

[*]An extended abstract of this paper will appear in *Advances in Cryptology–ASIACRYPT 2004 Proceedings.*

exchange have been proposed, e.g., [2, 6, 8, 10, 11, 14, 16, 17, 18, 26]. A comprehensive list of such protocols can be found in Jablon's research link [15].

Password-authenticated key exchange protocols are attractive for their simplicity, convenience, and strength against off-line dictionary attacks. Unlike other public-key based key exchange protocols such as SSL, the EKE-like protocols do not rely on the existence of a public key infrastructure (PKI). This is desirable in many environments where the deployment of a public key infrastructure is either not possible or would be overly complex. Over the last decade, many researchers have investigated the feasibility of implementing EKE using different public-key cryptosystems such as RSA, ElGamal, and Diffie-Hellman key exchange. Nonetheless, most of the well-known and secure variants of EKE are based on Diffie-Hellman key exchange. It seems that EKE works well with Diffie-Hellman key exchange, but presents subtleties when implemented with RSA and other public-key cryptosystems. In fact, many of the proposed protocols for password-authenticated key exchange based on RSA have been shown to be insecure [5, 23, 21]; the only one that remains secure is the SNAPI protocol developed by Mackenzie, et al. [21]. Unfortunately, the SNAPI protocol has to use a prime public exponent e larger than the RSA modulus n. This may render the SNAPI protocol impractical in resource limited platform due to the massive computation involved in the public-key encryption as well as the primality test of large public exponent.

1.1 Overview of Our Results

In this paper, we investigate efficient RSA-based password-authenticated key exchange protocols that can use both large and small primes as RSA public exponent, but without inducing large communication overhead on communication entities. For this purpose, we develop a new protocol for password-authenticated key exchange based on RSA. The new protocol, called *PEKEP*, involves two entities, Alice and Bob, who share a short password and Alice possesses a pair of RSA keys, n, e and d, where $ed \equiv 1 \pmod{\phi(n)}$. Unlike the protocol SNAPI, however, the new protocol PEKEP allows Alice to select both large and small primes for the RSA public exponent e. In the protocol PEKEP, Bob does not need to verify if e is relatively prime to $\phi(n)$, and furthermore, Bob does not have to test the primality of a large public exponent selected by Alice. Thus, the protocol PEKEP improves on SNAPI by reducing the cost of primality test of RSA public exponents. The protocol PEKEP also improves on the interactive protocols of [28, 25, 9] by reducing the size of messages communicated between Alice and Bob. Based on number-theoretic techniques, we prove that the protocol PEKEP is secure against the e-residue attack as described in [5, 23]. We also provide a formal security analysis of PEKEP under the RSA assumption and the random oracle model.

To further reduce the computational load on entities, we present a computationally-efficient key exchange protocol (called *CEKEP*) in this paper. The protocol CEKEP is derived from PEKEP by adding two additional flows between Alice and Bob. With the two additional flows, we show that the probability for an adversary to launch a successful e-residue attack against CEKEP is less than or equal to ε, where ε is a small number (e.g., $0 < \varepsilon \leq 2^{-80}$) selected by Bob. In the protocol CEKEP, the computational burden on Bob includes two modular exponentiations, each having an exponent of $\mathcal{O}(\lceil \log_2 \varepsilon^{-1} \rceil)$ bits. When $\varepsilon = 2^{-80}$, for example, the computational burden on Bob is lighter than that in a Diffie-Hellman based password-authenticated key exchange protocol. In the Diffie-Hellman based EKE variant, Bob needs to compute two modular exponentiations, each having an exponent of at least 160 bits. We show that the computational burden on Bob could be reduced further if Bob caches a hashed version of Alice's public key used in previous communication sessions.

It is worth pointing out that both protocols, PEKEP and CEKEP, do not require public parameters; Alice and Bob only need to establish a shared password *in advance* and do not need to establish other common parameters such as a prime number p and a generator g of the cyclic group modulo p. This is appealing in environments where entities have insufficient resources to generate or validate public parameters with certain properties, e.g., primality.

1.2 Related Work

In 1989, Lomas et al. [19] introduced the first password-authenticated key exchange protocol resistant to off-line dictionary attacks. Their protocol, however, assumed that one of the entities knows the other entity's public key and thus is not a strictly password-only protocol. This type of protocol was further studied by Gong [12] and by Halevi and Krawczyk [13].

The EKE protocol developed by Bellovin and Merritt is the first password-authenticated key exchange protocol that does not require one of the entities (say, Bob) to know the public key of the other entity (say, Alice). Following EKE, many researchers have provided a variety of extensions to achieve additional goals, e.g., protection against server compromise [6, 18, 26]. In their original paper [5], Bellovin and Merritt investigated the feasibility of implementing EKE using three different types of public-key cryptographic techniques: RSA, ElGamal, and Diffie-Hellman key exchange. They pointed out that EKE is only suitable for implementation using Diffie-Hellman key exchange. The problem with the EKE variant implemented using RSA is that Bob does not know the public key, (n, e), of Alice and thus can not verify whether e is relatively prime to $\phi(n)$. This fosters the so-called e-residue attack as described in [5]. Based on number-theoretic techniques, Patel [23] further investigated the security of the RSA-based EKE variant and concluded that simple modifications of EKE would not prevent the RSA-based EKE variant from off-line dictionary attacks. In 1997, Lucks [20] proposed an RSA-based password-authenticated key exchange protocol (called OKE) which was claimed to be secure against the e-residue attack. Later, Mackenzie et al. [21] found that the OKE protocol is still subject to the e-residue attack. In [21], Mackenzie et al. proposed an RSA-based password-authenticated key exchange protocol (SNAPI) and provided a formal security proof in the random oracle model. The SNAPI protocol mandates that the public exponent e be a prime number larger than the RSA modulus n. This ensures that e is relatively prime to $\phi(n)$.

To avoid using large public exponents, Zhu et al. [28] proposed an "interactive" protocol which is revised from an idea of [5]. In the interactive protocol, Bob sends to Alice a number of messages encrypted using Alice's public key. If Alice can successfully decrypt the encrypted messages, then Bob is ensured that the encryption based on Alice's public key is a permutation. In [25], Wong et al. revised the interactive protocol of [28] to reduce the message size involved in the interactive protocol. Recently, Catalano et al. [9] proposed an interactive protocol similar to that of [25] and provided a security proof in the random oracle model. A drawback of the interactive protocols [28, 25, 9] is the large communication overhead involved in the verification of RSA public key.

The rest of the paper is organized as follows. In Section 2, we review basic concepts of number theory used throughout this paper. We provide an overview of the security model for password-authenticated key exchange in Section 3. We present the protocol PEKEP in Section 4 and investigate its security against e-residue attacks. We present the protocol CEKEP in Section 5 and pursue formal security analysis of PEKEP and CEKEP in Section 6. We conclude in Section 7.

2 Preliminaries

Let $\{0,1\}^k$ denote the set of binary strings of length k and $\{0,1\}^*$ denote the set of binary strings of finite length. Without confusion, we sometimes use s_1, s_2 to denote the concatenation of two strings s_1 and s_2. A real-valued function $\epsilon(k)$ of non-negative integers is called *negligible* (in k) if for every $c > 0$, there exists $k_0 > 0$ such that $\epsilon(k) \leq 1/k^c$ for all $k > k_0$.

For every positive integer n, $n > 1$, it is well know that n can be expressed as a product of nontrivial powers of distinct primes, i.e., $n = p_1^{a_1} p_2^{a_2} \ldots p_r^{a_r}$, where $p_1, p_2, \ldots, p_r$ are primes and $a_1, a_2, \ldots, a_r$ are positive integers. Up to a rearrangement of the prime powers, this *prime-power* *factorization* is unique. Let $\mathbb{Z}_n$ denote the set of non-negative integers less than n and $\mathbb{Z}_n^*$ denote the set consisting of integers in $\mathbb{Z}_n$ that are relatively prime to n. The number of integers in $\mathbb{Z}_n^*$ is equal to the *Euler phi-function* $\phi(n)$.

Let a, b, and n be integers such that $n > 0$ and $\gcd(a,n) = c$. If $c \nmid b$, then the congruence $ax \equiv b \pmod{n}$ has no solution. If $c \mid b$, the congruence $ax \equiv b \pmod{n}$ has exactly c incongruent solutions modulo n. Let x_0 denote one of the solutions, then the c incongruent solutions are given by

$$x = x_0 + t(n/c), \quad t = 0, 1, \ldots, c-1. \tag{1}$$

Let g and n be positive integers relatively prime to each other. The least positive integer i such that $g^i \equiv 1 \pmod{n}$ is called the *order of g modulo n*. If the order of g is equal to $\phi(n)$, then g is called a *primitive root of n*. It is known (see [1, 24]) that a positive integer n, $n > 1$, possesses a primitive root if and only if $n = 2, 4, p^t$ or $2p^t$, where p is an odd prime and t is a positive integer. If n has a primitive root g, then the integers $g^0, g^1, g^2, \ldots, g^{\phi(n)-1}$ form a cyclic group under the modulo n multiplication. Due to this fact, we see that if a is a positive integer relatively prime to n, then there exists a unique integer i, $0 \le i \le \phi(n) - 1$, such that $a = g^i \bmod n$. The integer i is called the *index of a to the base g modulo n*, and is denoted by $\mathrm{ind}_g a$. With this notation, we have $a = g^{\mathrm{ind}_g a} \bmod n$.

If n and e are positive integers and a is an integer relatively prime to n, then we say that a is an *e-th power residue of n* if the congruence $x^e \equiv a \pmod{n}$ has a solution. If n has a primitive root, then a is an e-th power residue of n if and only if $a^{\phi(n)/b} \equiv 1 \pmod{n}$, where $b = \gcd(e, \phi(n))$.

3 Security Model

We consider two-party protocols for authenticated key-exchange using human-memorable passwords. In its simplest form, such a protocol involves two entities, say *Alice* and *Bob* (denoted by A and B), both possessing a secret password drawn from a small password space $\mathcal{D}$. Based on the password, Alice and Bob can authenticate each other and upon a successful authentication, establish a session key which is known to nobody but the two of them. There is present an active adversary, denoted by $\mathcal{A}$, who intends to defeat the goal for the protocol. The adversary has full control of the communications between Alice and Bob. She can deliver messages out of order and to unintended recipients, concoct messages of her own choosing, and create multiple instances of entities and communicate with these instances in parallel sessions. She can also enumerate, *off-line*, all the passwords in the password space $\mathcal{D}$. She can even acquire session keys of accepted entity instances. Our formal model of security for password-authenticated key exchange protocols is based on that of [2]. In the following, we review the operations of the adversary and formulate the definition of security. For details as well as motivations behind the model, we refer the readers to [2].

Initialization. Let I denote the identities of the protocol participants. Elements of I will often be denoted A and B (Alice and Bob). We emphasis that A and B are variables ranging over I and not fixed members of I. Each pair of entities, $A, B \in I$, are assigned a password w which is randomly selected from the password space $\mathcal{D}$. The initialization process may also specify a set of cryptographic functions (e.g., hash functions) and establish a number of cryptographic parameters.

Running the Protocol. Mathematically, a protocol Π is a probabilistic polynomial-time algorithm which determines how entities behave in response to received input. For each entity, there may be multiple instances running the protocol in parallel. We denote the i-th instance of entity A as Π_A^i. The adversary $\mathcal{A}$ can make queries to any instance; she has an endless supply of Π_A^i oracles ($A \in I$ and $i \in \mathbb{N}$). In response to each query, an instance updates its internal state and gives its output to the adversary. At any point in time, the instance may accept and possesses a session key sk, a session id sid, and a partner id pid. The query types, as defined in [2], include:

- $\mathsf{Send}(A, i, M)$: This sends message M to instance Π^i_A. The instance executes as specified by the protocol and sends back its response to the adversary. Should the instance accept, this fact, as well as the session id and partner id will be made visible to the adversary.
- $\mathsf{Execute}(A, i, B, j)$: This call carries out an honest execution between two instances Π^i_A and Π^j_B, where $A, B \in I, A \neq B$ and instances Π^i_A and Π^j_B were not used before. At the end of the execution, a transcript is given to the adversary, which logs everything an adversary could see during the execution (for details, see [2]).
- $\mathsf{Reveal}(A, i)$: The session key sk^i_A of Π^i_A is given to the adversary.
- $\mathsf{Test}(A, i)$: The instance Π^i_A generates a random bit b and outputs its session key sk^i_A to the adversary if $b = 1$, or else a random session key if $b = 0$. This query is allowed only once, at any time during the adversary's execution.
- $\mathsf{Oracle}(M)$: This gives the adversary oracle access to a function h, which is selected at random from some probability space Ω. The choice of Ω determines whether we are working in the standard model, or in the random-oracle model (see [2] for further explanations).

Note that the Execute query type can be implemented by using the Send query type. The Execute query type reflects the adversary's ability to passively eavesdrop on protocol execution. As well shall see, the adversary shall learn nothing about the password or the session key from such oracle calls. The Send query type allows the adversary to interact with entity instances and to carry out an active man-in-the-middle attack on the protocol execution.

Definition. Let Π^i_A and Π^i_B, $A \neq B$, be a pair of instances. We say that Π^i_A and Π^i_B are *partnered* if both instances have accepted and hold the same session id sid and the same session key sk. Here, we define the sid of Π^i_A (or Π^i_B) as the concatenation of all the messages sent and received by Π^i_A (or Π^i_B). We say that Π^i_A is *fresh* if: i) it has accepted; and ii) a Reveal query has not been called either on Π^i_A or on its partner. With these notions, we now define the advantage of the adversary $\mathcal{A}$ in attacking the protocol. Let Succ denote the event that $\mathcal{A}$ asks a single Test query on a fresh instance, outputs a bit b', and $b' = b$, where b is the bit selected during the Test query. The advantage of the adversary $\mathcal{A}$ is defined as $\mathsf{Adv}^{ake}_{\mathcal{A}} = 2Pr(\mathsf{Succ}) - 1$.

It is clear that a polynomial-time adversary $\mathcal{A}$ can always gain an advantage close to 1 if we do not limit her capability to perform on-line password-guessing attacks. In such an attack, the adversary picks a password π as her guess and then impersonates as an instance Π^i_A to start the protocol towards another instance Π^j_B. By observing the decision of Π^j_B (i.e., accepts or rejects), the adversary can test the correctness of the guessed password π. Furthermore, by analyzing, off-line, the transcript of the execution, the adversary may be able to test passwords other than π. For a secure protocol, we expect that the adversary can only test a single password in each on-line password-guessing attack. As suggested in [10], we use the Send queries to count the number of on-line guesses performed by the adversary. We only count *one* Send query for each entity instance, that is, if the adversary sends two Send queries to an entity instance, it should still count as a single password guess. Based on this idea, we have the following definition of secure password-authenticated key exchange protocol, which is the same as in [10].

Definition 1 *A protocol Π is called a secure password-authenticated key exchange protocol if for every polynomial-time adversary $\mathcal{A}$ that makes at most Q_{send} ($Q_{send} \leq |\mathcal{D}|$) queries of* Send *type to different instances, the following two conditions are satisfied:*

(1) *Except with negligible probability, each oracle call* $\mathsf{Execute}(A, i, B, j)$ *produces a pair of partnered instances* Π_A^i *and* Π_B^j.

(2) $\mathsf{Adv}_{\mathcal{A}}^{ake} \leq Q_{send}/|\mathcal{D}| + \epsilon$, *where* $|\mathcal{D}|$ *denotes the size of the password space and* ϵ *is a negligible function of security parameters.*

The first condition basically says that when the adversary carries out an honest execution between two instances Π_A^i and Π_B^j ($A \neq B$), both instances accept and hold the same session key and the same session id. The second condition means that the advantage of the adversary is at most negligibly more than $Q_{send}/|\mathcal{D}|$ if she sends at most Q_{send} queries of Send type to different entity instances, or equivalently, if she interacts on-line with at most Q_{send} entity instances using the Send query type. As noticed in [10], the bound $Q_{send}/|\mathcal{D}|$ for $\mathcal{A}$'s advantage is actually optimal when $Q_{send} \leq |\mathcal{D}|$.

4 Password Enabled Key Exchange Protocol

In this section, we present a new protocol, called *Password Enabled Key Exchange Protocol*, or simply, *PEKEP*. It is an RSA-based password-authenticated key exchange protocol, but allows using both large and small prime numbers as RSA public exponents. Let A and B denote the identities of Alice and Bob who share a password w drawn from a password space $\mathcal{D}$. Alice has also generated a pair of RSA keys n, e and d, where n is a large positive integer (e.g., $n > 2^{1023}$) equal to the product of two primes of (roughly) the same size, e is a positive integer relatively prime to $\phi(n)$, and d is a positive integer such that $ed \equiv 1 \pmod{\phi(n)}$. The encryption function is define by $E(x) = x^e \bmod n$, $x \in \mathbb{Z}_n$. The decryption function is $D(x) = x^d \bmod n$. For a positive integer m, we define E^m recursively as

$$E^m(x) = E(E^{m-1}(x)) = x^{e^m} \mod n.$$

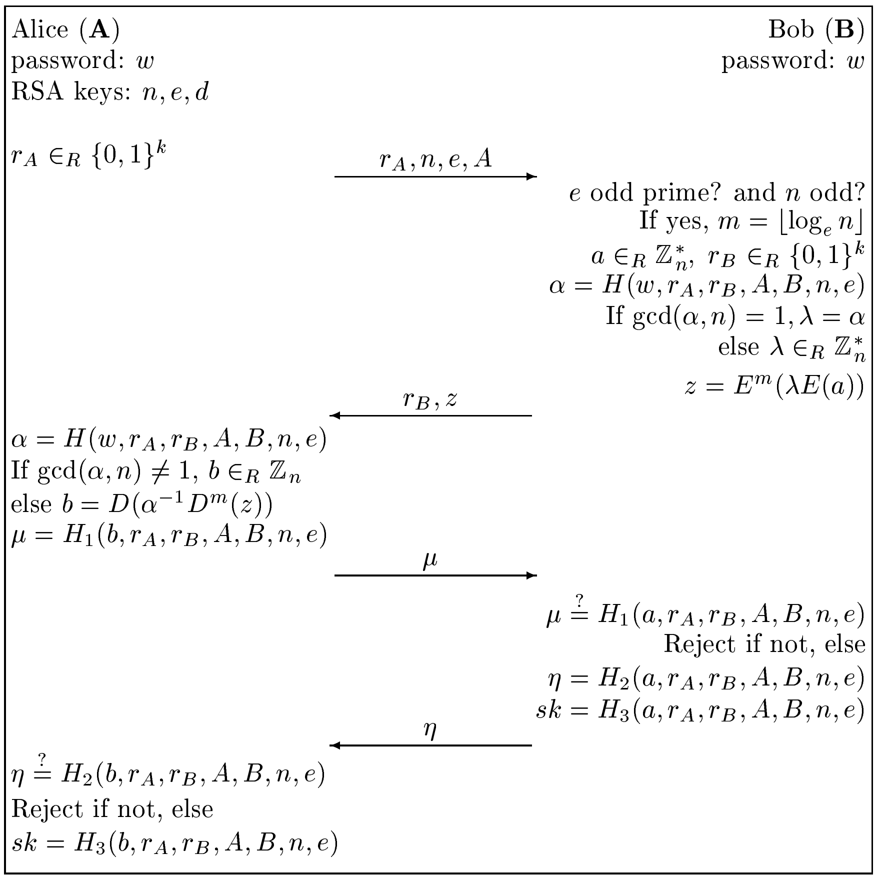
Likewise,

$$D^m(x) = D(D^{m-1}(x)) = x^{d^m} \mod n.$$

Define hash functions $H_1, H_2, H_3 : \{0,1\}^* \to \{0,1\}^k$ and $H : \{0,1\}^* \to \mathbb{Z}_n$, where k is a security parameter, e.g., $k = 160$. Note that H can be implemented using a standard hash function $h : \{0,1\}^* \to \{0,1\}^\ell$, where ℓ is the length of n, i.e., $\ell = \lceil \log_2 n \rceil$. On input $x, H(x) = h(x)$, if $h(x) < n$, and $H(x) = h(x) - \lceil n/2 \rceil$ if else. Assume that h is a random function, then for any integer $b \in \mathbb{Z}_n$, it can be proved that $|Pr(H(x) = b) - \frac{1}{n}| < 2^{-\ell}$; the bias is negligible. In the following, we assume that H_1, H_2, H_3 and H are independent random functions.

The protocol PEKEP is described in Fig. 1. Alice starts the protocol by sending her public key (n, e) and a random number $r_A \in_R \{0,1\}^k$ to Bob. Bob verifies if e is an odd prime and n is an odd integer. Bob may also verify that the integer n is large enough, e.g., $n > 2^{1023}$. If e is not an odd prime or n is not an odd integer, Bob rejects; otherwise, Bob computers an integer $m = \lfloor \log_e n \rfloor$ and selects two random numbers $a \in_R \mathbb{Z}_n^*$, and $r_B \in_R \{0,1\}^k$. Bob then computes $\alpha = H(w, r_A, r_B, A, B, n, e)$ and checks if $\gcd(\alpha, n) = 1$. If $\gcd(\alpha, n) \neq 1$, Bob assigns a random number of $\mathbb{Z}_n^*$ to λ; otherwise, Bob assigns α to λ. Next, Bob computes $z = E^m(\lambda E(a))$ and sends r_B and z to Alice. Subsequently, Alice computes α using her password w and checks if α and n are relatively prime. If $\gcd(\alpha, n) \neq 1$, Alice assigns a random number of $\mathbb{Z}_n$ to the variable b. If $\gcd(\alpha, n) = 1$ and z is an e^m-th power residue of n, Alice sets $b = D(\alpha^{-1} D^m(z))$. Next, Alice and Bob authenticate each other using a and b and generate a session key sk upon successful authentication.

In the protocol PEKEP, both Alice and Bob intend to reject when they detect the event $\gcd(\alpha, n) \neq 1$. To avoid leaking any information about this event, Alice and Bob use random numbers to compute their responses μ, z, and η. If $n = pq$ is the product of two large primes of about the same size, then the probability of such an event is negligible. When $\gcd(\alpha, n) = 1$, Alice and Bob agree on a secret



Alice (**A**)
password: w
RSA keys: n, e, d
$r_A \in_R \{0,1\}^k$
r_A, n, e, A
Bob (**B**)
password: w
e odd prime? and n odd?
If yes, $m = \lfloor \log_e n \rfloor$
$a \in_R \mathbb{Z}_n^*$, $r_B \in_R \{0,1\}^k$
$\alpha = H(w, r_A, r_B, A, B, n, e)$
If $\gcd(\alpha, n) = 1, \lambda = \alpha$
else $\lambda \in_R \mathbb{Z}_n^*$
$z = E^m(\lambda E(a))$
r_B, z
$\alpha = H(w, r_A, r_B, A, B, n, e)$
If $\gcd(\alpha, n) \neq 1$, $b \in_R \mathbb{Z}_n$
else $b = D(\alpha^{-1} D^m(z))$
$\mu = H_1(b, r_A, r_B, A, B, n, e)$
μ
$\mu \overset{?}{=} H_1(a, r_A, r_B, A, B, n, e)$
Reject if not, else
$\eta = H_2(a, r_A, r_B, A, B, n, e)$
$sk = H_3(a, r_A, r_B, A, B, n, e)$
η
$\eta \overset{?}{=} H_2(b, r_A, r_B, A, B, n, e)$
Reject if not, else
$sk = H_3(b, r_A, r_B, A, B, n, e)$

Figure 1: Password Enabled Key Exchange Protocol (PEKEP)

number $b = a$ and thus can use the secret number to authenticate each other and establish a shared session key.

Note that, when e is a prime number larger than n, Bob sets $m = 0$. In this case, the run of PEKEP is nearly identical to that of SNAPI. A minor difference between the two protocols is that, in SNAPI, Alice rejects immediately when she detects the event $\gcd(\alpha, n) \neq 1$; while in PEKEP, Alice sends a random number as her response to Bob and expects Bob to reject. As mentioned above, the event $\gcd(\alpha, n) \neq 1$ occurs with negligible probability if $n = pq$ is the product of two large primes of about the same size. In the protocol PEKEP, Bob only verifies if the public exponent e is an odd prime and the RSA modulus n is an odd integer; Bob does not verify if e is relatively prime to $\phi(n)$. This may foster the so-called e-residue attack as described in [5, 23]. In the e-residue attack, an adversary, say, *Eva*, selects $\pi_0 \in \mathcal{D}$ as her guess of Alice's password. She also selects an odd prime number e and an odd integer n such that $e \mid \phi(n)$, i.e., (n, e) is not a valid RSA public key. Then Eva impersonates as Alice and starts the protocol PEKEP by sending r_E, n, e, A to Bob, where $r_E \in \{0,1\}^k$ is a random number generated by Eva. After receiving r_B and z from Bob, Eva Computes μ and sends it back to Bob. If Bob accepts, then Eva has a successful guess of Alice's password (i.e., π_0). If Bob rejects, on the other hand, Eva excludes her guess π_0 from the password space $\mathcal{D}$. Furthermore, Eva may exclude more passwords by repeating, *off-line*, the following three steps:

1) Eva selects a password π from $\mathcal{D}$.

2) Eva computes $\alpha = H(\pi, r_E, r_B, A, B, n, e)$.

3) Eva tests if $\gcd(\alpha, n) = 1$. If not, Eva returns to step 1; otherwise, Eva verifies if the congruence $(\alpha x^e)^{e^m} \equiv z \pmod{n}$ has a solution in $\mathbb{Z}_n^*$. If the congruence has a solution, Eva returns to step 1. If the congruence has no solution in $\mathbb{Z}_n^*$, then Eva knows that π is not the password of Alice. Next, Eva excludes π from $\mathcal{D}$ and returns to step 1.

We say that Eva succeeds if she can exclude more than one password in the e-residue attack as described above. In the following, we show that the protocol PEKEP is secure against e-residue attacks.

Theorem 1 *Let n, $n > 1$, be an odd integer with prime-power factorization $n = p_1^{a_1} p_2^{a_2} \dots p_r^{a_r}$. Let m be a non-negative integer and e an odd prime such that for any prime-power $p_i^{a_i}$ of the factorization of n, $e^{m+1} \nmid \phi(p_i^{a_i}), 1 \le i \le r$. If z is an e^m-th power residue of n, then for any $\lambda \in \mathbb{Z}_n^*$, the congruence $(\lambda x^e)^{e^m} \equiv z \pmod{n}$ has solutions in $\mathbb{Z}_n^*$.*

Proof. To prove that $(\lambda x^e)^{e^m} \equiv z \pmod{n}$ has a solution in $\mathbb{Z}_n^*$, we only need to prove that, for each prime power $p_i^{a_i}$ of the factorization of n, the following congruence

$$(\lambda x^e)^{e^m} \equiv z \pmod{p_i^{a_i}} \tag{2}$$

has a solution in $\mathbb{Z}_{p_i^{a_i}}^*$.

Let $n_i = p_i^{a_i}, 1 \le i \le r$. Then $\phi(n_i) = p_i^{a_i - 1}(p_i - 1)$. Since n is odd, p_i is an odd prime. Thus, the integer n_i possesses a primitive root. Let g be a primitive root of n_i, that is, $g^{\phi(n_i)} = 1 \bmod n_i$, and for any $0 \le i, j \le \phi(n_i) - 1, i \ne j$, $g^i \ne g^j \bmod n_i$. Let $\gcd(e^m, \phi(n_i)) = e^c, 0 \le c \le m$. We consider the following two cases:

(1) If $c = 0$, then e and $\phi(n_i)$ are relatively prime. In this case, it is clear that the congruence $(\lambda x^e)^{e^m} \equiv z \pmod{n_i}$ has a unique solution in $\mathbb{Z}_{n_i}^*$.

(2) Next, we consider the case that $1 \le c \le m$. Since z is an e^m-th power residue of n, the congruence $y^{e^m} \equiv z \pmod{n}$ has solutions in $\mathbb{Z}_n^*$. By the Chinese Remainder Theorem, the following congruence

$$y^{e^m} \equiv z \pmod{n_i} \tag{3}$$

has solutions in $\mathbb{Z}_{n_i}^*$. Let $\mathrm{ind}_g z$ denote the index of z to the base g modulo n_i and let $y \in \mathbb{Z}_{n_i}^*$ be a solution of (3), then

$$g^{e^m \mathrm{ind}_g y - \mathrm{ind}_g z} \equiv 1 \pmod{n_i}.$$

Since the order of g modulo n_i is $\phi(n_i)$, we have

$$e^m \mathrm{ind}_g y \equiv \mathrm{ind}_g z \pmod{\phi(n_i)} \tag{4}$$

Also since $\gcd(e^m, \phi(n_i)) = e^c$, equation (4) has exactly e^c incongruent solutions modulo $\phi(n_i)$ when taking $\mathrm{ind}_g y$ as variable. This indicates that equation (3) has e^c solutions in $\mathbb{Z}_{n_i}^*$. Let y_0 denote one of the solutions of (3), by (1), the e^c incongruent solutions of (4) are given by

$$\mathrm{ind}_g y = \mathrm{ind}_g y_0 + t\phi(n_i)/e^c \pmod{\phi(n_i)}, \quad 0 \le t \le e^c - 1. \tag{5}$$

For any $\lambda \in \mathbb{Z}_n^*$, we have

$$\mathrm{ind}_g y - \mathrm{ind}_g \lambda \equiv \mathrm{ind}_g y_0 - \mathrm{ind}_g \lambda + t\phi(n_i)/e^c \pmod{\phi(n_i)}, \quad 0 \le t \le e^c - 1.$$

Under the condition that $e^{m+1} \nmid \phi(n_i)$, it is clear that $e \nmid \phi(n_i)/e^c$. Hence, $\phi(n_i)/e^c \equiv 1 \pmod{e}$. So, there exists an integer t, $0 \le t \le e - 1$, such that

$$\mathrm{ind}_g y_0 - \mathrm{ind}_g \lambda + t\phi(n_i)/e^c \equiv 0 \pmod{e},$$

which implies that there exists an integer $y \in \mathbb{Z}_{n_i}^*$, such that $y^{e^m} \equiv z \pmod{n_i}$ and $y\lambda^{-1}$ is an e-th power residue of n_i. Therefore, equation (2) has a solution in $\mathbb{Z}_{n_i}^*$, which proves the theorem. $\square$

In PEKEP, Bob sets m equal to $\lfloor \log_e n \rfloor$. Thus, for every prime-power $p_i^{a_i}$ of the factorization of n, we have $e^{m+1} > n \geq p_i^{a_i}$. By Theorem 1, for any $\lambda \in \mathbb{Z}_n^*$, the congruence $(\lambda x^e)^{e^m} \equiv z \pmod{n}$ has a solution in $\mathbb{Z}_n^*$, where z is the e-th power residue computed by Bob. Hence, by repeating (off-line) the three steps as described previously, Eva could not exclude any password from the space $\mathcal{D}$. So, the protocol PEKEP is secure against e-residue attacks. In Section 6, we will provide a formal analysis of PEKEP within the security model described in Section 3.

At the beginning of PEKEP, Bob needs to test the primality of the public exponent e selected by Alice. When e is small, e.g., $e = 17$, the primality test only induces moderate overhead on Bob. When e is large (e.g., $e > 2^{512}$), however, the computational load for the primality test would be tremendous. In the following, we show that primality test of large public exponents by Bob could be avoided with slight modification of PEKEP. In the protocol PEKEP, Bob can actually select a small prime number e' (e.g., $e' = 3$) and replaces Alice's public key (n, e) by (n, e'), that is, Bob computes m, α, z, η, sk using (n, e') instead of Alice's public key (n, e). Theorem 1 demonstrates that the replacement does not lead to e-residue attacks, even if e' is not relatively prime to $\phi(n)$. So, when the public exponent e received from Alice exceeds a threshold, Bob replaces e by a smaller prime number e' $(2 < e' < e)$ of his own choosing. Bob sends r_B, z, and e' to Alice in the second flow. After receiving e' from Bob, Alice tests if e' is relatively prime to $\phi(n)$. If $\gcd(e', \phi(n)) \neq 1$, Alice sends a random number $\mu \in \{0,1\}^k$ to Bob; Alice may select a smaller prime number for e in the next communication session. If $\gcd(e', \phi(n)) = 1$, Alice replaces her decryption key by d' and then proceeds as specified in Fig. 1, where $e'd' \equiv 1 \pmod{\phi(n)}$. If n is the product of two safe primes p and q, that is, $(p-1)/2$ and $(q-1)/2$ are also prime numbers, then for every odd prime number e' less than $(p-1)/2$ and $(q-1)/2$, e' is always prime to $\phi(n)$.

In each run of PEKEP, Bob computes $m + 1$ encryptions using Alice's public key (n, e), where $m = \lfloor \log_e n \rfloor$. The computation time for the $m + 1$ encryptions is $\mathcal{O}((\log_2 n)^3)$, which means that the computational load on Bob is about the same as that in SNAPI. As discussed above, however, Bob does not have to perform primality test of large public exponents. Hence, the protocol PEKEP still improves on SNAPI by reducing the cost of primality test of RSA public exponents. Since Alice has knowledge of $\phi(n)$, she only needs to perform two decryptions in each run of PEKEP; one using the decryption key $d_1 = d$ and another using the decryption key $d_2 = d^m \bmod \phi(n)$. Note that the computational load on Bob is high even when e is small. In Section 5, we present a computationally-efficient key exchange protocol which greatly reduces the computational load on Bob.

5 Computationally-Efficient Key Exchange Protocol

In this section, we present a *Computationally-Efficient Key Exchange Protocol* (CEKEP), which is described in Fig. 2. The protocol CEKEP is based on PEKEP, but the number of encryptions performed by Bob is less than $\lfloor \log_e n \rfloor$, where (n, e) is the public key of Alice. In the protocol CEKEP, Bob selects a small number ε, $0 < \varepsilon \leq 2^{-80}$, which determines the probability of a successful e-residue attack against the protocol CEKEP. Alice starts the protocol CEKEP by sending her public key n, e and two random numbers $\rho, r_A \in_R \{0,1\}^k$ to Bob. Bob verifies if e is an odd prime and n is an odd integer. If not, Bob rejects. Else, Bob computers an integer m based on e and ε as $m = \lceil \log_e \varepsilon^{-1} \rceil$. Then Bob selects a random number $\varrho \in_R \{0,1\}^k$ such that $\gamma = H(n, e, \rho, \varrho, A, B, m)$ is relatively prime to n. Bob sends ϱ and m to Alice. After receiving ϱ and m, Alice computes $u = D^m(\gamma)$ and sends it back to Bob. Subsequently, Bob verifies if Alice has made the right decryption, i.e.,

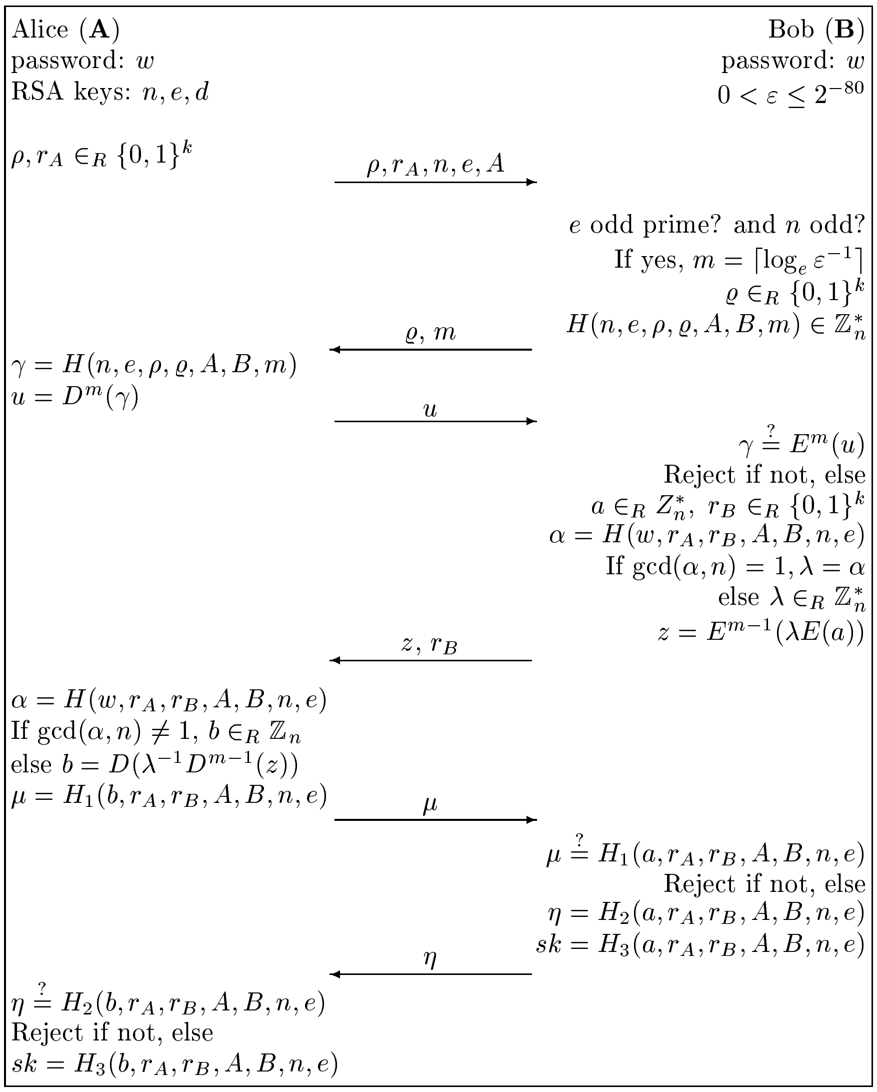

Alice (**A**)
password: w
RSA keys: n, e, d
Bob (**B**)
password: w
$0 < \varepsilon \le 2^{-80}$
$\rho, r_A \in_R \{0,1\}^k$
ρ, r_A, n, e, A
e odd prime? and n odd?
If yes, $m = \lceil \log_e \varepsilon^{-1} \rceil$
$\varrho \in_R \{0,1\}^k$
$H(n, e, \rho, \varrho, A, B, m) \in \mathbb{Z}_n^*$
$\varrho,\ m$
$\gamma = H(n, e, \rho, \varrho, A, B, m)$
$u = D^m(\gamma)$
u
$\gamma \overset{?}{=} E^m(u)$
Reject if not, else
$a \in_R Z_n^*,\ r_B \in_R \{0,1\}^k$
$\alpha = H(w, r_A, r_B, A, B, n, e)$
If $\gcd(\alpha, n) = 1, \lambda = \alpha$
else $\lambda \in_R \mathbb{Z}_n^*$
$z = E^{m-1}(\lambda E(a))$
$z,\ r_B$
$\alpha = H(w, r_A, r_B, A, B, n, e)$
If $\gcd(\alpha, n) \ne 1$, $b \in_R \mathbb{Z}_n$
else $b = D(\lambda^{-1} D^{m-1}(z))$
$\mu = H_1(b, r_A, r_B, A, B, n, e)$
μ
$\mu \overset{?}{=} H_1(a, r_A, r_B, A, B, n, e)$
Reject if not, else
$\eta = H_2(a, r_A, r_B, A, B, n, e)$
$sk = H_3(a, r_A, r_B, A, B, n, e)$
η
$\eta \overset{?}{=} H_2(b, r_A, r_B, A, B, n, e)$
Reject if not, else
$sk = H_3(b, r_A, r_B, A, B, n, e)$

Figure 2: Computationally-Efficient Key Exchange Protocol (CEKEP)

$E^m(u) = \gamma$. If $\gamma \ne E^m(u)$, Bob rejects. Else, Alice and Bob executes the rest of the protocol as in PEKEP.

A major difference between CEKEP and PEKEP is that the protocol CEKEP adds two additional flows between Alice and Bob. Through the two flows, Alice and Bob establish a random number $\gamma \in \mathbb{Z}_n^*$. Then Alice decrypts the random number γ repeatedly m times. If the m repeated decryption is correct, i.e., $\gamma = E^m(u)$, then it can be concluded that, except with probability as small as e^{-m}, the integer e^m does not divide $\phi(p_i^{a_i})$ for every prime-power $p_i^{a_i}$ of the factorization of n.

Theorem 2 *Let n, $n > 1$, be an odd integer with prime-power factorization $n = p_1^{a_1} p_2^{a_2} \ldots p_r^{a_r}$. Let m be a positive integer and e an odd prime. If there exists a prime power, say $p_i^{a_i}$, of the factorization of n such that $e^m \mid \phi(p_i^{a_i})$, then for an integer γ randomly selected from $\mathbb{Z}_n^*$, the probability that γ is an e^m-th power residue of n is less than or equal to e^{-m}.*

Proof. Let $n_i = p_i^{a_i}$ be a prime power of the factorization of n such that $e^m \mid \phi(n_i)$. Since n is odd, n_i possesses a primitive root. Let g be a primitive root of n_i. For an integer γ randomly selected from $\mathbb{Z}_n^*$, let $\mathrm{ind}_g\gamma$ denote the index of γ to the base g modulo n_i. Then γ is an e^m-th power residue of n_i if and only if the congruence $x^{e^m} \equiv \gamma \pmod{n_i}$ has a solution, or equivalently, if and only if

$$g^{e^m \,\mathrm{ind}_g x - \mathrm{ind}_g\gamma} \equiv 1 \pmod{n_i},$$

which is equivalent to

$$e^m \mathrm{ind}_g x \equiv \mathrm{ind}_g\gamma \pmod{\phi(n_i)}.$$

Since $e^m \mid \phi(n_i)$, γ is an e^m-th power residue of n_i if and only if $e^m \mid \mathrm{ind}_g\gamma$.

Let $n_i' = n/n_i$, then n_i and n_i' are relatively prime. For any integer $\beta \in \mathbb{Z}_n^*$, it is clear that $\beta \mod n_i$ and $\beta \mod n_i'$ are integers of $\mathbb{Z}_{n_i}^*$ and $\mathbb{Z}_{n_i'}^*$, respectively. On the other hand, for two integers $\alpha_1 \in \mathbb{Z}_{n_i}^*$ and $\alpha_2 \in \mathbb{Z}_{n_i'}^*$, by the Chinese Remainder Theorem, there is an unique integer $\alpha \in \mathbb{Z}_n^*$, such that $\alpha \equiv \alpha_1 \pmod{n_i}$, and $\alpha \equiv \alpha_2 \pmod{n_i'}$. So, the number of integers $\alpha \in \mathbb{Z}_n^*$ which satisfy the congruence $\alpha \equiv \alpha_1 \pmod{n_i}$ is $\phi(n_i')$. If γ is randomly selected from $\mathbb{Z}_n^*$, then for any integer s, $0 \le s \le \phi(n_i) - 1$, we have

$$Pr(g^s = \gamma \mod n_i) = \phi(n_i')/\phi(n) = 1/\phi(n_i),$$

which implies that

$$Pr(\mathrm{ind}_g\gamma = s) = 1/\phi(n_i).$$

Hence,

$$\begin{aligned} Pr(e^m \mid \mathrm{ind}_g\gamma) &= \sum_{e^m \mid s,\, 0 \le s < \phi(n_i)} Pr(\mathrm{ind}_g\gamma = s) \\ &= \phi(n_i)e^{-m}/\phi(n_i) \\ &= e^{-m} \end{aligned}$$

which indicates that, for an integer γ randomly selected from $\mathbb{Z}_n^*$, the probability that γ is an e^m-th power residue of n_i is equal to e^{-m}. So, the probability that γ is an e^m-th power residue of n does not exceed e^{-m}. $\square$

Theorem 2 demonstrates that, if there exits a prime-power $p_i^{a_i}$ of the factorization of n such that $e^m \mid \phi(p_i^{a_i})$, then for a random number $\gamma \in \mathbb{Z}_n^*$, the probability that Alice can decrypt γ repeatedly m times is less than or equal to e^{-m}. If the number u received from Alice satisfies the equation $E^m(u) = u^{e^m} = \gamma \mod n$, i.e., γ is an e^m-power residue of n, then Bob is ensured with probability greater than or equal to $1 - e^{-m}$ that, for every prime-power $p_i^{a_i}$ of the factorization of n, $e^m \nmid \phi(p_i^{a_i})$. Since $m = \lceil \log_e \varepsilon^{-1} \rceil$, $e^{-m} \le \varepsilon$. By Theorem 1, it is clear that the probability for an adversary to launch a successful e-residue attack against CEKEP is upper-bounded by ε

In the protocol CEKEP, Alice proves to Bob in an interactive manner (via flow 2 and flow 3) that for every prime-power $p_i^{a_i}$ of the factorization of n, $e^m \nmid \phi(p_i^{a_i})$. In the interactive procedure, however, only one decrypted message is sent from Alice to Bob. The communication overhead on Alice and Bob is greatly reduced in comparison with that in [28, 25, 9]. In CEKEP, the computational burden on Bob includes two modulo exponentiations, i.e., $u^{e^m} \bmod n$ and $(\lambda a^e)^{e^{m-1}} \bmod n$, where $m = \lceil \log_e \varepsilon^{-1} \rceil$. When $e < \varepsilon^{-1}$, each modulo exponentiation has an exponent consisting of $\mathcal{O}(\lceil \log_2 \varepsilon^{-1} \rceil)$ bits. The computation time for the two modulo exponentiations is $\mathcal{O}(2(\log_2 \varepsilon^{-1})(\log_2 n)^2)$. If $\varepsilon^{-1} \ll n$, then the computational load on Bob is greatly reduced in CEKEP in comparison with that in PEKEP (or in SNAPI). The parameter ε determines the computational load on Bob. It also determines the level of security against e-residue attacks. In practice, Bob can make a trade-off between the computational load and the security level offered by the protocol. When $\varepsilon = 2^{-80}$, for example, Bob needs to compute two modular exponentiation, each having an exponent of about 80 bits. In this case, the computational load on Bob is lighter than that in a Diffie-Hellman based password-authenticated key exchange protocol. In the Diffie-Hellman based EKE variant, for example, Bob needs to compute two modular exponentiation, each having an exponent of at least 160 bits.

The computational load on Bob could be reduced further if Bob maintains a cache in the protocol CEKEP (or PEKEP). The cache stores a hashed version of Alice's public key (n, e) used in a previous sessions, that is, $H_1(n, e, A)$. Initially, the cache is empty. In the next run of CEKEP (PEKEP), Bob computes the hashed value of the public key received from Alice and checks if it is in the cache. If it is in the cache, Bob sets $m = 1$ ($m = 0$ for PEKEP) and then directly sends r_B, z, m to Alice (Bob does not send ϱ, m to Alice). Otherwise, Bob and Alice behave as usual (i.e. as specified in Fig. 1 or Fig. 2) and at the end of a success run, Bob updates the cache using Alice's new public key. In general, Alice would most likely use the same RSA key pairs in many sessions, although for perfect forward secrecy, Alice would need to choose a new key pair in each session. In such a circumstance, Bob may compute a single RSA encryption in a run of CEKEP (PEKEP).

6 Formal Security Analysis

In this section, we analyze the security of PEKEP and CEKEP within the formal model of security given in Section 3. Our analysis is based on the random-oracle model of Bellare and Rogaway [4]. In this model, a hash function is modeled as an oracle which outputs a random number for each new query. If the same query is asked twice, identical answers are returned by the oracle. In our analysis, we also assume the intractability of the RSA problem.

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

$$\mathsf{Adv}_{\mathcal{C}}^{rsa}(t) = Pr(x^e = c \mod n : (e, d, n) \leftarrow GE(1^\ell), c \in_R \{0,1\}^\ell, x \leftarrow \mathcal{C}(1^\ell, c, e, n))$$

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

Under the above assumptions, we have the following Theorem 3. The proof of Theorem 3 is given in Appendix A.

Theorem 3 *Let $\mathcal{A}$ be an adversary which runs in time t and makes Q_{send}, $Q_{send} \leq |\mathcal{D}|$, queries of type* Send *to different instances. Then the adversary's advantage in attacking the protocol PEKEP is bounded by*

$$\mathsf{Adv}_{\mathcal{A}}^{ake} \leq \frac{Q_{send}}{|\mathcal{D}|} + (Q_{execute} + 3Q_{send})\mathsf{Adv}^{rsa}(\mathcal{O}(t)) + \mathcal{O}(\frac{(Q_{execute} + 2Q_{send})Q_{oh}}{2^k}),$$

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

It is easy to check that the protocol PEKEP satisfies the first condition of Definition 1. Theorem 3 indicates that the protocol PEKEP also satisfies the second condition of Definition 1 and hence is a secure password-authenticated key exchange protocol. Similarly, we can also show that the protocol CEKEP satisfies the two conditions of Definition 1. In summary, we have the following theorem 4.

Theorem 4 *Both protocols, PEKEP and CEKEP, are secure password-authenticated key exchange protocols under the RSA assumption and the random oracle model.*

We notice that the random oracle model in Theorem 4 is less desirable than a standard cryptographic assumption. To avoid the random oracle model, we could use the proof technique of [13], which require a public-key encryption scheme secure against chosen-ciphertext attacks. Unfortunately, the most commonly used RSA schemes (e.g. [3, 7]) which are secure against chosen-ciphertext attacks are also based on the random oracle model. Nevertheless, it is encouraging to see that efficient password-authenticated key exchange protocols with security proof in the random oracle model can be constructed without severe restriction on the public key of RSA.

7 Conclusion

In this paper, we investigate the design of RSA-based password-authenticated key exchange protocols that do not restrict the size of RSA public exponent. Based on number-theoretic techniques, we develop a password enabled key exchange protocol (PEKEP) which allows using both large and small primes as RSA public exponents. We show that the protocol PEKEP is secure against e-residue attacks. We also provide a formal security analysis of PEKEP under the RSA assumption and the random oracle model. Based on PEKEP, we develop a computationally-efficient key exchange protocol to mitigate the burden on communication entities. Both protocols, PEKEP and CEKEP, do not require public parameters; Alice and Bob only need to establish a shared password *in advance* and do not need to establish other common parameters such as a prime number p and a generator g of the cyclic group modulo p. This is appealing in environments where entities have insufficient resources to generate or validate public parameters with certain properties, e.g., primality.

References

[1] E. Bach and J. Shallit, *Algorithmic Number Theory*, vol. 1: *Efficient Algorithms*, MIT Press, 1997.

[2] M. Bellare, D. Pointcheval, and P. Rogaway, Authenticated key exchange secure against dictionary attack, *Advances in Cryptology - EUROCRYPT 2000 Proceedings*, Lecture Notes in Computer Science, vol. 1807, Springer-Verlag, 2000, pp. 139-155.

[3] M. Bellare and P. Rogaway, Optimal asymmetric encryption, *Advances in Cryptology - EUROCRYPT '94 proceedings*, Lecture Notes in Computer Science, vol. 950, Springer-Verlag, 1995, pp. 92–111.

[4] M. Bellare and P. Rogaway, Entity Authentication and key distribution, *Advances in Cryptology - CRYPTO'93 Proceedings*, Lecture Notes in Computer Science, vol. 773, Springer-Verlag, 1994, pp. 22-26.

[5] S. M. Bellovin and M. Merritt, Encrypted key exchange: Password-based protocols secure against dictionary attacks, *Proc. of the IEEE Symposium on Research in Security and Privacy*, Oakland, May 1992, pp. 72-84.

[6] S. M. Bellovin and M. Merritt, Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise, *Proc. of the 1st ACM Conference on Computer and Communications Security*, ACM, November 1993, pp. 244-250.

[7] D. Boneh, Simplified OAEP for the RSA and Rabin functions, *Advances in Cryptology - CRYPTO 2001 Proceedings*, Lecture Notes in Computer Science, vol. 2139, Springer-Verlag, 2001, pp. 275-291.

[8] V. Boyko, P. MacKenzie, and S. Patel, Provably secure password authenticated key exchange using Diffie-Hellman, *Advances in Cryptology - EUROCRYPT 2000 Proceedings*, Lecture Notes in Computer Science, vol. 1807, Springer-Verlag, 2000, pp. 156-171.

[9] D. Catalano, D. Pointcheval, and T. Pornin, IPAKE: Isomorphisms for password-based authenticated key exchange, to appear in *CRYPTO 2004*.

[10] R. Gennaro and Y. Lindell, A framework for password-based authenticated key exchange, *Advances in Cryptology - EUROCRYPT 2003 Proceedings*, Lecture Notes in Computer Science, vol. 2656, Springer-Verlag, 2003, pp.524-542.

[11] O. Goldreich and Y. Lindell, Session-key generation using human passwords only, *Advances in Cryptology - CRYPTO 2001 Proceedings*, Lecture Notes in Computer Science, vol. 2139, Springer-Verlag, 2001, pp.408-432.

[12] L. Gong, Optimal authentication protocols resistant to password guessing attacks, *Proc. IEEE Computer Security Foundation Workshop*, June 1995, pp. 24-29.

[13] S. Halevi and H. Krawczyk, Public-key cryptography and password protocols, *Proc. of the Fifth ACM Conference on Computer and Communications Security*, 1998, pp. 122-131.

[14] D. Jablon, Strong password-only authenticated key exchange, *Computer Communication Review, ACM SIGCOMM*, vol. 26, no. 5, 1996, pp. 5-26.

[15] D. Jablon, http://www.integritysciences.com.

[16] J. Katz, R. Ostrovsky, and M. Yung, Efficient password-authenticated key exchange using human-memorable passwords, *Advances in Cryptology – EUROCRYPT 2001 Proceedings*, Lecture Notes in Computer Science, vol. 2045, Springer-Verlag, 2001.

[17] K. Kobara and H. Imai, Pretty-simple password-authenticated key-exchange under standard assumptions, *IEICE Trans.*, vol. E85-A, no. 10, 2002, pp. 2229-2237.

[18] T. Kwon, Authentication and key agreement via memorable passwords, *Proc. Network and Distributed System Security Symposium*, February 7-9, 2001.

[19] M. Lamos, L. Gong, J. Saltzer, and R. Needham, Reducing risks from poorly chosen keys, *Proc. of the 12th ACM Symposium on Operating System Principles, ACM Operating Systems Review*, 1989, pp. 14-18.

[20] S. Lucks, Open key exchange: How to defeat dictionary attacks without encrypting public keys, *Proc. Security Protocol Workshop*, Lecture Notes in Computer Science, vol. 1361, Springer-Verlag, 1997, pp. 79-90.

[21] P. MacKenzie, S. Patel, and R. Swaminathan, Password-authenticated key exchange based on RSA, *Advances in Cryptology—ASIACRYPT 2000 Proceedings*, Lecture Notes in Computer Science, vol. 1976, Springer-Verlag, 2000, pp. 599–613.

[22] A. Menezes, P. C. van Oorschot, S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[23] S. Patel, Number theoretic attacks on secure password schemes, *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 5-7, 1997.

[24] K. H. Rosen, *Elementary Number Theory and Its Applications*, 4th ed., Addison Wesley Longman, 2000.

[25] D. Wong, A. Chan, and F. Zhu, More efficient password authenticated key exchange based on RSA, *INDOCRYPT 2003 Proceedings*, Lecture Notes in Computer Science, vol. 2904, Springer-Verlag, 2003, pp. 375-387.

[26] T. Wu, The secure remote password protocol , *Proc. Network and Distributed System Security Symposium*, San Diego, March 1998, pp. 97-111.

[27] T. Wu, A real-world analysis of Kerberos password security, *Proc. Network and Distributed System Security Symposium*, February 3-5, 1999.

[28] F. Zhu, D. Wong, A. Chan, and R. Ye, RSA-based password authenticated key exchange for imbalanced wireless networks, *Proc. Information Security Conference 2003 (ISC'02)*, Lecture Notes in Computer Science, vol. 2433, Springer-Verlag, 2002, pp.150-161.

A Proof of Theorem 3

We prove Theorem 3 using similar techniques as described in [10, 16]. 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 (not output by random oracles) for instances for which the Execute oracle is called. We then proceed 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 hybrid experiment, all the session keys are chosen uniformly at random. Thus, the adversary $\mathcal{A}$ can not distinguish them from random. We denote these hybrid experiments by $P_0, P_1, \ldots, P_4$ and by $\mathsf{Adv}(\mathcal{A}, P_i)$ the advantage of $\mathcal{A}$ when participating in experiment P_i.

Hybrid experiment P_0: This describes the real adversary attack. During the attack, the adversary $\mathcal{A}$ makes a number of oracle calls (i.e., Send, Execute, Reveal, and Test) as specified in section 3. In addition, the adversary $\mathcal{A}$ has access to four independent random oracles

$$H_1, H_2, H_3 : \{0,1\}^* \to \{0,1\}^k \quad \text{and} \quad H : \{0,1\}^* \to \mathbb{Z}_n.$$

Each random oracle H_i (or H) maintains a list of input-output pairs $(x_0, y_0), (x_1, y_1), \ldots$. On a new input x, H_i (or H) checks if x was queried before. If there exists x_i in the list such that $x = x_i$, then the random oracle returns the corresponding y_i as its reply. If x is not in the list, the random oracle generates a random number y and returns y as its reply. The random oracle adds the new pair (x, y) to the list. It is clear that $\mathsf{Adv}(\mathcal{A}) = \mathsf{Adv}(\mathcal{A}, P_0)$ and we wish to bound this advantage by $Q_{send}/|D| + \epsilon$, where ϵ is a negligible function and Q_{send} denotes the number of Send oracles that $\mathcal{A}$ makes to different instances.

Hybrid experiment P_1: In this experiment, the Execute oracle is modified so that the session keys of instances for which Execute is called are all chosen uniformly at random, that is, if the oracle $\mathsf{Execute}(A, i, B, j)$ is called between two instances Π^i_A and Π^j_B, then the session keys sk^i_A and sk^j_B are set equal to a random number selected from $\{0,1\}^k$, rather than the output of the random oracle H_3. In the following, we show that modifying the Execute oracle in this way affects the advantage of $\mathcal{A}$ by a negligible value.

Claim 1 *For every polynomial-time adversary $\mathcal{A}$ making $Q_{execute}$ oracle calls of type* Execute*,*

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

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

Proof. We prove this claim by showing how any advantage that $\mathcal{A}$ has in distinguishing P_1 from P_0 can be used to break RSA. First, note that if two Execute oracle calls generate the same set of random numbers, i.e., $r_A, r_B \in_R \{0,1\}^k$, and $a \in_R \mathbb{Z}_n^*$, then there are four instances sharing the same session key (generated using random oracle H_3) in experiment P_0. In this occurrence, $\mathcal{A}$ can distinguish P_1 and P_0 by simplify making the Reveal oracle calls. The probability of such occurrence is $Q_{execute}^2/(2^{2k}\phi(n))$, which is too small to be counted.

Fixing an oracle call $\mathsf{Execute}(A, i, B, j)$, let's assume that the random numbers $r_A, r_B \in_R \{0,1\}^k$, and $a \in_R \mathbb{Z}_n^*$ in this oracle call are not used by other oracle calls of type Execute. In experiment P_1, the session key sk_A^i (or sk_B^j) of Π_A^i and Π_B^j is the output of the random oracle H_3 on the input $(a, r_A, r_B, A, B, n, e)$. Without knowledge of a, the output of H_3 is indistinguishable from a random number uniformly selected from $\{0,1\}^k$. Hence, based on the transcript generated by $\mathsf{Execute}(A, i, B, j)$, the adversary $\mathcal{A}$ can distinguish P_0 and P_1 if and only if $\mathcal{A}$ can recover the integer a. Let p_a denote the probability that $\mathcal{A}$ recovers the integer a. To bound p_a, we consider the following two games G_1 and G_2.

Game G_1: the adversary $\mathcal{A}$ carries out an honest execution between two instances Π_A^i and Π_B^j as described below:

(1) Π_A^i sends out a random number $r_A \in_R \{0,1\}^k$ and an RSA public (n, e) (via $\mathcal{A}$) to Π_B^j.

(2) Π_B^j computers $m = \lfloor \log_e n \rfloor$ and selects two random numbers $a \in_R \mathbb{Z}_n^*$, and $r_B \in_R \{0,1\}^k$, then Π_B^j queries the random oracle H on $(w, r_A, r_B, A, B, n, e)$. Assume that the reply of H, denoted by α, is relatively prime to n. Next, Π_B^j computes $z = E^m(\alpha E(a))$ and sends r_B and z to Π_A^i.

(3) Π_A^i computes $b = D(\alpha^{-1}D^m(z))$ and queries the random oracle H_1 on $(b, r_A, r_B, A, B, n, e)$, then Π_A^i sends out the reply of H_1 (denoted by μ) to Π_B^j.

(4) After receiving μ from Π_A^i, Π_B^j queries the random oracle H_2 on $(a, r_A, r_B, A, B, n, e)$ and accepts. Next, Π_B^j sends the reply (denoted by η) of H_2 to Π_A^i.

(5) After receiving η from Π_B^j, Π_A^i accepts. The game ends here and the adversary $\mathcal{A}$ outputs her guess of the integer a.

Game G_2: this game is similar to game G_1 except that Π_A^i and Π_B^j do not query random oracles H_1 and H_2. Instances Π_A^i and Π_B^j execute according to the following steps:

(i) Π_A^i sends out a random number $r_A \in_R \{0,1\}^k$ and an RSA public (n, e) (via $\mathcal{A}$) to Π_B^j.

(ii) Π_B^j computers $m = \lfloor \log_e n \rfloor$ and selects two random numbers $a \in_R \mathbb{Z}_n^*$, and $r_B \in_R \{0,1\}^k$, then Π_B^j queries the random oracle H on $(w, r_A, r_B, A, B, n, e)$. Assume that the reply of H, denoted by α, is relatively prime to n. Next, Π_B^j computes $z = E^m(\alpha E(a))$ and sends r_B and z to Π_A^i.

(iii) After receiving r_B and z from Π_B^j, Π_A^i sends out a random number $\mu \in_R \{0,1\}^k$ to Π_B^j.

(iv) After receiving μ from Π_A^i, Π_B^j sends a random number $\eta \in_R \{0,1\}^k$ to Π_A^i and accepts.

(v) After receiving η from Π_B^j, Π_A^i accepts. The game ends here and the adversary $\mathcal{A}$ outputs her guess of the integer a.

Let $p_a(G_1)$ denote the probability that $\mathcal{A}$ makes a correct guess of a in game G_1. Likewise, let $p_a(G_2)$ denote the probability that $\mathcal{A}$ makes a correct guess of a in game G_2. It is clear that $p_a = p_a(G_1)$. In game G_1, $\mathcal{A}$ can select a random number $x \in \mathbb{Z}_n^*$ as her guess on a and verifies the correctness of x by comparing μ (or η) with the reply of the random oracle H_1 (or H_2) on $(x, r_A, r_B, A, B, n, e)$. Let $\mathsf{AskH}_{1,2}$ denote the event that $\mathcal{A}$ queries the random oracle H_1 or H_2 on $(a, r_A, r_B, A, B, n, e)$. Then, we have

$$\begin{aligned} p_a(G_1) &= p_a(G_1|\mathsf{AskH}_{1,2})Pr(\mathsf{AskH}_{1,2}) + p_a(G_1|\neg\mathsf{AskH}_{1,2})Pr(\neg\mathsf{AskH}_{1,2}) \\ &\leq Pr(\mathsf{AskH}_{1,2}) + p_a(G_1|\neg\mathsf{AskH}_{1,2}) \end{aligned}$$

If $\mathcal{A}$ did not query the random oracle H_1 or H_2 on $(a, r_A, r_B, A, B, n, e)$, then μ and η are indistinguishable from random numbers selected from $\{0,1\}^k$. Thus,

$$p_a(G_1|\neg\mathsf{AskH}_{1,2}) = p_a(G_2).$$

Let Q_{oh} denote the number of random oracle calls to H_1 and H_1 by $\mathcal{A}$. Then,

$$Pr(\mathsf{AskH}_{1,2}) = Q_{oh}/\phi(n).$$

Hence, it follows that

$$p_a = p_a(G_1) \leq p_a(G_2) + Q_{oh}/\phi(n). \tag{6}$$

In the following, we show that $p_a(G_2) \leq \mathsf{Adv}^{rsa}(\mathcal{O}(t))$. Given RSA public key (n, e) and integer $c \in_R \mathbb{Z}_n$, 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, in step (ii), Π_B^j computes $z = E^m(\alpha c)$. Algorithm $\mathcal{C}$ returns the output of $\mathcal{A}$ in game G_2. It is clear that if $\mathcal{A}$'s output (denoted by x) in game G_2 is correct, then x satisfies

$$(\alpha x^e)^{e^m} = z \mod n,$$

which means that x is equal to the RSA decryption of c. Hence,

$$p_a(G_2) = \mathsf{Adv}_{\mathcal{C}}^{rsa}(\mathcal{O}(t)) \leq \mathsf{Adv}^{rsa}(\mathcal{O}(t)). \tag{7}$$

By (6) and (7), we have

$$p_a \leq \mathsf{Adv}^{rsa}(\mathcal{O}(t)) + Q_{oh}/\phi(n). \tag{8}$$

Assume that $\mathcal{A}$ makes $Q_{execute}$ oracle calls of type $\mathsf{Execute}$ in the hybrid experiment P_1. Then

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

which completes the proof of Claim 1. □

At this point, the $\mathsf{Execute}$ oracle queries only provide negligible advantage to the adversary $\mathcal{A}$ because the session keys are chosen uniformly at random. We now proceed to show that the Send oracle calls are also not of too much help to the adversary. There are five Send oracles in the protocol PEKEP:

- $\mathsf{Send}_0(A, i)$: the instance Π_A^i generates a random number $r_A \in \{0,1\}^k$ as well as a pair of RSA public/private keys, e, d, n, and returns r_A, n, e, A to the adversary.

- $\mathsf{Send}_1(B, j, r_A, n, e, A)$: the instance Π_B^j tests if n is odd and e is odd prime. If not, Π_B^j rejects and returns a Reject notice to the adversary. Otherwise, Π_B^j sets $m = \lfloor \log_e n \rfloor$ and selects random numbers $a \in_B \mathbb{Z}_n^*$ and $r_B \in \{0,1\}^k$. Next, Π_B^j queries the random oracle H on $(w, r_A, r_B, A, B, n, e)$ and receives the reply α. If $\gcd(\alpha, n) \neq 1$, Π_B^j selects a random number λ from $\mathbb{Z}_n^*$; if else, Π_B^j sets λ equal to α. Last, Π_B^j returns $z = (\lambda a^e)^{e^m} \bmod n$ to the adversary.

- $\mathsf{Send}_2(A, i, r_B, z)$: the instance Π_A^i queries the random oracle H on $(w, r_A, r_B, A, B, n, e)$ and receives the reply α, where r_A, n, e was generated by Π_A^i for the oracle call $\mathsf{Send}_0(A, i)$. If $\gcd(\alpha, n) \neq 1$, Π_A^i selects a random number b from $\mathbb{Z}_n$, and if else, Π_A^i sets b equal to $(\alpha^{-1} z^{d^m})^d \bmod n$, where $ed = 1 \bmod \phi(n)$. Next, Π_A^i queries the random oracle H_1 on input $(b, r_A, r_B, A, B, n, e)$ and returns the reply (denoted by μ) of H_1 to the adversary.

- $\mathsf{Send}_3(B, j, \mu)$: the instance Π_B^j retrieves r_A, n, e, a, r_B from its internal state, where r_A, n, e were received from the $\mathsf{Send}_1(B, j, r_A, n, e, A)$ oracle call and a, r_B were generated during the reply to this oracle call. Then Π_B^j queries the random oracle H_1 on $(a, r_A, r_B, A, B, n, e)$. If the reply of H_1 is not equal to μ, Π_B^j returns a Reject notice to the adversary. Otherwise, Π_B^j accepts. In this case, Π_B^j queries the random oracles H_2 and H_3 on $(a, r_A, r_B, A, B, n, e)$. Then Π_B^j sets the session key sk_B^j equal to the reply of H_3 and returns the reply of H_2 to the adversary.

- $\mathsf{Send}_4(A, i, \eta)$: the instance Π_B^j first retrieves r_A, n, e, b, r_B from its internal state, where (r_A, n, e) was generated for the oracle call $\mathsf{Send}_0(A, i)$, r_B was received from $\mathsf{Send}_2(A, i, r_B, z)$ and b was computed during the reply to the oracle call $\mathsf{Send}_2(A, i, r_B, z)$. Then Π_A^i queries the random oracle H_1 on $(b, r_A, r_B, A, B, n, e)$. If the reply of H_1 is not equal to η, Π_A^i returns a Reject notice to the adversary. Otherwise, Π_A^i accepts and no message is returned to the adversary. In the case of acceptance, Π_A^i queries the random oracle H_3 on $(b, r_A, r_B, A, B, n, e)$ and sets the session key sk_A^i equal to the reply of H_3.

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 Π_A^i is said to have been Π_A^i-oracle-generated.

Hybrid experiment P_2: In this experiment, we consider an instance Π_B^j that receives a Π_A^i-oracle-generated message (r_A, n, e, A) in a Send_1 oracle call. If this occurs, then the experiment is modified as follows:

1. If both instances Π_B^j and Π_A^i accept, they are given the same random session key $sk \in_R \{0,1\}^k$.

2. If Π_B^j accepts but Π_A^i does not accept, then only Π_B^j receives a random session key and no session key is defined for Π_A^i.

We now show that the advantage of $\mathcal{A}$ in P_2 is close to its advantage in P_1.

Claim 2 *For every polynomial-time adversary $\mathcal{A}$ making Q_{send} oracle calls of type* Send *to different instances,*

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

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

Proof. Assume that Π_B^j returns (r_B, z) to the adversary after receiving a Π_A^i-oracle-generated message (r_A, n, e, A) in a Send_1 oracle call, where $z = (\lambda a^e)^{e^m} \bmod n, a \in_R \mathbb{Z}_n^*$, and $r_B \in_R \{0,1\}^k$. Except for a negligible probability $1/\phi(n)$, λ is equal to the the reply of the random oracle H on input $(w, r_A, r_B, A, B, n, e)$. Since the RSA public key (e, n) was generated by Π_A^i, not by $\mathcal{A}$, the private key d is not known to $\mathcal{A}$. As shown in the proof of Claim 1, the probability for $\mathcal{A}$ to recover the random number a is upper bounded by $\mathsf{Adv}^{rsa}(\mathcal{O}(t))$. So, except for a probability as small as $\mathsf{Adv}^{rsa}(\mathcal{O}(t))$, Π_B^j has received an Π_A^i-oracle-generated message μ in a Send_3 oracle when Π_B^j accepts. Similarly, if Π_A^i accepts, it has received an Π_B^j-oracle-generated message η in a Send_4 oracle call. If both Π_B^j and Π_A^i accept, they share the same session key which is equal to the output of the random oracle H_3 on $(a, r_A, r_B, A, B, n, e)$. Hence, the modification of the session keys of Π_B^j and Π_A^i affects the adversary's advantage by a value as small as $\mathsf{Adv}^{rsa}(\mathcal{O}(t))$. 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 $Q_{send}\mathsf{Adv}^{rsa}(\mathcal{O}(t))$. □

Hybrid experiment P_3: In this experiment, we consider an instance Π_A^i that receives a Π_B^j-oracle-generated message (r_B, z) in a Send_2 oracle call, while the instance Π_B^j has received a Π_A^i-oracle-generated message (r_A, n, e, A) in a Send_1 oracle call. In this case, if both instances Π_A^i and Π_B^j accept and their session keys were not replaced by a random session key in experiment P_2, they are given the same random session key $sk \in_R \{0,1\}^k$.

It is clear that the advantage of $\mathcal{A}$ in P_3 is the same as its advantage in P_2.

Claim 3 *For every polynomial-time adversary $\mathcal{A}$ making Q_{send} oracle calls of type* Send *to different instances,*

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

If an instance Π_A^i receives a Π_B^j-oracle-generated message (r_B, z) in a Send_2 oracle call but the message (r_A, n, e, A) received by Π_B^j in a Send_1 oracle call was not generated by Π_A^i, we treat the message (r_B, z) as adversarially-generated.

Hybrid experiment P_4: In this experiment, we consider an instance Π_A^i (or Π_B^j) that receives an adversarially-generated message in a Send_2 (or Send_1) oracle call. In this case, if Π_A^i (or Π_B^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.

Claim 4 *For every polynomial-time adversary $\mathcal{A}$ making Q_{send} oracle calls of type* Send *to different instances,*

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

At this point, we have given random session keys to all accepted instances that receive $\mathsf{Execute}$ or Send oracle calls. We next proceed to bound the adversary's success probability in P_4.

Claim 5 *For every polynomial-time adversary $\mathcal{A}$ making Q_{send}, $Q_{send} \le |\mathcal{D}|$, queries to the* Send_1 *and* Send_2 *oracles,*

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

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 Π_A^i that receives an adversarially-generated message (r_B, z) in a Send_2 oracle. Assume that Π_A^i returns (r_A, n, e, A) in a send_0 oracle, where (e, n) is RSA public key. After receiving the message (r_B, z), Π_A^i queries the random oracle H on $(w, r_A, r_B, A, B, n, e)$ and receives α from H. Without loss of generality, let's assume that $\gcd(\alpha, n) = 1$. Then Π_A^i computes $b = (\alpha^{-1} z^{d^m})^d$ and queries the random oracle H_1 on $(b, r_A, r_B, A, B, n, e)$. Next, Π_A^i returns the reply (denoted by μ) of H_1 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 $(b, r_A, r_B, A, B, n, e)$. Without knowledge of b, the probability for $\mathcal{A}$ to generate η is just 2^{-k}. Let p_b denote the probability that $\mathcal{A}$ can recover the integer b. The adversary's success probability in this case is bounded by

$$Pr(\mathsf{Succ}) \leq Q_{send_2}(p_b + 2^{-k}).$$

In the following, we bound p_b based on how z was generated by $\mathcal{A}$.

(i) If z was selected (by $\mathcal{A}$) at random from $\mathbb{Z}_n^*$, then similar to the proof of (8), we can prove that p_b is bounded by

$$p_b \leq \mathsf{Adv}^{rsa}(\mathcal{O}(t)) + Q_{oh}/\phi(n).$$

(ii) Next, assume that z was generated by $\mathcal{A}$ as follows: $\mathcal{A}$ selected a random number $a \in \mathbb{Z}_n^*$, as well as a password $w_i \in \mathcal{D}$, $\mathcal{A}$ queried the random oracle H on $(w_i, r_A, r_B, A, B, n, e)$ and received the reply λ_i, then $\mathcal{A}$ computed $z = (\lambda_i a^e)^{e^m} \bmod n$. In this scenario, the integer b is the solution of the following congruence

$$(\alpha x^e)^{e^m} = (\lambda_i a^e)^{e^m} \bmod n, \tag{9}$$

where α is the output of the random oracle H on $(w, r_A, r_B, A, B, n, e)$. If $\lambda_i = \alpha$, b is equal to a, which implies that $\mathcal{A}$ can recover b without making RSA decryption. If $\lambda_i \neq \alpha$, however, λ_i and α are independent random numbers and z can be treated as an random number of $\mathbb{Z}_n^*$. So, the probability for $\mathcal{A}$ to recover b is bounded by

$$\begin{aligned} p_b &\leq Pr(\lambda_i = \alpha) + \mathsf{Adv}^{rsa}(\mathcal{O}(t)) + Q_{oh}/\phi(n), \\ &\leq \frac{1}{|\mathcal{D}|} + \mathsf{Adv}^{rsa}(\mathcal{O}(t)) + \frac{Q_{oh}}{\phi(n)}. \end{aligned}$$

By (i) and (ii), the adversary's success probability in Case 1 is upper bounded by

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

Case 2: Consider an instance Π_B^j that receives an adversarially-generated message (r_A, n, e, A) in a Send_1 oracle, where n is an odd integer and e is odd prime. The instance Π_B^j sets $m = \lfloor \log_e n \rfloor$ and selects random numbers $a \in_B \mathbb{Z}_n^*$ and $r_B \in \{0,1\}^k$. Next, Π_B^j queries the random oracle H on $(w, r_A, r_B, A, B, n, e)$ and receives the reply α. Without loss of generality, assume that $\gcd(\alpha, n) = 1$. The instance Π_B^j returns $z = (\alpha a^e)^{e^m} \bmod n$ to the adversary. 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 $(a, r_A, r_B, A, B, n, e)$. Without knowledge of a, the probability for $\mathcal{A}$ to generate μ is just 2^{-k}. Let p_a denote the probability that $\mathcal{A}$ recovers the integer a.

Note that (e, n) was generated by $\mathcal{A}$. For every $\lambda \in \mathbb{Z}_n^*$ selected by $\mathcal{A}$, she can always find the solutions of the congruence

$$(\lambda x^e)^{e^m} = z \bmod n, \tag{10}$$

or equivalently, the following congruence

$$(xa^{-1})^{e^{m+1}} = (\alpha\lambda^{-1})^{e^m} \bmod n. \tag{11}$$

If $x = a$ is a solution of (11), then λ must be equal to α. Otherwise, for every prime power p^{a_i} of the factorization of n, $\phi(p_i^{a_i}) \mid e^m$, which leads to a contradiction. Thus, we have

$$p_a = Pr(\lambda = \alpha) = \frac{1}{|\mathcal{D}|}.$$

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

$$Pr(\mathsf{Succ}) \leq \frac{Q_{send_1}}{|\mathcal{D}|} + \frac{Q_{send_1}}{2^k}.$$

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

$$\begin{aligned} Pr(\mathsf{Succ}) &\leq \frac{Q_{send}}{|\mathcal{D}|} + Q_{send_2}\mathsf{Adv}^{rsa}(\mathcal{O}(t)) + \frac{Q_{send_2}Q_{oh}}{\phi(n)} + \frac{Q_{send}}{2^k} \\ &\leq \frac{Q_{send}}{|\mathcal{D}|} + Q_{send}\mathsf{Adv}^{rsa}(\mathcal{O}(t)) + \frac{Q_{send}Q_{oh}}{\phi(n)} + \frac{Q_{send}}{2^k} \end{aligned}$$

Note that $Q_{send} \leq |\mathcal{D}|$, we have $Q_{send}/|\mathcal{D}| \leq 1$. Therefore,

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

This completes the proof of Claim 5. □

By combining Claims 1 to 5, we have the following bound for $\mathcal{A}$'s advantage in the real attack,

$$\mathsf{Adv}(\mathcal{A}, P_0) \leq \frac{Q_{send}}{|\mathcal{D}|} + (Q_{execute} + 3Q_{send})\mathsf{Adv}^{rsa}(\mathcal{O}(t)) + \frac{(Q_{execute} + 2Q_{send})Q_{oh}}{\phi(n)} + \frac{Q_{send}}{2^{k-1}},$$

In general, $\phi(n) \leq 2^{-k}$. Hence, we have

$$\mathsf{Adv}(\mathcal{A}, P_0) \leq \frac{Q_{send}}{|\mathcal{D}|} + (Q_{execute} + 3Q_{send})\mathsf{Adv}^{rsa}(\mathcal{O}(t)) + \mathcal{O}(\frac{(Q_{execute} + 2Q_{send})Q_{oh}}{2^k}),$$

which completes the proof of Theorem 3. □